Universität Ulm

Fakultät für Informatik

Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

Thomas Bauer
Peter Dadam
Universität Ulm

Nr. 2000-02

Ulmer Informatik-Berichte

Februar 2000

Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

Thomas Bauer, Peter Dadam
Abteilung Datenbanken und Informationssysteme, Universität Ulm
{bauer, dadam}@informatik.uni-ulm.de

Zusammenfassung

Bei großen, unternehmensweiten Workflow- (WF) Anwendungen können die Belastung der WF-Server und das Kommunikationsaufkommen in den Teilnetzen zum Engpaß werden. In diesem Beitrag wird gezeigt, wie eine verteilte WF-Steuerung dergestalt realisiert werden kann, daß die Belastung der beteiligten Komponenten zur Ausführungszeit minimiert wird. Dazu kann die WF-Steuerung einer Instanz von einem WF-Server auf einen anderen migrieren. Ausgehend von den Bearbeiterzuordnungen wird zur Modellierungszeit für alle Aktivitäten jeweils der WF-Server ermittelt, durch dessen Wahl die Kommunikationskosten minimiert werden. Da Bearbeiterzuordnungen von vorangegangenen Aktivitäten abhängen können, ist eine statische Serverzuordnung nicht immer angebracht. Deshalb werden sog. logische Serverzuordnungsausdrücke eingeführt, durch die eine dynamische Serverzuordnung ohne aufwendige Laufzeitanalysen möglich wird.

1 Motivation und Einführung

Workflow-Management-Systeme (WfMSe) stellen eine vielversprechende Technologie für die Realisierung vorgangsorientierter, unternehmensweiter verteilter Anwendungssysteme dar. Sie sind in einer Phase, die mit den frühen Tagen relationaler Datenbanksysteme (RDBS) verglichen werden kann. Die eigentlichen Herausforderungen entstehen erst dann, wenn zu großen Systemen übergegangen wird. RDBS-Technologie zog Anwendungsentwickler und Benutzer mit einer semantisch höheren Query-Sprache, mit einem größeren Abstraktionsgrad, mit mehr Flexibilität (dynamisches Schema) und mit dem Potential zur Anfrageoptimierung an. Die Technologie mußte noch zeigen, daß ihre Funktionalität, ihre Systemanforderungen und im speziellen ihre Performanz ausreichend sind, um unternehmensweite Anwendungssysteme zu entwickeln.

Die Herausforderungen für WF-Technologie sind ähnlich: Es ist zu untersuchen, welche Funktionalität benötigt wird, um ein breites Spektrum realer WFs unterstützen zu können. Außerdem muß die benötige Performanz erreicht werden. Dabei ist zu untersuchen, wie WF-basierte Informations- und Anwendungssysteme entwickelt werden können, die mehrere Organisationseinheiten (OEen, vgl. Tabelle 1) umspannen und hunderte oder tausende von Benutzern bedienen (vgl. [KAGM96]). Im ADEPT-Projekt[1] werden beide Aspekte betrachtet. In diesem Beitrag konzentrieren wir uns auf den

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

Performanzaspekt.

Besonders kritisch für die Performanz von großen, unternehmensweiten, WF-basierten Anwendungssystemen ist die viele Kommunikation, die zwischen dem WF-Server und seinen Klienten notwendig ist. Oberhalb einer gewissen Schwelle ist der WF-Server alleine durch diese Kommunikation überlastet. Aus diesem Grund wurden in einigen Projekten Lösungen entwickelt, bei denen sich mehrere WF-Server diese Last teilen. Aber selbst wenn das Problem der Rechenleistung und Kommunikationsbandbreite des WF-Servers gelöst ist, kann es noch zur Überlastung des Kommunikationsnetzwerks kommen. Da diesem Aspekt bisher wenig Aufmerksamkeit gewidmet wurde, haben wir dieses Problem näher untersucht. Bei unserem Ansatz wird schon zur Modellierungszeit analysiert, mit welcher Wahrscheinlichkeit eine Aktivität zur Laufzeit in welchem Teilnetz ausgeführt werden wird. Eine WF-Instanz wird nicht mehr nur von einem WF-Server kontrolliert, sondern die Kontrolle der Instanz migriert von einem WF-Server zum anderen, wenn dies Kommunikation über Teilnetzgrenzen hinweg verhindert bzw. reduziert (siehe Abb. 1).

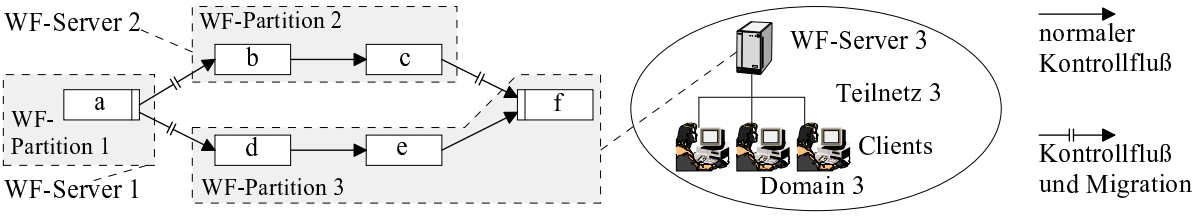


Abbildung 1 Partitionierung eines WF-Graphen und verteilte WF-Ausführung.

Erst Ergebnisse dieser Arbeit wurden in [BD97] veröffentlicht. Die konkreten Serverzuordnungen wurden schon zur Modellierungszeit berechnet. Die Berechnung basiert auf den Wahrscheinlichkeitsverteilungen (WVen) für die Aktivitätenausführung in den einzelnen Teilnetzen (diese werden wiederum von der Verteilung der Rollen der Benutzer abgeleitet, für Details siehe [BD97]). Die (Abschnitts-) Server eines WFs werden also komplett vor dessen Start ermittelt. Deshalb nennen wir diesen Ansatz „statisch definierte Migration“, im folgenden kurz „statische Migration“ genannt. Die statische Migration führt dann zu guten Ergebnissen, wenn Bearbeiterzuordnungen ($BearbZuordn$) der Art "$Rolle = Arzt$", "$Rolle = Pflegekraft \wedge Abteilung = Chirurgie$" oder "$Bearb = Dr.\ Brinkmann \vee Bearb = Dr.\ Frank$" verwendet werden, da in diesen Fällen die Menge der potentiellen Bearbeiter (als Voraussetzung für die Berechnung der Wahrscheinlichkeiten) bereits zur Modellierungszeit ermittelt werden kann.[2]

In der Praxis gibt es jedoch häufig Bearbeiterzuordnungen, die von anderen Aktivitäten abhängen („abhängige Bearbeiterzuordnungen“). So kann z.B. festgelegt sein, daß derselbe Arzt, der einen Patienten untersucht hat (Aktivität x), auch den zugehörigen Arztbrief schreiben soll (Nachfolgeraktivität k'), oder daß ein Patient von einer Pflegekraft derjenigen OE versorgt werden soll (Aktivität k), auf der er zuvor untersucht wurde (siehe Abb. 2). In diesem Fall steht erst nachdem Aktivität x ausgeführt wurde, die OE der Bearbeiter der weiteren Aktivitäten (k und k') fest. Auch wenn in einer WF-Vorlage abhängige Bearbeiterzuordnungen vorkommen, können prinzipiell die besten statischen Serverzuordnungen berechnet werden, indem bedingte Wahrscheinlichkeiten verwendet werden. Wenn aber statische Serverzuordnungen verwendet werden, so kann kein Wissen genutzt werden, das erst bei der Ausführung des WFs entsteht. Genau dies sollte aber bei den Aktivitäten k und k' geschehen. Wegen der abhängigen Bearbeiterzuordnung steht bei diesen Aktivitäten schon vor ihrer Ausführung fest,

[2] Die Berechnung basiert auf der vereinfachenden Annahme, daß sich an den Größenordnungen – und damit an den WVen – bis zur Ausführung von Instanzen dieses WF-Typs nichts wesentlich ändert.

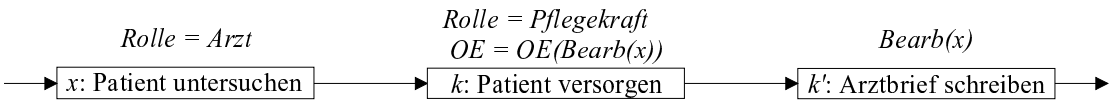


Abbildung 2 Beispiele für Bearbeiterzuordnungen, die von anderen Aktivitäten abhängig sind.

aus welcher OE die Bearbeiter stammen werden. Durch den Einsatz von variablen Serverzuordnungen (siehe auch [BD99a, BD99b]) ist es möglich, stets den optimalen WF-Server für diese Aktivitäten zu verwenden. Anstelle einer einfachen Serverzuordnung der Art "Aktivität x wird vom Server S_5 kontrolliert", werden logische Serverzuordnungsausdrücke verwendet, z.B. "Aktivität k wird vom Server im Domain des Bearbeiters kontrolliert, der Aktivität x ausgeführt hat". Es wird also Information genutzt, die zur Modellierungszeit bzw. beim Starten des WFs noch nicht existiert hat. Statische Serverzuordnungen sind weniger flexibel. Mit denen ist es lediglich möglich, den WF-Server in einer festen (möglichst günstigen) OE zu wählen. Läuft der WF aber in einer anderen OE, so ist diese Festlegung ungünstig; bei Verwendung variabler Serverzuordnungen ist sie stets optimal. Die zur variablen Serverzuordnung benötigten Ausdrücke werden zur Modellierungszeit systemunterstützt ermittelt und erlauben es, zur Laufzeit auf effiziente Weise den optimalen WF-Server zu bestimmen.

Im nächsten Abschnitt werden die Voraussetzungen für die Anwendbarkeit des Verfahrens geklärt. Abschnitt 3 diskutiert die Problemstellung und mögliche Lösungsansätze. Abschnitt 4 beschreibt die Algorithmen zum Ermitteln einer optimalen Verteilung. Die Auswirkungen von zusammengesetzten Bearbeiterzuordnungen – bei denen mehrere Aktivitäten z.B. einer Verzweigung referenziert werden – werden in Abschnitt 5 untersucht. Die Effizienz des Ansatzes wird in Abschnitt 6 mit Hilfe von Simulationen gezeigt. In Abschnitt 7 werden verwandte Ansätze diskutiert. Der Artikel schließt mit einer Zusammenfassung und einem Ausblick auf weitere Arbeiten.

2 Einbettung und Rahmenbedingungen

Der nachstehend vorgestellte Ansatz beschreibt, wie Lastverteilung mittels variabler Migration in ADEPT$_{distribution}$, der verteilten Variante des ADEPT-Systems, realsiert wird. In der weiteren Diskussion wird lediglich von den folgenden Annahmen ausgegangen:

1. Das WfMS verfügt über ein Organisationsmodell, in dem Personen sowohl mit OEen in der Aufbauorganisation als auch mit Rollen assoziiert werden können.
2. Die Zuordnung von Bearbeitern zu einer zur Bearbeitung anstehenden Aktivität erfolgt in der Regel zur Laufzeit mittels eines Auswahlprädikats, wie z.B. "$Rolle = Arzt \wedge Organisationseinheit = Radiologie$". In diesen Bearbeiterzuordnungen können auch andere Aktivitäten referenziert werden (abhängige Bearbeiterzuordnungen). Ist dies nicht der Fall, so wird angenommen, daß sich der tatsächliche Bearbeiter einer Aktivität unabhängig von anderen Aktivitäten ergibt. Der Bearbeiter ergibt sich also zufällig aus den potentiellen Bearbeitern einer Aktivität.
3. Das WfMS verwendet mehrere Teilnetze sowie mehrere WF-Server. Zur Vereinfachung der Diskussion 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-Typ und jede WF-Partition zur Steuerung eingesetzt werden kann. Ein Benutzer des WfMSs (WF-Klient) hat i.d.R. ein oder mehrere fest zugeordnete Teilnetze.
4. Jeder WF-Server kann alle beim WfMS angemeldeten WF-Klienten bedienen, nicht nur diejenigen, die sich in seinem Teilnetz (Domain) befinden.

Name	Bedeutung
$ActorProb_k(D\|S)$	Anteil der Bearbeiter im Domain (Teilnetz) D an der Gesamtheit der Bearbeiter von Aktivität k, wenn Aktivität k vom Server S kontrolliert wird
$BearbZuordn_k$	Bearbeiterzuordnung der Aktivität k
$Dependencies$	Menge der Abhängigkeiten $\langle \ldots \rangle$ zwischen den Bearbeitern der Aktivitäten des betrachteten WFs
$DepMigrProb_{x,k}(S_2\|S_1)$	Wahrscheinlichkeit für eine Migration zum WF-Server S_2 (beim Übergang von der Aktivität x zu k), wenn Aktivität x vom WF-Server S_1 kontrolliert wird
$DepServProb_k(S\|u)$	Wahrscheinlichkeit, daß Aktivität k vom Server S kontrolliert wird, wobei der Bearbeiter u dieser Aktivität vorgegeben ist
$ExProb_k(i,j)$	Wahrscheinlichkeit, daß Aktivität k vom Server des Teilnetzes i kontrolliert und von einem Bearbeiter in Teilnetz j ausgeführt wird
$Gewicht_k(u)$	Gesamtgewicht mit der der Bearbeiter u bei der Ausführung von Aktivität k gewichtet wird
$Join(s)$	zum Split-Knoten s gehörender Join-Knoten
$MigrProb_{k,l}(i,j)$	Wahrscheinlichkeit, daß beim Übergang von Aktivität k nach l vom Teilnetz i zum Teilnetz j migriert werden muß
OE	Organisationseinheit
$Pred_{EdgeTypes}(k)$	direkte Vorgänger von Aktivität k, analog zu $Succ_{EdgeTypes}(k)$
$Pred^*_{EdgeTypes}(k)$	(in)direkte Vorgänger von k, analog zu $Succ^*_{EdgeTypes}(k)$
$PotBearb_k$	Menge der potentiellen Bearbeiter der Aktivität k
$PotServZuordn_k$	Menge der für Aktivität k potentiell möglichen Serverzuordnungen
$RefAllowed(k,x)$	prüft, ob die Aktivität x in $ServZuordn_k$ referenziert werden darf
$ReferencedAct_k$	Aktivität, die die in $ServZuordn_k$ referenziert wird
$RelevantAct_k$	Aktivitäten, die für eine Referenzierung in $ServZuordn_k$ geeignet und relevant sind
$ServProb_k(S)$	Wahrscheinlichkeit, daß Aktivität k vom Server S (in Teilnetz S) kontrolliert wird
$ServZuordn_k$	Serverzuordnung der Aktivität k
WV	Wahrscheinlichkeitsverteilung
$Split(j)$	zum Join-Knoten j gehörender Split-Knoten
$Succ_{EdgeTypes}(k)$	bezeichnet für die Aktivität k und die Kantentypen $EdgeTypes \subseteq \{CONTROL\_E, SYNC\_E, LOOP\_E\}$ die Menge der direkten Nachfolgeraktivitäten von k bzgl. der in $EdgeTypes$ festgelegten Kantentypen: $\{$Akt. $x \mid \exists$ Kante $e = (k, x, edgeType)$ mit $edgeType \in EdgeTypes\}$
$Succ^*_{EdgeTypes}(k)$	bezeichnet die Menge der direkten oder indirekten Nachfolger: $\{$Akt. $x \mid x \in Succ_{EdgeTypes}(k) \ \vee$ $(\exists y : x \in Succ_{EdgeTypes}(y) \wedge y \in Succ^*_{EdgeTypes}(k))\}$
$\#User_k(D\|S)$	Anzahl der potentiellen Bearbeiter von Aktivität k im Domain D, wenn die Aktivität vom Server S kontrolliert wird
$\langle k, [x, B] \rangle$	Abhängigkeit: Aktivität k hat denselben Bearbeiter wie die Vorgängeraktivität x
$\langle k, [x, OE] \rangle$	Abhängigkeit: der Bearbeiter der Aktivität k gehört derselben OE an, wie der Bearbeiter der Vorgängeraktivität x

Tabelle 1: Übersicht über die verwendeten Abkürzungen.

5. Die potentiell möglichen Bearbeiter für eine gegebene Aktivität befinden sich nicht notwendigerweise immer im selben Teilnetz.[3]
6. Die Anzahl der Personen, die eine bestimmte Aktivität bearbeiten dürfen, ändert sich zwischen der Modellierungsphase und der WF-Ausführung nicht signifikant.
7. Die Topologie des Kommunikationsnetzwerks ändert sich ebenfalls nicht signifikant.

3 Problemstellung

Wie bereits oben erwähnt, ist es das Ziel unseres Ansatzes, Serverzuordnungen zu erhalten, durch welche die Kommunikationskosten minimiert werden. Dabei soll insbesondere der oben beschriebene Fall abhängiger Bearbeiterzuordnungen berücksichtigt werden. Für die Zuordnung von WF-Servern zu Aktivitäten gibt es im Prinzip mehrere Vorgehensweisen: Die einfachste und am häufigsten verwendete Methode ist, jeder Aktivität zur Modellierungszeit oder beim Start des WFs fest (statisch) einen Server zuzuordnen. Wie oben beschrieben wurde, ist diese Lösung für WFs mit abhängigen Bearbeiterzuordnungen, welche in der Praxis häufig anzutreffen sind, unbefriedigend. Die besten Serverzuordnungen würde man natürlich erhalten, wenn jeweils nach Beendigung einer Aktivität der WF-Server für die Nachfolgeraktivität aktuell bestimmt werden würde. Für diese Auswahl stünden dann die kompletten und aktuellen Laufzeitdaten der WF-Instanz zur Verfügung, so daß der tatsächlich am besten geeignete WF-Server bestimmt werden könnte. Leider scheidet diese Lösung im allgemeinen aus Aufwandsgründen aus. Das Durchführen der notwendigen Analysen würden die WF-Server stark belasten und damit die Performanz des WfMSs beeinträchtigen.[4]

In ADEPT$_{distribution}$ wird deshalb eine Strategie verfolgt, welche eine statische Vorberechnung (zur Modellierungszeit) mit dynamischen Aspekten (Berücksichtigung von Laufzeitdaten) kombiniert und so im wesentlichen die Vorteile der beiden Vorgehensweisen in sich vereint. Wie schon in Abschnitt 1 beschrieben wurde, werden anstelle einer festen Serverzuordnung bei der Analyse, wo erforderlich bzw. sinnvoll, logische Serverzuordnungsausdrücke (im folgenden kurz: Serverzuordnungen) erzeugt. Diese referenzieren die Laufzeitdaten der WF-Instanz und ermöglichen so auf einfache (und effiziente) Weise eine Festlegung des konkreten WF-Servers zur Laufzeit.

Die Frage ist nun, wie man zu geeigneten variablen Serverzuordnungen für die Aktivitäten kommt. Eine Möglichkeit wäre, sie bei der Modellierung einfach vorgeben zu lassen (analog zu den Bearbeiterzuordnungen). Das Problem hierbei ist, daß der WF-Modellierer, ohne weitere Unterstützung bzw. Information, die durch die Verteilung entstehende Last in den Teilnetzen in der Regel kaum wird abschätzen können. Es ist deshalb notwendig, daß er beim Ermitteln von geeigneten Serverzuordnungen seitens des WfMSs aktiv unterstützt wird. Das heißt, die WfMS-Modellierungskomponente muß für die in Betracht gezogenen Serverzuordnungen die Belastung jeder Systemkomponente (Server, Teilnetz, Gateway) geeignet abschätzen, so daß mögliche Überlastungen entdeckt und vermieden werden können.

Für diese Lastanalyse wird ein Kostenmodell benötigt, das es ermöglicht, auszurechnen, für welche Komponenten welche Kosten (als gewichtete Summe von Übertragungsmenge, Verarbeitungsmenge etc.) bei der jeweils betrachteten Serverzuordnung entstehen. Dieses Kostenmodell und die Bestimmung der Serverzuordnungen werden im nächsten Abschnitt beschrieben.

[3] Wäre dies immer der Fall, dann wäre keine variable Migration erforderlich. Alle Entscheidungen könnten bereits zur Modellierungszeit getroffen werden (statische Migration).

[4] Zur Erinnerung: Wir betrachten „Produktions-WfMSe“ mit potentiell sehr vielen gleichzeitig aktiven WF-Instanzen.

4 Ermitteln der Serverzuordnungen

Im folgenden wird der Zweck von abhängigen Bearbeiterzuordnungen ausführlich erläutert und es beschrieben, wie zur Modellierungszeit die optimale Verteilung der Aktivitäten einer WF-Vorlage auf WF-Server berechnet werden kann. Insbesondere wird auf die Bestimmung der Kosten und der WVen eingegangen.

4.1 Abhängige Bearbeiterzuordnungen

In der Einleitung wurde schon ein Beispiel für die Verwendung von abhängigen Bearbeiterzuordnungen beschrieben. In diesem Abschnitt wird nun genauer erläutert, warum abhängige Bearbeiterzuordnungen benötigt werden und welche Ausdrücke dafür in Frage kommen. Außerdem wird gezeigt, wie – auch bei mehrstufigen abhängigen Bearbeiterzuordnungen – schon zur Modellierungszeit die potentiellen Bearbeiter einer Aktivität k berechnet werden können.

4.1.1 Einsatz abhängiger Bearbeiterzuordnungen

In vielen Anwendungen sind für bestimmte Aktivitäten zwar prinzipiell Mitarbeiter aus verschiedenen OEen zuständig; für eine bestimmte WF-Instanz sind aber doch nur Mitarbeiter aus einer bestimmten OE zuständig. So werden bei einem Kreditantrag Arbeitsschritte, wie die Kundenbefragung, zwar von Angestellten aller Filialen ausgeführt; für einen konkreten Vorgang sind aber nur Angestellte der Filiale zuständig, in der der Kunde den Antrag gerade stellt. Abb. 3 zeigt ein Beispiel aus dem Krankenhausbereich: 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 (Aktivität 2-4).

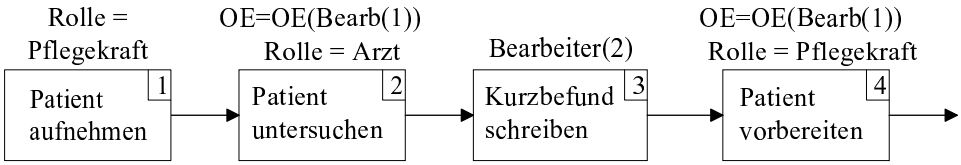


Abbildung 3 Beispiel für einen WF mit abhängigen Bearbeiterzuordnungen.

Damit eine abhängige Bearbeiterzuordnung von Aktivität k zur Ausführungszeit überhaupt aufgelöst werden kann, dürfen nur Aktivitäten referenziert werden, die vor k ausgeführt werden (Vorgängeraktivitäten). Aus Abb. 3 lassen sich die folgenden Typen von teilweise abhängigen Bearbeiterzuordnungen ermitteln:

- $BearbZuordn_k =$ unabhängiger Ausdruck
  z.B. "Rolle = Arzt" oder "$Rolle = Pflegekraft \wedge Abteilung = Chirurgie$"
- $BearbZuordn_k =$ "$Bearb(x)$"
  Aktivität k soll vom selben Bearbeiter ausgeführt werden wie Aktivität x
- $BearbZuordn_k =$ "$OE = OE(Bearb(x)) \wedge \ldots$"
  Der Bearbeiter von Aktivität k soll derselben OE angehören, wie der von Aktivität x. Des weiteren soll er irgendwelche weiteren Kriterien erfüllen.

Manchmal werden auch Bearbeiterzuordnungen benötigt, die von WF-Daten abhängig sind. Ein Beispiel hierfür ist eine Untersuchung, bei der die ausführende OE von der Untersuchungsart abhängig

ist: $BearbZuordn_k = "(Untersuchungsart = \textit{Röntgen:}\ OE = Radiologie \wedge Rolle = MTA) \vee (Untersuchungsart = \textit{Endoskopie:}\ OE = Chirurgie \wedge Rolle = Arzt)"$. Allgemein haben solche Bearbeiterzuordnungen die Form:

- $BearbZuordn_k = "Bedingung^1 : BearbZuordn^1 \vee Bedingung^2 : BearbZuordn^2 \vee \ldots"$

Außer diesen Typen von Bearbeiterzuordnungen sind auch noch weitere denkbar, auf die im folgenden aber nicht detailliert eingegangen wird. So kann z.B. die folgende Bearbeiterzuordnung verwendet werden, um ein 4-Augen-Prinzip zu realisieren:

- $BearbZuordn_k = "Bearb \neq Bearb(x)\ [\wedge OE = OE(Bearb(x))]\ \wedge \ldots"$
  Die Bearbeiter der Aktivitäten x und k sollen nicht identisch sein und evtl. derselben OE angehören. Die Auswahl des Bearbeiters von Aktivität k kann noch durch weitere Kriterien eingeschränkt werden.

Soll die Aktivität k vom Vorgesetzten des Bearbeiters der Aktivität x ausgeführt werden, so ergibt sich die folgende Bearbeiterzuordnung:

- $BearbZuordn_k = "Vorgesetzter(Bearb(x))"$

Je nach Anwendung sind auch noch weitere Typen von Bearbeiterzuordnungen denkbar.

4.1.2 Potentielle Bearbeiter einer Aktivität

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

Da die Menge der potentiellen Bearbeiter einer Aktivität häufig relevant ist, soll ein Algorithmus entwickelt werden, der es ermöglicht, diese Menge zu berechnen. Eine einfache Möglichkeit ist, für alle Aktivitäten des WFs sukzessive (d.h. in einer Reihenfolge entsprechend dem Kontrollfluß) diese Menge $PotBearb_k$ zu berechnen (siehe Algorithmus 1). Stößt man dabei auf eine abhängige Bearbeiterzuordnung, so wird aus der Menge $PotBearb_x$ der Bearbeiter der referenzierten Aktivität x die Menge der Bearbeiter der referenzierenden Aktivität k berechnet. So erhält man z.B. bei der $BearbZuordn_k = "Vorgesetzter(Bearb(x))"$ die Bearbeiter der Aktivität k, indem man die Menge der Bearbeiter u_1 der Aktivität x durchläuft und für jeden Benutzer u_2 prüft, ob er die Bearbeiterzuordnung erfüllt, unter der Annahme, daß u_1 die Aktivität x bearbeitet hat. Die WF-Vorlage wird in partieller Ordnung durchlaufen und in Bearbeiterzuordnungen können nur Vorgängeraktivitäten referenziert werden. Deshalb ist dem Algorithmus zu dem Zeitpunkt, wenn $PotBearb_k$ berechnet wird, $PotBearb_x$ schon bekannt.

Der große Vorteil von Algorithmus 1 ist, daß keine Fallunterscheidung für die verschiedenen Arten von abhängigen Bearbeiterzuordnungen benötigt wird. Deshalb bleibt der Algorithmus unverändert, auch wenn weitere abhängige Bearbeiterzuordnungen benötigt werden. Es muß lediglich möglich sein, zu prüfen, ob sich ein Benutzer u_2 als Bearbeiter von Aktivität k qualifiziert, wenn der Bearbeiter u_1 der referenzierten Aktivität x bekannt ist. Dies ist genau die Funktionalität, die auch das WfMS benötigt, wenn vor der Ausführung der Aktivität k die möglichen Bearbeiter ermittelt werden müssen. Der Algorithmus hat aber auch zwei Nachteile. Erstens müssen in einem großen WfMS, mit sehr vielen Benutzern, für jede Aktivität große Mengen verwaltet werden. Zweitens müssen diese Mengen neu berechnet werden, wenn sich die Menge der Benutzer des WfMSs oder deren Rollen-

Algorithmus 1 (Berechnung von $PotBearb_k$ als Menge)

for each Aktivität $k \in$ *Prozeßvorlage* (in partieller Ordnung entsprechend Kontrollfluß) **do**
  $PotBearb_k = \{\}$;
  case $BearbZuordn_k$ unabhängig:
    for each Benutzer u **do**
      if u erfüllt $BearbZuordn_k$ **then**
        $PotBearb_k = PotBearb_k \cup \{u\}$;
  case $BearbZuordn_k$ referenziert Aktivität x:
    for each Benutzer $u_1 \in PotBearb_x$ **do**
      for each Benutzer u_2 **do**
        if u_2 erfüllt $BearbZuordn_k$, falls u_1 die Aktivität x bearbeitet hat **then**
          $PotBearb_k = PotBearb_k \cup \{u_2\}$;

oder OE-Zuordnung ändert. Deshalb haben wir den Algorithmus 2 entwickelt, der diese Nachteile vermeidet. Er bietet damit eine elegante Möglichkeit, die Menge der potentiellen Bearbeiter einer Aktivität zu berechnen. Dazu wird für jede Aktivität k ein Ausdruck $PotBearb_k$ berechnet, der diese Menge beschreibt. Dabei werden die im vorherigen Abschnitt aufgeführten Bearbeiterzuordnungen berücksichtigt; für weitere evtl. benötigte Typen von Bearbeiterzuordnungen kann er entsprechend erweitert werden. Der Algorithmus ermittelt aus der Bearbeiterzuordnung der Aktivität k und den potentiellen Bearbeitern der referenzierten Aktivität x ($PotBearb_x$) die potentiellen Bearbeiter der Aktivität k ($PotBearb_k$).

Algorithmus 2 (Berechnung von $PotBearb_k$ als Ausdruck)

for each Aktivität $k \in$ *Prozeßvorlage* (in partieller Ordnung entsprechend Kontrollfluß) **do**
  $PotBearb_k = PotBearbErmitteln(BearbZuordn_k)$;

$PotBearbErmitteln(BearbZuordn)$:
  case $BearbZuordn$ unabhängig:
    return $BearbZuordn$;
  case $BearbZuordn =$ "$Bearb(x)$":
    return $PotBearb_x$;
  case $BearbZuordn =$ "$OE = OE(Bearb(x)) \wedge$ *Prädikat*":
    return $OE \in \{OE(PotBearb_x)\} \wedge$ *Prädikat*;
  case $BearbZuordn =$ "$\bigvee_i Bedingung^i : BearbZuordn^i$":
    $PotBearb =$ "";
    for each i **do**
      $PotBearb = PotBearb \vee PotBearbErmitteln(BearbZuordn^i)$;
    return $PotBearb$;

Mit Hilfe von $PotBearb_k$ läßt sich für jeden Benutzer u prüfen, ob er sich als Bearbeiter von Aktivität k qualifiziert.[5] Dadurch kann auch bei Verwendung von abhängigen Bearbeiterzuordnungen die Menge der für die Ausführung von Aktivität k in Frage kommenden Benutzer ermittelt werden. Wie in Kapitel 3 beschrieben wurde, lassen sich damit – wie bei unabhängigen Bearbeiterzuordnungen – die besten statischen Serverzuordnungen berechnen. Da in diese Berechnung nur die Gesamtmenge der potentiellen Bearbeiter der Aktivitäten einfließt (und nicht die Menge der Bearbeiter, die sich

[5] Wir schreiben dann im folgenden vereinfachend $u \in PotBearb_k$, egal ob es sich bei $PotBearb_k$ um eine mit Algorithmus 1 berechnete Menge oder um einen mit Algorithmus 2 berechneten Ausdruck handelt. Im letzteren Fall wäre eigentlich die Schreibweise u erfüllt $PotBearb_k$ korrekt.

tatsächlich für eine Aktivitäteninstanz qualifizieren), gehen die Abhängigkeiten zwischen den Aktivitäten verloren. Deshalb führt diese Vorgehensweise zu einem unzulänglichen Ergebnis und wird nicht weiter verfolgt. Allerdings wird $PotBearb_k$ von mehreren in diesem Beitrag vorgestellten Algorithmen benötigt. In diesen Algorithmen werden alle potentiellen Bearbeiter einer Aktivität durchlaufen und dabei jeweils die Beziehung zu der referenzierten Aktivität x berücksichtigt.

4.2 Kostenmodell

Ein Kostenmodell zur Berechnung der Kosten bzw. zur Bestimmung der Last für die unter Performanz-Aspekten kritischen Komponenten des WfMSs (Server, Teilnetz, Gateway) muß zumindest die folgenden Kosten berücksichtigen:
• Kosten für den Parametertransfer beim Starten und Beenden von Aktivitätenprogrammen
• Kosten für das Aktualisieren von Arbeitslisten
• Migrationskosten
• Kosten für die Kommunikation von Aktivitätenprogrammen mit externen Datenquellen
Zusätzlich zu den bereits im WF- und Organisationsmodell vorhandenen (relativ statischen) Informationen werden je Aktivität die geschätze Ausführungshäufigkeit und die im Mittel zu kommunizierende Datenmenge benötigt. Für bereits freigegebene WF-Typen können diese Daten durch die Analyse von Audit Trails ausgeführter WF-Instanzen gewonnen werden. Andernfalls müssen diese Daten geschätzt werden.

Bezeichne im folgenden $ExProb_k(i,j)$ die Wahrscheinlichkeit, daß Aktivität k vom Server in Teilnetz i kontrolliert und von einem Benutzer in Teilnetz j bearbeitet wird. $MigrProb_{k,l}(i,j)$ sei die Wahrscheinlichkeit, daß beim Übergang von Aktivität k nach l von Server i nach j migriert werden muß. $\#User_k(j|i)$ sei die Anzahl der potentiellen Bearbeiter von Aktivität k im Teilnetz j, wenn diese Aktivität vom Server im Teilnetz i kontrolliert wird. Diese Matrizen hängen von den gewählten Serverzuordnungen ab. Wie sie bestimmt werden können, wird später 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 WfMSs. Diese Formeln werden anschließend in eine Gesamtformel eingesetzt, mit der die in einer Komponente entstehende Last berechnet werden kann.

4.2.1 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 an Server i für die Ausführung von Aktivität k entsteht, ergibt sich als Wahrscheinlichkeit, daß Server i die Aktivität k kontrolliert, multipliziert mit der zu transportierenden Datenmenge:

$$Vol^{Akt}_{Server,i}(k) = \Big(\sum_j ExProb_k(i,j)\Big) \cdot (in\_parameter\_size_k + out\_parameter\_size_k)$$

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 $j \neq i$):

$$Vol^{Akt}_{TN,i}(k) = \Big(\sum_j ExProb_k(i,j) + \sum_{j \neq i} ExProb_k(j,i)\Big) \cdot (in\_parameter\_size_k + out\_parameter\_size_k)$$

Das anfallende Datenvolumen am Gateway von Teilnetz i nach j ($i \neq j$) ergibt sich wie folgt:
$Vol^{Akt}_{GW,i,j}(k) = ExProb_k(i,j) \cdot in\_parameter\_size_k + ExProb_k(j,i) \cdot out\_parameter\_size_k$

4.2.2 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 [BD98]), aus denen unterschiedliche Kosten resultieren. Die Klienten können ihre Arbeitslisten bei den Servern erfragen (Polling) oder die Server propagieren eine Änderung automatisch an die betroffenen Klienten. Eine Variante des zweiten Verfahrens ergibt sich, wenn Mindestzeitabstände für Übertragungen zum selben Klienten eingehalten werden und mehrere Änderungen zusammengefaßt werden. Dadurch wird die Anzahl der Übertragungen reduziert. 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 Klienten 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, wie schnell verschiedene Änderungen aufeinander folgen, so daß sie zusammengefaßt werden können. Außerdem ist dann relevant, wie schnell ein neuer Eintrag von einem Bearbeiter u_1 ausgewählt wird, so daß der Arbeitsschritt schon vor der nächsten Arbeitslistenaktualisierung eines anderen Bearbeiters u_2 nicht mehr verfügbar ist. Dies führt dazu, daß der Eintrag nie an diesen Bearbeiter u_2 übertragen wird. Aufgrund dieser Aspekte sind solche Abschätzungen 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. Deshalb ist die im folgenden beschriebene Berechnungsmethode völlig ausreichend. Wird ein anderes (optimiertes) Verfahren zum Aktualisieren der Arbeitslisten verwendet, so wird durch die beschriebene Abschätzung eine obere Schranke für die entstehenden Datenvolumina ermittelt. Dies gilt, weil das aufwendigste Verfahren analysiert wird, bei dem jede Änderung sofort an alle betroffenen Clients propagiert wird. Um realistischere Ergebnisse zu erhalten, können 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 Sinn machen, sie völlig zu ignorieren ($F = 0$).

Der WF-Server i muß Arbeitslisten aktualisieren, wenn Aktivität k 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 m zusammengezählt. Das Datenvolumen, das bei einem einzigen Einfügen und späteren Löschen eines Arbeitslisteneintrags für Aktivität k übertragen wird, ist $WL\_insert\_size_k$ und $WL\_delete\_size_k$. 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:
$Vol^{AL}_{Server,i}(k) = \sum_j ExProb_k(i,j) \cdot \sum_m \#User_k(m|i) \cdot (WL\_insert\_size_k + WL\_delete\_size_k)$

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:

$$Vol^{AL}_{TN,i}(k) = \Big(\sum_j ExProb_k(i,j) \cdot \sum_m \#User_k(m|i) + \sum_{l \neq i} \cdot \sum_j ExProb_k(l,j) \cdot \#User_k(i|l)\Big) \cdot (WL\_insert\_size_k + WL\_delete\_size_k)$$

Im Gateway von Teilnetz i nach j entsteht Datentransfer, wenn der Server in Teilnetz i die Arbeitsliste eines Benutzers aus Teilnetz j aktualisiert. Auch hier ist nur die Wahrscheinlichkeit relevant, mit der der Server i die Aktivität kontrolliert, unabhängig davon in welchem Teilnetz l sie später bearbeitet wird:

$$Vol^{AL}_{GW,i,j}(k) = \sum_l ExProb_k(i,l) \cdot \#User_k(m|i) \cdot (WL\_insert\_size_k + WL\_delete\_size_k)$$

4.2.3 Kommunikation mit externen Datenquellen

Aktivitätenprogramme, die auf dem Rechner des Klienten laufen, können mit externen Datenquellen (z.B. Datenbanken) kommunizieren. Um die daraus resultierenden Kosten berücksichtigen zu können, müssen die durchschnittlichen Ein- und Ausgabedatenmengen $Ext\_in_k(j)$ und $Ext\_out_k(j)$ vorgegeben (modelliert) werden, die Aktivität k mit Datenquellen in Teilnetz j austauscht. Dem WF-Server entsteht durch solche Kommunikationen kein Aufwand, da die Daten direkt zwischen Teilschrittprogramm und Datenquelle fließen:

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

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

$$Vol^{Ext}_{TN,j}(k) = \sum_i ExProb_k(i,j) \cdot \sum_l (Ext\_in_k(l) + Ext\_out_k(l)) + \sum_i \cdot \sum_{l \neq j} ExProb_k(i,l) \cdot (Ext\_in_k(j) + Ext\_out_k(j))$$

Von Teilnetz i nach j fließen Eingabedaten, wenn sich die Datenquelle in i und der Klient in j befindet, und Ausgabedaten, wenn der Klient in Teilnetz i und die Datenquelle in j ist:

$$Vol^{Ext}_{GW,i,j}(k) = \sum_l ExProb_k(l,j) \cdot Ext\_in_k(i) + \sum_l ExProb_k(l,i) \cdot Ext\_out_k(j)$$

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 Teilnetze für die Benutzer und für die Datenquellen ab, die mit Hilfe dieser Kostenabschätzung optimiert werden kann. Auch wenn der Modellierer bei dieser Festlegung 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 zu können.

4.2.4 Migrationen

Von besonderem Interesse sind in dem betrachteten Kontext natürlich die Migrationskosten beim Übergang von Aktivität k nach l. Hierbei müssen sowohl ein- wie auch ausgehende Migrationen berücksichtigt werden. Dabei ist zu beachten, daß keine Kommunikation stattfindet, wenn die Server für Aktivität k und l 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:

$$Vol^{Migr}_{Server,i}(k,l) = \sum_{j \neq i} \Big(MigrProb_{k,l}(i,j) + MigrProb_{k,l}(j,i)\Big) \cdot migration\_size_{k,l}$$

Das durch die Migration verursachte Datenvolumen in den betroffenen Teilnetzen ist genau so groß wie bei den Servern:
$Vol^{Migr}_{TN,i}(k,l) = Vol^{Migr}_{Server,i}(k,l)$.

Die Gateways müssen hierbei die folgenden Datenmengen transportieren:
$Vol^{Migr}_{GW,i,j}(k,l) = MigrProb_{k,l}(i,j) \cdot migration\_size_{k,l}$

4.2.5 Gesamtkosten

Für jede Systemkomponente müssen diese Datenmengen noch mit den Ausführungshäufigkeiten der einzelnen Aktivitäten $NodeFreq(k)$ bzw. der Kanten $EdgeFreq(k,l)$ 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):

$$\begin{aligned} Load_{Server,i} = \sum_{wf \in WF\_Types} \sum_{k \in wf} \Big(& NodeFreq(k) \cdot Vol^{Akt}_{Server,i}(k) \\ & + NodeFreq(k) \cdot Vol^{AL}_{Server,i}(k) \\ & + NodeFreq(k) \cdot Vol^{Ext}_{Server,i}(k) \\ & + \sum_{l \in wf \wedge l \neq k} EdgeFreq(k,l) \cdot Vol^{Migr}_{Server,i}(k,l) \Big) \end{aligned}$$

Verzögerungen bei den verschiedenen Aktionen eines WfMSs werden von den Benutzern unterschiedlich 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 diese bewußt wahrgenommenen Wartezeiten. Die „Last“ für den WF-Server i ($Load'_{TN,i}$ und $Load'_{GW,i,j}$ analog) ergibt sich damit als:

$$\begin{aligned} Load'_{Server,i} = \sum_{wf \in WF\_Types} \sum_{k \in wf} \Big(& G_{Akt} \cdot NodeFreq(k) \cdot Vol^{Akt}_{Server,i}(k) \\ & + G_{AL} \cdot NodeFreq(k) \cdot Vol^{AL}_{Server,i}(k) \\ & + G_{Ext} \cdot NodeFreq(k) \cdot Vol^{Ext}_{Server,i}(k) \\ & + G_{Migr} \cdot \sum_{l \in wf \wedge l \neq k} EdgeFreq(k,l) \cdot Vol^{Migr}_{Server,i}(k,l) \Big) \end{aligned}$$

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 damit 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 berücksichtigt 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.

Im folgenden Abschnitt wird beschrieben, welche Serverzuordnungsausdrücke für die Aktivitäten möglich sind. In Abschnitt 4.4 wird gezeigt, wie die Serverzuordnungen so gewählt werden können, daß K minimal wird.

4.3 Potentielle Serverzuordnungen

Die Frage, welche Serverzuordnungen als $ServZuordn_k$ in Frage kommen, läßt sich in zwei Aspekte aufteilen: welche Ausdrücke sind generell möglich und welche Aktivitäten werden von ihnen referenziert. Diese beiden Gesichtspunkte werden in diesem Abschnitt diskutiert.

4.3.1 Serverzuordnungsausdrücke

Bei der statischen Migration werden als Serverzuordnungsausdrücke die Server-IDs von WF-Servern verwendet. Im Fall variabler Serverzuordnungen sind, wie oben bereits erwähnt, als Serverzuordnungen auch Ausdrücke erlaubt, die Laufzeitdaten der Instanz referenzieren. In den meisten praktisch relevanten Fällen wird man sich hierbei auf die Lokation des Servers oder des Bearbeiters einer anderen Aktivität beschränken. In Sonderfällen können auch weitere WF-interne Daten (z.B. der Startzeitpunkt einer Aktivität), beliebige mathematische Funktionen, logische Ausdrücke oder die Einbeziehung WfMS-externer Daten (z.B. Parameterdaten von Aktivitäten) Sinn machen. Während Serverzuordnungen der erstgenannten Art (s.u.: 1. - 4.) systemseitig vom WF-Modell abgeleitet werden können, müssen die Zuordnungsausdrücke für die Sonderfälle (5.) explizit vom Modellierer vorgegeben werden.

Mögliche Serverzuordnungen sind:

1. $ServZuordn_k = \text{"}S_i\text{"}$
   Der Aktivität k wird der Server S_i fest (statisch) zugeordnet.
2. $ServZuordn_k = \text{"}Server(x)\text{"}$
   Die Aktivität k soll vom selben Server kontrolliert werden wie die Aktivität x.
3. $ServZuordn_k = \text{"}Domain(Bearb(x))\text{"}$
   Der Aktivität k wird der Server zugewiesen, der sich im Domain des Bearbeiters befindet, der die

Aktivität x ausgeführt hat. Ist Aktivität k derselbe Bearbeiter zugeordnet wie Aktivität x, so ist der Server von Aktivität k durch diese Zuordnung stets optimal.

4. $ServZuordn_k = "f(Server(x))"$ oder $ServZuordn_k = "f(Domain(Bearb(x)))"$
   Auf die Serverzuordnungen vom Typ 2 und 3 kann noch eine Funktion f angewandt werden. Dies macht z.B. Sinn, wenn Aktivität k zwar von Bearbeitern derselben OE wie Aktivität x ausgeführt wird, diese aber eine andere Rolle haben und in einem anderen Domain angesiedelt sind. Die Funktion f muß dann von dem ersten Domain auf den zweiten abbilden. Ein weiteres Beispiel für die Verwendung einer Funktion ist eine Aktivität k, welcher der Chef des Bearbeiters von Aktivität x zugeordnet ist. Dieser kann einem anderen Domain angehören (z.B. wenn er einer anderen Abteilung zugeordnet ist), so daß f den Domain des Mitarbeiters auf den des Chefs abbildet.
   Da die WVen der beiden Aktivitäten x und k berechnet werden können (siehe Abschnitt 4.5), kann eine optimale Funktion f automatisch erzeugt werden. Wie dies exakt funktioniert, wird im Anhang A beschrieben.
5. $ServZuordn_k =$ beliebiger vorgegebener Ausdruck, der nicht Typ 1 - 4 entspricht
   Der Modellierer kann auch einen beliebigen Serverzuordnungsausdruck vorgeben. Da dieser vom WfMS nicht analysiert werden kann, können die WVen nicht automatisch berechnet werden. Deshalb muß der Modellierer auch diese vorgeben, da sie von den Algorithmen aus Abschnitt 4.5 für ihre Analysen benötigt werden.

Bei Verwendung variabler Serverzuordnungen kann in $ServZuordn_k$ eine Aktivität x referenziert werden. Um diese beschrieben zu können, wird die Funktion $ReferencedAct$ eingeführt:

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

$$ReferencedAct_k = \begin{cases} NULL & \text{, falls } ServZuordn_k \text{ vom Typ 1 oder 5} \\ x & \text{, falls } ServZuordn_k \text{ von der Bauart 2 - 4 (s.o.)} \end{cases}$$

Zur Ausführungszeit der WFs muß vor dem Start der Aktivität k der Ausdruck $ServZuordn_k$ ausgewertet werden. Damit dies immer möglich ist, muß sichergestellt sein, daß die in $ServZuordn_k$ referenzierte Aktivität $x = ReferencedAct_k$ garantiert vor der Aktivität k ausgeführt wird. Das bedeutet, daß die Aktivität x immer vor der Aktivität k ausgeführt werden muß. Außerdem muß, falls die Aktivität k tatsächlich ausgeführt wird, auch die Aktivität x wirklich ausgeführt worden sein, d.h., sie darf sich dann nicht in einem nicht gewählten Zweig einer OR-Verzweigung befinden. Nur dann sind die benötigen Laufzeitdaten (Bearbeiter, Server) verfügbar. Wie diese Bedingungen sichergestellt werden können, wird im nächsten Abschnitt diskutiert.

4.3.2 Für eine Referenzierung relevante Aktivitäten

Nachdem geklärt wurde, welche Ausdrücke als $ServZuordn_k$ in Frage kommen, wird in diesem Abschnitt diskutiert, welche Aktivitäten x darin referenziert werden sollen. Dazu wird festgelegt, welche Aktivitäten in einer Serverzuordnung überhaupt referenziert werden dürfen. Dann wird betrachtet, bei welchen Aktivitäten es Sinn macht, sie zu referenzieren. Schließlich wird für Paare von Aktivitäten untersucht, wie sich die Festlegung des Bearbeiters für eine Aktivität auf die Menge der potentiellen Bearbeiter der anderen Aktivitäten auswirkt.

4.3.2.1 Für Referenzierung mögliche Aktivitäten

Damit die Aktivität x in $ServZuordn_k$ referenziert werden darf, muß sie garantiert vor Aktivität k ausgeführt werden (vgl. Abschnitt 4.3.1). Dazu muß die Aktivität $x = ReferencedAct_k$ im Ablaufgraphen vor k liegen und die Aktivität x muß stets ausgeführt werden, falls k ausgeführt wird. Ein Beispiel, in dem nur die zweite Bedingung verletzt ist, ist in Abb. 4 dargestellt. In diesem Fall darf in $ServZuordn_k$ die Aktivität x' nicht referenziert werden, obwohl x' eine Vorgängeraktivität von k ist ($x' \in Pred^*_{\{CONTROL\_E,SYNC\_E\}}(k)$)[6]. Solche Fälle müssen explizit ausgeschlossen werden.

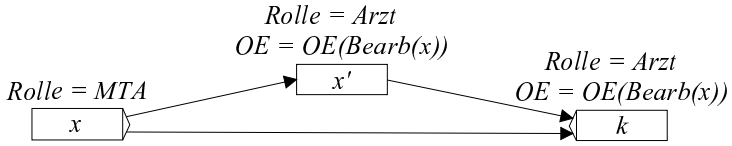


Abbildung 4 In $ServZuordn_k$ darf die Aktivität x' nicht referenziert werden.

Um sicherzustellen, daß die Aktivität x stets ausgeführt wird, falls die Aktivität k ausgeführt wird, ist zu prüfen, ob sich x in einem Ast einer bedingten Verzweigung befindet, in dem sich k nicht befindet. Wegen der Blockstrukturierung von ADEPT (vgl. [RD98]) ist dies gegeben, wenn sich zwischen x und k ein Verzweigungs-Endknoten und nicht der zugehörige Verzweigungs-Anfangsknoten befindet. Ist dies der Fall, so ist nicht erlaubt, daß die Aktivität x in $ServZuordn_k$ referenziert wird. Die in Algorithmus 3 definierte Funktion $RefAllowed$ prüft diese Bedingung für die Aktivität k und die referenzierte Aktivität x. Außerdem wird überprüft, ob die Aktivität x im Ablaufgraphen vor k liegt.

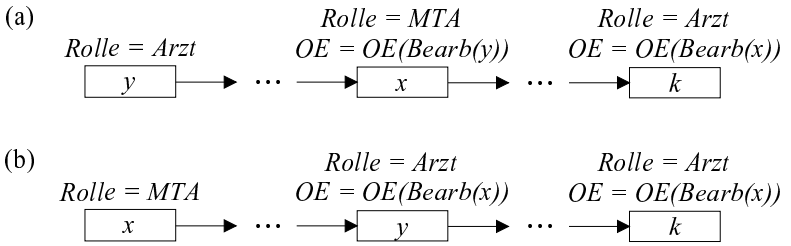
Algorithmus 3 (Prüfen, ob Referenzierung erlaubt: $RefAllowed(x, k)$)
if $x \notin Pred_{\{CONTROL\_E,SYNC\_E\}}(k)$ **then return** False;
for each j:
  $V_{in}(j) = ONE\_Of\_ALL$ // j ist OR-Join-Knoten
  $\wedge\ j \in Pred_{\{CONTROL\_E\}}(k)\ \wedge\ j \in Succ_{\{CONTROL\_E\}}(x)$ **do** // j liegt zwischen x und k
 $s = Split(j)$; // s ist zu j gehöriger Splitknoten
 if $s \notin Succ_{\{CONTROL\_E\}}(x)$ **then return** False; // Splitknoten s liegt nicht zwischen x und k
return True;

4.3.2.2 Für Referenzierung relevante Aktivitäten

Die Funktion $RefAllowed$ prüft, ob es möglich ist, eine Aktivität x in $ServZuordn_k$ zu referenzieren. Daraus folgt noch nicht, daß dies auch sinnvoll ist. Deshalb wird in diesem Abschnitt untersucht, für welche Aktivitäten x es interessant ist, sie in $ServZuordn_k$ zu referenzieren.

Die naheliegendste Idee ist, nur die Aktivität x zu berücksichtigen, die in $BearbZuordn_k$ referenziert wird. Aber auch transitiv abhängige Aktivitäten können relevant sein. In dem in Abb. 5a dargestellten Beispiel hat der Bearbeiter der (von Aktivität k transitiv abhängigen) Aktivität y dieselbe Rolle Arzt wie der Bearbeiter von Aktivität k. Der Bearbeiter der direkt abhängigen Aktivität x hat die Rolle MTA. Da mit der in Abb. 5c dargestellten Bearbeiterverteilung der Bearbeiter von Aktivität x immer im Domain 1 angesiedelt sind, ist diese Information zur Festlegung der Serverlokation von Aktivität k wertlos. Der Bearbeiter von Aktivität y (Rolle Arzt) befindet sich je nach OE im Domain 1 oder 2. Da für den Bearbeiter von Aktivität k dasselbe gilt, ist der Domain des Bearbeiters der transitiv abhängigen Aktivität y die optimale Lokation für den Server von Aktivität k.

[6] Zur Definition von $Pred^*_{EdgeTypes}(k)$ siehe Tabelle 1.



(c) Verteilung der Bearbeiter der verschiedenen OEen auf die Domains 1 und 2:

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 WFs, bei denen die indirekt abhängige Aktivität y für $ServZuordn_k$ relevant ist.

Doch nur die transitiven Abhängigkeiten zu berücksichtigen, reicht nicht aus, wie Abb. 5b zeigt. Mit der Bearbeiterverteilung aus Abb. 5c befindet sich der Bearbeiter der einzigen von Aktivität k (transitiv) abhängigen Aktivität x immer im Domain 1 (wegen Rolle MTA). Der Bearbeiter der Aktivität y hat aber wie der Bearbeiter von k 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 k. Daraus folgt, daß alle Aktivitäten für eine Referenzierung relevant sind, die indirekt (vor- oder rückwärts) mit Aktivität k in Beziehung stehen. Alle anderen Aktivitäten sind nicht relevant, da ihre Bearbeiter und deren OEen in keiner Beziehung zum Bearbeiter von Aktivität k stehen.

Für die in Abschnitt 4.5.2 beschriebenen Analysen ist es notwendig, zu wissen, ob die Aktivität k und die in $ServZuordn_k$ referenzierte Aktivität x denselben Bearbeiter haben, oder ob deren Bearbeiter nur derselben OE angehören. Deshalb ermittelt der nachfolgend beschriebene Algorithmus 4 nicht nur die für eine Referenzierung relevanten Aktivitäten, sondern er ermittelt für jede von ihnen auch, den Typ T der Abhängigkeit ($T \in \{B, OE\}$). Dabei haben Einträge im Ergebnis $Dependencies$ folgende Bedeutung:
$\langle k, [x, B] \rangle$ Aktivität k hat denselben Bearbeiter wie Aktivität x
$\langle k, [x, OE] \rangle$ Aktivität k hat einen Bearbeiter aus derselben OE wie Aktivität x
$\langle k, [NULL] \rangle$ steht für eine unabhängige Bearbeiterzuordnung von Aktivität k
Falls die Bearbeiter der Aktivitäten k und x in keiner Beziehung zueinander stehen, so befindet sich sich kein entsprechender Term in $Dependencies$.

Algorithmus 4 berechnet zuerst aus den Bearbeiterzuordnungen aller Aktivitäten initiale (nicht transitive) Abhängigkeiten und sammelt diese in der Menge $Dependencies$ auf (Funktion $initial\_dep$). Anschließend wird versucht, aus jedem Paar von Abhängigkeiten $Dep1$ und $Dep2$ eine neue Abhängigkeit $Dep3$ zu generieren. Diese Aufgabe übernimmt die Funktion $substitute\_dep$, welche die in Abb. 5a und b dargestellten Fälle indirekter Abhängigkeiten berücksichtigt. Wurde ein Paar von Abhängigkeiten gefunden, das sich kombinieren läßt, so muß noch der Typ T_{neu} 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). Andernfalls kann nichts ausgesagt werden.

Der nun folgende Algorithmus 4 analysiert Zusammenhänge zwischen den Bearbeiterzuordnungen der Aktivitäten, ist also von den Serverzuordnungen unabhängig. Er wird deshalb nicht in einer Schlei-

fe von Algorithmus 6 (s.u.) aufgerufen, sondern nur einmal (vor dem Berechnen der Serverzuordnungen) ausgeführt.

Algorithmus 4 (Abhängigkeiten zwischen den Bearbeitern der Aktivitäten)

$Dependencies = \{\}$;
for each Aktivität $k \in$ *Prozeßvorlage* **do**
  $Dependencies = Dependencies \cup initial\_dep(k)$;
do
  $changed =$ False;
  for each $Dep1 \in Dependencies$ **do**
    for each $Dep2 \in Dependencies$ **do**
    $Dep3 = substitute\_dep(Dep1, Dep2)$;
    if $Dep3 \neq NULL \wedge Dep3 \notin Dependencies$ **then**
      $Dependencies = Dependencies \cup \{Dep3\}$;
      $changed =$ True;
while $changed =$ True;
$\boldsymbol{initial\_dep(k)}$**:**
case $BearbZuordn_k =$ "$Bearb(x)$":
  return $\{\langle k, [x, B]\rangle, \langle x, [k, B]\rangle\}$;
case $BearbZuordn_k =$ "$OE(Bearb(x)) \wedge \ldots$":
  return $\{\langle k, [x, OE]\rangle, \langle x, [k, OE]\rangle\}$;
case $BearbZuordn_k =$ unabhängig:
  return $\{\langle k, [NULL]\rangle\}$;
$\boldsymbol{substitute\_dep(Dep1, Dep2)}$**:**
$\langle l1, D1\rangle = Dep1$; $\langle l2, D2\rangle = Dep2$;
if $D1 = [NULL] \vee D2 = [NULL]$ **then return** $NULL$;
$[l1', T1] = D1$; $[l2', T2] = D2$; // $Dep1 = \langle l1, [l1', T1]\rangle,\ Dep2 = \langle l2, [l2', T2]\rangle$
if $l1' = l2$ **then** // Fall 1: Fall aus Abb. 5a
  $T_{neu} = combine(\{T1, T2\})$;
  return $\langle l1, [l2', T_{neu}]\rangle$;
else if $l1' = l2'$ **then** // Fall 2: Fall aus Abb. 5b
  $T_{neu} = combine(\{T1, T2\})$;
  return $\langle l1, [l2, T_{neu}]\rangle$;
else return $NULL$;
$\boldsymbol{combine(T)}$**:**
if $\forall T_i \in T : T_i \in \{B\}$ **then return** B;
else if $\forall T_i \in T : T_i \in \{B, OE\}$ **then return** OE;
else return $NULL$;

Wenn sich in $dependencies$ ein Eintrag $\langle k, [x, T]\rangle$ befindet, so ist x für eine Referenzierung in $ServZuordn_k$ „interessant“. Damit die Aktivität x in $ServZuordn_k$ referenziert werden darf, muß sie außerdem garantiert vor Aktivität k ausgeführt werden ($RefAllowed(k, x) =$ True). Damit ergeben sich die in $ServZuordn_k$ referenzierbaren und relevanten Aktivitäten $RelevantAct_k$ wie folgt:
$RelevantAct_k = \{x \mid \langle k, [x, T]\rangle \in Dependencies \ \wedge RefAllowed(k, x)\}$

4.3.2.3 Bearbeiter von Aktivitätenpaaren

Die Menge $Dependencies$ wurde 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 Aktiviäten. Deshalb können sie auch verwendet

werden, um zu entscheiden, ob sich ein Benutzer noch für die Bearbeitung einer Aktiviät qualifiziert, wenn die Bearbeiter anderer Aktiviäten vorgegeben sind. Da diese Funktionalität von mehreren in dieser Arbeit vorgestellten Algorithmen benötigt wird, wird in diesem Abschnitt ein entsprechender Algorithmus vorgestellt.

Die in der Menge $Dependencies$ enthaltenen Abhängigkeiten lassen erkennen, welches die potentiellen Bearbeiter einer Aktivität k sind, wenn die Bearbeiter anderer Aktivitäten vorgegeben sind. Mit dieser Information läßt berechnen, ob ein Bearbeiter u die Aktivität k bearbeiten darf, falls der Bearbeiter u' einer anderen Aktivität x gegeben ist. Dazu prüft Algorithmus 5 zuerst, ob der Bearbeiter u – unabhängig vom Bearbeiter der Aktivität x – die Aktivität k bearbeiten darf, d.h. (vereinfacht), ob er z.B. die richtige Rolle hat. Dies ist gegeben, falls $u \in PotBearb_k$ gilt. Anschließend wird getestet, ob die Bearbeiter u' und u eine Abhängigkeit zwischen den Aktivitäten x und k verletzen. Existiert zwischen den Aktivitäten x und k eine Abhängigkeit vom Typ B (selber Bearbeiter), so darf u die Aktivität k 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 x und k, so darf der Bearbeiter u die Aktivität k immer bearbeiten, da dann auch keine Abhängigkeit verletzt sein kann. Daß u die Aktivität k aufgrund seiner Rolle o.ä. prinzipiell bearbeiten darf, wurde ja schon am Anfang des Algorithmus geprüft.

Algorithmus 5 ($ActorPossible(x, u', k, u)$)
if $u \notin PotBearb_k$ **then return** False;
if $\langle k, [x, B] \rangle \in Dependencies$ **then**
  return $u = u'$;
else if $\langle k, [x, OE] \rangle \in Dependencies$ **then**
  return $OE(u) = OE(u')$;
else /* $\langle k, [NULL] \rangle \in Dependencies$ */
  return True;

Der vorgestellte Algorithmus kann auch verwendet werden, wenn die Bearbeiter mehrerer Aktiviäten vorgegeben sind. Sind die Bearbeiter $u'_1 \ldots u'_n$ der Aktivitäten $x_1 \ldots x_n$ gegeben, so qualifiziert sich der Bearbeiter u für die Aktivität k, falls gilt:
$ActorPossible(x_1, u'_1, k, u) = True \ \wedge \ \ldots \ \wedge \ ActorPossible(x_n, u'_n, k, u) = True$

4.4 Bestimmung der optimalen Serverzuordnungen

Zum Ermitteln der optimalen Verteilung könnten im Prinzip für alle Aktivitäten und alle möglichen Serverzuordnungen die Kosten analysiert und die optimale Variante ausgewählt werden. Da ein WF aber durchaus $\#Akt > 100$ Aktivitäten umfassen kann, die jeweils alle ihre Vorgänger in der Serverzuordnung referenzieren können, scheidet diese Vorgehensweise wegen ihrer Komplexität $O(\#Akt^{\#Akt})$ im allgemeinen aus. Wir schlagen deshalb den nachfolgend beschriebenen Algorithmus 6 vor, der ein polynomiales Laufzeitverhalten hat. Er basiert auf einem Greedy-Ansatz und liefert für die praktisch relevanten Fälle das optimale Ergebnis bzw. eines, das dicht an diesem Optimum liegt.

Algorithmus 6 berechnet für jede Aktivität die optimale Serverzuordnung und legt diese im Array $ServZuordn$ ab. In der Phase 1 wird zunächst eine zulässige Ausganglösung bestimmt. Diese berücksichtigt noch keine Migrationskosten. Für jede Aktivität werden alle möglichen Serverzuordnungen,

d.h. Ausdrücke vom Typ 1 - 4 aus Abschnitt 4.3.1, ausprobiert ($PotServZuordn_k$). In diesen Ausdrücken werden alle relevanten Aktivitäten referenziert, d.h. Aktivitäten x mit $x \in RelevantAct_k$. Aus den potentiell möglichen Serverzuordnungen wird diejenige ausgewählt, die zu den minimalen Kosten für die Ausführung dieser Aktivität führt ($calculate\_costs()$ verwendet hierzu das Kostenmodell aus Abschnitt 4.2).

Da die Aktivitäten isoliert betrachtet werden, ergeben sich i.d.R. auch nicht rentable Migrationen. In Phase 2 wird überprüft, welche dieser Migrationen sinnvoll sind. Dazu wird ausgerechnet, ob es günstiger ist, eine Gruppe P von Aktivitäten (die alle vom selben WF-Server kontrolliert werden) mit einer direkten Vorgänger- oder Nachfolgergruppe zusammenzufassen, so daß die Migration dazwischen entfällt. Die Motivation dafür ist, daß es sich nicht lohnt, wegen wenigen Aktivitäten die gesamte Prozeßinstanz zu einem Server hin und zurück zu migrieren. Der Algorithmus betrachtet zuerst kleine zusammenhängende Teilgraphen P (Partitionen) von Aktivitäten, die alle vom selben Server (vgl. Algorithmus 12 und 13) kontrolliert werden. Diese werden mit Nachbar-Teilgraphen 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 zu kommunizierende Datenmenge bei der Ausführung der WFs, da Gruppen von Aktivitäten genau dann zusammengefaßt werden, wenn dies die Kommunikationskosten reduziert.

Algorithmus 6 (Berechnung der Serverzuordnung für einen WF-Typ)

Phase 1:

for each Aktivität $k \in$ *Prozeßvorlage* (in partieller Ordnung entsprechend Kontrollfluß) **do**
  $MinCost = \infty$;
  for each $TestServZuordn_k \in PotServZuordn_k$ **do**
    $Cost = calculate\_costs(ServZuordn_*, k)$; [7] /* Kosten nur für Aktivität k berechnen */
    if $Cost < MinCost$ **then**
      $ServZuordn_k = TestServZuordn_k$;
      $MinCost = Cost$;

Phase 2:

$MinCost = calculate\_costs(ServZuordn_*, all)$; /* Kosten gesamter WF (mit Migrationskosten) */
for *PartGröße* = 1 **to** *#Aktivitäten(Prozeßvorlage)* **do**
  for each $P : |P| =$*PartGröße* $\wedge$ P ist max. Teilgraph mit $\forall l_1, l_2 \in P : SameServer(l_1, l_2)$ **do**
    $OptAct = NULL$;
    for each $a \notin P : \exists l \in P :$
      $a \in Pred_{\{CONTROL\_E, SYNC\_E, LOOP\_E\}}(l) \vee$
      $a \in Succ_{\{CONTROL\_E, SYNC\_E, LOOP\_E\}}(l)$ **do**
      for each $l \in P$ **do** $TestServZuord_l = ServZuordn_a$;
      for each $l \notin P$ **do** $TestServZuord_l = ServZuordn_l$;
      $TestCost = calculate\_costs(TestServZuordn_*, all)$;
      if $TestCost < MinCost$ **then**
        $OptAct = a$;
        $MinCost = TestCost$;
    if $OptAct \neq NULL$ **then**
      for each $l \in P$ **do**
        $ServZuordn_l = ServZuordn_{OptAct}$;

4.5 Berechnung der Wahrscheinlichkeitsverteilungen

Nachdem erläutert wurde, welche Serverzuordnungen für eine Aktivität in Frage kommen und wie die durch sie entstehenden Kosten berechnet werden, bleibt die Frage, wie die WVen $ExProb_k(i,j)$, $MigrProb_{k,l}(i,j)$ und $\#User_k(D|S)$ für eine zu untersuchende Verteilung berechnet werden können.

Da Serverzuordnungen von Laufzeitdaten der Instanz abhängen, kann dieselbe Aktivität k bei verschiedenen WF-Instanzen durch unterschiedliche Server kontrolliert werden. Die entsprechende Wahrscheinlichkeit, daß Aktivität k vom Server S kontrolliert wird, wird mit $\boldsymbol{ServProb_k(S)}$ = P(Aktivität k wird vom Server S kontrolliert) bezeichnet. Da sich verschiedene Instanzen in unterschiedlichen OEen befinden können (und dabei unterschiedliche Server verwenden können), kann die Bearbeiter-WV jeweils unterschiedlich sein. Der Anteil an den Bearbeitern von Aktivität k aus dem Domain D, für den Fall, daß der Server S die Aktivität k kontrolliert, wird in $\boldsymbol{ActorProb_k(D|S)}$ = P(der Bearbeiter von Aktivität k befindet sich im Domain D | der Server der Aktivität k befindet sich im Domain S) festgehalten. Das Hauptproblem bei der Kostenanalyse besteht nun darin, für gegebene Serverzuordnungen die WVen $ServProb_k(S)$ und $ActorProb_k(D|S)$ zu berechnen. Sind diese erst bekannt, so können $ExProb_k(i,j)$ und damit die Kosten für die Aktivitätenausführung berechnet werden. $ActorProb_k(D|S)$ muß dazu nur noch mit der Wahrscheinlichkeit gewichtet werden, mit der Server S den Schritt k kontrolliert ($ServProb_k(S)$). Die Wahrscheinlichkeit, daß der Server S verwendet wird und ein Bearbeiter im Domain D die Aktivität k bearbeitet, beträgt $ExProb_k(S,D) = ServProb_k(S) \cdot ActorProb_k(D|S)$.

Für einzelne Aktivitäten können $ServProb_k(S)$ und/oder $ActorProb_k(D|S)$ 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.

4.5.1 Berechnung der Server-Wahrscheinlichkeitsverteilung $ServProb_k(S)$

Die Server-WV $ServProb_k(S)$ für die Aktivität k gibt an, mit welcher Wahrscheinlichkeit Server S der Server von Aktivität k ist. Sie ergibt sich aus der Serverzuordnung $ServZuordn_k$ und der Server- bzw. Bearbeiter-WV der referenzierten Aktivität x. Diese WVen sind bei der Analyse von Aktivität k schon bekannt, da x vor k ausgeführt wird und die Prozeßvorlage in partieller Ordnung durchlaufen wird. Die Berechnung der Server-WV wird nun für die in Abschnitt 4.3.1 definierten Serverzuordnungen beschrieben. Der Fall 5 wird dabei (und im weiteren) nicht berücksichtigt, da bei dieser Serverzuordnung die WV vom Modellierer vorgegeben werden muß. $ServProb_k(S)$ kann mit dem folgenden Algorithmus berechnet werden:

4.5.2 Berechnung der Bearbeiter-WV $ActorProb_k(D|S)$

Die Bearbeiter-WV $ActorProb_k(D|S)$ gibt an, mit welcher Wahrscheinlichkeit der Bearbeiter von Aktivität k aus dem Domain D stammt, wenn die Aktivität vom Server S kontrolliert wird. Sie ist vom konkret verwendeten Server S dieser Aktivität abhängig. Als Beispiel 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 Server 1 verwendet. Damit ergibt sich eine Bearbeiter-WV

Algorithmus 7 (Berechnung der Server-WV $ServProb_k(S)$)

case $ServZuordn_k = "S_i"$: (1)
$ServProb_k(S) = \delta_{S,S_i}$ [8]

case $ServZuordn_k = "Server(x)"$: (2)
$ServProb_k(S) = ServProb_x(S)$

case $ServZuordn_k = "Domain(Bearb(x))"$: (3)
$ServProb_k(i) = ActorProb_x(i)$ mit $ActorProb_x(D) = \sum_S ActorProb_x(D|S) \cdot ServProb_x(S)$

case $ServZuordn_k = "f(Server(x))"$: (4a)
$ServProb_k(S) = f(ServProb_x(S))$

case $ServZuordn_k = "f(Domain(Bearb(x)))"$: (4b)
$ServProb_k(i) = f(ActorProb_x(i))$ mit $ActorProb_x(D) = \sum_S ActorProb_x(D|S) \cdot ServProb_x(S)$

Erklärungen:

(1) Da die Aktivität k stets vom Server S_i kontrolliert wird, ist $ServProb_k(S) = 1$ für $S = S_i$ und sonst $= 0$.

(2) Für Akt. k wird derselbe Server verwendet wie für x, weshalb die Server-WVen gleich sind.[9]

(3) Der Server von Aktivität k ist im Domain des Bearbeiter von Aktivität x angesiedelt. Deshalb ergibt sich die Server-WV von k aus der Bearbeiter-WV von x. Da dabei nicht relevant ist, welcher Server Aktivität x kontrolliert hat, wird eine vom Server unabhängige Bearbeiter-WV $ActorProb_x(i)$ erzeugt, indem die Bearbeiter-WVen der einzelnen Server gewichtet aufaddiert werden. Diese werden jeweils mit der Wahrscheinlichkeit gewichtet, daß der betrachtete Server die Aktivität x kontrolliert hat.

(4a) Die Berechnung von $ServProb_k(S)$ erfolgt wie im Fall 2, wobei auf das Ergebnis noch die Funktion f angewandt werden muß (wie dies exakt funktioniert, zeigt Algorithmus 11 in Abschnitt 4.5.4).

(4b) Wie Fall 3, dann auf das Ergebnis f anwenden.

$ActorProb_k(D|1) = (1, 0, 0)$. Analog ergibt sich für Server 2 die WV $ActorProb_k(D|2) = (0, 1, 0)$ und $ActorProb_k(D|3) = (0, 0, 1)$ für Server 3.

Die Bearbeiter-WV $ActorProb_k(D|S)$ kann durch Algorithmus 8 bestimmt werden. Der Algorithmus durchläuft alle Bearbeiter der Aktivität k 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 für den entsprechenden Server S und Domain D in der Bearbeiter-WV $ActorProb_k(D|S)$ berücksichtigt. Die Berechnung von $ActorProb_k(D|S)$ erfolgt also entsprechend der Definition der Bearbeiter-WV (siehe Abschnitt 4.5). Die Aktivität k kann trotz desselben Bearbeiters durch unterschiedliche Server – jeweils mit einer gewissen Wahrscheinlichkeit – kontrolliert werden. Diese Wahrscheinlichkeiten werden mit der Funktion $DepServ(k, "Bearb = u")$ berechnet und im Vektor $DepServProb_k(S|u)$ gespeichert. Der Bearbeiter wird anteilig bei jedem dieser Server berücksichtigt.

Um die Kosten für Arbeitslisten-Updates abschätzen zu können, muß $\#User_k(D|S)$ – die Anzahl der potentiellen Bearbeiter von Aktivität k im Domain D, falls die Aktivität vom Server S kontrolliert wird – berechnet werden. Dies geschieht im Algorithmus 8, indem alle Bearbeiter gezählt werden, die Aktivität k bearbeiten dürfen, falls sie vom Server S kontrolliert wird ($\#User_k|S$). Da sich diese Bearbeiter entsprechend $ActorProb_k(D|S)$ auf die verschiedenen Domains verteilen, berechnet sich $\#User_k(D|S)$ damit als: $\#User_k(D|S) = \#User_k|S \cdot ActorProb_k(D|S)$

Die Berechnung der Server-WV $DepServProb_k(S|u)$ für einen bestimmten Bearbeiter u geschieht in ähnlicher Weise, wie für die bearbeiterunabhängige WV $ServProb_k(S)$ in Abschnitt 4.5.1 be-

Algorithmus 8 (Berechnung der Bearbeiter-WV $ActorProb_k(D|S)$)
for each $u \in PotBearb_k$ **do**
  $D = Domain(u)$;
  $DepServProb_k(S|u) = DepServ(k, "Bearb = u")$;
  for each S **do**
    if $DepServProb_k(S|u) \neq 0$ **then**
      $ActorProb_k(D|S) = ActorProb_k(D|S) + DepServProb_k(S|u)$;
      $\#User_k|S = \#User_k|S + 1$;
normalisiere jede Zeile von $ActorProb_k(D|S)$ so, daß $\forall S : \sum_D ActorProb_k(D|S) = 1$;

schrieben. Allerdings hängt $DepServProb_k(S|u)$ nicht nur von der Serverzuordnung der betrachteten Aktivität k ab, sondern auch von deren Bearbeiterzuordnung. Im folgenden werden einige Beispiele beschrieben, eine vollständige Diskussion aller Fälle, die durch Kombination der möglichen Server- und Bearbeiterzuordnungen entstehen, findet sich im Abschnitt 4.5.4.

- Ist die Serverzuordnung statisch ($ServZuordn_k = "S_i"$), so ist die Berechnung trivial, da der Server immer gleich ist: $DepServProb_k(S|u) = \delta_{S,S_i}$.
- Ist die Bearbeiterzuordnung unabhängig von anderen Aktivitäten (z.B. $"Rolle = Arzt"$), so ist die Server-WV unabhängig vom Bearbeiter u, weshalb die allgemeine Server-WV übernommen werden kann: $DepServProb_k(S|u) = P_k^S$.
- In Abb. 6a wird Aktivität k vom selben Arzt bearbeitet wie Aktivität x. Der Server wird im Domain des Bearbeiters von Aktivität x allokiert. Es soll $DepServProb_k(S|u)$ für u = Dr. Brinkmann aus Domain 3 berechnet werden. Wegen $BearbZuordn_k = "Bearb(x)"$ hat Dr. Brinkmann auch Aktivität x ausgeführt. Deshalb ist der Server von Aktivität k im Domain von Dr. Brinkmann angesiedelt. Da dies der Domain 3 ist, ergibt sich $DepServProb_k(S|u) = (0, 0, 1)$.
- Wird, wie in Abb. 6b dargestellt, nicht derselbe Bearbeiter verwendet, sondern lediglich dieselbe OE, so ist das Verfahren etwas komplizierter. Es soll $DepServProb_k(S|u)$ für Schwester Hildegard aus der OE Abteilung 2 berechnet werden. Dazu müssen alle Bearbeiter (derselben OE Abteilung 2) mit der Rolle Arzt durchlaufen werden, weil dies genau die Bearbeiter sind, die die Aktivität x ausgeführt haben können, wenn Aktivität k von Schwester Hildegard ausgeführt wird. Für jeden dieser Ärzte wird der Domain ermittelt, da er – falls dieser Arzt die Aktivität x ausgeführt hat – die Lokation des Servers bestimmt. Dieser Domain wird dann in $DepServProb_k(S|u)$ berücksichtigt. Nach dem Durchlaufen aller potentieller Bearbeiter wird $DepServProb_k(S|u)$ noch so normalisiert, daß $\sum_S DepServProb_k(S|u) = 1$ gilt.

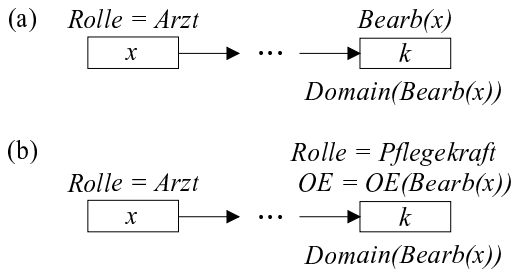


Abbildung 6 Beispiele zur Berechnung der Server-WV $DepServProb_k(S|u)$.

4.5.3 Gewichtung der Bearbeiter

Es gibt Benutzer, die Teilzeit arbeiten, die nur einen Teil ihres Arbeitstags mit dem WfMS arbeiten oder die in mehreren Domains jeweils einen Teil ihrer Arbeitszeit verbringen. Diese dürfen nicht wie Benutzer behandelt werden, die „full time“ im selben Domain arbeiten, da sie im Tagesdurchschnitt bzw. pro Teilnetz weniger Last erzeugen. Ein solcher Sachverhalt läßt sich z.B. dadurch modellieren, daß jedem Benutzer des WfMSs Gewichte zugeordnet werden. Diese können verwendet werden, um bei der Berechnung der Bearbeiter-WV $ActorProb_k(D|S)$ die $DepServProb_k(S|u)$ der einzelnen Bearbeiter gewichtet zu addieren.

Die Gewichte können auch von der aktuell betrachteten Aktivität k abhängen ($G_k(u)$). Die Gewichte eines Bearbeiters werden zwar häufig für alle Aktivitäten identisch sein ($\forall n : G_k(u) = G(u)$), 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 $G_k(u)$ größer bzw. kleiner als $G(u)$.

Für die Gewichte eines Bearbeiters ist sogar eine noch feinere Unterscheidung möglich. Für einen Benutzer u können zwei Arten von benutzerspezifischen Gewichten $G_k(u)$ und $G'_k(u)$ unterschieden werden. Mit $G_k(u)$ legt der Modellierer fest, welchen Anteil der Zeit der Benutzer die *Aktivität k ausführt* (so daß durch den Parametertransfer Last erzeugt wird). Mit $G'_k(u)$ wird angegeben, welchen Anteil der üblichen Arbeitszeit er durchschnittlich am WfMS *angemeldet ist* (so daß seine Arbeitsliste aktualisiert werden muß). Ist die Aktivität k eine „Lieblingsaktivität“ des Benutzers u, so ist $G_k(u)$ größer als der Standardwert $G(u)$ dieses Benutzers. Auch das Gewicht $G'_k(u)$, 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 $G'_k(u) < G'(u)$ gewählt. Im folgenden wird aber auf die Unterscheidung von $G_k(u)$ und $G'_k(u)$ verzichtet und das Gewicht eines Bearbeiters u als $G_k(u)$ 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 ($G_k(u)$) oder für Arbeitslisten-Updates ($G'_k(u)$) 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 k auswählt, ist so groß, wie sein Anteil an dem „Gesamtgewicht für Aktivität k“:
P(Benutzer u bearbeitet die Aktivität k) = $G_k(u) \,/ \sum_{u' \in PotBearb_k} G_k(u')$

Außer diesen benutzerspezifischen Gewichten $G_k(u)$, werden durch die variable Serverzuordnung weitere OE-spezifische Gewichte $G_k(OE)$ notwendig. Der Grund dafür ist, daß nicht mehr angenommen werden kann, daß alle potentiellen Bearbeiter gleich wahrscheinlich[10] für die Bearbeitung

[10] Hinter dieser Annahme steckt die Unabhängigkeit der Bearbeiter voneinander.

einer Aktivität sind. Um dies zu erläutern, nehmen wir an, daß für den in Abbildung 6b dargestellten Ablauf die Verteilung von Bearbeitern auf OEen aus Tabelle 2 gelte. $AnteilBearb_k(OE) = \frac{\sum_{u \in PotBearb_k \wedge OE(u)=OE} G_k(u)}{\sum_{u \in PotBearb_k} G_k(u)}$ beschreibt dabei den Anteil der Bearbeiter von Aktivität k, die der Organisationseinheit OE angehören ($PotBearb_k$ ist die Menge der potentiellen Bearbeiter von Aktivität k, vgl. Abschnitt 4.1.1).

OE	$\boldsymbol{AnteilBearb_x}$ **(Ärzte)**	$\boldsymbol{AnteilBearb_k}$ **(Pflegekräfte)**
Abteilung 1	0,2 (2 Bearbeiter mit $G_k(u) = 1$)	0,4 (16 Bearbeiter mit $G_k(u) = 1$)
Abteilung 2	0,5 (5 Bearbeiter mit $G_k(u) = 1$)	0,35 (14 Bearbeiter mit $G_k(u) = 1$)
Abteilung 3	0,3 (3 Bearbeiter mit $G_k(u) = 1$)	0,25 (10 Bearbeiter mit $G_k(u) = 1$)

Tabelle 2: Beispiel für die Verteilung von Bearbeitern auf verschiedene OEen.

Obwohl 40% der Bearbeiter von Aktivität k (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 x festgelegt. Da die Ärzte von Abteilung 1, die Aktivität x ausführen, aber nur einen Anteil von 20% haben, wird Aktivität k auch nur mit einer Wahrscheinlichkeit von 20% in Abteilung 1 ausgeführt. Algorithmus 9 berechnet für jede Aktivität k, wie groß der Anteil der einzelnen OEen bei der Ausführung dieser Aktivität ist ($AnteilAusf_k(OE)$). Ist Aktivität k abhängig von einer Aktivität x ($\langle k, [x, T] \rangle \in Dependencies$ mit $T \in \{B, OE\}$), so bestimmt Aktivität x den Anteil der OEen bei der Ausführung von Aktivität k. Andernfalls ($\langle k, [NULL] \rangle \in Dependencies$) entspricht der Anteil einer OE bei der Ausführung ihrem Anteil an den Bearbeitern. Aus den Werten für $AnteilBearb_k(OE)$ und $AnteilAusf_k(OE)$ berechnet Algorithmus 9 das *OE-Gewicht* $G_k(OE)$, 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. Damit erfolgt die Berechnung von $Gewicht_k(u)$ auf korrekte Art und Weise. Diese Aussage wird im Anhang B.1 bewiesen. Ist Aktivität k von keiner anderen Aktivität abhängig, so ergibt sich das OE-Gewicht als 1.

Im obigen Beispiel ist für die Abteilung 1 $AnteilAusf_k(Abt1) = AnteilAusf_x(Abt1) = AnteilBearb_x(Abt1) = 0,2$. Damit ergibt sich das OE-Gewicht als $G_k(Abt1) = 0,2/0,4 = 0,5$. Mit diesem Wert wird jede Pflegekraft der Abteilung 1 bei der Berechnung von $ActorProb_k(D|S)$ zusätzlich gewichtet, so daß sich für alle Bearbeiter aus Abteilung 1 ein Gesamtgewicht $Gewicht_k(u) = 0,5$ ergibt. Mit diesem Gesamtgewicht wird der Bearbeiter u bei der Berechnung von $ActorProb_k(D|S)$ in Algorithmus 8 gewichtet:

4.5.4 Berechnung der Wahrscheinlichkeitsverteilung $DepServProb_k(S|u)$

Wie in Abschnitt 4.5.2 beschrieben, muß zur Berechnung der Bearbeiter-Wahrscheinlichkeitsverteilung $ActorProb_k(D|S)$ die Server-WV $DepServProb_k(S|u)$ berechnet werden. Dabei ist zu beachten, daß der Bearbeiter u von Aktivität k vorgegeben ist. $DepServProb_k(S|u)$ gibt also für einen bestimmten Bearbeiter u an, mit welcher Wahrscheinlichkeit die Aktivität k vom Server S kontrolliert wird. Dabei muß sowohl die Bearbeiter- wie auch die Serverzuordnung gegeben sein, da die Berechnung von beiden abhängt. Algorithmus 10 behandelt die bei der Berechnung von $DepServ(k, Cond)$

Algorithmus 9 (Berechnung der OE-abhängigen Gewichte)

/* Anteil der OEen an den Bearbeitern von Aktivität k berechnen */

$\forall OE : AnteilBearb_k(OE) = 0;$

for each OE **do**

$$AnteilBearb_k(OE) = \sum_{\substack{u \in PotBearb_k \\ \wedge\, OE(u)=OE}} G_k(u) \;/ \sum_{u \in PotBearb_k} G_k(u);$$

/* Anteil der OEen an der Ausführung von Aktivität k berechnen */

for each OE **do**

if $\exists x : \langle k, [x, B] \rangle \in Dependencies \vee \langle k, [x, OE] \rangle \in Dependencies$ **then**

$AnteilAusf_k(OE) = AnteilAusf_x(OE)$

else

$AnteilAusf_k(OE) = AnteilBearb_k(OE);$

$G_k(OE) = AnteilAusf_k(OE) \;/\; AnteilBearb_k(OE);$

/* Berechnung der Gesamtgewichte für die Bearbeiter */

for each $u \in PotBearb_k$ **do**

$OE = OE(u);$

$Gewicht_k(u) = G_k(OE) \cdot G_k(u);$

Algorithmus 8: Berechnung $ActorProb_k(D|S)$ unter Berücksichtigung der Gewichte

...

if $DepServProb_k(S|u) \neq 0$ **then**

$ActorProb_k(D|S) = ActorProb_k(D|S) + DepServProb_k(S|u) \cdot Gewicht_k(u);$

$\#User_k|S = \#User_k|S \;+\; Gewicht_k(u);$

...

auftretenden Fälle. $Cond$ beschreibt dabei die betrachtete Bearbeitermenge, also beim Aufruf den Benutzer u ($Bearb = u$).

Wir erwähnen zuerst noch einmal die einfachen Fälle, die auch schon in Abschnitt 4.5.2 diskutiert wurden. Ist die Serverzuordnung statisch ($ServZuordn_k = \text{"}S_i\text{"}$, Fall 1a - c in Algorithmus 10), so ist der Server immer gleich und die Serververteilung ergibt sich als $DepServProb_k(S|u) = \delta_{S,S_i}$. Ist die Bearbeiterzuordnung unabhängig von Aktivität x (Fall 1a, 2a und 3a), so wird für jeden Bearbeiter die allgemeine Server-WV $ServProb_k(S)$ übernommen. Im Fall 1a gilt nach Algorithmus 7 $ServProb_k(S) = \delta_{S,S_i}$, so daß kein Widerspruch zu den Aussagen vom Anfang dieses Absatzes besteht. Die Fälle 2a und 3a machen scheinbar keinen Sinn, da eine abhängige Serverzuordnung verwendet wird, obwohl der Bearbeiter der Aktivität k nicht von anderen Aktivitäten abhängt. Diese Fälle können aber sehr wohl auftreten, nämlich wenn Aktivitäten zusammengefaßt werden um Migrationen einzusparen (Phase 2 von Algorithmus 6), so daß Aktivität k in eine Partition mit abhängigen Serverzuordnungen aufgenommen wird.

Es bleiben noch die Fälle, in denen $BearbZuordn_k$ von Aktivität x abhängig ist ($\langle k, [x, T] \rangle \in Dependencies$ mit $T \in \{B, OE\}$) und in $ServZuordn_k$ die Aktivität x referenziert wird. Lautet $ServZuordn_k = \text{"}Server(x)\text{"}$, so ergibt sich $DepServProb_k(S|u)$ durch Rekursion. Wir betrachten zuerst den Fall, daß Aktivität k vom exakt selben Benutzer bearbeitet wird wie Aktivität x (Fall 2b). Da die Aktivitäten k und x in diesem Fall denselben Bearbeiter und denselben Server haben, hat jeder Bearbeiter bei den beiden Aktivitäten auch dieselbe Serververteilung. Das Ergebnis ergibt sich also durch direkte Rekursion, wobei die Bedingung $Cond$ – die die Bearbeitermenge von Aktivität x beschreibt – unverändert übernommen wird. Wird Aktivität x lediglich von irgendeinem Benutzer derselben OE bearbeitet wie Aktivität k (Fall 2c), so wird das Ergebnis ebenfalls

durch Rekursion berechnet. Es ist aber nicht garantiert, daß die Aktivitäten k und x vom exakt selben Bearbeiter ausgeführt werden (nur dieselbe OE ist sicher). Deshalb wird die Rekursion mit einer geänderten Bedingung $Cond'$ aufgerufen, falls $Cond$ einen Bearbeiter spezifiziert hat. $Cond'$ darf für Aktivität x nun nur noch die OE des Bearbeiters festlegen. Falls durch $Cond$ nur die OE der Bearbeiter vorgegeben ist, kann die Bedingung unverändert für die Rekursion verwendet werden.

Lautet $ServZuordn_k = \texttt{"}Domain(Bearb(x))\texttt{"}$, so wird keine Rekursion ausgeführt; ggf. ist damit das Rekursionsende erreicht. Stattdessen wird die Menge der potentiellen Bearbeiter von Aktivität x durchlaufen, da der Server von Aktivität k im Domain des Bearbeiters von Aktivität x angesiedelt ist. Für jeden Benutzer, der prinzipiell Aktivität x bearbeiten darf und zusätzlich die Bearbeiterbedingung $Cond$ erfüllt, wird der zugehörige Domain ermittelt. Die Domains dieser Bearbeiter bestimmen den Server von Aktivität k, so daß das gewichtete Aufaddieren dieser Domains zur korrekten Server-WV für Aktivität k führt (für die in $Cond$ spezifizierten Benutzer). Schließlich wird das Ergebnis noch normiert, so daß die Summe aller Werte 1 ergibt.

Algorithmus 10 (Berechnung von $DepServ(k, Cond)$)

case	$\langle k, [NULL]\rangle \in Dependencies$	$\langle k, [x, B]\rangle \in Dependencies$	$\langle k, [x, OE]\rangle \in Dependencies$
$ServZuordn_k = \texttt{"}S_i\texttt{"}$	(1a) $\forall S : SV(S) = \delta_{S,S_i}$	(1b) $\forall S : SV(S) = \delta_{S,S_i}$	(1c) $\forall S : SV(S) = \delta_{S,S_i}$
$ServZuordn_k = \texttt{"}Server(x)\texttt{"}$	(2a) $\forall S : SV(S) = ServProb_k(S)$	(2b) $SV(S) = DepServ(x, Cond)$	**case** $cond = \texttt{"}Bearb = u\texttt{"}$: (2c) $oe = OE(u)$; $Cond' = \texttt{"}OE = oe\texttt{"}$; **case** $cond = \texttt{"}OE = oe\texttt{"}$: $Cond' = Cond$; $SV(S) = DepServ(x, Cond')$
$ServZuordn_k = \texttt{"}Domain(Bearb(x))\texttt{"}$	(3a) $\forall S : SV(S) = ServProb_k(S)$	(3b) $\forall S : SV(S) = 0$; **for** each $u \in PotBearb_x$ mit u erfüllt $Cond$ **do** $S = Domain(u)$; $SV(S) = SV(S) + Gewicht_x(u)$; normalisiere SV(S), so daß $\sum_S SV(S) = 1$	

return SV(S);

Wird eine Funktion f auf die Serverzuordnung angewandt, d.h. $ServZuordn_k = \texttt{"}f(Server(x))\texttt{"}$ oder $ServZuordn_k = \texttt{"}f(Domain(Bearb(x)))\texttt{"}$, so muß f auch noch auf das Ergebnis $SV(S)$ angewandt werden. Um die Funktion f auf einen Vektor anzuwenden, initialisiert Algorithmus 11 einen Ergebnisvektor $SV'(S)$ mit 0. Für jeden Eintrag in $SV(S)$ wird der aus der Abbildung f resultierende Server $S' = f(S)$ berechnet. Der entsprechende Eintrag aus $SV(S)$ wird dann an der Stelle S' im Ergebnis SV' berücksichtigt.

Algorithmus 11 (Anwenden einer Funktion f auf eine WV $SV(S)$)

$\forall S : SV'(S) = 0$;
for each S **do**
  $S' = f(S)$;
  $SV'(S') = SV'(S') + SV(S)$;

Hat der Modellierer für eine Aktivität k eine Serverzuordnung vorgegeben, die nicht analysiert wer-

den kann (Fall 5. in Abschnitt 4.3.1), so muß er auch $ServProb_k(S)$ und $ActorProb_k(D|S)$ vorgeben. Da $ActorProb_k(D|S)$ damit schon bekannt ist, wird Algorithmus 8 aus Abschnitt 4.5.2 und damit Algorithmus 10 nicht verwendet. Allerdings kann es vorkommen, daß die benutzerabhängige Serververteilung $DepServProb_k(S|u)$ einer Aktivität mit einer derartigen Serverzuordnung bei einer Rekursion benötigt wird (Fall 2b oder 2c). Als Ergebnis kann immer die allgemeine Server-WV $ServProb_k(S)$ verwendet werden, die ebenfalls vom Modellierer vorgegeben werden muß. Hat der Modellierer zusätzlich OE-spezifische Server-WVen angegeben, so wird aus $Cond$ die betroffene OE ermittelt, womit die zugehörige Server-WV verwendet werden kann.

Mit den beschriebenen Mechanismen lassen sich weitere relevante Typen von Bearbeiterzuordnungen (siehe 4.1.1) behandeln. Für diese können weitere Arten von Abhängigkeiten und weitere Bearbeiterbedingungen $Cond$ definiert werden. Soll z.B. die Aktivität k vom Vorgesetzten des Bearbeiters der Aktivität x erledigt werden, so läßt sich dies durch eine Serverzuordnung der Form "$f(Domain(Bearb(x)))$" effizient unterstützen. Dabei stellt f die Abbildung vom Domain der Angestellten einer Abteilung zum Domain ihres Vorgesetzten dar. $SV(S)$ wird, wie im Fall 3b beschrieben, durch das Durchlaufen aller Untergebenen des Chefs berechnet. Dafür kann eine Bedingung der Art "$Vorgesetzter = u$" verwendet werden. – Wird für Aktivität k ein Bearbeiter aus derselben OE OE_1 wie für Aktivität x verwendet, der aber von diesem verschieden sein muß (4-Augen-Prinzip), so muß beim Durchlaufen aller Bearbeiter beim Rekursionsende (3b) der Bearbeiter von Aktivität k ausgelassen werden. Die Bedingung kann hier "$OE = OE_1 \wedge Bearb \neq u$" lauten. Die $BearbZuordn_k$ = "$\neg Bearb(x) \wedge Rolle \ldots$" wird ebenso behandelt, wobei aber alle Bearbeiter mit einer passenden Rolle berücksichtigt werden müssen (außer dem von Aktivität k), und nicht nur die einer bestimmten OE. $Cond$ lautet dann "$Bearb \neq u$", die richtige Rolle der Bearbeiter u der Aktivität x wird in Algorithmus 10 Fall 3b durch die Bedingung $u \in PotBearb_x$ sichergestellt.

4.5.5 Wahrscheinlichkeitsverteilung für die Startaktivität einer WF-Instanz

In zahlreichen Anwendungsbereichen sollen dieselben „Standard-WFs“ in verschiedenen OEen verwendet werden. Diese WFs werden (fast) ausschließlich von Bearbeitern der jeweiligen OE ausgeführt. Es werden also dieselben WF-Templates in unterschiedlichen OEen verwendet, wobei die WF-Instanzen vom jeweils lokalen WF-Server kontrolliert werden sollen. Um eine solche Verteilung zu modellieren, könnte man einen speziellen Serverzuordnungsausdruck einführen, der festlegt, daß eine Aktivität von dem Server kontrolliert wird, auf dem die entsprechende WF-Instanz gestartet wurde. Dieser zusätzliche Serverzuordnungsausdruck wird in ADEPT nicht benötigt; es genügen die in Abschnitt 4.3.1 beschriebenen Ausdrücke. Der Grund dafür ist, daß der erste Schritt jedes WFs ein expliziter Startknoten st ist, dem 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 Instanz gestartet wurde. Dies ist der Server im Domain dieses Benutzers. Diese beiden Daten der Startaktivität können nun in den Server- und Bearbeiterzuordnungen der darauf folgenden Aktivitäten referenziert werden. Einen WF, der am Startserver verbleibt, erhält man damit, indem für jede Aktivität k die Serverzuordnung $ServZuordn_k$ = "$Server(st)$" verwendet wird.

Um (automatisch) ermitteln zu können, ob solche Serverzuordnungen günstig sind, müssen auch für die Startaktivität st die Bearbeiter- und Server-WVen berechnet werden. Die Server-WV $ServProb_{st}(S)$ kann aus den Domains der Benutzer abgeleitet werden, die eine Instanz dieses WF-Typs T starten dürfen. Welche dies sind, wird durch ein Prädikat des WF-Typs beschrieben (analog zu den Bearbeiterzuordnungen). Bei der Berechnung von $ServProb_{st}(S)$ werden die Gewichte $G_T(u)$ berücksichtigt, die angeben, ob der Benutzer u einen WF dieses Typs T überdurchschnittlich oft bzw.

selten startet (analog zu den Gewichten der Benutzer für die Ausführung von Aktivitäten):

$$\forall S\text{: } ServProb_{st}(S) = \frac{\sum\limits_{\substack{u \in Starter_T \\ \wedge Domain(u)=S}} G_T(u)}{\sum\limits_{u \in Starter_T} G_T(u)}$$

mit $Starter_T = \{u \mid \text{Benutzer } u \text{ darf WFs vom Typ } T \text{ starten}\}$

Für die Bearbeiter-WV gilt stets $ActorProb_{st}(D|S) = \delta_{S,D}$, weil der Benutzer u WFs auf seinem lokalen Server S startet. Deshalb gehören der Server S und der Benutzer u immer demselben Domain D an. Die WVen $ServProb_{st}(S)$ und $ActorProb_{st}(D|S)$ können nun bei der Berechnung von $ServProb_k(S)$ und $ActorProb_k(D|S)$ nachfolgender Aktivitäten verwendet werden.

4.6 Migrationskosten

In Phase 1 von Algorithmus 6 werden nur die optimalen Serverzuordnungen der Aktivitäten betrachtet, ohne die Migrationskosten zu berücksichtigen. In Phase 2 werden auch diese Kosten mit einbezogen. Um entscheiden zu können, ob überhaupt Kosten anfallen, muß berechnet werden, ob zwischen zwei aufeinanderfolgenden Aktivitäten (k und l) eine Migration stattfindet. Falls dem so ist, muß die Wahrscheinlichkeit $MigrProb_{k,l}(i,j)$ bestimmt werden, mit der die WF-Instanz vom Server i zum Server j migriert wird.

4.6.1 Bestimmung identischer Serverzuordnungen

Dieser Abschnitt untersucht die Frage, welche Aktivitäten stets vom gleichen Server kontrolliert werden (dazu müssen die Serverzuordnungsausdrücke nicht identisch sein). Die Gleichheit bezieht sich dabei auf dieselbe WF-Instanz; die Server unterschiedlicher Instanzen können verschieden sein. Für die Aktivitäten l und k wird derselbe Server verwendet ($SameServer(l,k) = True$), wenn einer der in Abb. 7 dargestellten Fälle vorliegt. Bei a) wird in der Serverzuordnung von k explizit angegeben, daß derselbe Server wie für l verwendet werden soll und bei b) ergibt sich dies durch denselben Serverzuordnungsausdruck (z.B. "$Domain(Bearb(a))$"). Im Fall c) haben die Aktivitäten a und b denselben Bearbeiter. In dessen Domain befinden sich die Server von l und k, die somit identisch sind. Ein häufiger Spezialfall von c) ist, daß die Aktivitäten b und l zusammenfallen, so daß sich eine Kette von Aktivitäten mit demselben Server und demselben Bearbeiter ergibt. In d) ist exemplarisch der Fall dargestellt, daß in zwei Serverzuordnungen dieselbe Funktion f verwendet wird und innerhalb dieser Funktion äquivalent Ausdrücke stehen. Da $Server(a)$ und $Server(b)$ denselben Server beschreiben, werden – nach Anwendung von f auf diesen Server – auch die Aktivitäten l und k vom selben Server kontrolliert. Falls die inneren Ausdrücke identisch sind, erfüllen die Serverzuordnungen zusätzlich die Bedingung für den Fall b).

Soll geprüft werden, ob die Aktivitäten k und x vom selben Server kontrolliert werden, so muß man analysieren, ob eine „transitive Abhängigkeit“ der beiden Aktivitäten voneinander besteht. Algorithmus 12 ermittelt dazu die Menge der Aktivitäten, die vom selben Server kontrolliert werden wie Aktivität k ($Server\_wie\_k$). Diese Menge enthält zuerst nur die Aktivität k. Dann wird in einer Schleife für jede Aktivität l geprüft, ob sie nach den Regeln aus Abb. 7 (Funktion $gleicherServerDirekt()$) vom selben Server kontrolliert wird, wie eine Aktivität m aus der Menge $Server\_wie\_k$. Falls dem so ist, wird l in diese Menge aufgenommen. Dies wird solange durchgeführt, bis sich die Menge nicht

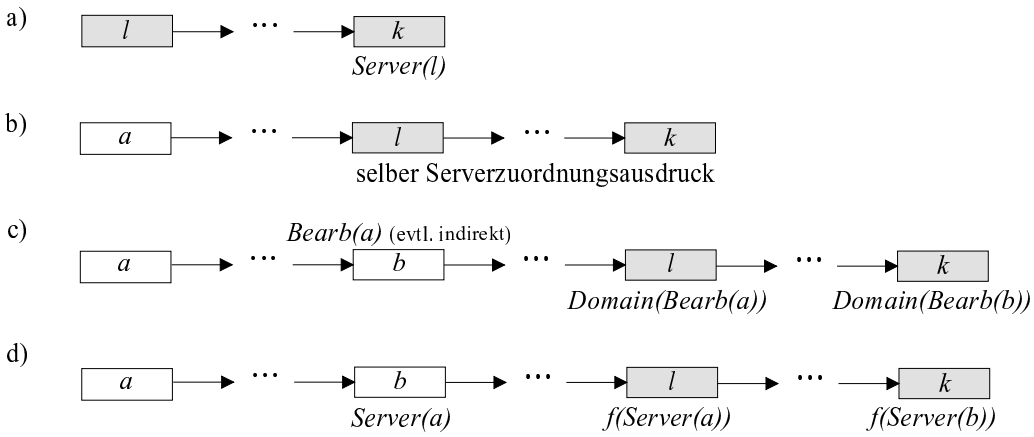


Abbildung 7 Fälle in denen die Aktivitäten k und l vom selben Server kontrolliert werden.

mehr ändert oder die Aktivität x aufgenommen wurde. Im letzteren Fall werden die Aktivitäten k und x vom selben Server kontrolliert.

Algorithmus 12 jedesmal auszuführen, wenn die Gleichheit von Serverzuordnungen getestet werden soll, bedeutet einen sehr großen Aufwand. Eine effiziente Alternative stellt die einmalige Berechnung von *Serverklassen* dar. Für die Aktivitäten k und x muß danach nur noch geprüft werden, ob sie derselben Klasse angehören:
$SameServer(x, k)$: **return** $ServerKlasse_x = ServerKlasse_k$;

Die Berechnung der Serverklasse $ServerKlasse_k$ einer Aktivität k erfolgt nach Phase 1 und vor Phase 2 von Algorithmus 6, also direkt nachdem die optimale Serverzuordnung für k (isoliert betrachtet) ermittelt wurde. Algorithmus 13 testet, ob die Aktivität k in eine der schon ermittelten Serverklassen fällt. Dazu wird für alle Aktivitäten l, deren Serverklasse schon feststeht, geprüft, ob die Aktivitäten k und l vom selben Server kontrolliert werden (Funktion $gleicherServerDirekt()$ aus Algorithmus 12). Falls dem so ist, wurde die Serverklasse von Aktivität k gefunden und die Suche kann beendet werden. Wurde keine Aktivität l gefunden, die diese Bedingung erfüllt, so begründet Aktivität k eine neue Serverklasse.

In Phase 2 von Algorithmus 6 können sich Serverzuordnungen noch ändern, wenn Partitionen P innerhalb derer alle Aktivitäten vom selben Server kontrolliert werden, zusammengefaßt werden. Dann ergibt sich die neue Serverklasse der Aktivitäten mit geänderter Serverzuordnung (Partition P) als Serverklasse der Aktivität a, deren Serverzuordnung übernommen wurde:
for each $l \in P$: $ServerKlasse_l = ServerKlasse_a$;
Eine erneute Berechnung der Serverklassen ist damit nicht notwendig.

4.6.2 Migrationswahrscheinlichkeiten

Die Wahrscheinlichkeiten für die Migrationen beim Übergang von Aktivität x nach k können in Form einer *Migrationsmatrix* angegeben werden. Der Eintrag der i-ten Zeile und j-ten Spalte gibt an, wie groß die bedingte Wahrscheinlichkeit ist, daß zum Server j migriert werden muß, wenn Aktivität x vom Server i kontrolliert wird. Es gilt also:
$DepMigrProb_{x,k}(S_2|S_1) =$ P(Server S_2 kontrolliert Aktivität k | Server S_1 kontrolliert Aktivität x)
Damit ergibt sich die Migrationswahrscheinlichkeit als:

Algorithmus 12 (Berechnung von $SameServer(x, k, ServZuordn_*)$)

$Server\_wie\_k = \{k\}$;
do
  $changed =$ False;
  for each Aktivität l **do**
    for each Aktivität $m \in Server\_wie\_k$ **do**
      if $gleicherServerDirekt(l, m)$ **then**
        $Server\_wie\_k = Server\_wie\_k \cup \{l\}$;
        $changed =$ True;
while $changed = True \wedge x \notin Server\_wie\_k$;
if $x \in Server\_wie\_k$ **then**
  return True
else
  return False;

$\boldsymbol{gleicherServerDirekt(l, k)}$**:**
// die Serverzuordnungen der Aktivitäten l oder k sind noch nicht festgelegt:
if $ServZuordn_l = NULL \vee ServZuordn_k = NULL$ **then return** False;
// Fall aus Abb. 7a:
if $ServZuordn_l = \text{"}Server(k)\text{"} \vee ServZuordn_k = \text{"}Server(l)\text{"}$ **then return** True;
// Fall aus Abb. 7b:
if $ServZuordn_l = ServZuordn_k$ **then return** True;
// Fall aus Abb. 7c:
if $ServZuordn_l = \text{"}Domain(Bearb(a))\text{"}) \wedge ServZuordn_k = \text{"}Domain(Bearb(b))\text{"} \wedge$
  $\langle a, [b, B] \rangle \in Dependencies$ **then return** True;
// Fall aus Abb. 7d:
if $ServZuordn_l = \text{"}f_1(ServZuordn_1)\text{"}) \wedge ServZuordn_k = \text{"}f_1(ServZuordn_1)\text{"} \wedge$
  $\forall s : f_1(s) = f_2(s)$ **then**
  $ServZuordn_{l'} = ServZuordn_1$;
  $ServZuordn_{k'} = ServZuordn_2$;
  return $SameServer(l', k', ServZuordn_* \cup \{ServZuordn_{l'}, ServZuordn_{k'}\})$;
return False;

$MigrProb_{x,k}(S_1, S_2) = ServProb_k(S_1) \cdot DepMigrProb_{x,k}(S_2|S_1)$
Im folgenden wird beschrieben, wie die Matrix $DepMigrProb$ ermittelt wird.

Werden die Aktivitäten x und k stets vom selben Server kontrolliert ($SameServer(l, k) = True$), so muß nie migriert werden, so daß sich $DepMigrProb_{x,k}(S_2|S_1) = \delta_{i,j}$ ergibt. Andernfalls bietet die Nutzung der Server-WV $ServProb_k(S_2)$ eine einfache Möglichkeit zur Approximation der Migrationswahrscheinlichkeiten. Die Server-WV gibt an, mit welcher Wahrscheinlichkeit sich die Instanz nach der Migration an Server S_2 befindet. Die Migrationswahrscheinlichkeiten ergeben sich damit als:
$\forall S_1, S_2 : DepMigrProb_{x,k}(S_2|S_1) = ServProb_k(S_2)$

Bei dieser Berechnung wird allerdings die Annahme vorausgesetzt, daß die Server der beiden Aktivitäten unabhängig voneinander gewählt werden. Das ist jedoch nicht der Fall, wenn z.B. die Server- und Bearbeiterzuordnungen der beiden Aktivitäten jeweils von derselben OE abhängen. Deshalb wird zur Berechnung der Migrationswahrscheinlichkeiten von der Aktivität x nach k der Algorithmus 14 verwendet, wenn eine Abhängigkeit zwischen den Bearbeitern der Aktivitäten besteht ($\langle k, [x, *] \rangle \in Dependencies$).

Algorithmus 13 (Berechnen der Serverklassen)
$maxKlasse = 0$;
$\forall$ Aktivitäten k: $ServerKlasse_k = NULL$;
for each Aktivität $k \in$ *Prozeßvorlage* **do**
  for each Aktivität l mit $ServerKlasse_l \neq NULL$ **do**
    if $gleicherServerDirekt(k, l)$ **then**
      $ServerKlasse_k = ServerKlasse_l$;
      exit loop;
  if $ServerKlasse_k = NULL$ **then**
    $maxKlasse = maxKlasse + 1$;
    $ServerKlasse_k = maxKlasse$;

Algorithmus 14 berechnet die tatsächlichen Migrationswahrscheinlichkeiten, da er alle aufgrund von $BearbZuordn_k$ erlaubten Kombinationen von Bearbeitern u_1 und u_2 für die Aktivitäten x und k durchläuft, die Wahrscheinlichkeit ihres Auftretens berechnet und die bei diesem Paar auftretende Migration in $DepMigrProb$ berücksichtigt. Die Wahrscheinlichkeit, daß Benutzer u_1 die Aktivität x bearbeitet, beträgt $\frac{Gewicht_x(u_1)}{Gesamtgewicht_1}$. Die bedingte Wahrscheinlichkeit, daß Benutzer u_2 unter dieser Voraussetzung Aktivität k bearbeitet, beträgt $\frac{Gewicht_k(u_2)}{Gesamtgewicht_2(u_1)}$. Mit diesen Wahrscheinlichkeiten gewichtet geht die Migration in $DepMigrProb$ ein. Ergibt die Berechnung von $DepServProb_x(S|u_1)$ bzw. $DepServProb_x(S|u_2)$ für das betrachtete Bearbeiterpaar u_1 und u_2, daß mehrere Server betroffen sind, so werden die Bearbeiter anteilsmäßig bei jedem dieser Server berücksichtigt. Dadurch ergibt sich der Faktor $DepServProb_x(S_1|u_1) \cdot DepServProb_x(S_2|u_2)$. 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 Aktivität x vom Server S_1 kontrolliert wird. Zeilen der Form (0 ...0) ergeben sich, wenn Server S_1 die Aktivität x nie kontrolliert und sind deshalb nicht relevant ($ServProb_x(S_1) = 0 \Rightarrow MigrProb_{x,k}(i,j) = ServProb_x(S_1) \cdot DepMigrProb_{x,k}(S_2|S_1) = 0$). Sie können deshalb bei der Normalisierung mit beliebigen Werten belegt werden. Im Anhang B.2 wird bewiesen, daß im Algorithmus 14 alle Fälle berücksichtigt werden und daß die Faktoren korrekt gewählt sind.

Algorithmus 14 (Migrationswahrscheinlichkeiten (keine Unabhängigkeit))
for each S_1, S_2 **do** $DepMigrProb_{x,k}(S_2|S_1) = 0$;
$Gesamtgewicht_1 = \sum_{u \in PotBearb_x} Gewicht_x(u)$;
for each $u_1 \in PotBearb_x$ **do**
  $DepServProb_x(S|u_1) = DepServ(x, "Bearb = u_1")$;
  $Bearb_{u_1} = \{u \mid ActorsPossible(x, u_1, k, u) = True\}$;
  $Gesamtgewicht_2(u_1) = \sum_{u \in Bearb_{u_1}} Gewicht_k(u)$;
  for each $u_2 \in Bearb_{u_1}$ **do**
    $DepServProb_x(S|u_2) = DepServ(k, "Bearb = u_2")$;
    for each S_1, S_2 **do**
      $DepMigrProb_{x,k}(S_2|S_1) = DepMigrProb_{x,k}(S_2|S_1)$
        $+ \frac{Gewicht_x(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_k(u_2)}{Gesamtgewicht_2(u_1)} \cdot DepServProb_x(S_1|u_1) \cdot DepServProb_x(S_2|u_2)$;
normalisiere jede Zeile von $DepMigrProb_{x,k}(S_2|S_1)$ so, daß $\forall S_1 : \sum_{S_2} DepMigrProb_{x,k}(S_2|S_1) = 1$;

Der mit dem soeben beschriebenen Algorithmus behandelte Fall ist in der Praxis sehr relevant, da WFs häufig für die Dauer mehrerer Aktivitäten innerhalb derselben OE verbleiben. Außerdem bilden die Benutzer einer OE oft exakt einen Domain. Angenommen, in dem in Abb. 2 dargestellten Beispiel werden die Serverzuordnungen $ServZuordn_k = "Domain(Bearb(x))"$ und $ServZuordn_{k'} = "Domain(Bearb(k))"$ verwendet, so sind die Server der Aktivitäten k und k' nicht unbedingt gleich (siehe Abb. 7). Wird weiter angenommen, daß zwei Server jeweils mit der Wahrscheinlichkeit 0,5 verwendet werden, so ergibt sich durch die einfache Approximation vom $DepMigrProb_{k,k'}(S_2|S_1)$ durch $ServProb_{k'}(S_2)$ eine Migrationswahrscheinlichkeit von 50%. In Wirklichkeit tritt aber keine Migration auf. Algorithmus 14 berücksichtigt dies, da bei allen Kombinationen von Bearbeitern für die Aktivitäten k und k' jeweils derselbe Server (nämlich der in der OE dieser Bearbeiter) ermittelt wird, womit sich $DepMigrProb_{k,k'}(S_2|S_1) = \delta_{S_1,S_2}$ ergibt. Befindet sich lediglich der Großteil der Benutzer einer OE im selben Domain, so ergibt sich annähernd $DepMigrProb_{k,k'}(S_2|S_1) = \delta_{S_1,S_2}$. In beiden Fällen wird erkannt, daß die Migrationskosten wesentlich niedriger sind, als wenn Unabhängigkeit angenommen worden wäre.

4.7 Einsparen von Migrationen

Mit den bisher gezeigten Verfahren lassen sich die Kosten für jede beliebige Verteilung berechnen. Algorithmus 6 analysiert – für jede Aktivität isoliert betrachtet – alle relevanten Serverzuordnungen. Damit ist nach der Phase 1 die optimale Verteilung bekannt, allerdings ohne Berücksichtigung der Migrationskosten. In der Phase 2 werden diese mit einbezogen, indem überprüft wird, ob die Migrationen rentabel sind. Dabei wird untersucht, ob es günstiger ist, einige von ihnen wegzulassen, indem man Partitionen zusammenfaßt (d.h. demselben Server zuordnet). Im statischen Fall ist dieses Zusammenfassen sehr einfach möglich, indem die Serverzuordnungen (IDs von WF-Servern) entsprechend ersetzt werden. Bei variablen Serverzuordnungen ist dies wegen der Abhängigkeiten der Serverzuordnungen voneinander nicht so einfach möglich. In diesem Abschnitt wird untersucht, welche Problemfälle auftreten können und wie Partitionen trotzdem zusammengefaßt werden können.

4.7.1 Zusammenfassen mit nachfolgenden Partitionen

Das Zusammenfassen mit einer auf die Partition P folgenden Aktivität m ist nicht immer möglich (vgl. Abb. 8a). 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 m' bezieht ($m' = ReferencedAct(m)$) und m' keine Vorgängeraktivität von n ist. Beim Zusamenfassen entlang einer normalen Kontrollkante tritt dieser Fall auf, wenn $m' \in P$ gilt (Abb. 8a). Soll entlang einer Synchronisationskante zwischen Aktivitäten paralleler Zweige (Abb. 8b) bzw. entlang einer Schleifenrücksprungkante (Abb. 8c) zusammengefaßt werden, so kann es sich bei m auch um eine Aktivität in einem parallelen Zweig bzw. um eine Nachfolgeraktivität von P handeln. In all diesen Fällen ist diese Art des Zusammenfassens nicht möglich. Im Fall a und c kann die Migration trotzdem weggelassen werden, indem man der Partition von Aktivität m den Server von P zuordnet, also „andersherum“ zusammenfaßt.

Das Problem, eine Partition P vom selben Server kontrollieren zu lassen, wie die Aktiviät m, kann in mehrere Teilprobleme zerlegt werden: $\forall n \in P$ soll die Aktivität n vom selben Server kontrolliert werden, wie die Aktivität m. Der Algorithmus 15 versucht für die Aktivität n eine Serverzuordnung

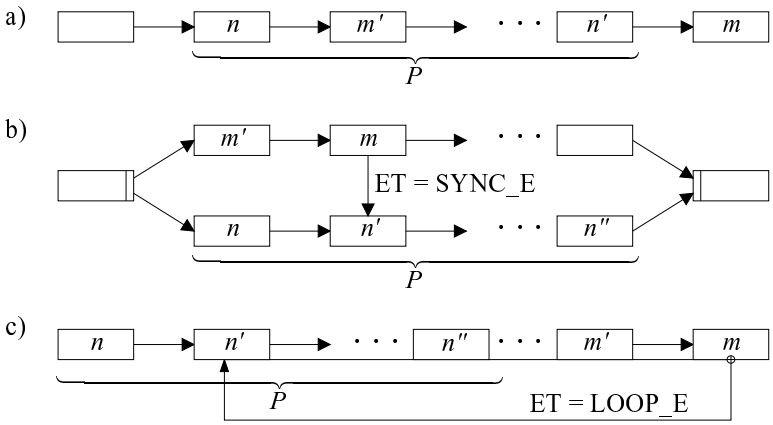


Abbildung 8 Die Partition P soll vom Server der Aktivität v bzw. y kontrolliert werden.

zu berechnen, so daß n vom selben Server kontrolliert wird, wie die Aktiviät m. Eine Aktivität n dem Server einer Vorgängeraktivität m zuzuordnen ist stets möglich, indem die Serverzuordnung $ServZuordn_n = "Server(m)"$ verwendet wird. Schwieriger ist der Fall, wenn m keine Vorgängeraktivität von n ist. Wird für Aktivität m eine statische Serverzuordnung verwendet, so kann diese für n übernommen werden. Lautet $ServZuordn_m = "Server(l)"$, so muß l eine Vorgängeraktivität von m sein. Deshalb besteht die Chance, daß l auch eine Vorgängeraktivität von n ist, womit die Aktivität l in $ServZuordn_n$ referenziert werden darf. Da bei Verwendung dieser Serverzuordnung die Server von m und l identisch sind, wird rekursiv versucht, die Aktiviät n dem Server von l zuzuordnen. Wird die Aktiviät m vom Server im Domain des Bearbeiters einer Aktivität l kontrolliert, so ergibt sich $Server_n \equiv Server_m$, wenn $ServZuordn_n = "Domain(Bearb(l))"$ verwendet wird. Dies ist aber nur möglich, falls die Aktivität l in $ServZuordn_n$ referenziert werden darf. Andernfalls ist das Zusammenfassen von n und m 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 für Aktiviät m ein vom Modellierer vorgegebener Serverzuordnungsausdruck verwendet, der nicht Typ 1-5 aus Abschnitt 4.3.1 entspricht, so kann die Semantik dieses Ausdrucks nicht verstanden werden, so daß das 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.

4.7.2 Abhängige Aktivitäten

Beim Zusammenfassen von Aktivitäten werden die Serverzuordnungen der Aktivitäten $p \in P$ geändert. Dies kann unerwünschte Auswirkungen auf eine Aktivität n haben, die später im Ablauf folgt. Soll diese Aktivität nicht vom Zusammenfassen betroffen sein ($n \notin P$), so darf die Wahl des Servers für diese Aktivität nicht durch die Änderung beeinflußt werden. Bei den folgenden Serverzuordnungen der Aktivität n tritt ein Problem auf (es gilt jeweils $m \in P$):
1. $ServZuordn_n = "Server(m)"$
2. $ServZuordn_n = "f(Server(m))"$

Der Algorithmus 16 berechnet für die Aktivität n eine Serverzuordnung $ErgServZuordn$, so daß, trotz veränderter Serverzuordnungen der Aktivitäten $p \in P$, der Server von Aktivität n unverändert bleibt. Im ersten oben beschriebenen Problemfall läßt sich Unabhängigkeit von der Ände-

Algorithmus 15 ($ModifySZ(n, m, ServZuordn_*, ErgServZuordn) : Boolean$)

// Aktivität m darf in $ServZuordn_n$ referenziert werden
if $RefAllowed(n, m)$ **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 $ModifySZ(n, l, ServZuordn_*, ErgServZuordn)$;
case $ServZuordn_m = "Domain(Bearb(l))"$:
  if $RefAllowed(n, l)$ **then**
    $ErgServZuordn = "Domain(Bearb(l))"$;
    return True;
  else
    return False;
case $ServZuordn_m = "f(Server(l))"$:
  ZusammenfassenMöglich = $ModifySZ(n, l, ServZuordn_*, ErgServZuordn')$;
  if *ZusammenfassenMöglich* **then**
    $ErgServZuordn = "f(ErgServZuordn')"$;
    return True;
  else
    return False;
case $ServZuordn_m = "f(Domain(Bearb(l)))"$:
  if $RefAllowed(n, l)$ **then**
    $ErgServZuordn = "f(Domain(Bearb(l)))"$;
    return True;
  else
    return False;
default case: // vom Modellierer vorgegebener Ausdruck
  return False;

rung erreichen, indem $ServZuordn_m$ als $ServZuordn_n$ verwendet wird. Dadurch wird Aktivität n von dem Server kontrolliert, der für Aktivität x und damit auch für m geplant war. Dazu wird in Algorithmus 16 $ServZuordn_n$ durch $ServZuordn_m$ ersetzt. Die Rekursion ist notwendig, falls in der neuen $ServZuordn_n$ weiterhin eine Aktivität $p \in P$ referenziert wird. Lautet $ServZuordn_n = "f(Server(m))"$, so kann $ServZuordn_m$ nicht direkt übernommen werden. Wird auf $ServZuordn_m$ aber noch die Funktion f angewandt, so bleibt der Server von Aktivität n unverändert. Bei allen anderen Serverzuordnungen von Aktivität n kann diese unverändert übernommen werden, da sich durch die Änderung der Serverzuordnungen für Partition P nur der Server der entsprechenden Aktivitäten ändert.[11] Alle anderen Daten bleiben unverändert und können deshalb weiterhin referenziert werden. Die Seiteneffekte auf Aktvitäten $n \notin P$ lassen sich also stets beheben. Deshalb gibt es keine Fälle, in denen sie das Zusammenfassen von Partitionen verhindern.

[11] Wird in einer Serverzuordnung vom Typ 5 der Server einer Aktivität $p \in P$ referenziert, so muß auch dieser Teil der Serverzuordnung (wie beschrieben) ersetzt werden.

Algorithmus 16 ($CopySZ(n, P, ServZuordn_*, ErgServZuordn) : Boolean$)
case $ServZuordn_n$ = "$Server(m)$":
  if $m \in P$ **then**
    $ServZuordn_n = ServZuordn_m$; // in $ServZuordn_*$ wird $ServZuordn_n$ geändert
    $CopySZ(n, P, ServZuordn_*, ErgServZuordn)$;
  else
    $ErgServZuordn$ = "$Server(m)$";
case $ServZuordn_n$ = "$f(Server(m))$":
  if $m \in P$ **then**
    $ServZuordn_n$ = "$f(ServZuordn_m)$"; // in $ServZuordn_*$ wird $ServZuordn_n$ geändert
    $CopySZ(n, P, ServZuordn_*, ErgServZuordn)$;
  else
    $ErgServZuordn$ = "$f(Server(m))$";
default case: // statische Serverzuordn., Bearbeiter referenziert, sonstiger vorgegebener Ausdruck[11]
  $ErgServZuordn = ServZuordn_n$;

4.7.3 Verteilungsalgorithmus

Wie wir in den vorherigen Unterabschnitten gesehen haben, kann bei variablen Serverzuordnungen in Phase 2 von Algorithmus 6 eine zu analysierende Verteilung ($TestServZuordn_*$) nicht durch das direkte Übernehmen der Serverzuordnungen erzeugt werden. Stattdessen müssen die Funktionen $ModifySZ$ und $CopySZ$ verwendet werden. Damit ergibt sich die folgende Änderung für die Phase 2 von Algorithmus 6:

4.8 Verhalten zur Ausführungszeit der WFs

Die bisher beschriebenen Algorithmen laufen zur Modellierungszeit ab. Zur Ausführungszeit der WFs müssen nur die Serverzuordnungen ausgewertet werden. Dies geschieht nach Beendigung jeder Aktivität für alle direkten Folgeaktivitäten, indem die referenzierten Laufzeitdaten der WF-Instanz eingesetzt werden. Wird dabei ein Server ermittelt, der vom aktuellen verschieden ist, so wird die Migration der WF-Instanz angestoßen. Migration bedeutet, daß die von den Vorgängeraktivitäten erzeugte Statusinformation und die Variablenwerte zum neuen Server transportiert werden. Um die Datenmenge klein zu halten werden nur die aktuell gültigen Variablenwerte übertragen und auf deren Historie verzichtet. Die zu transportierende Datenmenge kann noch reduziert werden (z.B. Daten nicht über mehrere parallele Zweige zum selben Join-Knoten transportieren), auf diese Optimierungsmöglichkeiten soll in dieser Arbeit aber nicht eingegangen werden.

Durch die Verwendung variabler Serverzuordnungen ergibt sich an Synchronisationspunkten (Aktivitäten an denen parallele Zweige zusammengeführt werden oder Synchronisationskanten eingehen) eine Schwierigkeit. Da der WF-Server der Synchronisationsaktivität von jeder beliebigen Vorgängeraktivität abhängig sein kann, ist dieser Server evtl. nicht in allen parallelen Zweigen bekannt. In dem in Abbildung 9 dargestellten Beispiel kann nach Beendigung von Aktivität c mit den zur Verfügung stehenden Daten nicht berechnet werden, von welchen Server die Nachfolgeraktivität e kontrolliert wird. Dies hängt von Daten (Bearbeiter der Aktivität d) des unteren Zweiges ab, die im oberen Zweig nicht zur Verfügung stehen, da sie auf einem anderen WF-Server liegen. Deshalb kann die Migration von Aktivität c nach e nicht ausgeführt werden.

Ergänzte Version von Algorithmus 6

...

Phase 2:

$MinCost = calculate\_costs(ServZuordn_*, all)$; /* Kosten gesamter WF (mit Migrationskosten) */

for *PartGröße* = 1 **to** *#Aktivitäten(Prozeßvorlage)* **do**

  for each $P : |P|$ =*PartGröße* $\wedge$ P ist max. Teilgraph mit $\forall l_1, l_2 \in P : SameServer(l_1, l_2)$ **do**

    $OptAct = NULL$;

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

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

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

      ZusammenfassenMöglich $= True$;

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

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

        $ModifySZ(l, a, ServZuordn_*, TestServZuordn_l)$;

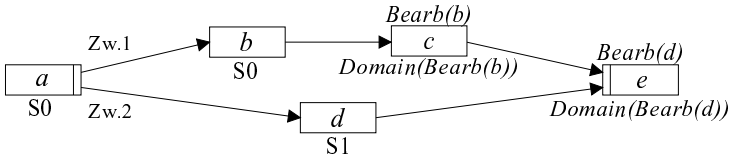


Abbildung 9 Beispiel für einen WF, bei dem der Server der Join-Aktivität e nicht in allen parallelen Zweigen berechnet werden kann.

Welche Möglichkeiten gibt es nun, die benötigte Information (Zielserver der Migration) den betroffenen Quellservern zur Verfügung zu stellen? Die ID des Zielservers direkt an alle betroffenen Quellserver der Migration zu senden, ist leider nicht möglich, da diese ihrerseits nicht immer allgemein bekannt sind. So ist in Abb. 9 nach Beendigung von Aktivität d im unteren Zweig nicht bekannt, welcher Server die Aktivität c im oberen Zweig kontrolliert. Die benötigte Information kann natürlich verbreitet werden, indem bei Erreichen einer Join-Aktivität der entsprechende Server per Broadcast allen anderen Servern des WfMSs bekannt gemacht wird. Dies führt in großen Systemen allerdings zu sehr vielen unnötigen Nachrichten. Eine weitere Möglichkeit ist, einen bestimmten Server zur Informationsvermittlung zu verwenden. Dies verschlechtert aber die Verfügbarkeit des WfMSs, weil es blockiert ist, wenn diese zentrale Informationsvermittlung ausfällt. Da zu einem fest gewählten Punkt kommuniziert wird, wird auch dann weit entfernte Kommunikation notwendig, wenn sich alle Server und die Klienten eines WFs in einem räumlich beschränkten Gebiet befinden. Es ist vorteilhaft, die Struktur des WFs auszunutzen, um die benötigte Information zu verbreiten. Der folgende Ansatz macht dies.

Algorithmus 17 beschreibt den Ablauf einer Migration zur Aktivität $TargetAct$. Kann der WF-Server S dieser Aktivität im aktuellen Zweig berechnet werden, so wird mit Hilfe der WF-Vorlage

die Menge $Pred_{\{CONTROL\_E,SYNC\_E\}}(TargetAct)$ der direkten Vorgängeraktivitäten des Migrationsziels berechnet. Die Server dieser Aktivitäten müssen das Migrationsziel kennen. Gibt es in $Pred_{\{CONTROL\_E,SYNC\_E\}}(TargetAct)$ eine Aktivität v, die keine Nachfolgeraktivität der in $ServZuordn_{TargetAct}$ referenzierten Aktivität x ist, so kann der Server von v den Migrationszielserver S nicht berechnen. In diesem Fall wird ihm S wie folgt 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 Migrationszielserver S erfragen. Damit der Zuordnungsausdruck des Dispatchers von diesem Server ausgewertet werden kann, muß der Zuordnungsausdruck statisch sein oder es muß eine Aktivität referenziert werden, die Vorgänger aller $v \in Pred_{\{CONTROL\_E,SYNC\_E\}}(TargetAct)$ ist. Wenn der Zielserver S einer Migration nicht berechnet werden kann, 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 Kommunikationen einzusparen, ist es auch möglich, diese Anfragen Huckepack [Tan92] mit anderen Nachrichten zu transportieren.

Algorithmus 17 (Migration zur Aktivität $TargetAct$)
if ($S = ServZuordn_{TargetAct}$) kann berechnet werden **then**
  if $ServZuordn_{TargetAct}$ referenziert Aktivität $x \wedge$
  $\neg\forall v \in Pred_{\{CONTROL\_E,SYNC\_E\}}(TargetAct)$: $v \in Succ_{\{CONTROL\_E,SYNC\_E\}}(x)$ **then**
    $NotifyMigration(InstId, TargetAct, S) \longrightarrow Dispatcher_{TargetAct}$;
else
  Nachfragen: $S = InfoMigration(InstId, TargetAct) \longrightarrow Dispatcher_{TargetAct}$;
  if $S = unknown$ **then wait**; **goto** Nachfragen;
$Migrate(Instance) \longrightarrow S$;

Das in Algorithmus 17 beschriebene Verfahren ist recht effizient, da nur wenige sehr kleine Nachrichten ausgetauscht werden. Es kann aber noch optimiert werden. Um mehrfache Aufrufe von $InfoMigration()$ zu vermeiden, kann sich der Dispatcher noch nicht beantwortbare Anfragen merken und beantworten, sobald er die benötigte Information besitzt.

Nach Beendigung einer Aktivität findet der Übergang zu den Nachfolgeraktivitäten asynchron statt. Deshalb kann dies überlappend geschehen. Warum diese Asynchonität notwendig ist, wird anhand von Abb. 10 erläutert: Wird zuerst mit der Signalisierung der Synchronisationskante $b \rightarrow e$ begonnen, so kann die Aktivität c erst gestartet werden, wenn diese Aktion abgeschlossen ist. Dies ist nach Algorithmus 17 erst möglich, wenn der WF-Server von Aktivität e feststeht. Da dies aber u.U. erst sehr viel später der Fall ist, wird die Ausführung der Aktivität c unzulässigerweise verzögert. Durch den asnchronen (überlappenden) Übergang von Aktivität b nach e und von b nach c wird dies vermieden.

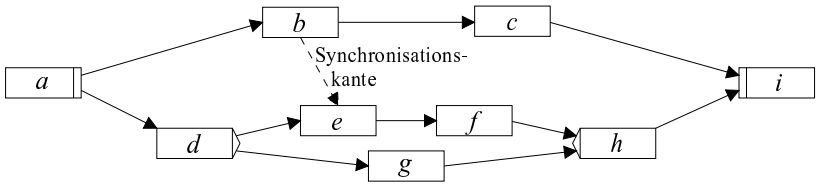


Abbildung 10 Synchronisation mit einer Aktivität aus eine exklusiven Verzweigung.

Steht der gewählte Zweig einer exklusiven Aufspaltung (z.B. bei Aktivität d in Abb. 10) fest, so ergibt sich im ADEPT-Modell [RD98] für die Aktivitäten aller anderen Zweige der Zustand $skipped$.

Wird z.B. der Zweig der Aktivität g gewählt, so können die Aktivitäten e und f nicht mehr ausgeführt werden, so daß sich der Zustand $skipped$ ergibt. Wie im zentralen Fall werden dann alle von diesen Aktivitäten ausgehenden Synchonisationskanten signalisiert (falls es z.B. in Abb. 10 eine Synchonisationskante von f nach c gäbe, so würde diese nach Beendigung von Aktivität d und der Entscheidung für den unteren Zweig signalisiert). Dies wird von dem Server durchgeführt, der die Berechnung dieses Zustands durchführt, also in diesem Beispiel vom Server der Aktivität d. Es findet also keine Migration zu dem für Aktivität f vorgesehenen Server statt, um die Synchronisationskante $f \rightarrow c$ zu signalisieren. Dadurch wird eine Migration vermieden, die (weil die Aktivität f überhaupt nicht ausgeführt wird) unnötig ist. Außerdem gibt es Fälle, in denen der Server der Aktivität f nicht bestimmbar ist, so daß auch nicht zu ihm migriert werden kann. So ist $ServZuordn_f = "Domain(Bearb(e))"$ zulässig, da e immer vor f ausgeführt wird, falls f ausgeführt wird. Diese Serverzuordnung kann im vorliegenden Fall aber nicht ausgewertet werden, da die Aktivität e nicht ausgeführt wird.

Wird in dem in Abb. 10 dargestellten Beispiel der untere Ausführungszweig (Aktivität g) gewählt, so ergibt sich eine Schwierigkeit bei der Signalisierung der Synchronisationskante $b \rightarrow e$. Da die Aktivität e nicht ausgeführt wird, gibt es auch keinen WF-Server, der ihre Ausführung kontrolliert. Das Problem ist nun, daß die Aktivität b in der Schleife von Algorithmus 17 ständig versucht, diesen Server vom $Dispatcher_e$ zu erfragen. Damit dieser Vorgang terminiert, muß auch dann ein Server an den Dispatcher gemeldet werden, wenn die Aktivität nie ausgeführt wird, sondern den Zustand $skipped$ erreicht[12]. Für die ausgelassenen Aktivitäten kann z.B. der Server gemeldet werden, der die Berechnung des Zustandes durchführt; hier also der Server der Aktivität d. Ist diese Information beim $Dispatcher_e$ verfügbar, so kann die Synchronisationskante $b \rightarrow e$ signalisiert werden, so daß die Bearbeitung der Aktivität b beendet werden kann.

Es bleibt noch zu klären, wie der Zuordnungsausdruck für den Dispatcher gewählt werden soll. Da mit dem Dispatcher nur kleine Nachrichten ausgetauscht werden, ist es eine akzeptable Möglichkeit, die vom Algorithmus 6 berechnete beste statische Serverzuordnung zu verwenden. Eine einfache Alternative dazu stellt die Verwendung des Startservers der WF-Instanz dar. Um eine hohe Lokalität zwischen den Aktivitäten aus $Pred_{\{CONTROL\_E,SYNC\_E\}}(TargetAct)$ und dem Dispatcher zu erhalten, sollte aber der Server einer Aktivität gewählt werden, die im WF-Ablauf nahe bei $TargetAct$ liegt. Hierfür bietet sich z.B. der letzte gemeinsame Vorgängeraktivität der Aktivitäten $v \in Pred_{\{CONTROL\_E,SYNC\_E\}}(TargetAct)$ bzgl. des Kontrollflusses an.

5 Komplexe Bearbeiterzuordnungen

Wie wir gezeigt haben können Bearbeiterzuordnungen von vorausgehenden Aktivitäten abhängig sein. Es ist sogar möglich, daß eine Bearbeiterzuordnung von mehreren Aktivitäten abhängt, z.B. wenn Aktivitäten einer Verzweigung referenziert werden sollen. In Abschnitt 5.1 wird der Zweck solcher Bearbeiterzuordnungen ausführlich diskutiert. Wenn der Modellierer eine komplexe Bearbeiterzuordnung für eine Aktivität k verwendet, gibt es zwei prinzipielle Vorgehensweisen beim Ermitteln einer geeigneten Serverzuordnung für diese Aktivität:
1. Die Bearbeiterzuordnung wird nicht als abhängige Bearbeiterzuordnung (im Sinne von Kapitel 4.1.1) behandelt. Dann wird lediglich die daraus resultierende Bearbeitermenge $PotBearb_k$ (siehe Abschnitt 4.1.2) ermittelt, um einen geeigneten Server auszuwählen.
2. Aus der komplexen abhängigen Bearbeiterzuordnung wird eine optimale variable Serverzuordnung

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

abgeleitet.
Da im Fall 1 die Abhängigkeiten ignoriert werden, ergibt sich für die Aktivität k eine statische Serverzuordnung. Die Vorteile von variablen Serverzuordnungen können damit nicht genutzt werden. Deshalb haben wir uns für die Variante 2 entschieden. Damit variable Serverzuordnungen automatisch berechnet werden können, muß eine Methode entwickelt werden, die es erlaubt, die Kosten im komplexen Fall zu berechnen. Um die bisher entwickelten Algorithmen weiter nutzen zu können, wird der Fall komplexer abhängiger Bearbeiterzuordnungen auf den in Kapitel 4 beschriebenen Fall nicht zusammengesetzter Bearbeiterzuordnungen zurückgeführt. Wie dies exakt funktioniert, wird in dem nun folgenden Kapitel beschrieben.

5.1 Verwendung komplexer Bearbeiterzuordnungen

In Abbildung 11 sind die elementaren Fälle komplexer Bearbeiterzuordnungen dargestellt. Diese können zu noch komplexeren kombiniert werden. In den abgebildeten Beispielen wurden stets Bearbeiterzuordnungen des Typs $BearbZuordn_k$ = "$Bearb(x)$" verwendet. Das im folgenden Gesagte gilt aber auch für andere Arten von Bearbeiterzuordnungen (z.B. $BearbZuordn_k$ = "$OE(Bearb(x)$") $\wedge$ $Rolle = \ldots$).

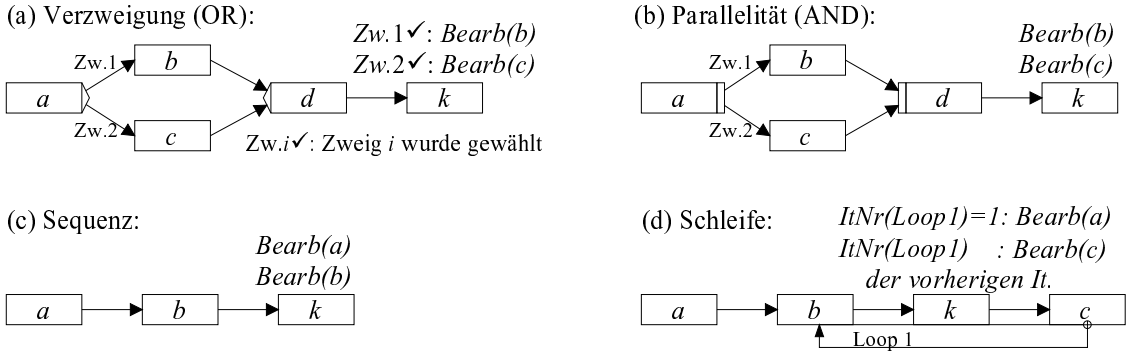


Abbildung 11 Beispiele für Bearbeiterzuordnungen, die sich auf mehrere Aktivitäten beziehen.

In Abbildung 11a bezieht sich die Bearbeiterzuordnung auf die Aktivitäten einer Verzweigung. Eine zusammengesetzte Bearbeiterzuordnung ist in diesem Fall die einzige Möglichkeit, Aktivitäten aus der Verzweigung zu referenzieren. Durch eine einfache Bearbeiterzuordnung kann maximal ein Zweig referenziert werden. Da dieser aber nicht in allen Fällen durchlaufen wird, wäre die Bearbeiterzuordnung manchmal undefiniert. Deshalb wird jeweils eine Aktivität jedes Zweiges referenziert. Natürlich ist für manche der Zweige auch eine unabhängige Bearbeiterzuordnung (z.B. über eine Rolle) zulässig und evtl. auch sinnvoll.

Im Fall einer Parallelität (b) werden stets alle Zweige durchlaufen. Deshalb liefern alle referenzierten Aktivitäten einen Bearbeiter. Die Menge der potentiellen Bearbeiter von Aktivität k ist die Vereinigungsmenge der Bearbeiter dieser Aktivitäten. Dasselbe gilt für Sequenzen (Fall c), da auch hier alle Aktivitäten ausgeführt werden.

In Abbildung 11d ist eine Schleife dargestellt. Die Bearbeiterzuordnung von Aktivität k soll sich auf den Bearbeiter einer (evtl. derselben) Aktivität des vorherigen Schleifendurchlaufs beziehen. Dadurch ergibt sich das Problem, daß die Bearbeiterzuordnung in der ersten Iteration undefiniert ist. Deshalb muß dieser Fall in der Bearbeiterzuordnung gesondert behandelt werden, indem sich der Bearbeiter z.B. aus einer vor der Schleife ausgeführten Aktivität ergibt. Es können natürlich auch für weitere

Iterationen spezielle Bearbeiterzuordnungen festgelegt werden. Entscheidend bei Bearbeiterzuordnungen in Schleifen ist aber, daß für jede Iteration jeweils nur ein Teil der Bearbeiterzuordnung gültig ist (ähnlich wie bei einer Verzweigung).

Die einzelnen Teile der Bearbeiterzuordnungen in (a) - (d) sind OR-verknüpft (wenn auch manchmal nur ein Teil gleichzeitig gültig sein kann). Eine AND-Verknüpfung macht wenig Sinn, da es dann von den konkret gewählten Bearbeitern bei der WF-Ausführung abhängig ist, ob überhaupt noch ein Bearbeiter den Zuordnungsausdruck erfüllt. Dadurch kann es in Prozessen zu Inkonsistenzen kommen, die zur Modellierungszeit nicht entdeckt werden können.

5.2 Potentiell mögliche Serverzuordnungen

Auch bei komplexen Bearbeiterzuordnungen werden die in Abschnitt 4.3.1 beschriebenen Serverzuordnungsausdrücke verwendet. Allerdings können diese zu noch komplexeren Ausdrücken zusammengesetzt werden. Wie diese im Detail aussehen, wird im nächsten Abschnitt beschrieben.

Das Ermitteln der für eine Referenzierung in einer Serverzuordnung relevanten Aktivitäten funktioniert prinzipiell wie im Abschnitt 4.3.2 für den nicht komplexen Fall beschrieben; Algorithmus 4 wird weiterhin verwendet. Allerdings können in $BearbZuordn_k$ jetzt mehrere Aktivitäten $x_1 \ldots x_n$ referenziert werden. Deshalb muß die Funktion $inital\_dep$ so erweitert werden, daß auch komplexe Bearbeiterzuordnungen berücksichtigt werden können. Für das in Abb. 11a dargestellte Beispiel ergibt sich z.B. die Abhängigkeit $\langle k, [EXCL(Zw.1\surd\colon [b, B]; Zw.2\surd\colon [c, B])]\rangle$. Es müssen also wie bisher alle abhängigen Aktivitäten und der Typ der Abhängigkeit (selber Bearbeiter oder selbe OE) und zusätzlich das entsprechende Konstrukt (Verzweigung: $EXCL$[13], Parallelität oder Sequenz: OR, Schleife: $LOOP$) mit der zugehörigen Zusatzinformation (Zweig-Id, Iterationsnummer) ermittelt werden.

Auch die Funktion $substitute\_dep$ muß erweitert werden. Es reicht nicht mehr aus, komplette Abhängigkeiten zu ersetzen, sondern es müssen auch Fragmente substituiert werden. Dabei werden diese ggf. auch durch komplexe Abhängigkeiten ersetzt. Falls z.B. in dem in Abb. 11a dargestellten Fall zusätzlich die Abhängigkeit $\langle b, [a, OE]\rangle$ existiert (wegen $BearbZuordn_b$), so läßt sich zusammen mit der oben angegeben Abhängigkeit für Aktivität k $\langle k, [EXCL(Zw.1\surd\colon [a, OE]; Zw.2\surd\colon [c, B])]\rangle \in Dependencies$ ableiten. Außerdem muß $substitute\_dep$ so erweitert werden, daß im Fall von $EXCL$ und $LOOP$ gesamte Blöcke ersetzt werden können (z.B. $[EXCL(\ldots)]$ durch $[k, OE]$). Diese Blöcke müssen bis auf den Typ (B oder OE) in den beiden Abhängigkeiten identisch sein. Auch dies soll an einem Beispiel erläutert werden: in Abb. 11a folge auf die Aktivität k eine Aktivität k' mit $BearbZuordn_{k'} = \text{"}Zw.1\surd\colon OE(Bearb(b)) \wedge Rolle = Arzt; Zw.2\surd\colon Bearb(c)\text{"}$. Daraus folgt $\langle k', [EXCL(Zw.1\surd\colon [b, OE]; Zw.2\surd\colon [c, B])]\rangle$. Mit dieser Abhängigkeit kann man schließen, daß k und k' stets in derselben OE bearbeitet werden: $\langle k', [k, OE]\rangle$. Derselbe Bearbeiter ist nicht garantiert, da der Bearbeiter von Aktivität k' – für den Fall daß Zweig 1 gewählt wird – nur aus derselben OE wie der von b stammt. Eine analoge Schlußfolgerung gilt nicht für den Fall OR (Parallelität, Sequenz), da hier mehrere Teile der Bearbeiterzuordnung gleichzeitig gültig sind. Deshalb kann der Bearbeiter von Aktivität k auf einem anderen Teil resultieren wie der von Aktivität k', womit zwischen den Bearbeitern von k und k' kein Zusammenhang mehr besteht. Dagegen ist im Fall LOOP eine solche Ersetzung möglich. Allerdings muß – wie immer bei Abhängigkeitsbeschreibungen von Schleifen – darauf geachtet werden, daß sich die zu ersetzenden Teile der beiden Abhängigkeiten auf dieselbe Iteration beziehen.

[13] $EXCL$ deutet an, daß die Teilausdrücke durch ein exclusives Oder verbunden sind, d.h., daß genau einer der Zweige $Zw.1$, $Zw.2$, ... gewählt wird.

5.3 Berechnung der Serverzuordnungen

Aus der Bearbeiterzuordnung einer Aktivität wird beim Berechnen der Verteilung ihre Serverzuordnung abgeleitet. In diesem Abschnitt werden die Algorithmen beschrieben, die bei den elementaren zusammengesetzten Bearbeiterzuordnungen (siehe Abbildung 11) verwendet werden, um die aus einer Serverzuordnung resultierenden Kosten zu berechnen. Die Behandlung der nicht elementaren (verschachtelten) Fälle wird im nächsten Abschnitt diskutiert.

Eine Serverzuordnung muß eindeutig einen Server festlegen. In den nun folgenden Unterabschnitten wird für jeden Typ von elementarer Bearbeiterzuordnung untersucht, welche Serverzuordnungen diese Bedingung erfüllen und wie die zugehörigen Kosten berechnet werden können. Durch eine komplexe Bearbeiterzuordnung kann sich eine komplexe Serverzuordnung ergeben. Wie immer wird diese aber nur dann verwendet, wenn die dadurch entstehenden Kosten minimal sind. So gibt es auch Fälle, in denen es günstiger ist, z.B. eine vor der betrachteten Verzweigung liegende Aktivität zu referenzieren oder den Server statisch festzulegen.

5.3.1 Verzweigungen

Im Falle der in Abbildung 11a dargestellten Verzweigung wird entweder Zweig 1 oder Zweig 2 ausgeführt, da genau ein Zweig durchlaufen wird. Deshalb ist es möglich eine Serverzuordnung der Art "$Zw.1\surd$: $ServZuordn^1$; $Zw.2\surd$: $ServZuordn^2$"Z zu verwenden. Da stets genau einer der Fälle eintritt, ist die Serverzuordnung immer definiert und eindeutig. In $ServZuordn^1$ können Aktivitäten referenziert werden, die in Zweig 1 liegen (z.B. Aktivität b); $ServZuordn^2$ bezieht sich auf Zweig 2. Welche Aktivitäten x für eine Referenzierung in $ServZuordn_i$ in Frage kommen, ergibt sich aus den Abhängigkeiten $\langle k, [EXCL(\ldots Zw.i\surd\text{: } [x, T]; \ldots)]\rangle$. Die möglichen $ServZuordn^i$ sind, wie in Abschnitt 4.3.1 beschrieben, statische Serverzuordnungen und Ausdrücke, die eine abhängige Aktivität x referenzieren. Für jede betrachtete $ServZuordn^i$ und den jeweils entsprechenden Teil der Bearbeiterzuordnung $BearbZuordn^i_k$ werden die WVen $ServProb^i_k(S)$ und $ActorProb^i_k(D|S)$ und die daraus resultierenden Datenvolumina $Vol^i(k)$ getrennt berechnet (wie für den nicht komplexen Fall in Abschnitt 4.5 beschrieben). Aus den für den i-ten Zweig in Frage kommenden $ServZuordn^i$ wird diejenige ausgewählt, die zur minimalen Last K führt. Dazu wird die Zielfunktion K, wie in Abschnitt 4.2 beschrieben, aus den $Vol^i(k)$ berechnet. Allerdings wird dabei nur der i-te Zweig ($BearbZuordn^i_k$ und $ServZuordn^i$) berücksichtigt. Zur Berechnung des Gesamtergebnisses von Aktivität k werden jeweils die optimalen Teilergebnisse gewichtet aufaddiert. Die Gewichte p_i entsprechen dabei den Ausführungswahrscheinlichkeiten der Zweige[14] i.

$$ServProb_k(S) = \sum_i p_i \cdot \; ServProb^i_k(S)$$

$$ActorProb_k(D|S) = \sum_i p_i \cdot \; ActorProb^i_k(D|S)$$

$$Vol(k) = \sum_i p_i \cdot \; Vol^i(k)$$

Es wurde also ein Vorgehen gewählt, bei dem für jeden Zweig i die optimale Serverzuordnung getrennt berechnet wird und diese dann zu der Gesamt-Serverzuordnung zusammengesetzt werden. Die

[14] Diese Wahrscheinlichkeiten gehören zu den statistischen Daten, die entweder durch eine Analyse der Audit Trails des WfMSs gewonnen werden oder vom Modellierer vorgegeben werden. Sie werden auch benötigt, um die Ausführungswahrscheinlichkeiten der Aktivitäten eines Zweiges zu berechnen, um so auf den Erwartungswert der durch ihre Ausführung entstehenden Last schließen zu können.

Alternative wäre, alle in den Abhängigkeiten für Aktivität k $\langle k, [EXCL(\ldots)]\rangle$ ermittelten Kombinationen von Aktivitäten herzunehmen und die Kosten für alle daraus resultierenden Kombinationen von Serverzuordnungen zu berechnen. Die Serverzuordnung mit den niedrigsten Kosten würde dann ausgewählt. Diese Alternative verursacht durch die hohe Anzahl der zu untersuchenden Gesamt-Serverzuordnungen (kombinatorische Vielfalt) einen großen Aufwand. Sie bringt keinen Vorteil, da auch das gewählte Verfahren die optimale Lösung ermittelt, da die einzelnen Zweige voneinander unabhängig sind. Dies ist der Fall, da sich die Gesamtkosten $Vol(k)$ durch Addition der Einzelkosten $Vol^i(k)$ ergeben und die Gewichtungsfaktoren p_i fest sind.

5.3.2 Disjunktion von Teil-Bearbeiterzuordnungen

Tritt in einem Ablauf Parallelität auf (Fall b), so werden alle Zweige durchlaufen. Die Bearbeiterzuordnung stellt die Disjunktion mehrerer Teilbeschreibungen dar. Die Serverzuordnung darf aber keine Vereinigungsmenge mehrerer Server darstellen, da sie dann nicht mehr eindeutig wäre. In der Serverzuordnung darf also maximal eine Aktivität referenziert werden. Dasselbe gilt, wenn in einer Bearbeiterzuordnung die Aktivitäten einer Sequenz referenziert werden (Fall c). Dies läßt sich zu dem Fall verallgemeinern, daß eine Bearbeiterzuordnung aus mehreren durch Disjunktion verbundenen Teilen besteht (egal, ob diese andere Aktivitäten referenzieren oder nicht). Ein Beispiel hierfür ist:
$BearbZuordn_k =$ "$(OE(Bearb(x)) \wedge Rolle = Arzt) \vee (OE = HNO \wedge Rolle = Arzt)$"

Obwohl nur die in Abschnitt 4.3.1 beschriebenen Serverzuordnungen verwendet werden, ist die Berechnung der Kosten komplizierter als in Kapitel 4. Da die Bearbeiterzuordnung zusammengesetzt ist, läßt sich der Fall bei der Berechnung von $DepServProb_k(S|u)$ nicht mehr geschlossenen behandeln (vgl. Abschnitt 4.5.4). Für jeden Teil der Bearbeiterzuordnung $BearbZuordn_k^i$ werden deshalb die WVen $ServProb_k^i(S)$ und $ActorProb_k^i(D|S)$ getrennt berechnet. Diese Teile werden dann gewichtet addiert, wobei die Gewichte p_i der Wahrscheinlichkeit entsprechen, daß der tatsächliche Bearbeiter der Aktivität aus dem gerade betrachteten Teil der Bearbeiterzuordnung resultiert. Sind die WVen berechnet, so werden sie verwendet, um die resultierenden Kosten (Datenvolumina) zu berechnen, anhand derer die optimale Serverzuordnung ausgewählt wird.
$ServProb_k(S) = \sum_i p_i \cdot \; ServProb_k^i(S)$
$ActorProb_k(D|S) = \sum_i p_i \cdot \; ActorProb_k^i(D|S)$

Es bleibt noch zu klären, wie die Gewichte p_i für die einzelnen Teile der Bearbeiterzuordnung ermittelt werden. Dabei soll p_i der Wahrscheinlichkeit entsprechen, daß der tatsächliche Bearbeiter von Aktivität k aus $BearbZuordn_k^i$ resultiert. Diese Wahrscheinlichkeit entspricht dem Anteil der Bearbeiter aus $BearbZuordn_k$, der sich aus $BearbZuordn_k^i$ ergibt. Dabei wird angenommen, daß alle Bearbeiter die Aktivität k mit der gleichen Wahrscheinlichkeit ausführen, unabhängig davon, aus welcher Teil-Bearbeiterzuordnung sie stammen. Trifft diese Annahme nicht zu, so kann der Modellierer geeignete Gewichte p_i manuell vorgeben oder die berechneten Werte überschreiben.

Algorithmus 18 berechnet die Gewichte p_i für $BearbZuordn_k$ = "$BearbZuordn_k^1 \vee \ldots \vee BearbZuordn_k^n$". Zuerst werden alle Bearbeiter von Aktivität k durchlaufen, wobei ihre Gewichte aufaddiert werden. Außerdem werden die Bearbeiter gezählt. Beides erfolgt getrennt nach OEen und für jeden Teil von $BearbZuordn_k$. Ein Bearbeiter u, der aus $A(u)$ Teilen von $BearbZuordn_k$ stammt, wird jeweils nur mit einem Anteil von $1/A(u)$ berücksichtigt.

In einer 2. Phase berechnet der Algorithmus für jede $BearbZuordn_k^i$ die durchschnittlich resultierende Anzahl von Bearbeitern B_i. Dabei muß nach dem Typ von $BearbZuordn_k^i$ unterschieden werden:
1. Wird direkt der Bearbeiter von Aktivität x referenziert, so geht aus diesem Teil der Bearbeiterzuordnung maximal ein Bearbeiter hervor. B_i wird berechnet, indem für jede OE das durchschnittliche Gewicht ihrer Bearbeiter berechnet wird. Diese Gewichte werden aufaddiert. Dies erfolgt gewichtet mit der Wahrscheinlichkeit $AnteilAusf_k^i$ (siehe Abschnitt 5.5.1), daß die Aktivität k in dieser OE ausgeführt wird (wobei nur $BearbZuordn_k^i$ berücksichtigt wird).
2. Hat $BearbZuordn_k^i$ die Form "$OE(Bearb(x)) \wedge \ldots$", so qualifiziert sich nicht nur ein Bearbeiter für Aktivität k, sondern alle Bearbeiter der OE, die diese Bedingung erfüllen. Deshalb ist in diesem Fall nicht deren durchschnittliches Gewicht, sondern deren Gesamtgewicht relevant. Das Gesamtgewicht aller OEen wird wie im Fall 1 gewichtet aufaddiert.
3. Ist $BearbZordn_k^i$ von anderen Aktivitäten unabhängig, dann qualifizieren sich stets alle Bearbeiter, so daß B_i unabhängig von irgendwelchen OEen berechnet wird.
B_i stellt nun die „Größe der Bearbeitermenge“, die durch $BearbZuordn_k^i$ resultiert, dar. In der 3. Phase berechnet Algorithmus 18 nun die Wahrscheinlichkeit p_i, mit der ein potentieller Bearbeiter von Aktivität k sich wegen $BearbZuordn_i(k)$ qualifiziert. p_i ist gleich dem Anteil von B_i an der Summe aller B_i.

Algorithmus 18 (p_i bei einer Disjunktion von Bearbeiterzuordnungen)

Phase 1:
/* Für jede OE, jeden Teil $BearbZuordn_k^i$: Gesamtgewicht und Anzahl Bearbeiter ermitteln */
$\forall i = 1 \ldots n$: berechne $PotBearb_k^i$ wie $PotBearb_k$,
wobei anstelle von $BearbZuordn_k$ nur der Teil $BearbZuordn_k^i$ verwendet wird;
for each u **do**
  $A(u) = \left|\{j \mid u \in PotBearb_k^i\}\right|$;
  $OE = OE(u)$;
  for i = 1 **to** n **do**
    if $u \in PotBearb_k^i$ **then**
      $Gesamtgewicht^i(OE) = Gesamtgewicht^i(OE) + G_k(u) \,/\, A(u)$;
      $AnzahlBearb^i(OE) = AnzahlBearb^i(OE) + 1/A(u)$;

Phase 2:
/* Größe der Bearbeitermenge B_i von $BearbZuordn_k^i$ berechnen */
for i = 1 **to** n **do**
  case $BearbZuordn_k^i$ = "$Bearb(x)$": (1)
    $B_i = \sum_{OE} \frac{Gesamtgewicht^i(OE)}{AnzahlBearb^i(OE)} \cdot AnteilAusf_k^i(OE)$;
  case $BearbZuordn_k^i$ = "$OE(Bearb(x)) \wedge \ldots$": (2)
    $B_i = \sum_{OE} Gesamtgewicht^i(OE) \cdot AnteilAusf_k^i(OE)$;
  case $BearbZuordn_i(k)$ unabhängig von x: (3)
    $B_i = \sum_{OE} Gesamtgewicht^i(OE)$;

Phase 3:
/* Wahrscheinlichkeiten p_i aus Größe des entsprechenden B_i berechnen */
$Summe = \sum_{i=1}^{n} B_i$;
for i = 1 **to** n **do**
  $p_i = B_i / Summe$;

5.3.3 Schleifen

Da die Bearbeiterzuordnung für jede Iteration einer Schleife verschieden sein kann, ergeben sich jeweils unterschiedliche Bearbeiter-WVen und Kosten. Dies kann zu einer unterschiedlichen optimalen Serverzuordnung für jede Iteration führen. Selbst wenn nur die erste Iteration in den Bearbeiterzuordnungen gesondert behandelt wird, kann dies die Bearbeiter-WVen unter Umständen noch für einige Iterationen beeinflussen. Der Grund dafür ist, daß Bearbeiterzuordnungen von nachfolgenden Aktivitäten abhängen können, und das auch noch mehrstufig. Das besondere Verhalten der ersten Iteration wandert also bei jeder Iteration nur um eine Stufe nach vorne. Deshalb kann es einige Schleifendurchläufe dauern, bis ein „eingeschwungener" Zustand erreicht ist. Es gibt sogar Fälle, in denen nie ein eingeschwungener Zustand erreicht wird. Ein Beispiel hierfür sind zwei parallele Aktivitäten in einer Schleife, zwischen denen bei jeder Iteration die Bearbeiter getauscht werden.

Werden Schleifen verschachtelt, so hat die Verwendung von iterations-abhängigen Serverzuordnungen in der äußeren Schleife Auswirkungen auf die innere Schleife. Ob in dem in Abbildung 12 dargestellten WF eine Migration zwischen den Aktivitäten x_1 und y_1 (ebenso zwischen y_n und x_2) stattfindet, hängt auch vom Server von Aktivität x_1 ab. Damit sind für die Serverzuordnungen der inneren Schleife auch die gültigen Serverzuordnungen der äußeren Schleife relevant. Da in der inneren Schleife die Server der äußeren referenziert werden können (z.B. $ServZuordn_{y_n} = "Server(x_1)"$), hängen auch WVen der inneren Schleife von den aktuell gültigen Serverzuordnungen der äußeren Schleife ab. Deshalb hängen die optimalen Serverzuordnungen der inneren Schleife davon ab, in welcher Iteration sich die äußere Schleife befindet.[15] Für Aktivitäten in Schleifen ergeben sich Serverzuordnungen der Form:

$$ServZuordn_k = "\bigvee_i \Big(\bigwedge_{LoopId} ItNr(LoopId)\ op\ ItNr_{LoopId,i} \Big) \colon ServZuordn^i"$$

mit $op \in \{=, \geq\}$

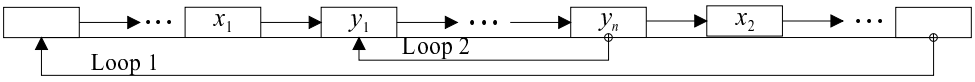


Abbildung 12 Verschachtelte Schleifen.

Um für jede Iteration die jeweils optimale Serverzuordnung bestimmen zu können, müssen bei der Analyse einige Schleifendurchläufe durchgespielt werden. Dies wird prinzipiell solange fortgesetzt, bis sich gegenüber der letzten Iteration nichts mehr verändert hat. Um den Aufwand zu begrenzen und da es Fälle gibt, in denen kein „eingeschwungener" Zustand erreicht wird, wird die Analyse nach n Iterationen abgebrochen. Dabei führt ein großes n zu einem genauen Ergebnis und einem hohen Analyseaufwand. In Anhang B.3 wird gezeigt, daß der Anteil der nicht analysierten Iteratinen kleiner als e^{-a} ist, wenn $n = a \cdot \#It$ ($\#It$: (durchschnittliche) Anzahl der Durchläufe)[16] Iterationen analysiert werden. Wird z.B. $n = 4 \cdot \#It$ gewählt, so ist die Wahrscheinlichkeit, daß noch weitere Iterationen ausgeführt werden, kleiner als 2%. Für diese Iterationen wird das Verhalten (und die Serverzuordnung) der letzten analysierten Iteration angenommen. Auch ihre Kosten werden berücksichtigt, allerdings nicht mit dem exakten Wert, sondern mit den WVen der Iteration n. Da dies eine

[15] Prinzipiell wäre dieselbe Form auch für Bearbeiterzuordnungen in verschachtelten Schleifen möglich (z.B. $BearbZuordn_{y_n} = "(ItNr(Loop1) = 1 \wedge ItNr(Loop2) = 1)\colon Bearb(x_1); (ItNr(Loop1) = 1 \wedge ItNr(Loop2) \geq 2)\colon Rolle\ Arzt; (ItNr(Loop1) \geq 2 \wedge ItNr(Loop2) \geq 1)\colon Bearb(y_1)"$). Da diese aber für den Modellierer schwer zu verstehen sind und in der Praxis wohl nicht benötigt werden, wird auf diese Variante nicht weiter eingegangen.

[16] Die durchschnittliche Anzahl von Iterationen ($\#It$) wird aus den Audit Trails ermittelt oder vom Modellierer vorgegeben. Diese Information ist auch notwendig, um die durch die Schleife verursachten Kosten abschätzen zu können.

recht gut Näherung darstellt, ist der Fehler noch kleiner als e^{-a}.

Nach dem Durchspielen der Iterationen sind die optimalen Serverzuordnungen, die WVen und die Kosten für jede einzelne Iteration bekannt. Diese müssen nun noch zu einem Gesamtergebnis zusammengesetzt werden. Wie eine zusammengesetzte Serverzuordnung aussieht, wurde schon erklärt. Die WVen $ServProb_k(S)$ und $ActorProb_k(D|S)$ werden für Aktivitäten benötigt, die auf die Schleife folgen und Aktivität k referenzieren. Die Daten werden verwendet, um die Server- und Bearbeiter-WVen dieser Aktivitäten zu berechnen. Für diese Aktivitäten sind die Werte der letzten ausgeführten Schleifeniteration relevant, da sich auch ihre Server- und Bearbeiterzuordnungen auf die letzte Iteration beziehen. Die WVen werden mit Hilfe der folgenden statistischen Überlegung zusammengesetzt. Wenn durchschnittlich $\#It$ Iterationen der Schleife durchlaufen werden, so beträgt die Wahrscheinlichkeit zum Verlassen der Schleife bei jeder Iteration $q = 1/\#It$, falls das Verlassen der Schleife zufällig erfolgt.[17] Die Wahrscheinlichkeit, daß die Schleife nach exakt i Iterationen verlassen wird, beträgt damit $p_i = (1-q)^{i-1} \cdot q$ (die Schleife wurde $i-1$ mal nicht verlassen und wird dann verlassen). Die Server- und Bearbeiter-WVen werden mit p_i gewichtet aufaddiert (der Wahrscheinlichkeit, mit der die Schleife nach Iteration i verlassen wird, so daß es die letzte Iteration war). Natürlich muß auch der Anteil der nicht mehr analysierten Iterationen berücksichtigt werden. Dies geschieht durch ein größeres Gewicht für die letzte analysierte Iteration n.

$$ServProb_k(S) = \sum_{i=1}^{n-1} p_i \cdot \; ServProb_k^i(S) + (1 - \sum_{i=1}^{n-1} p_i) \cdot \; ServProb_k^n(S)$$

$$ActorProb_k(D|S) = \sum_{i=1}^{n-1} p_i \cdot \; ActorProb_k^i(D|S) + (1 - \sum_{i=1}^{n-1} p_i) \cdot \; ActorProb_k^n(D|S)$$

Bei Schleifen werden die Kosten (Datenvolumina) für die einzelnen Schleifendurchläufe $Vol^i(k)$ getrennt berechnet und anschließend aufaddiert. Es wird (wie bei Verzweigungen) für jede Iteration i einzeln die $ServZuordn^i$ ermittelt, die die Zielfunktion K minimiert. Erst dann wird das Gesamtergebnis für die jeweils optimalen $ServZuordn^i$ berechnet. Beim Aufaddieren der Datenvolumina ist nicht die Wahrscheinlichkeit p_i für das Verlassen der Schleife nach Iteration i relevant, sondern die Wahrscheinlichkeit, daß diese Iteration überhaupt ausgeführt wird. Diese Wahrscheinlichkeit beträgt $p_i' = (1-q)^{i-1}$ (die Iteration i wird ausgeführt, wenn die Schleife in den ersten $i-1$ Iterationen nicht verlassen wurde). Diese Wahrscheinlichkeit wird als Gewicht beim Aufaddieren der Kosten verwendet:

$$Vol(k) = \sum_{i=1}^{n-1} p_i' \cdot Vol^i(k) + (1 - \sum_{i=1}^{n-1} p_i') \cdot \; Vol^n(k)$$

5.4 Kombination der Konstrukte

Die Serverzuordnungen aus Abschnitt 5.3 resultieren aus den elementaren komplexen Bearbeiterzuordnungen (vgl. Abbildung 11). Diese Bearbeiterzuordnungen können zu noch komplexeren zusammengesetzt werden, die sich prinzipiell auf beliebig viele Aktivitäten beziehen können. Ein Beispiel hierfür wäre für Abbildung 11a $BearbZuordn_k$ = "$(Zw.1\surd\colon\ Bearb(b); Zw.2\surd\colon\ Bearb(c)) \vee Bearb(d)$". Dadurch können sich Serverzuordnungen ergeben, die aus den soeben beschriebenen zusammengesetzt sind. Aus komplexen Bearbeiterzuordnungen ergeben sich entsprechend komplexe Abhängigkeiten, in diesem Beispiel $\langle k, [OR(EXCL(Zw.1\surd\colon\ [b, B]; Zw.2\surd\colon\ [c, B]); [d, B])]\rangle$.

[17] Es gibt auch Anwendungen, in denen die Schleife nicht zufällig nach irgendeiner Iteration verlassen wird. Ist das Kriterium für das Schleifenende z.B. das Erreichen eines Zählerstandes, so werden stets exakt $\#It$ Iterationen durchlaufen. $ServProb_k(S)$ und $ActorProb_k(D|S)$ ergeben sich dann aus ihrem Wert der $\#It$-ten (und damit letzten) Schleifeniteration.

Komplexe Abhängigkeiten können sich aber auch ergeben, wenn bei der Modellierung nur elementare zusammengesetzte Bearbeiterzuordnungen verwendet werden, nämlich dann, wenn Algorithmus 4 die aus ihnen resultierenden Abhängigkeiten zu komplexeren kombiniert. Im folgenden wird erläutert, wie in diesen Fällen die Serverzuordnung ermittelt und die resultierenden WVen und Kosten berechnet werden können.

Die Verschachtelung der Abhängigkeits-Klauseln orientiert sich an der Verschachtelung der Konstrukte (Verzweigung, Parallelität, Schleifen) von ADEPT. Deshalb können sie aus den elementaren Fällen (vgl. Abbildung 11) zusammengesetzt werden. Dadurch ergibt sich eine hierarchische Struktur, wie in Abbildung 13 für das oben beschriebene Beispiel dargestellt. Auf der untersten Ebene können, wie in Abschnitt 5.3 beschrieben, alle relevanten Serverzuordnungen durchprobiert und jeweils die Kosten analysiert werden. Darauf können die in Abschnitt 5.3 beschriebenen Algorithmen rekursiv angewandt werden. Dies ist möglich, da ihre Eingabeparameter ($ServProb_k(S)$ und $ActorProb_k(D|S)$) Teilmenge ihrer Ausgabeparameter sind. Dadurch ergibt sich für beliebig komplexe Bearbeiterzuordnungen eine Gesamt-Serverzuordnung mit den zugehörigen WVen und Kosten.

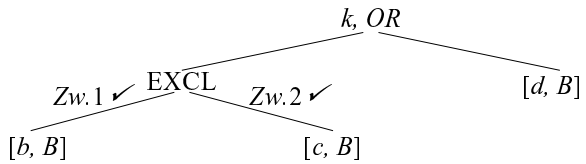


Abbildung 13 Hierarchische Struktur von $\langle k, [OR(EXCL(Zw.1\surd\text{: } [b, B]; Zw.2\surd\text{: } [c, B]); [d, B])]\rangle$.

Bei jeder Rekursion müssen alle relevanten Serverzuordnungen für die darunterliegende Ebene durchprobiert werden. Deshalb ergibt sich eine Komplexität, die exponentiell zur Rekursionstiefe ist. Da die Rekursionstiefe aber der Verschachtelungstiefe in den Abhängigkeits-Klauseln entspricht, ist mit sehr kleinen Rekursionstiefen zu rechnen. Diese Verschachtelungstiefe entspricht der Verschachtelungstiefe bei der Bearbeiterzuordnung, wobei aber auch indirekte Abhängigkeiten berücksichtigt sind. Ein Modellierer wird aber keine sehr komplizierten Bearbeiterzuordnungen verwenden, da diese in der Praxis nicht benötigt werden und sehr schwer zu durchschauen sind.

5.5 Veränderungen durch komplexe Bearbeiter- und Serverzuordnungen

Komplexe Bearbeiter- und Serverzuordnungen haben Einfluß auf in Kapitel 4 beschriebene Algorithmen. So muß die Berechnung des OE-Gewichts angepaßt werden, wenn komplexe Bearbeiterzuordnungen verwendet werden. Außerdem müssen bei der Prüfung, ob Aktivitäten vom selben Server kontrolliert werden, auch komplexe Serverzuordnungen berücksichtigt werden.

5.5.1 Berechnung von OE-Gewichten bei komplexen Bearbeiterzuordnungen

In Abschnitt 4.5.3 wurde das OE-Gewicht $G_k(OE)$ eingeführt, um berücksichtigen zu können, daß bei abhängigen Bearbeiterzuordnungen der Anteil der Bearbeiter aus einer OE von der referenzierten Aktivität abhängt. Für die Berechnung des OE-Gewichts benötigt man für jede OE den Anteil ihrer Bearbeiter und ihren Anteil bei der Ausführung von Aktivität k. Der Anteil der Bearbeiter $AnteilBearb_k(OE)$ berechnet sich als (vgl. Algorithmus 9):

$$AnteilBearb_k(OE) = \sum_{\substack{u \in PotBearb_k \\ \wedge\ OE(u)=OE}} G_k(u) \ / \sum_{u \in PotBearb_k} G_k(u)$$

Bei der Berechnung von $AnteilAusf_k(OE)$ muß unterschieden werden, ob die Bearbeiterzuordnung von Aktivität k eine andere Aktivität x referenziert ($AnteilAusf_k(OE) = AnteilAusf_x(OE)$) oder nicht ($AnteilAusf_k(OE) = AnteilBearb_k(OE)$).

Bei Verwendung komplexer Bearbeiterzuordnungen können mehrere Aktivitäten referenziert werden. Für jeden Teil $BearbZuordn_k^i$ von $BearbZuordn_k$ wird $AnteilBearb_k^i(OE)$ und $AnteilAusf_k^i(OE)$ separat berechnet. Aus diesen Werten wird $AnteilBearb_k(OE)$ und $AnteilAusf_k(OE)$ berechnet, indem sie gewichtet addiert werden. Die Gewichte entsprechen dabei den Wahrscheinlichkeiten p_i, daß $BearbZuordn_k^i$ dafür verantwortlich ist, daß der Bearbeiter u die Aktivität k bearbeiten darf. Wie die Wahrscheinlichkeiten p_i berechnet werden, wurde bereits in den Abschnitten 5.3.1 - 5.3.3 erläutert. Es gilt also:
$\forall OE : AnteilBearb_k(OE) = \sum_i p_i \cdot AnteilBearb_k^i(OE)$
$\forall OE : AnteilAusf_k(OE) = \sum_i p_i \cdot AnteilAusf_k^i(OE)$

Da damit die benötigten Werte für die Gesamtbearbeiterzuordnung verfügbar sind, kann $G_k(OE)$ und $Gewicht_k(u)$ – wie in Algorithmus 9 beschrieben – berechnet werden.

5.5.2 Komplexe Serverzuordnungen bei der Berechnung von Serverklassen

In Abschnitt 5.2 wurde diskutiert, unter welchen Umständen Aktivitäten bei komplexen Bearbeiterzuordnungen denselben Bearbeiter bzw. einen Bearbeiter aus derselben OE haben. Analog hat die Verwendung von komplexen Serverzuordnungen Auswirkungen darauf, wann Aktivitäten von selben Server kontrolliert werden (Funktion $gleicherServerDirekt()$ in Algorithmus 12). Aktivitäten können immer vom selben Server kontrolliert werden, obwohl dies mit den in Abschnitt 4.6.1 aufgeführten Regeln nicht erkannt wird, da diese keine komplexen Serverzuordnungen berücksichtigen. In dem in Abb. 14a dargestellten Beispiel referenzieren die Aktivitäten k und k' Aktivitäten einer Verzweigung. Dabei beschreiben die Teilserverzuordnungen für Zweig i jeweils denselben Server. Deshalb verwenden die beiden Aktivitäten jeweils denselben Server, egal welcher Zweig gewählt wird. Allgemein kann man schließen:
Falls $ServZuordn_k = "Zw.1\surd: ServZuordn_k^1; \ldots Zw.n\surd: ServZuordn_k^n"$
und $ServZuordn_{k'} = "Zw.1\surd: ServZuordn_{k'}^1; \ldots Zw.n\surd: ServZuordn_{k'}^n"$ mit
$\forall i = 1 \ldots n : ServZuordn_k^i$ referenziert denselben Server wie $ServZuordn_{k'}^i$,
dann werden die Aktivitäten k und k' vom selben Server kontrolliert.

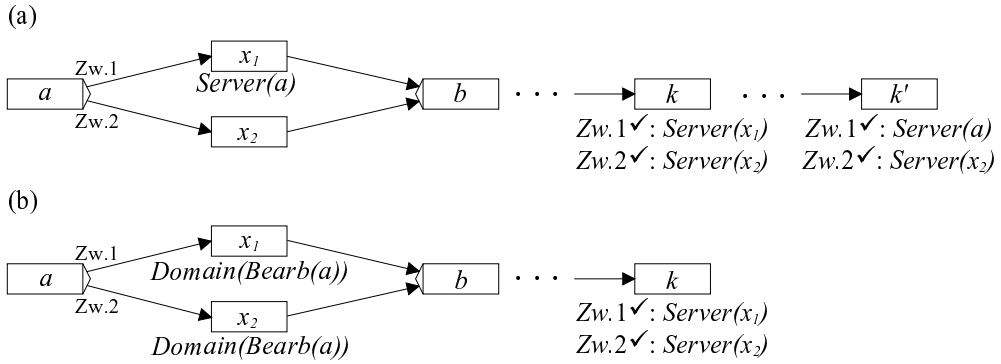


Abbildung 14 Aktivitäten mit gleichem Server bei Verwendung komplexer Serverzuordnungen.

Auch in dem in Abb. 14b dargestellten Beispiel referenziert Aktivität k die Aktivitäten einer Ver-

zweigung. Da alle referenzierten Aktivitäten $x_1 \ldots x_n$ vom selben Server kontrolliert werden und Aktivität k (in allen Fällen) auch diesen Server verwendet, kann geschlossen werden, daß der Server von Aktivität k derselbe ist wie für die Aktivitäten $x_1 \ldots x_n$. Dabei müssen die Teilserverzuordnungen von Aktivität k nicht unbedingt $Server(x_i)$ lauten, es sind auch andere Fälle denkbar (z.B. $ServZuordn_k = "Zw.1\surd\colon Domain(Bearb(a)); Zw.2\surd\colon Server(x_2)"$). Allgemein gilt:
Falls $ServZuordn_k = "Zw.1\surd\colon ServZuordn_k^1; \ldots Zw.n\surd\colon ServZuordn_k^n"$ mit
$\forall i = 1 \ldots n$: aus $ServZuordn_k^i$ folgt $SameServer(k, x_i)$
dann werden die Aktivitäten $k, x_1, \ldots x_n$ alle vom selben Server kontrolliert.

Für die Konstrukte Parallelität und Sequenz wurde in Abschnitt 5.3.2 festgestellt, daß Serverzuordnungen der Art $"Server(x_1) \vee Server(x_2)"$ nicht möglich sind, da der Server einer Aktivität eindeutig sein muß. Deshalb kann dieser Fall auch in dem hier diskutierten Zusammenhang nicht auftreten. Bei Schleifen ist die Iterationsnummer nur innerhalb der Schleife gültig. Deshalb können auf eine Schleifen folgende Aktivitäten keine Serverzuordnungen der Art $"ItNr(Loop1) = 1 : ServZuordn_1; ItNr(Loop1) \geq 2 : ServZuordn_2"$ verwenden. Innerhalb der Schleife analysiert Algorithmus 13 die Teilserverzuordnungen $ServZuordn^i$, da die Iterationen bei der Analyse „durchgespielt“ werden. Dadurch werden die Serverzuordnungen auf die nicht zusammengesetzten Fälle aus Abschnitt 4.3.1 zurückgeführt, wobei zusätzlich auf die Iterationsnummern geachtet werden muß. Es ergeben sich also für die Funktion $gleicherServerDirekt()$ aus Algorithmus 12 durch Parallelität, Sequenz und Schleifen keine neuen Fälle.

6 Effizienz des ADEPT$_{distribution}$-Ansatzes

Um den Nutzen von variablen Serverzuordnungen zu demonstrieren, wird das ADEPT$_{distribution}$-Verteilungsmodell mit Hilfe einer Simulation mit anderen Verteilungsmodellen verglichen (siehe auch [BD99a], für Details und weitere Simulationen siehe [BD99c]). Dazu wurde die Ausführung des folgenden einfachen Klinik-WFs unter verschiedenen Verteilungsmodellen simuliert:
3 Aktivitäten in der Ambulanz
1 Aktivität durch einen Stationsarzt (nimmt Patient auf Station 1-5 auf)
5 Aktivitäten durch einen Arzt dieser Station x
1 Aktivität im Labor
5 Aktivitäten durch einen Arzt der Station x

Die Simulation soll einen Vergleich der Verteilungsmodelle ermöglichen und dient nicht dazu, etwaige Überlastungen zu untersuchen. Deshalb wurde selbst für den zentralen Fall angenommen, daß der WF-Server nicht überlastet ist. Aus den bei der Simulation entstandenen Daten wurde die von den WF-Servern zu bewältigende Gesamtlast, die Last pro WF-Server, die Belastung pro Teilnetz und die Last für die Gateways ermittelt. Abb. 15 zeigt das dabei erzielte Ergebnis. Die Werte wurden so normiert, daß sich für die Last im zentralen Fall jeweils der Wert 100 ergibt.

6.1 Zentraler Server

Die Belastung für den einzigen WF-Server wird als Vergleichswert 100 verwendet. Dasselbe gilt für die durchschnittliche Belastung der Teilnetze und der Gateways. Der zentrale WF-Server befindet sich in einem bestimmten Teilnetz, das somit mit dessen gesamter Kommunikation belastet wird. Deshalb stellt es einen potentiellen Flaschenhals dar. Das zeigt sich auch an der hohen Last 387,1 für dieses

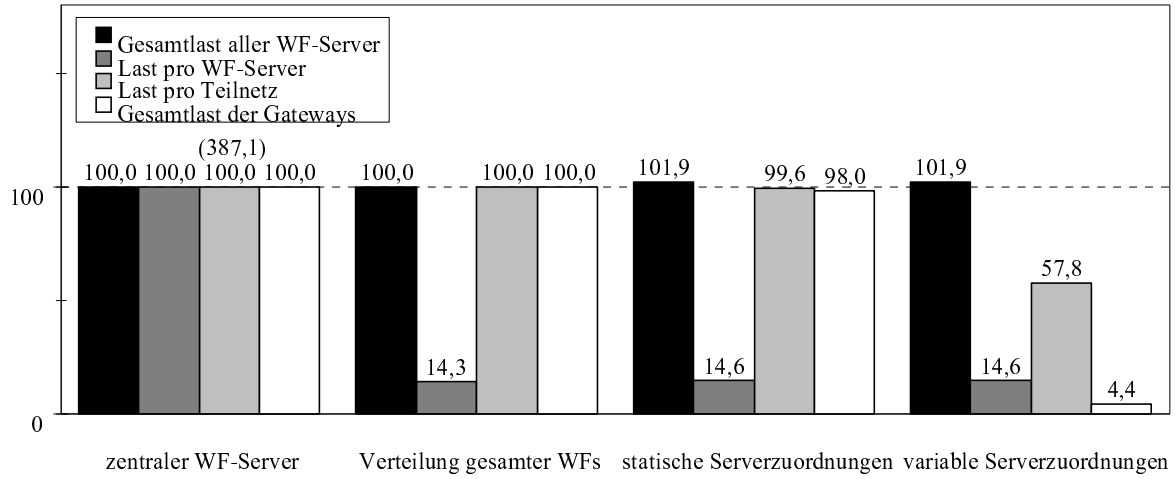


Abbildung 15 Ergebnis der Simulation eines Klinik-WFs unter verschiedenen Verteilungsmodellen.

Teilnetz, wohingegen die anderen Teilnetze mit durchschnittlich 52,1 nur relativ schwach belastet sind.

6.2 Verteilung von gesamten Workflows

Werden den WF-Servern gesamte WF-Typen zugeordnet, so ist die für die Server entstehende Gesamtlast genauso groß wie im zentralen Fall, da ebenfalls keine Migrationen stattfinden. Allerdings verteilt sich diese Last rechnerisch[18] auf die 7 zur Verfügung stehenden WF-Server, so daß sich für die Last pro WF-Server mit 14,3 der beste Wert aller Verteilungsmodelle ergibt. Die Belastung der Teilnetze und der Gateways ist mit der des zentralen Falls identisch, da in beiden Fällen der WF-Server der Station 1 verwendet wurde. Allerdings entfällt der Flaschenhals im Teilnetz des zentralen WF-Servers, da unterschiedliche WF-Typen von verschiedenen WF-Servern kontrolliert werden können.

6.3 Statische Serverzuordnung

Wird der WF partitioniert, so werden auch Migrationen (hier: von der Ambulanz zu einer Station) notwendig. Deshalb entsteht bei diesem Verteilungsmodell eine etwas höhere Last für die WF-Server, als bei den bisher vorgestellten. Diese Last verteilt sich aber auf mehrere WF-Server. Durch die geeignete Wahl des WF-Servers für die Partitionen wird die Belastung der Teilnetze und der Gateways reduziert. Da bei dem betrachteten WF statische Serverzuordnungen wenig geeignet sind, fällt die Verbesserung in dieser Simulation sehr klein aus.

6.4 Variable Serverzuordnung

Durch variable Serverzuordnungen ist es möglich, die von einem Stationsarzt bearbeiteten Aktivitäten durch den WF-Server der entsprechenden Station kontrollieren zu lassen. Dadurch kann die Belastung der Teilnetze und der Gateways drastisch reduziert werden.

[18] Da die Ausführung von Instanzen nur eines einzigen WF-Typs simuliert wurde, entsteht die Last eigentlich nur an einem WF-Server.

Wir haben in ADEPT das Ziel verfolgt, die Belastung des Kommunikationssystems zu reduzieren. Dies ist, wie mit dieser Simulation gezeigt wurde, durch die Verwendung von variablen Serverzuordnungen gelungen. Wegen der dabei notwendigen Migrationen steigt die Gesamtlast der WF-Server gegenüber dem zentralen Fall an, was durch die Verwendung zusätzlicher Server ausgeglichen werden kann.

7 Diskussion

Dieser Abschnitt bietet einen Überblick über Architekturkonzepte für skalierbare WfMSe und stellt einige konkrete Ansätze vor. Für eine detaillierte Diskussion möchten wir auf [BD98, BD99c] verweisen. Das eine Extrem für die Architektur eines WfMSs ist ein *zentraler Server*. Diesem sind alle Aktivitäten zugeordnet. Da er deshalb die gesamte Systemlast bewältigen muß, stellt er einen potentiellen Flaschenhals dar. Einige Forschungsprototypen, die sich nicht primär um Skalierbarkeit kümmern (z.B. Panta Rhei [EG96], WASA [WHKS98]), und die meisten kommerziellen Systeme fallen in diese Kategorie. Das andere Extrem sind *voll verteilte Systeme*, die überhaupt keine Server verwenden. Die WFs migrieren zwischen den Benutzern, die sie bearbeiten. Auch hier sind keine Serverzuordnungen notwendig. Beispiele sind Exotica/FMQM [AMG+95] und INCAS [BMR96].

Zwischen diesen beiden Extremen liegen Ansätze, bei denen *mehrere WF-Server* verwendet werden, aber nicht jede Benutzermaschine gleichzeitig als Server fungiert. Da bei diesen Systemen eine Strategie für die Zuordnung der Server zu den Aktivitäten notwendig ist, werden sie nach diesem Kriterium klassifiziert. In [AKA+94] werden identische Replikate (Cluster) der WF-Engine verwendet. Eine WF-Instanz wird komplett von einem *zufällig ausgewählten Cluster* kontrolliert. Deshalb sind keine Serverzuordnungen notwendig. In [SM96] werden die Systeme CodAlf und BPAFrame beschrieben, die den WF-Server einer Aktivität jeweils auf dem Rechner *der Anwendung* (z.B. beim DBMS) allokieren. Stehen mehrere Anwendungsserver zur Verfügung, so sorgt ein Trader für eine Verteilung der Last. Ein ähnlicher Ansatz wird von METEOR_2 [DKM+97, MSKW96, SK97] verfolgt. Generelle Nachteile dieses Vorgehens sind, daß zwischen dem WF-Server und den Bearbeitern u.U. eine weit entfernte Kommunikation notwendig wird. Außerdem ist nicht für jede Anwendung eine Ort vorgegebenen (z.B. für einen Editor).

Die meisten Ansätze versuchen, wie $\text{ADEPT}_{distribution}$, die *Server nahe bei den Bearbeitern* der jeweiligen Aktivität zu plazieren. Bei MENTOR [WWWK96a, WWWK96b, WWK+97, MWW+98] werden State-/Activitycharts so partitioniert, daß sich die Äquivalenz der verteilten Ausführung zum zentralen Fall formal zeigen läßt. Da alle Bearbeiter einer Aktivität demselben „Processing Entity“ angehören müssen, bietet es sich an, diesen Server auszuwählen. Dieselbe Verteilungsstrategie wird von WIDE [CGP+96, CGS97] verwendet, wobei – anstelle einer Migration – entfernte Objektzugriffe mittels CORBA durchgeführt werden. In MOBILE [HS96, SNS99] werden die verschiedenen Aspekte eines WFs von verschiedenen Servern behandelt. Außerdem wird beim Start eines (Sub-)WFs zur Laufzeit entschieden, von welchem WF-Server er kontrolliert werden soll.

$\text{ADEPT}_{distribution}$ bezieht in den Verteilungsalgorithmus die Kommunikationskosten mit ein und berücksichtigt damit, daß das Kommunikationssystem zum Flaschenhals werden kann. Es ist unseres Wissens der einzige Ansatz, bei dem eine Lastanalyse durchführt wird, um durch eine geeignete Verteilung die Kommunikationskosten zu minimieren. Bei den anderen Systemen gibt es keine variable Serverzuordnung, der Einfluß abhängiger Bearbeiterzuordnungen auf die Verteilung wird nicht berücksichtigt. Es gibt aber Systeme, die bei der Serverzuordnung flexibel sind [SM96], um

eine Lastbalancierung zu erreichen oder den Ausfall von Komponenten zu kompensieren. Dies ist in ADEPT$_{distribution}$ ebenfalls vorgesehen, ist aber nicht Thema dieses Beitrags.

8 Zusammenfassung und Ausblick

WfMSe mit expliziter Prozeß- und Datenmodellierung und der dynamischen Zuordnung von Bearbeitern zu den auszuführenden Tätigkeiten sind eine vielversprechende Technologie für die Realisierung unternehmensweiter, vorgangsorientierter Anwendungssysteme. Die mit diesen Systemen erreichbare Flexibilität schlägt sich jedoch in einem relativ hohen Kommunikationsaufkommen zwischen den Steuerungskomponenten (den WF-Servern) und den Anwendungskomponenten (den WF-Klienten) nieder. In unternehmensweiten WfMS-basierten Anwendungssystemen, mit hunderten von WF-Klienten und tausenden von aktiven WF-Instanzen spielt die daraus resultierende Belastung der WF-Server, aber auch der zugrundeliegenden (Teil-)Netze eine wesentliche Rolle für das Antwortzeitverhalten und damit für die Einsetzbarkeit überhaupt.

Eine Möglichkeit, um die Netzbelastung zu reduzieren, ist, die Kommunikation zwischen WF-Servern und ihren WF-Klienten jeweils möglichst lokal innerhalb eines Teilnetzes zu halten, d.h., teilnetzübergreifende Kommunikation nach Möglichkeit zu vermeiden. Dies ist dadurch zu erreichen, daß eine WF-Instanz nicht mehr komplett von dem initiierenden WF-Server gesteuert wird, sondern daß die Steuerung während der Ausführung ggf. auf andere WF-Server „migriert“. In [BD97] haben wir eine Modellierungskomponente und deren formale Grundlagen (vor allem die Bestimmung von Wahrscheinlichkeitsverteilungen und Kostenformeln) beschrieben, die bei der Modellierung von WFs die Bestimmung geeigneter „WF-Abschnitte“ erlaubt, die dann jeweils von einem anderen WF-Server gesteuert werden. Hierbei wurden die „Abschnittsserver“ zur Modellierungszeit ermittelt und festgelegt. Dies führt dann zu guten Ergebnissen, wenn die möglichen Bearbeiter einer Tätigkeit z.B. allein durch ihre Rolle oder ihre Organisationseinheit beschrieben sind.

In diesem Beitrag haben wir untersucht, wie auch komplizierte Bearbeiterzuordnungen durch ein WfMS adäquat unterstützt werden können. Im Mittelpunkt standen hierbei sog. „abhängige“ Bearbeiterzuordnungen, bei denen die in Frage kommenden Bearbeiter oder die Organisationseinheiten einer Aktivität von vorangegangenen Aktivität abhängen; ein praktisch sehr relevanter Fall. Wir haben gezeigt, wie sich zum einen zur Modellierungszeit geeignete Abschätzungen durchführen lassen, und wie zum anderen die WF-Steuerung so realisiert werden kann, daß zur Laufzeit bei Bedarf dynamisch entschieden werden kann, welcher WF-Server für die Steuerung ausgewählt werden soll. Hierzu haben wir sog. Serverzuordnungsausdrücke sowie entsprechende Modelle für die Wahrscheinlichkeits- und Kostenabschätzungen eingeführt. Die Korrektheit des Kostenmodells und der Verteilungsalgorithmen wurde durch eine Implementierung und durch Messungen verifiziert [End98]. Außerdem wurde durch Simulationen gezeigt, daß sich die Netzlast durch die hier beschriebenen Verfahren tatsächlich reduzieren läßt.

Das Kostenmodell wird primär benötigt, um eine geeignete Verteilung der Aktivitäten auf die WF-Server (Serverzuordnungen) zu ermitteln. Es erlaubt aber auch Optimierungen in eine andere Richtung. Änderungen an den Bearbeiterzuordnungen der Aktivitäten oder an der Verteilung der Bearbeiter auf die Teilnetze haben Einfluß auf die Belastung der Teilnetze. Es gibt Szenarien, bei denen in diesen Punkten Freiheitsgrade existieren (z.B. weil die Netzlast ein so großes Problem ist, daß sogar ein Umbau des Kommunikationsnetzwerks in Erwägung gezogen wird oder weil dieses erst noch aufgebaut werden muß). In solchen Szenarien kann das Kostenmodell verwendet werden, um den durch eine mögliche Veränderung erzielten Gewinn abzuschätzen. Das Kostenmodell ermöglicht

also auch eine Optimierung des Organisationsmodells. Ähnlich wie bei der Optimierung des Ortes für externe Datenquellen (vgl. Abschnitt 4.2.3) wird bei der Optimierung des Organisationsmodells vom WfMS kein Vorschlag berechnet. Stattdessen ermöglicht das Kostenmodell, verschiedene Alternativen zu vergleichen. Für den Fall unabhängiger Bearbeiterzuordnungen haben wir in [BD97] sogar einen Algorithmus vorgestellt, der eine optimale Verteilung der Benutzer auf die Teilnetze (aufgrund der Rollenzuordnungen) ermittelt. Dieser kann auch bei abhängigen Bearbeiterzuordnungen – für die einzelnen OEen getrennt – verwendet werden, indem die abhängige Bearbeiterzuordnung durch $PotBearb_k$ ersetzt wird und nur Bearbeiter der betrachteten OE berücksichtigt werden. Mit dem Algorithmus wird dann für die Benutzer einer bestimmten OE ermittelt, welche Bearbeiter demselben Teilnetz zugeordnet werden sollen, weil sie ähnliche Aktivitäten bearbeiten. Bearbeiter unterschiedlicher OEen gehören i.d.R. sowieso verschiedenen Teilnetzen an, da die unterschiedlichen OEen geographisch voneinander entfernt sind. Außerdem bearbeiten Benutzer unterschiedlicher OEen bei Verwendung abhängiger Bearbeiterzuordnungen verschiedene WF-Instanzen. Da sie deshalb nicht dieselben Aktivitäteninstanzen bearbeiten, sollten sie auch nicht demselben Teilnetz angehören.

Um ein komplettes Bild zu erhalten, müssen auch transaktionale Aspekte und ihr Einfluß auf die Kommunikationslast berücksichtigt werden. Dies war hier aus Platzmangel nicht möglich. Aufgrund unserer bisher durchgeführten Analysen sind wir aber sicher, daß unser Ansatz darunter nicht mehr leidet (wenn überhaupt) als die meisten anderen verteilten Ansätze. Um tiefere Einblicke in dieses Thema zu gewinnen, haben wir begonnen, die in dieser Arbeit beschriebenen Konzepte in den ADEPT-Prototypen zu integrieren.

Literatur

[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.

[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: *Proceedings of the IFIP Working Conference on Information Systems for Decentralized Organisations,* Trondheim, August 1995.

[BD97] T. Bauer und P. Dadam: *A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration.* In: *Proc. Second IFCIS Conference 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: *Proceedings Workshop Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI (Informatik '99),* S. 25–32, Paderborn, Oktober 1999.

[BD99b] T. Bauer und P. Dadam: *Efficient Distributed Workflow Management Based on Variable Server Assignments.* Ulmer Informatik-Berichte 99-09, Universität Ulm, Fakultät für Informatik, November 1999. Erscheint in: Proc. 12th Conference on Advanced Information Systems Engineering, Stockholm, Juni 2000.

[BD99c] T. Bauer und P. Dadam: *Verteilungsmodelle für Workflow-Management-Systeme – Klassifikation und Simulation.* Informatik Forschung und Entwicklung, 14(4):203–217, Dezember 1999.

[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.

[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: *7th International Workshop on Research Issues in Data Engineering,* Birmingham, April 1997.

[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.

[EG96] J. Eder und H. Groiss: *Ein Workflow-Managementsystem auf der Basis aktiver Datenbanken.* In: J. Becker, G. Vossen (Herausgeber): *Geschäftsprozeßmodellierung und Workflow-Management.* International Thomson Publishing, 1996.

[End98] H. Enderlin: *Realisierung einer verteilten Workflow-Ausführungskomponente auf Basis von IBM FlowMark.* Diplomarbeit, Universität Ulm, Fakultät für Informatik, 1998.

[HS96] P. Heinl und H. Schuster: *Towards a Highly Scaleable Architecture for Workflow Management Systems.* In: *Proceedings of the 7th International Workshop on Database and Expert Systems Applications, DEXA'96,* S. 439–444, Zurich, September 1996.

[KAGM96] M. Kamath, G. Alonso, R. Günthör und C. Mohan: *Providing High Availability in Very Large Workflow Management Systems.* In: *Proceedings of the 5th International Conference on Extending Database Technology,* S. 427–442, Avignon, März 1996.

[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.

[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.

[SK97] A. Sheth und K. J. Kochut: *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems.* In: *Proceedings of the 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.

[SNS99] H. Schuster, J. Neeb und R. Schamburger: *A Configuration Management Approach for Large Workflow Management Systems.* In: *Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration*, San Francisco, Februar 1999.

[Tan92] A.S. Tanenbaum: *Computer-Netzwerke*. Wolfram's Fachverlag, 1992.

[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.

[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: *Proceedings of the 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: *Proceedings of the 12th IEEE International Conference on Data Engineering*, S. 556–565, New Orleans, LA, März 1996.

Die meisten unserer Veröffentlichungen sind auf folgender Web-Seite verfügbar:
http://www.informatik.uni-ulm.de/dbis/papers

A Berechnung der optimalen Abbildungsfunktion f

In Abschnitt 4.3.1 wurden die möglichen Serverzuordnungen beschrieben. In einer Serverzuordnung ist es möglich, den Server einer vorangegangenen Aktivität ($ServZuordn_k$ = "$Server(x)$") oder den Domain des Bearbeiters einer solchen Aktivität ($ServZuordn_k$ = "$Domain(Bearb(x))$") zu referenzieren. Des weiteren kann auf eine solche Serverzuordnung auch noch eine Funktion f angewandt werden: $ServZuordn_k$ = "$f(Server(x))$" bzw. $ServZuordn_k$ = "$f(Domain(Bearb(x)))$". Eine optimale Funktion f kann automatisch ermittelt werden, muß also nicht vom Modellierer vorgegeben werden. Wie dies funktioniert wird in diesem Abschnitt beschrieben.

Soll für Aktivität k soll geprüft werden, ob die Serverzuordnung $ServZuordn_k$ = "$f(Server(x))$" zu einem guten Ergebnis führt, so wird mit Algorithmus 19 eine optimale Funktion f berechnet.

Dazu werden alle Bearbeiter u_1 von Aktivität x durchlaufen, wobei jeweils die Serververteilung $DepServProb_x(S|u_1)$ ermittelt wird. Dann werden alle Bearbeiter u_2 von Aktivität k durchlaufen, die der $BearbZuordn_k$ zufolge möglich sind, wenn der Benutzer u_1 die Aktivität x bearbeitet hat. Der Domain $Domain_2$ jedes Bearbeiters u_2 wird ermittelt, da er (für diesen Bearbeiter) den optimalen Ort für den Server von Aktivität k darstellt. Für jedes Paar von Bearbeitern u_1 und u_2 wird die Abbildung vom Server S von Aktivität x zum optimalen Server von Aktivität k ($Domain_2$) in *Häufigkeit* vermerkt. Der Eintrag wird mit der Wahrscheinlichkeit $\frac{Gewicht_x(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_k(u_2)}{Gesamtgewicht_2(u_1)}$ für das Auftreten dieses Paares (siehe Abschnitt 4.6.2) gewichtet. Da für Aktivität x mehrere Server S möglich sind, wird der Eintrag zusätzlich mit $DepServProb_x(S|u_1)$ gewichtet. Wegen $\sum_S DepServProb_x(S|u_1) = 1$, 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 von Aktivität x der optimale Zielserver S_2 für die Abbildungsfunktion f ausgewählt. Dies ist der mit dem größten Eintrag *Häufigkeit*(S_1, S_2) für diesen 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.

Algorithmus 19 (Ermitteln einer optimalen Abbildungsfunktion f)

$\forall S_1, S_2$: *Häufigkeit*$(S_1, S_2) = 0$;
$Gesamtgewicht_1 = \sum_{u \in PotBearb_x} Gewicht_x(u)$;
for each $u_1 \in PotBearb_x$ **do**
  $DepServProb_x(S|u_1) = DepServ(x, "Bearb = u_1")$;
  $Bearb_{u_1} = \{u \mid ActorPossible(x, u_1, k, u) = True\}$;
  $Gesamtgewicht_2(u_1) = \sum_{u \in Bearb_{u_1}} Gewicht_k(u)$;
  for each $u_2 \in Bearb_{u_1}$ **do**
    $Domain_2 = Domain(u_2)$;
    $\forall S$: *Häufigkeit*$(S, Domain_2) =$ *Häufigkeit*$(S, Domain_2)$
      $+ \frac{Gewicht_x(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_k(u_2)}{Gesamtgewicht_2(u_1)} \cdot DepServProb_x(S|u_1)$ $(*)$
for each S_1 **do**
  $f(S_1) := S_2$ mit $\forall S_2'$: *Häufigkeit*$(S_1, S_2') \leq$ *Häufigkeit*(S_1, S_2);

Um für Aktivität k die Eignung der Serverzuordnung $ServZuordn_k = "f(Domain(Bearb(x)))"$ zu prüfen, kann ein ähnlicher Algorithmus verwendet werden. Allerdings ist dann nicht die Server-WV von Aktivität x relevant, sondern deren Bearbeiter-WV. Deshalb wird für jeden Bearbeiter u_1 von Aktivität x nicht $DepServProb_x(S|u_1)$, sondern der $Domain_1 = Domain(u_1)$ ermittelt, weil dieser in $ServZuordn_k$ referenziert wird und damit den Server von Aktivität k determiniert. Die mit $(*)$ markierte Zeile in Algorithmus 19 muß damit wie folgt lauten:

$$\begin{aligned} &\textit{Häufigkeit}(Domain_1, Domain_2) = \textit{Häufigkeit}((Domain_1, Domain_2) \\ &\quad + \frac{Gewicht_x(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_k(u_2)}{Gesamtgewicht_2(u_1)} \end{aligned}$$

B Beweise

Im folgenden werden einige der in dieser Arbeit gemachten Aussagen bewiesen.

B.1 Korrektheit von Algorithmus 9

Um die Korrektheit von Algorithmus 9 zeigen zu können, benötigen wir Lemma 1: Das Gesamtgewicht aller Bearbeiter einer Aktivität k wird durch die Multiplikation mit $G_k(OE)$ in Algorithmus 8 nicht verändert. Es kommt nur zu Verschiebungen zwischen den OEen.

Lemma 1: $\sum\limits_{u \in PotBearb_k} Gewicht_k(u) = \sum\limits_{u \in PotBearb_k} G_k(u)$

Beweis:

$$\sum_{u \in PotBearb_k} Gewicht_k(u)$$

$$= \sum_{OE} \sum_{\substack{u \in PotBearb_k \\ \wedge\, OE(u)=OE}} Gewicht_k(u)$$

$$= \sum_{OE} \sum_{\substack{u \in PotBearb_k \\ \wedge\, OE(u)=OE}} G_k(OE) \cdot G_k(u)$$

$$= \sum_{OE} G_k(OE) \cdot \sum_{\substack{u \in PotBearb_k \\ \wedge\, OE(u)=OE}} G_k(u)$$

$$= \sum_{OE} \frac{AnteilAusf_k(OE)}{AnteilBearb_k(OE)} \cdot \sum_{\substack{u \in PotBearb_k \\ \wedge\, OE(u)=OE}} G_k(u)$$

$$= \sum_{OE} \frac{AnteilBearb_x(OE)}{AnteilBearb_k(OE)} \cdot \sum_{\substack{u \in PotBearb_k \\ \wedge\, OE(u)=OE}} G_k(u)$$

mit $BearbZuordn_k$ ist (indirekt) von Aktivität x abhängig:

$\langle k, [x, T] \rangle$ mit $T \in \{B, OE\} \in Dependencies$

und $BearbZuordn_x$ hängt von keiner weiteren Aktivität ab:

$\langle x, [NULL] \rangle \in Dependencies$

d.h.: $\forall OE : AnteilAusf_k(OE) = AnteilAusf_x(OE) = AnteilBearb_x(OE)$

$$= \sum_{OE} \frac{AnteilBearb_x(OE)}{\sum\limits_{\substack{u \in PotBearb_k \\ \wedge\, OE(u)=OE}} G_k(u) \quad / \quad \sum\limits_{u \in PotBearb_k} G_k(u)} \cdot \sum_{\substack{u \in PotBearb_k \\ \wedge\, OE(u)=OE}} G_k(u)$$

$$= \sum_{u \in PotBearb_k} G_k(u) \cdot \sum_{OE} AnteilBearb_x(OE)$$

$$= \sum_{u \in PotBearb_k} G_k(u) \cdot \sum_{OE} \Big(\sum_{\substack{u \in PotBearb_x \\ \wedge\, OE(u)=OE}} G_x(u) \quad / \quad \sum_{u \in PotBearb_x} G_x(u) \Big)$$

$$= \sum_{u \in PotBearb_k} G_k(u) \cdot \frac{1}{\sum\limits_{u \in PotBearb_x} G_x(u)} \cdot \sum_{OE} \sum_{\substack{u \in PotBearb_x \\ \wedge\, OE(u)=OE}} G_x(u)$$

$$= \sum_{u \in PotBearb_k} G_k(u) \cdot \frac{1}{\sum\limits_{u \in PotBearb_x} G_x(u)} \cdot \sum_{u \in PotBearb_x} G_x(u)$$

$$= \sum_{u \in PotBearb_k} G_k(u)$$

Mit Hilfe von Lemma 1 ist es nun möglich, Lemma 2 zu beweisen. Lemma 2 besagt, daß der Anteil der Bearbeiter der Organisationseinheit OE an Aktivität k (jeweils mit $Gewicht_k(u)$ gewichtet)

dem Anteil dieser OE an der Ausführung dieser Aktivität ($AnteilAusf_k(OE)$) entspricht. Da alle Bearbeiter einer OE mit demselben Faktor $G_k(OE)$ gewichtet werden, kann es innerhalb dieser Bearbeiter zu keinen „Verzerrungen“ der Gewichte kommen. Damit ist gezeigt, daß die Berechnung von $Gewicht_k(u)$ in Algorithmus 9 auf korrekte Art und Weise erfolgt.

Lemma 2: Wenn Aktivität x die OE der Bearbeiter von Aktivität k bestimmt, gilt $\forall OE$:

$$\sum_{\substack{u \in PotBearb_k \\ \wedge\ OE(u)=OE}} Gewicht_k(u) \quad / \sum_{u \in PotBearb_k} Gewicht_k(u) \ = \ AnteilAusf_k(OE)$$

Beweis: $\forall OE$ gilt:

$$\sum_{\substack{u \in PotBearb_k \\ \wedge\ OE(u)=OE}} Gewicht_k(u) \quad / \sum_{u \in PotBearb_k} Gewicht_k(u)$$

$$\overset{Lemma\ 1}{=} \sum_{\substack{u \in PotBearb_k \\ \wedge\ OE(u)=OE}} G_k(OE) \cdot G_k(u) \quad / \sum_{u \in PotBearb_k} G_k(u)$$

$$= \sum_{\substack{u \in PotBearb_k \\ \wedge\ OE(u)=OE}} \frac{AnteilAusf_k(OE)}{AnteilBearb_k(OE)} \cdot G_k(u) \quad / \sum_{u \in PotBearb_k} G_k(u)$$

$$= \frac{AnteilAusf_k(OE)}{AnteilBearb_k(OE)} \cdot \sum_{\substack{u \in PotBearb_k \\ \wedge\ OE(u)=OE}} G_k(u) \quad / \sum_{u \in PotBearb_k} G_k(u)$$

$$= \frac{AnteilAusf_k(OE)}{\sum\limits_{\substack{u \in PotBearb_k \\ \wedge\ OE(u)=OE}} G_k(u) \ / \sum\limits_{u \in PotBearb_k} G_k(u)} \cdot \sum_{\substack{u \in PotBearb_k \\ \wedge\ OE(u)=OE}} G_k(u) \quad / \sum_{u \in PotBearb_k} G_k(u)$$

$$= \ AnteilAusf_k(OE)$$

B.2 Korrektheit von Algorithmus 14

Lemma 3 zeigt, daß Algorithmus 14 in den for-Schleifen alle relevanten Fälle berücksichtigt. Der Grund dafür ist, daß die Summe der Wahrscheinlichkeiten (für alle betrachteten Kombinationen der Bearbeitern von Aktivität x und k) 1 ergibt, was der Wahrscheinlichkeit entspricht, daß die Aktivitäten x und k von genau einem Benutzerpaar bearbeitet werden.

Lemma 3: $\sum\limits_{u_1 \in PotBearb_x} \ \sum\limits_{u_2 \in Bearb_{u_1}} \left(\frac{Gewicht_x(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_k(u_2)}{Gesamtgewicht_2(u_1)} \right) = 1$

Beweis:

$$\sum_{u_1 \in PotBearb_x} \ \sum_{u_2 \in Bearb_{u_1}} \left(\frac{Gewicht_x(u_1)}{Gesamtgewicht_1} \cdot \frac{Gewicht_k(u_2)}{Gesamtgewicht_2(u_1)} \right)$$

$$= \sum_{u_1 \in PotBearb_x} \left(\frac{Gewicht_x(u_1)}{Gesamtgewicht_1} \cdot \sum_{u_2 \in Bearb_{u_1}} \frac{Gewicht_k(u_2)}{Gesamtgewicht_2(u_1)} \right)$$

$$= \frac{1}{Gesamtgewicht_1} \cdot \underbrace{\sum_{u_1 \in PotBearb_x} \left(Gewicht_x(u_1) \right.}_{Gesamtgewicht_1} \cdot \frac{1}{Gesamtgewicht_2(u_1)} \cdot \underbrace{\left. \sum_{u_2 \in Bearb_{u_1}} Gewicht_k(u_2) \right)}_{Gesamtgewicht_2(u_1)}$$

$$= 1$$

Außerdem gilt analog $\sum_{u_2 \in Bearb_{u_1}} \frac{Gewicht_k(u_2)}{Gesamtgewicht_2(u_1)} = 1$, so daß die Gewichtung des Bearbeiters u_1 von Aktivität x nicht durch die Multiplikationen mit $\frac{Gewicht_k(u_2)}{Gesamtgewicht_2(u_1)}$ beeinträchtigt wird.

Das Gesamtgewicht eines Bearbeiterpaares u_1 und u_2 wird durch die Multiplikation mit dem Faktor $DepServProb_x(S_1|u_1) \cdot DepServProb_x(S_2|u_2)$ nicht beeinflußt, weil wegen $\sum_{S_1} DepServProb_x(S_1|u_1) = \sum_{S_2} DepServProb_x(S_2|u_2) = 1$ gilt:
$\sum_{S_1} \sum_{S_2} DepServProb_x(S_1|u_1) \cdot DepServProb_x(S_2|u_2) = 1$.

B.3 Anteil der nicht analysierten Iterationen einer Schleife

Mit Hilfe von Lemma 4 ist es möglich, abzuschätzen, wie groß der Anteil der nicht analysierten Iterationen einer Schleife (und damit der Fehler) maximal ist, wenn $n = a \cdot \#It$ Iterationen analysiert werden. Die Wahrscheinlichkeit, daß Iteration i ausgeführt wird, beträgt $p_i = (1-q)^{i-1} \cdot q$ mit $q = 1/\#It$ (vgl. Abschnitt 5.3.3). Der Anteil der nicht mehr analysierten Iterationen ist damit die Summe der p_i für $i > n$. Es kann $\#It \geq 1$ angenommen werden, da in ADEPT die Schleifenbedingung am Ende der Schleife geprüft wird (Repeat-Until), weshalb immer mindestens ein Schleifendurchlauf ausgeführt wird.

Lemma 4: $\sum\limits_{i=a\cdot x+1}^{\infty} p_i \leq e^{-a}$ mit $p_i = \left(1 - \frac{1}{x}\right)^{i-1} \cdot \frac{1}{x}$ für $x \geq 1$ und $a > 0$

Beweis:

$$\sum_{i=a\cdot x+1}^{\infty} p_i = 1 - \sum_{i=1}^{a\cdot x} p_i = 1 - \sum_{i=1}^{a\cdot x} \left(1-\frac{1}{x}\right)^{i-1} \cdot \frac{1}{x} = 1 - \frac{1}{x} \cdot \sum_{i=1}^{a\cdot x} \left(1-\frac{1}{x}\right)^{i-1} = 1 - \frac{1}{x} \cdot \sum_{i=0}^{a\cdot x-1} \left(1-\frac{1}{x}\right)^{i}$$

$$\overset{19}{=} 1 - \frac{1}{x} \cdot \frac{1-\left(1-1/x\right)^{a\cdot x}}{1-(1-1/x)} = 1 - \left(1 - \left(1-\frac{1}{x}\right)^{a\cdot x}\right) = \left(1-\frac{1}{x}\right)^{a\cdot x} \leq e^{-a},$$

weil 1) $\lim\limits_{x\to\infty} \left(1-\frac{1}{x}\right)^{a\cdot x} = e^{-a}$ und 2) $\left(1-\frac{1}{x}\right)^{a\cdot x}$ monoton wachsend für $x \geq 1$:

1) $\lim\limits_{x\to\infty} \left(1-\frac{1}{x}\right)^{a\cdot x} = \lim\limits_{x\to\infty} e^{a\cdot x \cdot \ln(1-1/x)} = e^{-a}$, weil

$$\lim_{x\to\infty} x \cdot \ln(1-1/x) = \lim_{x\to\infty} \frac{\ln(1-1/x)}{1/x} \overset{20}{=} \lim_{x\to\infty} \frac{\left(\ln(1-1/x)\right)'}{\left(1/x\right)'} = \lim_{x\to\infty} \frac{\frac{1}{1-1/x} \cdot \left(0 - \frac{-1}{x^2}\right)}{\frac{-1}{x^2}}$$

$$= \lim_{x\to\infty} \frac{-1}{1-1/x} = -1$$

2) $\left(\left(1-\frac{1}{x}\right)^{a\cdot x}\right)' = \left(e^{a\cdot x\cdot\ln(1-1/x)}\right)' = e^{a\cdot x\cdot\ln(1-1/x)} \cdot \left(a \cdot x \cdot \ln(1-\frac{1}{x})\right)'$

$$= \left(1-\frac{1}{x}\right)^{a\cdot x} \cdot \left(a \cdot \ln(1-\frac{1}{x}) + a \cdot x \cdot \frac{1}{1-1/x} \cdot (0 - \frac{-1}{x^2})\right)$$

$$= \underbrace{a}_{>0} \cdot \underbrace{(1-\frac{1}{x})^{a\cdot x}}_{\geq 0 \text{ für } x\geq 1} \cdot \underbrace{\left(\ln(1-\frac{1}{x}) + \frac{1}{x-1}\right)}_{=:f(x)} \geq 0, \text{ weil } f(x) \geq 0 \text{ für } x \geq 1$$

$f(x) \geq 0$, weil a) $\lim\limits_{x\to\infty} f(x) = 0$ und b) $f(x)$ monoton fallend für $x \geq 1$:

[19] geometrische Reihe: $\sum\limits_{k=0}^{n} q^k = \frac{1-q^{n+1}}{1-q}$

[20] Satz von l' Hospital: $\lim\limits_{x\to\infty} \frac{f(x)}{g(x)} = \lim\limits_{x\to\infty} \frac{f'(x)}{g'(x)}$, falls $\lim\limits_{x\to\infty} f(x) = \lim\limits_{x\to\infty} g(x) = 0$

a) $\lim\limits_{x\to\infty} \underbrace{\ln(\underbrace{1-\frac{1}{x}}_{\to 1})}_{\to 0} + \underbrace{\frac{1}{x-1}}_{\to 0} = 0$

b) $\left(\ln(1-\frac{1}{x}) + \frac{1}{x-1}\right)' = \frac{1}{1-1/x} \cdot \left(0 - \frac{-1}{x^2}\right) + \left(-1 \cdot (x-1)^{-2}\right) = \frac{1}{x \cdot (x-1)} - \frac{1}{(x-1)^2}$

$= \frac{(x-1)-x}{x \cdot (x-1)^2} = \frac{-1}{\underbrace{x}_{\geq 0} \cdot \underbrace{(x-1)^2}_{\geq 0}} \leq 0$