Informatik Forsch. Entw. (2001) 16: 76–92

Informatik
Forschung und Entwicklung

Effiziente Übertragung von Prozessinstanzdaten in verteilten Workflow-Management-Systemen

Thomas Bauer, Manfred Reichert, Peter Dadam

Universität Ulm, Abteilung Datenbanken und Informationssysteme, D-89069 Ulm
(e-mail: {bauer, reichert, dadam}@informatik.uni-ulm.de, http://www.informatik.uni-ulm.de/dbis)

Eingegangen am 22. Juni 2000 / Angenommen am 25. Januar 2001

Zusammenfassung. Zur Unterstützung von unternehmensweiten und -übergreifenden Geschäftsprozessen muss ein Workflow-Management-System (WfMS) eine große Anzahl von Workflow-Instanzen steuern können. Daraus resultiert eine hohe Last für die Workflow-Server und das zugrunde liegende Kommunikationssystem. Ein in der Workflow-Literatur viel diskutierter Ansatz zur Bewältigung der Last ist es, die Workflow-Instanzen verteilt durch mehrere Workflow-Server zu kontrollieren. Beim Wechsel der Kontrolle zwischen zwei Workflow-Servern werden dann Migrationen notwendig, bei denen Daten der jeweiligen Workflow-Instanz vom Quell- zum Zielserver übertragen werden müssen, um dort mit der Steuerung fortfahren zu können. Deshalb belasten Migrationen das Kommunikationssystem zusätzlich. In diesem Beitrag werden Verfahren entwickelt, mit denen die bei Migrationen entstehende Kommunikationslast reduziert werden kann, so dass die Skalierbarkeit des WfMS signifikant verbessert wird. Falls Geschäftsbereiche aus Kostengründen nur über langsame Kommunikationsverbindungen angebunden sind, wird dadurch der Einsatz eines WfMS überhaupt erst ermöglicht.

Schlüsselwörter: Workflow-Management, verteilte Ausführung, Skalierbarkeit, Migration, Kommunikationskosten

Abstract. For the support of enterprise-wide and cross-enterprise business processes, a workflow management system (WfMS) must be able to control a large number of workflow instances. This, in turn, results in a very high load for the workflow servers as well as for the underlying communication network. To cope with this load, in the workflow literature, several approaches suggest to control workflow instances piecewise by multiple, distributed workflow servers. When transferring the control from one server of a workflow instance to another, a migration becomes necessary. During such a migration, data of the corresponding workflow instance have to be transmitted from the source to the target server. Afterwards, the target server may continue with the control. Although very advantageous for many applications, migrations themselves burden the communication network. In this paper, methods are presented, which allow to reduce the communication load caused by migrations and which, therefore, contribute to improve the scalability of the WfMS significantly. If business units are only connected by a communication network with a low bandwidth (e.g., due to cost reasons), such an approach is a prerequisite for the broad usage of a WfMS.

Keywords: workflow management, distributed execution, scalability, migration, communication costs

CR Subject Classification: H.4.1, H.1.0, C.2.4

1 Einleitung

WfMS ermöglichen die rechnerunterstützte Ausführung von Arbeitsabläufen (engl. Workflows) in einer verteilten Systemumgebung. Ein entscheidender Vorteil des Einsatzes von WfMS ist, dass sie helfen, große Anwendungssysteme überschaubarer zu gestalten. Dazu wird der applikationsspezifische Code der Anwendungen, die zur Ausführung einzelner Prozessschritte verwendet werden, von der Definition und Steuerung der Ablauflogik des Geschäftsprozesses getrennt. Anstelle eines großen monolithischen Programmpakets erhält man einzelne Aktivitäten, welche die Anwendungsprogramme repräsentieren. Ihre Ablauflogik wird in einer separaten Kontrollflussdefinition festgelegt, welche die Ausführungsreihenfolgen und -bedingungen der einzelnen Aktivitäten vorgibt. Zur Ausführungszeit sorgt das WfMS dafür, dass diese Aktivitäten entsprechend der festgelegten Ablauflogik zur Bearbeitung kommen.

Bei unternehmensweiten und -übergreifenden prozessorientierten Anwendungssystemen entsteht wegen der großen Benutzerzahl eine hohe Last für das WfMS.[1] Um diese bewältigen zu können, erfolgt bei vielen Ansätzen (z.B. [AKA+94, CGP+96, DKM+97, Jab97, MWW+98, SM96]) eine Aufteilung auf mehrere Workflow-Server. Dabei kann eine Workflow-Instanz (abschnittsweise) von verschiedenen *Workflow-Servern* gesteuert werden, d.h. die Kontrolle über

[1] In [KAGM96, SK97] werden Anwendungen beschrieben, bei denen die Zahl der Benutzer des WfMS bis auf einige zehntausend anwachsen kann oder mehrere zehntausend Workflow-Instanzen gleichzeitig im System sein können. Ähnliches gilt auch für E-Business-Anwendungen.

sie wechselt zur Ausführungszeit zwischen den Servern. Ein solcher Wechsel erfordert eine sog. *Migration*, bei der die bei der Ausführung der Workflow-Instanz erzeugten Daten[2] an den Zielserver übertragen werden, bevor dort mit der Kontrolle fortgefahren wird. Dadurch kann bei verteilter Workflow-Steuerung dasselbe logische Verhalten wie im zentralen Fall erreicht werden, obwohl mehrere Workflow-Server nacheinander oder gleichzeitig an der Ausführung einer Workflow-Instanz beteiligt sind. Bei den meisten in der Literatur diskutierten Ansätze werden bei einer solchen Migration alle Laufzeitdaten der Workflow-Instanz zum Zielserver transferiert. Dies führt im Allgemeinen aber zu einer redundanten Übertragung von Daten. So kann der Zielserver einer Migration bereits früher einmal an der Ausführung beteiligt gewesen sein, so dass ihm gewisse Instanzdaten schon bekannt sind. In diesem Fall ist es unnötig, bereits vorhandene Daten nochmals zu übertragen. In diesem Beitrag werden zur Ausführungszeit der Workflow-Instanzen zur Anwendung kommende Verfahren vorgestellt, die solche und ähnliche Fälle ausschließen, wodurch das Kommunikationsaufkommen bei Migrationen reduziert wird. Die Ausnutzung des existierenden Optimierungspotenzials ist insbesondere für unternehmensweite und -übergreifende Anwendungen unverzichtbar, da bei den entsprechenden Systemen die Kommunikation häufig (zumindest teilweise) über ein WAN (Wide Area Network) oder eine sonstige langsame Verbindung (Internet, ISDN) abgewickelt wird. Auch bei Workflow-Anwendungen mit mobilen Benutzern sind solche Optimierungen essentiell, da die Kommunikation hier häufig über Modems oder Mobiltelefone erfolgt. Dennoch gibt es unseres Wissens bisher keine Arbeiten, die systematisch untersuchen, wie Migrationen effizient realisiert werden können.

Die Verbesserung der Skalierbarkeit von WfMS ist aber nur eine von vielen Anforderungen, die sich im unternehmensweiten Einsatz stellen. Um WfMS für ein breites Spektrum von Anwendungen einsetzbar zu machen, müssen weitere *fortschrittliche Workflow-Konzepte* unterstützt werden. Zu diesen zählen unter anderen:

- die dynamische Abänderbarkeit laufender Workflows (z.B. das Einfügen von Aktivitäten) [KDB98, RD98]
- die Spezifikation und Überwachung von Zeitbedingungen (z.B. zeitliche Minimal- und Maximalabstände zwischen Aktivitäten garantieren) [MO99, Gri97]
- das (partielle) Zurücksetzen von Workflows [Ley97, RD98]
- die Verwendung komplexer (abhängiger) Bearbeiterzuordnungen [BF99, Kub98]

Um solche fortschrittlichen Konzepte realisieren zu können, darf die verteilte Workflow-Steuerung nicht isoliert betrachtet werden. Beispielsweise müssen zur Unterstützung dieser Konzepte bei Migrationen gewisse Informationen über beendete Aktivitäten übertragen werden. Bei der Entwicklung der in dieser Arbeit vorgestellten Verfahren wird deshalb darauf geachtet, dass diese mit den gestellten Anforderungen vereinbar sind.

[2] Wir nehmen im Folgenden an, dass das Workflow-Schema vollständig auf all denjenigen Servern repliziert wird, die potenziell an der Ausführung einer entsprechenden Instanz beteiligt sein können.

Im nachfolgenden Abschnitt werden grundlegende Aspekte der zentralen und verteilten Workflow-Steuerung erörtert. Außerdem werden die Problemstellungen untersucht, die sich im Zusammenhang mit der effizienten Realisierung von Migrationen ergeben. In Abschnitt 3 werden mögliche Vorgehensweisen zur Migration von Workflow-Kontrolldaten (z.B. Information zum Status einer Workflow-Instanz) vorgestellt und ein geeignetes Verfahren näher untersucht. Dabei mögliche weitergehende Optimierungen werden im Abschnitt 4 diskutiert. Die Migration der von Workflow-Aktivitäten gelesenen bzw. geschriebenen Parameterdaten wird in Abschnitt 5 betrachtet. In Abschnitt 6 wird die Effektivität der vorgestellten Verfahren durch ein Zahlenbeispiel belegt. Abschnitt 7 diskutiert verwandte Ansätze. Der Beitrag schließt mit einer Zusammenfassung.

2 Grundlagen und Problemstellung

In diesem Abschnitt fassen wir kurz einige für das weitere Verständnis des Beitrags relevante Grundlagen zusammen. So werden das im Folgenden verwendete Workflow-Metamodell und die verteilte Workflow-Ausführung vorgestellt. Danach untersuchen wir detailliert die im Zusammenhang mit der effizienten Verwirklichung von Migrationen zu lösenden Problemstellungen.

2.1 Das Workflow-Metamodell

Um einen Arbeitsprozess zu spezifizieren, wird in einem WfMS üblicherweise ein *Workflow-Schema* modelliert (oberer Teil von Abb. 1). Ein solches Schema legt die Ausführungsreihenfolge und -bedingungen (*Kontrollfluss*) der einzelnen Prozessschritte (*Aktivitäten*) fest. Bei vielen WfMS kann darüber hinaus auch der *Datenfluss* zwischen den Aktivitäten definiert werden. Aus Abb. 1 zum Beispiel ist ersichtlich, welche Aktivitäten die Prozessvariable d_1 lesen bzw. schreiben. Für einzelne Aktivitäten wird festgelegt, ob sie automatisch gestartet oder manuell bearbeitet werden sollen. Für manuelle Aktivitäten muss zur Modellierungszeit zusätzlich eine *Bearbeiterzuordnung* angegeben werden, welche die zur Bearbeitung der Aktivität berechtigten Benutzer festlegt. Eine solche Bearbeiterzuordnung spezifiziert üblicherweise die *Rolle*, die potenzielle Bearbeiter einnehmen, oder die *Organisationseinheit*, der sie angehören sollen. Es sind aber auch komplexere Ausdrücke möglich. Ausgehend von einem solchen Workflow-Schema können zur Ausführungszeit *Workflow-Instanzen* erzeugt werden, die dann über ihre komplette Lebenszeit vom WfMS gesteuert werden. Wenn eine manuelle Aktivität zur Bearbeitung ansteht, so wird sie in die *Arbeitslisten* der entsprechenden Benutzer eingetragen. Nachdem einer von ihnen die Aktivität zur Bearbeitung ausgewählt hat, wird das zugehörige *Aktivitätenprogramm* gestartet, so dass er die Aktivität bearbeiten kann. Nach Beendigung des Programms schaltet das WfMS zur nächsten Aktivität weiter, die dann bearbeitet werden kann.

Um in diesem Beitrag konkrete Algorithmen für Migrationen angeben zu können, wird ein formales Workflow-Metamodell benötigt. Wir verwenden das Metamodell

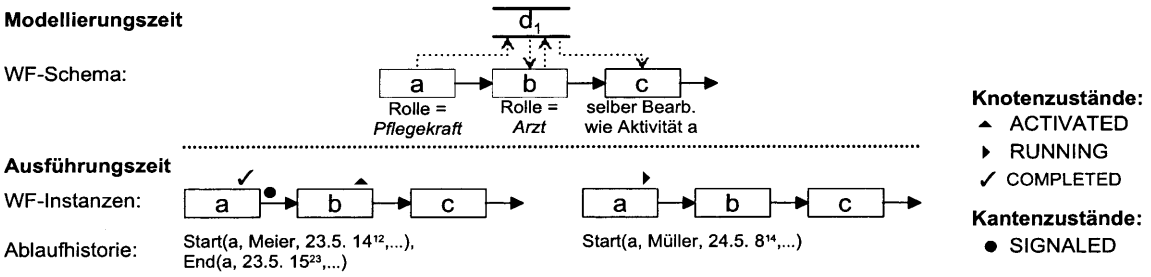

Abb. 1. Modellierung von Workflow-Schemata und Ausführung von Workflow-Instanzen.

[RD98] von ADEPT[3] [DRK00], die gewonnenen Erkenntnisse sind aber auch auf andere Workflow-Metamodelle übertragbar. Bei ADEPT können zur Spezifikation des Kontrollflusses unter anderem die Konstrukte Sequenz, bedingte und parallele Verzweigung und Schleife verwendet werden. Zur Festlegung des Datenflusses wird angegeben, welche prozessglobalen Variablen (*Datenelemente*) von welchen Aktivitäten gelesen bzw. geschrieben werden. Wie in Abb. 1 dargestellt, können Bearbeiterzuordnungen nicht nur Rollen referenzieren (z.B. Aktivität a), sondern es werden auch die in der Praxis häufig benötigten *abhängigen Bearbeiterzuordnungen* unterstützt. Mit diesen kann eine Aktivität z.B. demselben Bearbeiter wie eine Vorgängeraktivität oder einem Benutzer derselben Organisationseinheit zugeordnet werden.

Im unteren Teil von Abb. 1 sind zur Ausführungszeit erzeugte Workflow-Instanzen dargestellt. Damit der Workflow-Server die für eine bestimmte Workflow-Instanz aktuell zur Ausführung anstehenden Aktivitäteninstanzen ermitteln kann, benötigt er entsprechende Zustandsinformation. In ADEPT werden dazu den Knoten und Kanten des Ausführungsgraphen einer Workflow-Instanz sog. Zustandsmarkierungen[4] zugeordnet. Für ihre Festlegung bzw. Veränderung (und die damit zusammenhängende Aktivierung von Aktivitäten) gibt es wohldefinierte Regeln. Der Zustand einer Aktivität ist initial NOT_ACTIVATED. Er geht, wenn alle Vorbedingungen (Signalisierung der eingehenden Kanten) erfüllt sind, in ACTIVATED über, was bedeutet, dass die Aktivität gestartet werden kann. Bei manuell ausgeführten Aktivitäten ermittelt der Workflow-Server (mit Hilfe des Organisationsmodells) dann die potenziellen Bearbeiter und erzeugt die entsprechenden Arbeitslisteneinträge. Wenn die Aktivität von einem Benutzer ausgewählt wird, wird sie aus den entsprechenden Arbeitslisten der anderen potenziellen Bearbeiter wieder entfernt. Nachdem die Datenelemente für die Versorgung der Eingabeparameter des Aktivitätenprogramms an den entsprechenden Workflow-Client übertragen wurden, kann dieses gestartet werden (Zustand RUNNING). Nach seiner Beendigung werden die Ausgabeparameter zum Workflow-Server transportiert und in entsprechenden Datenelementen persistent gespeichert. Dann geht der Zustand der Aktivität in TERMINATED über, woraufhin die ausgehenden Kanten markiert werden. Dies wiederum führt zur Berechnung der Zustandsmarkierung der Nachfolgeraktivitäten.

Für die Unterstützung der eingangs erwähnten fortschrittlichen Workflow-Konzepte, wie dynamische Änderungen, Überwachung von Zeitbedingungen und Realisierung abhängiger Bearbeiterzuordnungen, reicht die Zustandsinformation der Knoten und Kanten alleine noch nicht aus. Hier wird zusätzlich eine *Ablaufhistorie* benötigt, in welcher beim Starten und Beenden einer Aktivitäteninstanz ein entsprechender Eintrag erzeugt wird. Wie in Abb. 1 (vereinfacht) dargestellt, enthalten solche Einträge Informationen zum tatsächlichen Bearbeiter sowie zum Start- und Endezeitpunkt der Aktivität. Auf der Basis dieser Ablaufhistorie ist es dann z.B. möglich, abhängige Bearbeiterzuordnungen auszuwerten oder Zeitpläne für die Ausführung der Workflow-Instanz zu erstellen, da aus ihr die Bearbeiter und die Start- und Endezeiten der Aktivitäten hervorgehen.

2.2 Verteilte Workflow-Ausführung

Wie bei zahlreichen Ansätzen (siehe [BD99]) besteht bei ADEPT$_{distribution}$, dem Konzept zur verteilten Workflow-Ausführung von ADEPT, die Möglichkeit, eine Workflow-Instanz nicht nur durch einen einzigen Workflow-Server kontrollieren zu lassen, sondern ihre Ausführung verteilt durch mehrere Server zu steuern [BD97]. Dazu wird das zugehörige Workflow-Schema bei der Modellierung in *Partitionen* unterteilt, die zur Ausführungszeit von unterschiedlichen Servern kontrolliert werden können (vgl. Abb. 2). Wird bei der Ausführung einer Instanz dieses Workflow-Schemas eine Aktivität beendet und gehört ihr Nachfolger einer anderen Partition an, so migriert die Kontrolle über diesen Workflow vom Server der aktuellen Partition (*Quellserver der Migration*) zum Server der Nachfolgerpartition (*Zielserver*). Dabei muss eine Beschreibung des Zustands der Workflow-Instanz zum Zielserver transportiert werden, um dort mit der Kontrolle fortfahren zu können.[5] Da eine Migration die Übertragung der vom Quellserver verwalteten Workflow-Kontrolldaten, workflow-relevanten Daten und Anwendungsdaten (vgl. [WMC99]) der Workflow-Instanz erfordert, verursacht sie aber Kommunikationskosten. Um keinen unnötigen Aufwand zu generieren, kommunizieren Workflow-Server in ADEPT ausschließlich bei

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

[4] Diese Markierung bleiben (anders als bei Petri-Netzen) auch nach Beendigung der Aktivität erhalten, so dass anhand des Zustands unmittelbar erkannt werden kann, ob eine bestimmte Aktivität schon ausgeführt wurde (für Details siehe [RD98]).

[5] Eine Alternative hierzu wäre, die Ausführung jeder einzelnen Aktivität am Zielserver zu veranlassen (im Stile eines Remote Procedure Calls). Diese Vorgehensweise hat aber den Nachteil, dass für jede einzelne Aktivität Nachrichten übertragen werden müssen und dass Daten evtl. redundant bei der Initiierung mehrerer Aktivitäten transferiert werden.

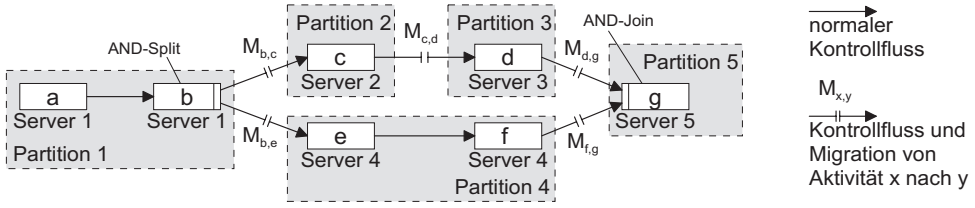

Abb. 2. Partitionierung eines Workflow-Schemas und verteilte Workflow-Ausführung.

Migrationen, d.h. Partitionen parallel ausgeführter Aktivitäten (z.B. Partition 2 und 4 in Abb. 2) werden von den entsprechenden Workflow-Servern unabhängig voneinander kontrolliert. Bei der Aktivitätenausführung findet dementsprechend keine Synchronisation oder Kommunikation zwischen diesen Workflow-Servern statt, weshalb einem Server der Zustand von parallel ausgeführten Aktivitäten in der Regel nicht bekannt ist. So weiß der Workflow-Server der Partition 4 zum Beispiel nicht, ob die parallel ausgeführte Aktivität c schon beendet ist oder nicht. Die verschiedenen an einer Workflow-Instanz beteiligten Workflow-Server verfügen also über (teilweise) unterschiedliche Zustandsinformation und dementsprechend über unterschiedliche Ablaufhistorien.

Eine Partitionierung des Workflow-Schemas und die dadurch notwendig werdenden Migrationen zur Ausführungszeit werden auch von einigen anderen in der Literatur diskutierten Ansätzen unterstützt (z.B. [CGP+96, MWW+98]). ADEPT verfolgt zusätzlich das Ziel, die Kommunikationskosten eines derart verteilten Ansatzes zu minimieren. Unsere Erfahrungen mit existierenden WfMS, Modellrechnungen und Simulationen [BD97, BD99, BD00] haben gezeigt, dass zwischen einem Workflow-Server und seinen Clients zahlreiche Nachrichten und große Datenmengen ausgetauscht werden. Mit steigender Anzahl von Benutzern und Workflow-Instanzen kann dies dazu führen, dass das Kommunikationssystem überlastet wird. Aus diesem Grund wird in ADEPT der Workflow-Server für jede Aktivität so gewählt, dass die insgesamt anfallenden Kommunikationskosten minimiert werden. Dazu wird – vereinfacht ausgedrückt – der Workflow-Server einer Aktivität so festgelegt, dass er in demjenigen Teilnetz liegt, dem die Mehrzahl ihrer potenziellen Bearbeiter angehört. Dadurch gelingt es weitgehend, teilnetzübergreifende Kommunikation zwischen einem Workflow-Server und seinen Clients zu vermeiden. Außerdem werden die Antwortzeiten verbessert und die Verfügbarkeit erhöht, da bei der Ausführung von Aktivitäten kein Gateway oder WAN benötigt wird (für Details und die zugehörigen Algorithmen siehe [BD97]).

Die Zuordnung von Workflow-Servern zu Aktivitäten (bzw. Partitionen) ist in ADEPT$_{distribution}$ sowohl statisch als auch dynamisch möglich, abhängig von den für die einzelnen Schritte definierten Bearbeiterzuordnungen. Werden die Bearbeiter einer Aktivität direkt durch ihre Rolle und Organisationseinheit festgelegt, so werden statische Serverzuordnungen verwendet. Für viele Anwendungen werden zusätzlich abhängige Bearbeiterzuordnungen benötigt (z.B. derselbe Bearbeiter wie Aktivität n). Da der entsprechende Bearbeiter erst im Verlauf der Workflow-Ausführung feststeht, ist es in solchen Fällen vorteilhaft, den Workflow-Server dieser Aktivitäteninstanz erst zur Ausführungszeit festzulegen. Für diesen Fall ist es möglich, den Workflow-Server so auszuwählen, dass er sich in der Organisationseinheit der entsprechenden Bearbeiter befindet. Zu diesem Zweck wurde für ADEPT$_{distribution}$ das Konzept der *variablen Serverzuordnungen* [BD00] entwickelt, bei denen der tatsächliche Server einer Aktivitäteninstanz erst nach Beendigung der Vorgängeraktivitäten festgelegt wird. Auch bei einigen anderen Ansätzen kann ein Workflow-Server dynamisch zur Ausführungszeit festgelegt werden [AMG+95, BMR96, SM96].

2.3 Problemstellung und Beitrag

Bei Migrationen von Workflow-Instanzdaten zwischen Workflow-Servern entstehen hohe Kosten (vgl. Abschnitt 6). Das Ziel dieser Arbeit ist es, Verfahren zu entwickeln, mit denen dieses Datenvolumen reduziert werden kann. In bisherigen Veröffentlichungen zu ADEPT$_{distribution}$ [BD97, BD00] haben wir Verfahren zur Reduzierung der Kommunikationslast betrachtet, die auf der *Vorberechnung von geeigneten Serverzuordnungen* basieren. Diese werden zur *Modellierungszeit* eingesetzt. Dagegen werden bei den in der vorliegenden Arbeit vorgestellten Verfahren die Kommunikationskosten durch zur *Ausführungszeit* durchgeführte *Optimierungen* reduziert, welche die Serverzuordnung nicht verändern. Die Reduktion der Kommunikationskosten wird erreicht, indem nicht alle für die betroffene Workflow-Instanz verfügbaren Daten zum Zielserver übertragen werden, sondern nur diejenigen, die dieser Server tatsächlich für die korrekte Workflow-Ausführung benötigt und noch nicht besitzt. Ein Workflow-Server kann beispielsweise bereits über Daten verfügen, wenn er früher an der Kontrolle der Workflow-Instanz beteiligt war, oder wenn ihm bestimmte Informationen bei vorausgehenden Migrationen übermittelt worden sind. Da die Migrationen durch entsprechende Maßnahmen „billiger“ werden, können sie ggf. häufiger und damit auch für kleinere Prozessfragmente durchgeführt werden. Die Workflow-Steuerung erfolgt dementsprechend meist im Teilnetz derjenigen Benutzer, welche die Aktivität bearbeiten, so dass sich die Antwortzeiten und die Verfügbarkeit des WfMS verbessern.

Um einen Eindruck zu vermitteln, welche Art von Optimierungen im Zusammenhang mit der Durchführung von Prozessmigrationen im Einzelnen möglich sind, wollen wir anhand von Abb. 2 einige Beispiele betrachten:

– Die Zustandsinformation (inkl. Ablaufhistorie) zu den Aktivitäten a und b wird bei herkömmlichen Migrationen über beide parallele Zweige zur AND-Join-Aktivität g transportiert. Das heißt, sie würde sowohl bei der Migration $M_{d,g}$ als auch bei $M_{f,g}$ zum Server 5 übertragen. Wie man leicht sieht, kann eine dieser beiden (redundanten) Übertragungen eingespart werden.

– Angenommen, die Aktivität c schreibt einen Ausgabeparameter in ein Datenelement, auf das von der Aktivität g lesend zugegriffen wird (nicht aber von d). Dann ist es wenig sinnvoll, das Datenelement entlang des Kontrollflusses vom Server 2 über den Server 3 zum Server 5 zu übertragen. Eine Optimierung wäre es, das Datenelement direkt vom Server 2 zum Server 5 zu übertragen. Dies ist besonders rentabel, wenn sehr große Datenelemente (z.B. CAD-Pläne oder Multimediadaten wie Videos und Bitmaps) betroffen sind. Erwähnt sei an dieser Stelle, dass eine Speicherung solcher Daten in einer zentralen Datenbank nachteilig wäre, da dann evtl. mehrfach (bei der Ausführung mehrerer Aktivitäten) weit entfernt auf sie zugegriffen werden müsste. Es ist günstiger, die Daten einmal zu migrieren, um sie dann im lokalen Teilnetz verfügbar zu haben.
– Wird von der Aktivität g ein Datenelement benötigt, das von der Aktivität a geschrieben und von c und d gelesen wurde, dann kann der Wert dieser Variablen von den Workflow-Servern 1, 2 und 3 bezogen werden. Es existiert insofern Optimierungspotential, als dass das Datenelement von demjenigen Server bezogen werden kann, zu dem die kostengünstigste Kommunikationsverbindung besteht. Eine redundante Übertragung sollte dabei vermieden werden. Angenommen, die Server 1 und 3 sind mit dem Server 5 nur über eine ISDN-Verbindung verbunden, wohingegen der Server 2 in einem zum Server 5 benachbarten Teilnetz mit einer Verbindung über ein schnelles Gateway liegt. Dann ist es wesentlich günstiger, wenn das (große) Datenelement vom Server 2 bezogen wird.

Die Beispiele zeigen, dass bei „naiver“ Realisierung der verteilten Workflow-Steuerung Daten unnötigerweise zu einer Partition übertragen werden können. Eine solche redundante Übertragung sollte vermieden werden. Im Folgenden werden Verfahren vorgestellt, mit denen Migrationen effizient durchgeführt werden können, indem die entstehenden Migrationskosten minimiert werden. Dazu werden im Einzelnen Verfahren zur Migration der Kontrolldaten einer Workflow-Instanz (Zustand, Ablaufhistorie) und Verfahren zur optimalen und redundanzfreien Übertragung von Datenelementen vorgestellt.

3 Möglichkeiten zur Migration von Workflow-Kontrolldaten

In diesem Abschnitt untersuchen wir, wie die Zustandsinformation einer Workflow-Instanz (Workflow-Kontrolldaten nach [WMC99]) migriert werden soll. Dazu analysieren wir zuerst die prinzipiell möglichen Vorgehensweisen und beschreiben anschließend ein ausgewähltes Verfahren.

3.1 Auswahl eines geeigneten Verfahrens

Im Folgenden werden alternative Ansätze zur Migration der Kontrolldaten einer Workflow-Instanz (Zustand und Ablaufhistorie) diskutiert[6]:

Ansatz 1: Minimallösung. Die Übertragung der Zustandsinformation ist mit minimalem Kommunikationsaufwand möglich, wenn dem Zielserver lediglich die Quell- und Zielaktivität der Migration mitgeteilt wird. Damit weiß er, um welche Migration es sich handelt, und somit, an welcher Stelle im Workflow-Ausführungsgraphen er die Abarbeitung fortsetzen muss.

Der Nachteil dieses einfachen Ansatzes ist das Fehlen jeglicher Zusatzinformation aus der Ablaufhistorie. Dadurch können die meisten der in der Einleitung erwähnten fortschrittlichen Workflow-Konzepte nicht realisiert werden. Die fehlende Information zu Bearbeitern von Vorgängeraktivitäten etwa schließt die Verwendung von abhängigen Bearbeiterzuordnungen aus. Für die Berechnung von Zeitplänen für die Bearbeitung der Aktivitäteninstanzen werden die in der Ablaufhistorie vermerkten Ausführungszeitpunkte der Vorgängeraktivitäten benötigt.

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

Zwar ist bei diesem Ansatz die Zustandsinformation verfügbar, allerdings fehlt auch hier jegliche Zusatzinformation (z.B. zu Bearbeitern oder Ausführungszeitpunkten der Aktivitäten).

Ansatz 3: Übertragung der Zustände und der Ablaufhistorie. Um das letztgenannte Problem zu lösen, kann die gesamte zu dieser Workflow-Instanz gespeicherte Information (inkl. Ablaufhistorie) übertragen werden. Werden an Synchronisationspunkten unterschiedliche Zustände für eine Aktivität empfangen, so muss wie beim Ansatz 2 der aktuell gültige Zustand bestimmt werden.

Da bei diesem Ansatz alle Kontrolldaten der Workflow-Instanz migriert werden, treten die oben beschriebenen Probleme nicht auf. Der Nachteil ist aber, dass Ausführungsinformationen redundant übertragen werden, da sich z.B. die Information, dass eine Aktivität beendet wurde, sowohl in der Ablaufhistorie als auch in den Aktivitätenmarkierungen widerspiegelt.

Ansatz 4: Übertragung (nur) der Ablaufhistorie. Die gesamte von Nachfolgeraktivitäten benötigte Information zu einer Workflow-Instanz findet sich in ihrer Ablaufhistorie wieder. Deshalb kann, ausgehend vom (auf den Servern repliziert vorhandenen) Workflow-Schema, durch das „Nachspielen“

[6] Bei einer Migration wird zusätzlich zu der explizit aufgeführten Information stets auch die global eindeutige ID der Workflow-Instanz transferiert.

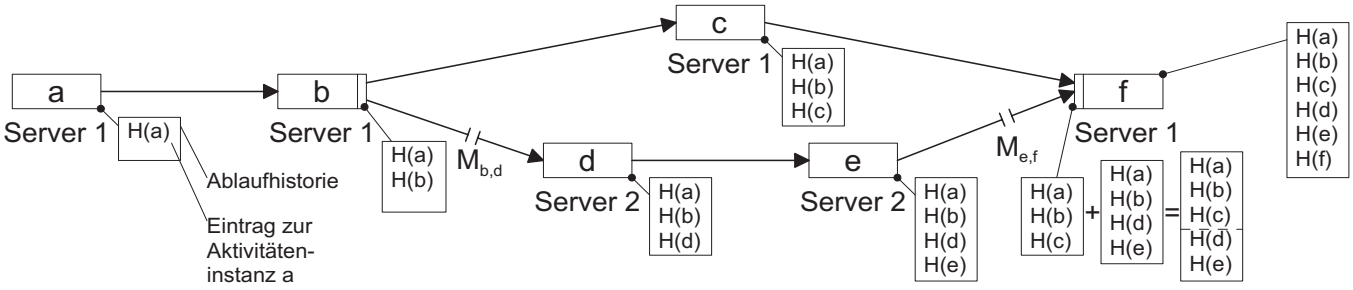

Abb. 3. Beispiel für das Zusammenführen von Ablaufhistorien.

der in der Ablaufhistorie vermerkten Aktionen der Zustand der Workflow-Instanz rekonstruiert werden. Die Zustandsinformation einer Workflow-Instanz kann somit übertragen werden, indem ausschließlich die Ablaufhistorie transferiert wird.

Bewertung. Ein gravierender Nachteil der Ansätze 1 und 2 ist das Fehlen von Zusatzinformation zur Workflow-Instanz. Sollen bei Zugrundelegung dieser Ansätze fortschrittliche WfMS-Konzepte realisiert werden, muss häufig Information nachträglich angefordert werden. Dies führt dazu, dass eine große Anzahl von Übertragungen stattfindet, da die Kontrolldaten, die bei den anderen Ansätzen auf einmal übertragen werden, durch mehrere Anforderungen besorgt werden müssen. Im Extremfall müssen von allen Vorgängeraktivitäten Daten nachgefordert werden, z.B. wenn die Start- und Endezeitpunkte aller Aktivitäten benötigt werden, um Zeitpläne für die nachfolgenden Aktivitäten zu berechnen [DRK00]. Das Nachfordern von Kontrolldaten beeinträchtigt außerdem die Verfügbarkeit des WfMS, da die Bearbeitung eines Workflows nur fortschreiten kann, wenn die Server, von denen Informationen benötigt werden, gerade verfügbar sind. Aus diesen Gründen scheiden die Ansätze 1 und 2 aus.[7] Der Ansatz 3 disqualifiziert sich wegen des großen Umfangs der zu transferierenden Datenmenge. Abgesehen davon resultiert aus den zusätzlich übertragenen Daten kein Vorteil gegenüber dem Ansatz 4. Der letztgenannte Ansatz stellt einen guten Kompromiss dar, da hier die Datenmenge in einem vernünftigen Rahmen bleibt und alle benötigten Informationen vorhanden sind. Im Folgenden wird dieser Ansatz deshalb näher untersucht.

3.2 Migration der Ablaufhistorie

Nachdem sich gezeigt hat, dass die günstigste Variante zur Migration von Zustandsinformation darin besteht, die Ablaufhistorie zu übertragen, betrachten wir nun diesen Ansatz etwas detaillierter. Von Interesse ist dabei vor allem, wie an Synchronisationspunkten des Ausführungsgraphen (AND-Join) die Ablaufhistorien der verschiedenen Vorgängeraktivitäten zusammengeführt werden.

Bei einem Workflow ohne parallel ausgeführte Zweige ist der Ablauf des Verfahrens denkbar einfach: Da eine Ablaufhistorie stets von nur einer Vorgängeraktivität empfangen wird, kann die lokal evtl. schon vorhandene Historie[8] durch diese ersetzt werden. Durch das „Nachspielen“ der in der Ablaufhistorie vorhandenen Einträge erhält man dann den aktuellen Zustand des Ausführungsgraphen.

Interessanter ist der Fall, dass parallele Zweige unterschiedlicher Server zusammengeführt werden müssen (z.B. bei der AND-Join-Aktivität g in Abb. 2), da dann zwei unterschiedliche Versionen der Ablaufhistorie aufeinander treffen. Beim Zusammenführen von Historieninformation verschiedener Workflow-Server muss gewährleistet sein, dass das Ergebnis wieder eine korrekte Historie darstellt, d.h., dass eine Ablaufhistorie entsteht, die auch auf einem zentralen System mit nur einem einzigen Workflow-Server entstanden sein könnte. Es müssen sich also alle im Workflow-Schema definierten Reihenfolgebeziehungen von Aktivitäten in der Ablaufhistorie widerspiegeln, auch wenn diese Aktivitäten von unterschiedlichen Servern kontrolliert wurden. Außerdem soll ein Workflow-Server stets den vollständigen für ihn relevanten Zustand einer von ihm kontrollierten Workflow-Instanz kennen. Das heißt, ein Workflow-Server, der gerade die Aktivitäteninstanz a kontrolliert, kennt stets alle Historieneinträge (und damit den Zustand) der Vorgängeraktivitäten von a, weil diese bei Migrationen zu ihm weitergereicht wurden. Über die Historieneinträge von parallel zu a ausgeführten Aktivitäten muss dieser Server im Allgemeinen aber nicht verfügen.

Die bei einer Migration empfangene und die auf diesem Server schon lokal vorhandene (bzw. davor empfangene) Ablaufhistorie müssen zu einer einzigen Ablaufhistorie zusammengeführt werden. Dies geschieht, indem die Historieneinträge „gemischt“ werden. Wie dieses Mischen erfolgt, wollen wir am Beispiel der in Abb. 3 dargestellten AND-Join-Aktivität f betrachten. Wenn sich eine Aktivitäteninstanz (z.B. a) im Kontrollfluss vor einer anderen Aktivität (z.B. d) befindet, so müssen auch die zu ihr gehörenden Einträge in der resultierenden Historie vor den Einträgen der anderen Aktivität einsortiert werden. Die Reihenfolge von Historieneinträgen parallel ausgeführter Aktivitäten (z.B. c und d) kann dagegen beliebig gewählt werden. Man kann zeigen, dass bei einer Migration empfangene Historieneinträge, die dem Zielserver noch nicht bekannt sind, einfach an die lokal schon vorhandene Ablaufhistorie an-

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

[8] Dieser Fall ist gegeben, wenn der Migrationszielserver bereits früher einmal eine Partition des Ausführungsgraphen dieser Workflow-Instanz kontrolliert hat.

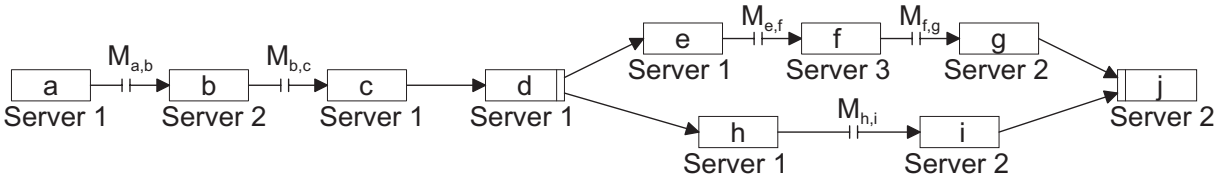

Abb. 4. Beispiel für die Migration von Workflow-Kontrolldaten.

gehängt werden können [Zei99], um diesen Bedingungen zu genügen (vgl. Abb. 3). Die Position schon vorhandener Einträge bleibt unverändert. Dieses Verfahren arbeitet korrekt, weil für alle Aktivitäten gilt: Wenn ein Historieneintrag zu einer Aktivität am Zielserver bekannt ist, dann sind dort auch die Historieneinträge zu allen Vorgängeraktivitäten bekannt (siehe oben). Deshalb kann es sich bei einer Aktivitäteninstanz, von der erstmalig ein Eintrag empfangen wird (z.B. d), nicht um eine Vorgängeraktivität von schon bekannten Aktivitäten (a, b und c) handeln. Diese Aktivität kann also nur parallel zu diesen Aktivitäten oder nach diesen ausgeführt worden sein. Deshalb ist es korrekt, den erstmalig empfangenen Historieneintrag am Ende der Ablaufhistorie zu platzieren. Dabei darf die Reihenfolge neuer Einträge untereinander (z.B. d und e) natürlich nicht verändert werden. Da mittels einer Hash-Tabelle (mit konstantem Zeitaufwand) geprüft werden kann, ob ein Eintrag schon bekannt ist, und neue Einträge einfach an die Historie angehängt werden, ergibt sich ein äußerst effizienter Algorithmus.

4 Optimierte Übertragung von Workflow-Kontrolldaten

Der im vorherigen Abschnitt diskutierte Ansatz 4 ist am besten für die Übertragung der Kontrolldaten einer Workflow-Instanz geeignet, da er das günstigste Kommunikationsverhalten aufweist. Bei dem in Abschnitt 3.2 skizzierten Verfahren kann es allerdings zu redundanten Datenübertragungen kommen. So werden bei dem in Abb. 3 dargestellten Beispiel die Historieneinträge der Aktivitäten a und b bei der Migration $M_{e,f}$ zum Server 1 übertragen, obwohl sie diesem schon bekannt sind. Im dem nun folgenden Abschnitt werden verbesserte Verfahren vorgestellt, bei denen zum Zielserver nur die wirklich benötigten Historieneinträge übertragen werden.

4.1 Versenden von Ablaufhistorieneinträgen

Die nächstliegende Idee zur Reduzierung des Datenvolumens beim Übertragen von Workflow-Kontrolldaten besteht darin, nur den benötigten Ausschnitt der Ablaufhistorie und nicht die komplette Ablaufhistorie an den Migrationszielserver zu senden. Aus Platzgründen verzichten wir hier auf eine algorithmische Darstellung dieses Verfahrens. Stattdessen erläutern wir nur das Grundprinzip und diskutieren die entstehenden Nachteile. In Abschnitt 4.2 wird dann ein verbessertes Verfahren vorgestellt, das diese Nachteile vermeidet.

Um bei einer Migration nicht immer die gesamte Ablaufhistorie einer Workflow-Instanz übertragen zu müssen, berechnet der Quellserver, welcher Teil dieser Historie vom Zielserver für die weitere Ausführung noch benötigt wird. Dementsprechend wird nur dieser Ausschnitt bei der Migration übertragen. Dazu sucht der Quellserver in der Historie nach Einträgen, deren zugehörige Aktivitäteninstanz vom Zielserver kontrolliert wurde (sofern vorhanden). Bei der Übertragung der Ablaufhistorie können diese Einträge dann weggelassen werden. Dasselbe gilt für Historieneinträge, die sich auf Vorgängeraktivitäteninstanzen (bzgl. des Kontrollflusses) dieser Aktivitäteninstanzen beziehen, da auch diese Historieneinträge dem Zielserver schon bekannt sind. Der Grund dafür ist, dass ein Workflow-Server alle Historieneinträge kennt, die zu Vorgängeraktivitäten der von ihm kontrollierten Aktivitäteninstanzen gehören (vgl. Abschnitt 3.2).

Durch dieses Verfahren werden Historieneinträge, von denen der Quellserver einer Migration sicher weiß, dass sie auf dem Zielserver schon bekannt sind, an diesen nicht mehr übertragen. So werden bei dem in Abb. 4 dargestellten Beispiel einer parallelen Verzweigung die Historieneinträge zu den Aktivitäteninstanzen a und b weder bei der Migration $M_{f,g}$ noch bei $M_{h,i}$ an den Server 2 übertragen, da dieser bereits die Aktivität b kontrolliert hat und deshalb die zu Aktivität b und zur Vorgängeraktivität a gehörenden Historieneinträge schon kennt.

Bei dem oben beschriebenen Verfahren kann es aber immer noch zur redundanten Übertragung von Historieninformation kommen. Das Problem des Verfahrens ist, dass der Quellserver einer Migration nicht exakt weiß, welche Historieneinträge beim Zielserver schon vorhanden sind. Nehmen wir in dem Beispiel aus Abb. 4 an, dass die Migration $M_{f,g}$ vor $M_{h,i}$ ausgeführt wird. Dann müssen bei der Migration $M_{f,g}$ die Historieneinträge der Aktivitäten c und d übertragen werden. Bei der später ausgeführten Migration $M_{h,i}$ kann der Quellserver 1 aber nicht wissen, dass die entsprechenden Einträge dem Zielserver 2 schon bekannt sind. Diese nicht vorhandene Information führt dazu, dass er die Einträge unnötigerweise überträgt. Das vorgestellte Verfahren ermöglicht also zwar eine Reduzierung der zu übertragenden Datenmenge, redundante Datenübertragung lässt sich damit aber nicht völlig ausschließen.

4.2 Anfordern von Ablaufhistorieneinträgen

Die redundante Übertragung von Ablaufhistorieneinträgen kann ausgeschlossen werden, wenn der Zielserver einer Migration dem Quellserver Information darüber liefert, welche Historieneinträge ihm bereits bekannt sind.[9] Ein Verfahren

[9] Eine Alternative dazu wäre, dass die anderen Server des WfMS Information darüber austauschen, welche Historieneinträge schon zu diesem Zielserver migriert wurden. Da ein solches Verfahren aber sehr komplex wäre und außerdem nicht weniger Kommunikation erfordern würde, wird dieser Ansatz nicht weiter verfolgt.

Algorithmus 1 (Zielserver: Anfordern von Ablaufhistorieneinträgen)
input
$SourceAct$: Quellaktivitäteninstanz der Migration
$OldHistory$: am Zielserver bekannter Teil der Ablaufhistorie
output
$LastActivities$: Menge von Aktivitäteninstanzen, bis zu denen die Ablaufhistorie dem Zielserver bekannt ist
begin
$LastActivities$ = $\emptyset$;
// für Migration sind nur Vorgängeraktivitäteninstanzen der Quellaktivität (inklusive) relevant
$RelevantHistory$ = $OldHistory \cap HistoryEntries(\{SourceAct\} \cup Predecessors(SourceAct))$;
while $RelevantHistory \neq \emptyset$ **do**
// L ist der hinterste Eintrag der Historie $RelevantHistory$, l die zugehörige Aktivitäteninstanz
L = $LastEntry(RelevantHistory)$;
l = $ActivityInstance(L)$;
$LastActivities$ = $LastActivities \cup \{l\}$;
// entferne Einträge zu l und aller Vorgänger bzgl. Kontrollfluss aus Ablaufhistorie
$RelevantHistory$ = $RelevantHistory - HistoryEntries(\{l\} \cup Predecessors(l))$;
end.

hierfür wird nun vorgestellt. Dabei kann eine redundante Übertragung von Historieneinträgen nur dann ausgeschlossen werden, wenn für eine Workflow-Instanz nie mehrere Migrationen zeitlich überlappend bei demselben Workflow-Server eingehen. Ansonsten kann der Zielserver nicht wissen, welche Historieneinträge noch durch die anderen gleichzeitig ablaufenden Migrationen geliefert werden. Deshalb verwendet der Zielserver einer Migration Sperren, um sicherzustellen, dass zu jedem Zeitpunkt für eine Workflow-Instanz maximal eine Migration eingehen kann, d.h. er blockiert ggf. kurzfristig die weitere Bearbeitung einer eingehenden Migration.

Das im Folgenden vorgestellte Verfahren basiert auf der Idee, dass der Zielserver der durchzuführenden Migration beim Quellserver diejenigen Teile der Ablaufhistorie anfordert, die bei ihm noch nicht vorhanden sind. Das Verfahren gliedert sich in 3 Phasen:

1. Der Quellserver informiert den Zielserver über die anstehende Migration. Dazu überträgt er ihm außer der ID der zu migrierenden Workflow-Instanz jeweils die ID der Quell- und Zielaktivitäten der Migration. Für das Beispiel der Migration $M_{f,g}$ aus Abb. 4 ergibt sich also die Übertragung: <WF-Inst-ID, f, g >
2. Der Zielserver der Migration fordert nun beim Quellserver den noch fehlenden Teil der Ablaufhistorie an. Dazu muss er ihm mitteilen, über welche Historieneinträge er bereits verfügt. Die einfachste Lösung, die IDs aller zugehörigen Aktivitäteninstanzen zu übertragen, wäre aber unnötig aufwendig. Sind die Historieneinträge einer Aktivitäteninstanz dem Zielserver bekannt, so kennt er nämlich auch die Einträge zu Vorgängeraktivitäten dieser Aktivitäteninstanz. Deshalb genügt es, die Menge $LastActivities$ zu übertragen, welche die IDs derjenigen Aktivitäteninstanzen enthält, von denen dem Zielserver keine Historieneinträge zu Nachfolgeraktivitäten bekannt sind. Dies sind sozusagen die „letzten dem Zielserver bekannten Aktivitäten“ der Workflow-Instanz. Es kann vorkommen, dass die Menge $LastActivities$ mehrere Aktivitäten enthält, da der Zielserver einer Migration Historieneinträge zu Aktivitäten besitzen kann, die verschiedenen parallelen Zweigen angehören. Dann muss die jeweils letzte bekannte Aktivität jedes Zweiges in $LastActivities$ enthalten sein.

   Mit dem Algorithmus 1 kann der Zielserver einer Migration die Menge $LastActivities$ berechnen. Dazu reduziert er die lokal bei ihm vorhandene Ablaufhistorie $OldHistory$ der Workflow-Instanz auf die Vorgängeraktivitäten der Migrationsquellaktivität, da nur dieser Teil der Workflow-Instanz für die Migration relevant ist. Dadurch ergibt sich die Ablaufhistorie $RelevantHistory$. Der hinterste Eintrag L von $RelevantHistory$ gehört zur einer Aktivitäteninstanz l, deren Historieneinträge am Zielserver bekannt sind. Da der Zielserver auch über die Historieneinträge aller Vorgängeraktivitäten von l verfügt, entfernt er diese aus der Ablaufhistorie $RelevantHistory$. Die Aktivitäteninstanz l wird in $LastActivities$ aufgenommen, weil die Vorgängeraktivitäten von l (inkl. l) am Zielserver bekannt sind, nicht aber die Nachfolgeraktivitäten. Der gesamte Vorgang wird solange wiederholt, bis in der Historie $RelevantHistory$ keine Einträge mehr existieren. Ist dies erreicht, so wurde die vollständige Menge $LastActivities$ ermittelt. Im Beispiel der Migration $M_{f,g}$ aus Abb. 4 ergibt sich $LastActivities$ = $\{b\}$, weil der Zielserver (Server 2) nur über Information zu den Aktivitäten a und b verfügt (b wurde vom Server 2 kontrolliert). Da a eine Vorgängeraktivität von b ist, wird a nicht in $LastActivities$ aufgenommen.
3. Nachdem der Quellserver der Migration die Menge $LastActivities$ empfangen hat, kann er mit Algorithmus 2 die zu übertragende Ablaufhistorie $MigrHistory$ berechnen. Dazu reduziert er die bei ihm vorhandene Ablaufhistorie $LocalHistory$ auf Vorgänger der Quellaktivität der Migration (wie der Zielserver in Algorithmus 1). Anschließend entfernt er alle Einträge zu Aktivitäteninstanzen $l \in LastActivities$ und zu deren Vorgängern aus $MigrHistory$, so dass sich die zu übertragende Ablaufhistorie ergibt. Betrachten wir nochmals das oben angesprochene Beispiel der Migration $M_{f,g}$, bei dem in Schritt 2 die Menge $LastActivities$ = $\{b\}$ ermittelt wurde. In Schritt 3 werden dann die Einträge zu den Aktivitäteninstanzen a und b aus der lokal vorhandenen Historie (mit Einträgen zu den Aktivitäten a,

Algorithmus 2 (Quellserver: Anfordern von Ablaufhistorieneinträgen)
input
$SourceAct$: Quellaktivitäteninstanz der Migration
$LastActivities$: Menge der letzten am Zielserver bekannten Aktivitäteninstanzen
$LocalHistory$: am Quellserver vorhandene Ablaufhistorie
output
$MigrHistory$: der bei der Migration zu übertragende Teil der Ablaufhistorie
begin
// für Migration sind nur Vorgängeraktivitäten der Quellaktivität relevant
$MigrHistory = LocalHistory \cap HistoryEntries(\{SourceAct\} \cup Predecessors(SourceAct))$;
// Einträge zu bekannten Aktivitäteninstanzen aus Ablaufhistorie entfernen
for each $l \in LastActivities$ **do**
// Einträge zu l und Vorgängern von l aus Historie streichen
$MigrHistory = MigrHistory - HistoryEntries(\{l\} \cup Predecessors(l))$;
end.

b, c, d, e, f) entfernt, so dass die Historieneinträge der Aktivitäten c, d, e, f übertragen werden.

Wird nach der oben beschriebenen Migration $M_{f,g}$ die Migration $M_{h,i}$ zum selben Zielserver 2 durchgeführt,[10] so kennt dieser Server die Einträge zu den Aktivitäten a, b, c, d, e und f bereits. Durch Anwendung von Algorithmus 1 ergibt sich somit die Menge $LastActivities = \{d\}$. Algorithmus 2 entfernt dementsprechend die Einträge zu den Aktivitäten a, b, c und d aus der für diese Migration relevanten Historie (mit Einträgen zu a, b, c, d und h), so dass nur die Historieneinträge der Aktivität h übertragen werden. Da bei der Migration $M_{f,g}$ Historieneinträge zu den Aktivitäten c, d, e und f übermittelt wurden (s.o.), findet also keine redundante Informationsübertragung statt. Im Allgemeinen ist die redundante Übertragung von Historieneinträgen bei dem beschriebenen Verfahren stets ausgeschlossen: Ist ein Historieneintrag zu einer Aktivität n am Zielserver der Migration bereits bekannt, so wird dieser in der Schleife von Algorithmus 1 aus $OldHistory$ entfernt (sonst wäre die Schleife noch nicht verlassen worden). Deshalb wird der entsprechende Eintrag beim Durchlaufen der Schleife von Algorithmus 2 auch aus $MigrHistory$ entfernt und damit nicht übertragen.

5 Übertragung von Datenelementen

Nachdem wir in den Abschnitten 3 und 4 untersucht haben, wie die Kontroll- bzw. Statusdaten einer Workflow-Instanz bei Migrationen effizient übertragen werden können, wenden wir uns nun der Versorgung und Übernahme der Parameterdaten von Aktivitätenprogrammen zu ([WMC99]: Workflow-relevante Daten und Anwendungsdaten). In ADEPT werden diese Daten als globale Prozessvariablen (sog. Datenelemente) verwaltet. Die im Folgenden beschriebenen Verfahren lassen sich aber auch bei anderen Realisierungsformen für die Modellierung von Datenflüssen anwenden (z.B. bei Ein- und Ausgabencontainern wie in MQ-Series Workflow [LR00]). Bei den in der Literatur diskutierten Verfahren zur Migration von Datenelementen werden beim Übergang zwischen zwei Partitionen üblicherweise die Werte aller Datenelemente zur Nachfolgerpartition übertragen. Dies führt zu einer starken Belastung des Kommunikationssystems, insbesondere dann, wenn sehr große Datenelemente (z.B. multimediale Dokumente) verwendet und vom WfMS nicht nur Referenzen auf diese Daten verwaltet werden.[11] Im Folgenden werden Verfahren vorgestellt, mit denen die bei der Übertragung von Datenelementen entstehende Belastung des Kommunikationssystems deutlich reduziert werden kann. Bei der Entwicklung dieser Verfahren muss beachtet werden, dass von einem WfMS sowohl recht kleine als auch sehr große Datenelemente verwaltet werden können und dass nicht jedes Verfahren für beide Fälle gleich gut geeignet ist. Zusätzliche Schwierigkeiten ergeben sich aus der Tatsache, dass die Verfahren nicht auf den Fall statischer Serverzuordnungen beschränkt sein dürfen, sondern auch bei Verwendung variabler Serverzuordnungen anwendbar sein müssen.

5.1 Datenhistorie

Der Wert eines globalen Datenelements kann im Verlauf der Ausführung einer Workflow-Instanz mehrfach geschrieben werden und sich dadurch verändern. Da die „überschriebenen“ Werte später bei einem eventuellen (partiellen) Zurücksetzen noch für die Ausführung von Kompensationsaktivitäten der Workflow-Instanz benötigt werden, werden sie in ADEPT bis zu deren Beendigung aufbewahrt. Zu jedem Datenelement existiert deshalb eine Historie von Werten (zu denen jeweils auch die ID der schreibenden Aktivitäteninstanz verwaltet wird). Beim Lesen des Datenelements ist nur der zuletzt von einer Vorgängeraktivität des Lesers geschriebene Wert[12] gültig.[13] Um das Datenvolumen bei Migrationen zu reduzieren, wird bei allen im Folgenden vorgestellten Verfahren nur der jeweils aktuelle Wert übertragen. Dieser aktuelle Wert muss natürlich auch dann an den Migrationszielserver übertragen werden, wenn dort schon ein älterer (und damit ungültiger) Wert des Datenelements vorliegt. Im Falle einer (selten auftretenden) Kompensation muss der bei der Ausführung der zu kompensierenden Aktivität gültige Wert des Datenelements verwendet werden.

[10] Wie zu Beginn dieses Abschnitts erläutert, ist die überlappende Ausführung von Migrationen ausgeschlossen. Wird in dem Beispiel die Migration $M_{h,i}$ vor $M_{f,g}$ ausgeführt, so ergibt sich dasselbe Verhalten.

[11] Die Nachteile, die sich ergeben, wenn ein WfMS nur Referenzen auf Daten verwaltet, werden in Abschnitt 7 ausführlich diskutiert.

[12] Da in ADEPT das Schreiben eines Datenelements durch Aktivitäten paralleler Zweige nicht erlaubt ist (Vermeidung von Lost Updates), ist dieser Wert eindeutig identifizierbar.

[13] ADEPT verfolgt hiermit bzgl. der Verwaltung und Speicherung von Datenelementen einen ähnlichen Ansatz wie z.B. ConTracts [RS95].

Algorithmus 3 (Übertragung kleiner Datenelemente)
input
$TargetAct$: Zielaktivitäteninstanz der Migration
$SmallDataElements$: Menge der am Quellserver bekannten kleinen Datenelemente
$MigrHistory$: der zu übertragende Teil der Ablaufhistorie (von Algorithmus 2 ermittelt)
output
$MigrDataElements$: bei der Migration zu übertragende kleine Datenelemente
begin
$MigrDataElements = \emptyset$;
for each $d \in SmallDataElements$ **do**
// a hat diejenige Version von d geschrieben, die für $TargetAct$ gültig ist
$a = LastWriter(d, TargetAct)$;
// der zu a gehörende Ende-Historieneintrag wird bei der betrachteten Migration übertragen
if $END(a, \ldots) \in MigrHistory$ **then**
$MigrDataElements = MigrDataElements \cup \{d\}$;
end.

Dieser ist noch auf demjenigen Server vorhanden, welcher die Aktivität kontrolliert hat.

5.2 Kleine Datenelemente

Viele der von einem WfMS verwalteten Datenelemente, wie Integer-Werte oder kurze Strings, sind relativ „klein“, so dass für sie die im nachfolgenden Abschnitt vorgestellten Optimierungen nicht lohnend sind. Hier macht es keinen Sinn, die ohnehin bereits geringe Datenmenge noch weiter zu reduzieren und dabei zu riskieren, dass zusätzliche Kommunikationszyklen notwendig werden. In diesem Abschnitt wird ein Verfahren vorgestellt, das keine zusätzlichen Kommunikationszyklen erfordert, das ohne großen Berechnungsaufwand auskommt und das vermeidet, dass ein kleines Datenelement von mehreren Quellservern an den Zielserver einer Migration übertragen wird. Die (durchschnittliche) Größe, bis zu der ein Datenelement als klein gilt, kann vom Administrator des WfMS konfiguriert werden, so dass sich ein möglichst günstiges Laufzeitverhalten ergibt. In der Regel ist es sinnvoll, ein Datenelement der Menge der großen Datenelemente $LargeDataElements$ zuzuordnen, wenn es mehrere Netzwerkpakete ausfüllt. Andernfalls wird es der in diesem Abschnitt betrachteten Menge $SmallDataElements$ zugeordnet.

Die Menge der bei einer Migration $M_{SourceAct,TargetAct}$ zusammen mit den Ablaufhistorieneinträgen $MigrHistory$ zu übertragenden kleinen Datenelemente ($MigrDataElements$) kann mit Algorithmus 3 ermittelt werden. Er basiert auf der folgenden Idee: Für alle Datenelemente $d \in SmallDataElements$ kann diejenige Aktivitäteninstanz a bestimmt werden, die den für das Migrationsziel (d.h. den für die Zielaktivität $TargetAct$ der Migration) gültigen Wert von d geschrieben hat. Diese Aktivitäteninstanz kann unter Verwendung des Workflow-Schemas und der Ablaufhistorie der Workflow-Instanz ermittelt werden. Das Datenelement d wird bei der betrachteten Migration genau dann übertragen, wenn der Historieneintrag für die Beendigung der Aktivität a in der bei der Migration übertragenen Ablaufhistorie $MigrHistory$ enthalten ist (siehe Abschnitt 4.2). Da sichergestellt ist, dass ein solcher Historieneintrag höchstens einmal zu jedem Server übertragen wird, gilt dies auch für jede Version eines Datenelements. Bei dem vorgestellten Verfahren ist eine redundante Übertragung von Datenelementen somit ausgeschlossen. Des Weiteren wird kein zusätzlicher Kommunikationszyklus benötigt, da die Datenelemente der Menge $MigrDataElements$ zusammen mit der Menge der Historieneinträge $MigrHistory$ zum Migrationszielserver übertragen werden. Es kann aber vorkommen, dass Datenelemente zu diesem Server übertragen werden, obwohl sie dort überhaupt nicht benötigt werden. Da die hier betrachteten Datenelemente klein sind, ist dies akzeptabel. Ein Verfahren, bei dem solche unnötigen Übertragungen ausgeschlossen sind, wird im folgenden Abschnitt für große Datenelemente vorgestellt.

Angenommen, bei dem in Abb. 5 dargestellten Beispiel wird die Migration $M_{b,d}$ vor $M_{c,d}$ ausgeführt. Dann werden bei Verwendung des in Abschnitt 4.2 beschriebenen Verfahrens bei der Migration $M_{b,d}$ die Historieneinträge der Aktivitäten a und b zum Server 2 übertragen. Damit wird das von Aktivität a geschriebene kleine Datenelement d_1 bei dieser Migration ebenfalls transferiert. Bei der Migration $M_{c,d}$ werden die Historieneinträge von Aktivität c übertragen, weshalb auch das von c geschriebene Datenelement d_2 übermittelt wird. Das Datenelement d_1 wird bei dieser Migration aber nicht (redundant) an den Server 2 übertragen.

5.3 Große Datenelemente

Da für die Übertragung großer Datenelemente meist mehrere Netzwerkpakete benötigt werden, ist eine zusätzliche Kommunikation akzeptabel. Es ist außerdem sehr vorteilhaft, wenn ein solches Datenelement bei einer Migration nicht zum Zielserver transportiert werden muss, falls dieser es nicht benötigt. In dem Beispiel aus Abb. 6 wird das Datenelement d_1 von der Aktivität a geschrieben und von c gelesen (aber nicht von b). Deshalb können Kommunikationskosten eingespart werden, wenn das Datenelement d_1 direkt vom Server 1 zum Server 3 transportiert wird (ohne den Umweg über den Server 2 der Aktivität b). Um dies zu ermöglichen, wird im Folgenden ein Verfahren entwickelt, bei dem nicht nur die redundante Übertragung von Datenelementen ausgeschlossen wird, sondern auch nur solche Datenelemente zu einem Server transportiert werden, die von diesem tatsächlich benötigt werden. Dies ist insbesondere deshalb nicht trivial, weil im Falle von bedingten Verzweigungen im Voraus nicht feststeht, welche Aktivitäten der Zielpartition einer Migration tatsächlich ausgeführt werden, und damit, welche Datenelemente benötigt werden.

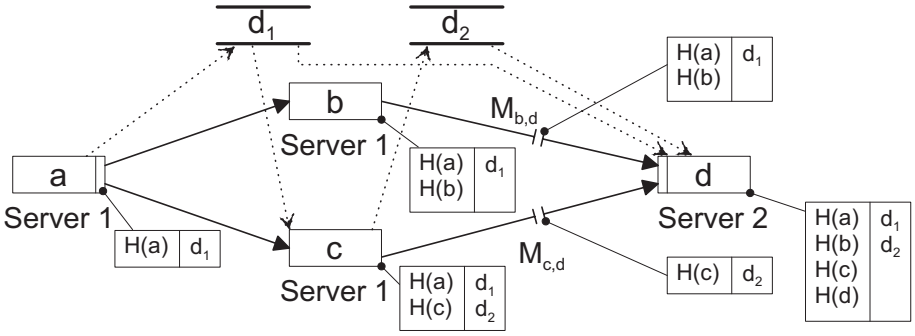

Abb. 5. Migration kleiner Datenelemente (Migration $M_{b,d}$ wird vor $M_{c,d}$ ausgeführt).

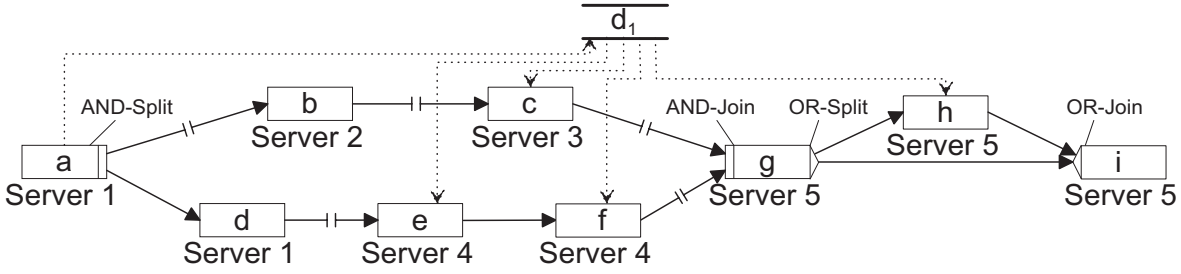

Abb. 6. Übertragung von großen Datenelementen bei Migrationen.

5.3.1 Versenden von Datenelementen

In [Zei99] wird ein Verfahren beschrieben, bei dem – nach Ausführung der „letzten“ Aktivität einer Partition – ein in dieser Partition geschriebenes Datenelement an die Workflow-Server derjenigen Partitionen gesendet wird, in denen dieses Datenelement potenziell gelesen wird. In dem Beispiel aus Abb. 6 würde das Datenelement d_1 also (nach Beendigung der letzten Aktivität a) vom Server 1 zum Server 3 gesendet, da es in dessen Partition von c gelesen wird. Dieses Verfahren weist allerdings einige Nachteile auf: So lassen sich Fälle konstruieren, in denen (ebenso wie beim Versenden der Ablaufhistorie) redundante Übertragungen auftreten. Außerdem ist das Verfahren bei variablen bzw. dynamischen Serverzuordnungen nicht einsetzbar. Der Grund dafür ist, dass zum Zeitpunkt des Versendens eines Datenelements der Server der im Ablauf evtl. erst viel später folgenden Zielpartition noch gar nicht bekannt ist. Schließlich weist dieses Verfahren ein schlechteres Kommunikationsverhalten auf als der im nachfolgenden Abschnitt beschriebene Ansatz, da ein Datenelement ausschließlich von demjenigen Workflow-Server bezogen werden kann, auf dem es geschrieben wurde.

5.3.2 Anfordern von Datenelementen

Um die gesamte bei der Aktivitätenausführung vorhandene Zustandsinformation nutzen zu können, ist es vorteilhaft, große Datenelemente bei einer Migration nicht zu versenden, sondern sie anzufordern. Um dies zu realisieren, sind zwei Verfahren denkbar: (1) Alle von einer Partition benötigten Datenelemente werden bei ihrer Aktivierung angefordert. (2) Vor dem Start jeder Aktivitäteninstanz werden die von ihr benötigten Datenelemente angefordert. Im Zusammenhang mit bedingten Verzweigungen führt das erste Verfahren zu Problemen, da zu Beginn einer Partition evtl. noch nicht feststeht, welche Datenelemente tatsächlich benötigt werden. Im Folgenden gehen wir deshalb auf das zweite Verfahren näher ein.

Eine Aktivitäteninstanz kann in ADEPT erst dann in den Zustand ACTIVATED übergehen, wenn alle eingehenden Kanten markiert wurden. Zu diesem Zeitpunkt sind alle Vorgängeraktivitäten beendet. Deshalb kann ein Datenelement prinzipiell vom Workflow-Server jeder Aktivität angefordert werden, welche es geschrieben oder gelesen hat. Im Beispiel aus Abb. 6 kann das von der Aktivität h benötigte Datenelement d_1 vom Server der Aktivität a und von den Servern der parallel ausgeführten Aktivitäten c bzw. e und f angefordert werden. Im Allgemeinen können die Workflow-Server, die über eine bestimmte Version eines Datenelements verfügen, mit Hilfe der zu diesem Zeitpunkt schon übertragenen Ablaufhistorie (siehe Abschnitt 4.2) ermittelt werden. Das Datenelement wird dann von demjenigen Workflow-Server angefordert, zu dem die kostengünstigste Netzwerkverbindung besteht. Um diesen Server bestimmen zu können, wird eine „Kommunikationskosten-Matrix“ vorgegeben, so dass ggf. WAN-Kommunikation vermieden werden kann.

Bevor für eine Aktivitäteninstanz Datenelemente angefordert werden, wird das in Abschnitt 4.2 beschriebene Verfahren zur Migration von Zustandsinformation für alle Vorgängeraktivitäten abgeschlossen (sofern diese überhaupt von einem fremden Workflow-Server kontrolliert wurden). Somit ist die relevante Ablaufhistorie für die zu aktivierende Aktivität $ActualAct$ bekannt. Mit dem Algorithmus 4 kann dann ermittelt werden, welche der von dieser Aktivität noch benötigten Datenelemente ($RequiredDataElements$) von welchem Workflow-Server angefordert werden sollen. Dazu wird für jedes Datenelement d dieser Menge die Aktivitäteninstanz w bestimmt, von welcher die für $ActualAct$ gültige Version des Datenelements geschrieben wurde. Die

Algorithmus 4 (Anfordern großer Datenelemente)
input
$ActualAct$: die aktuell auszuführende Aktivitäteninstanz
$LargeDataElements$: Menge der großen Datenelemente
$LocalDataElements$: Menge der auf dem lokalen Server schon vorhandenen Datenelemente
output
$OptServers$: Menge von Tupeln $< d, s >$, die angeben, von welchem Server s das große Datenelement d angefordert werden soll
begin
$OptServers = \emptyset$;
// Menge der von der Aktivitäteninstanz $ActualAct$ gelesenen großen Datenelemente
$RequiredDataElements = \{d \in LargeDataElements \mid d \in ReadSet(ActualAct)\}$;
// schon auf dem lokalen Server vorhandene Datenelemente nicht mehr anfordern
$RequiredDataElements = RequiredDataElements - LocalDataElements$;
for each $d \in RequiredDataElements$ **do**
// w hat diejenige Version von d geschrieben, die für $ActualAct$ gültig ist
$w = LastWriter(d, ActualAct)$;
// R enthält diejenigen Aktivitäteninst., die das von w geschriebene Datenelement d gelesen haben
$R = \{r \mid w \in Predecessors(r) \wedge r \in Reader(d)\}$;
// s_opt ist der bzgl. der Kommunikationskosten vom lokalen Server aus am günstigsten erreichbare
// Server, von dem d bezogen werden kann
$Sources = \{Server(w)\} \cup \{Server(r) \mid r \in R\}$;
wähle $s\_opt \in Sources$ mit $CommQual(Server(ActualAct), s\_opt)$ ist minimal;
$OptServers = OptServers \cup \{< d, s\_opt >\}$;
end.

Aktivität w ist also der „letzte“ Vorgänger von $ActualAct$, der das Datenelement d geschrieben hat. Vom Workflow-Server dieser Aktivität könnte das Datenelement nun angefordert werden. Darüber hinaus verfügen auch die Server aller Aktivitäteninstanzen $r \in R$, welche diese Version des Datenelements gelesen haben, über dessen aktuellen Wert. Diese Aktivitäten $r \in R$ sind diejenigen, welche lesend auf d zugegriffen haben und außerdem im Kontrollfluss auf w folgen. Das Datenelement d kann also vom Workflow-Server der Aktivität w und von den Servern der Aktivitäten $r \in R$ angefordert werden. Von diesen Servern $s \in Sources$ wird derjenige ausgewählt, der die bzgl. der Kommunikationskosten günstigste Verbindung zu dem Server aufweist, der $ActualAct$ kontrolliert und damit der Empfänger des Datenelements ist.

Bei dem vorgestellten Verfahren könnten Datenelemente redundant übertragen werden, wenn durch einen Workflow-Server für mehrere parallele Zweige dasselbe Datenelement angefordert würde. Dies wird verhindert, indem Datenelemente derselben Workflow-Instanz nicht überlappend angefordert werden. Dann kann es zu keiner redundanten Übertragung kommen, da nur Versionen von Datenelementen angefordert werden, die auf dem Zielserver noch nicht vorhanden sind. So wird in dem Beispiel aus Abb. 6 bei der Aktivierung der Aktivität f das Datenelement d_1 nicht angefordert, weil es von der Aktivität e gelesen wurde und deshalb auf diesem Server schon vorhanden ist. Ein Datenelement wird erst bei der Aktivierung derjenigen Aktivität angefordert, welche es liest. Dies hat den Vorteil, dass es nur dann übertragen werden muss, wenn diese Aktivität auch tatsächlich aktiviert wird. Wird in dem Beispiel aus Abb. 6 der Zweig mit der Aktivität h nicht gewählt, so benötigt der Server 5 das Datenelement d_1 nicht. Bei dem vorgestellten Verfahren wird d_1 durch den Server 5 dann auch nicht angefordert. Der Algorithmus 4 wird nämlich für $ActualAct = h$ niemals ausgeführt, da die Aktivität h nicht aktiviert wird.

Dass große Datenelemente für jede Aktivitäteninstanz einzeln angefordert werden, führt zu einer Verzögerung bei der Aktivierung dieser Aktivitäteninstanz. Da die Aktivität zu diesem Zeitpunkt noch nicht in die Arbeitslisten der Benutzer eingetragen wurde, bemerken diese die Verzögerung nicht. Die Verzögerungen erhöhen zwar die Ausführungszeit des Workflows, sie fallen aber im Vergleich zur Gesamtausführungszeit (Tage oder Wochen) nicht ins Gewicht. Ist ein Workflow-Server, von dem ein Datenelement angefordert werden soll, für längere Zeit nicht erreichbar, so kann dies die Verfügbarkeit des WfMS verschlechtern. Um dem entgegenzuwirken, sollte das Datenelement in diesem Fall (falls möglich) von einem anderen Server aus der Menge $Sources$ angefordert werden. Da die von einer Aktivität benötigen Datenelemente erst bei ihrer Aktivierung angefordert werden, entsteht ein Problem, wenn disconnected Clients [AGK+96] verwendet werden. Diese verfügen über einen eigenen Workflow-Server und sollen evtl. mehrere Aktivitäten ausführen können, ohne mit anderen Workflow-Servern zu kommunizieren. Deshalb müssen die Datenelemente, die von den im Disconnected-Modus auszuführenden Aktivitäten benötigt werden, schon vor dem Verbindungsabbau angefordert werden. Dabei muss in Kauf genommen werden, dass auch Datenelemente für möglicherweise nicht ausgeführte Aktivitäten (z.B. Aktivität h aus Abb. 6) angefordert werden müssen. Dieser Nachteil tritt aber auch bei klassischen Verfahren zur Migration von Datenelementen auf, dann aber nicht nur im Disconnected-Modus.

6 Effektivität der vorgestellten Verfahren

In [BD99] wurde anhand von Simulationen bereits gezeigt, dass eine verteilte Workflow-Ausführung bzgl. der Kommunikationskosten in vielen Fällen wesentlich günstiger ist als eine zentrale Workflow-Steuerung mit zentraler Datenhal-

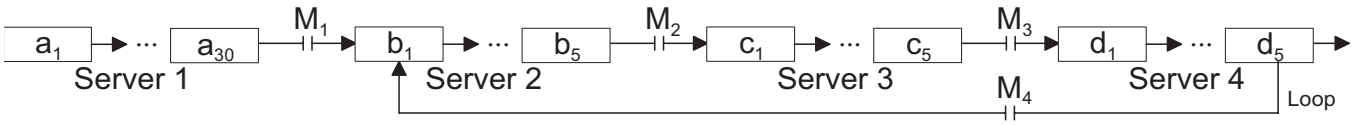

Abb. 7. Dem Zahlenbeispiel zugrunde liegender Workflow.

tung. Das bedeutet, dass das Konzept der Partitionierung von Workflow-Schemata zur Modellierungszeit und die damit verbundenen Migrationen zur Ausführungszeit generell notwendig sind.

Es bleibt zu evaluieren, ob die im vorliegenden Beitrag vorgestellten Verfahren geeignet sind, das bei Migrationen zu übertragende Datenvolumen gegenüber einer konventionellen Realisierung zu reduzieren. Diese Evaluation wollen wir im Folgenden anhand eines Zahlenbeispiels vornehmen. Hierzu wird der in der Praxis sehr relevante Fall eines Workflows mit Schleifen betrachtet. Schleifen werden z.B. zur Realisierung zahlreicher Krankenhausprozesse [DRK00] benötigt, etwa für Chemotherapie-Behandlungsprozesse, die üblicherweise mehrere (identische) Behandlungszyklen umfassen. Ein solcher Zyklus kann seinerseits mehrere Organisationseinheiten involvieren, mehrere Tage dauern und mehrere Dutzend Aktivitäten umfassen [Sch96]. Aus Gründen der besseren Lesbarkeit verzichten wir im Folgenden auf die Darstellung eines solchen komplexen Prozesses und wählen stattdessen eine abstrakte Darstellung. Bei Prozessen ohne Iterationen existiert in der Regel ein kleineres Optimierungspotential, dessen konkrete Größe davon abhängt, wieviele und welche Partitionen einer Workflow-Instanz vom selben Workflow-Server kontrolliert werden.

Im Folgenden sei der Workflow aus Abb. 7 gegeben. Wir betrachten die Migration M_2 vom Server 2 zu der vom Server 3 kontrollierten Partition, die aus den fünf Aktivitäten c_1 bis c_5 besteht. Diese Aktivitäten befinden sich innerhalb einer Schleife, die insgesamt 15 Aktivitäten umfasst. Vor der Schleife befinden sich die 30 Aktivitäten $a_1 \ldots a_{30}$.

Zuerst wollen wir die für die *Übertragung von Workflow-Kontrolldaten* (siehe Abschnitt 4.2) entstehenden Kosten einer konventionellen mit denen einer optimierten Migration vergleichen. Dabei wird pessimistischerweise angenommen, dass der Workflow bei der ersten Iteration der Schleife zum ersten Mal vom Server 3 kontrolliert wird, so dass bei der entsprechenden Migration keine Optimierungen möglich sind. Bei jeder weiteren Iteration sind dem Server 3 die Ablaufhistorieneinträge der Aktivitäten $a_1 \ldots a_{30}$ schon bekannt. Außerdem kennt er bereits die Einträge zu den Aktivitäten $b_1 \ldots b_5$ und $c_1 \ldots c_5$ der direkten Vorgängeriteration. Für alle davor liegenden Schleifendurchläufe sind ihm die Einträge zu den Aktivitäten $b_1 \ldots b_5$, $c_1 \ldots c_5$ und $d_1 \ldots d_5$ bekannt. Nehmen wir nun an, dass (durchschnittlich) n = 10 Iterationen der Schleife ausgeführt werden.

Bei einer konventionellen Realisierung der Migration entsteht der folgende Übertragungaufwand:
1. Iteration: 30 Einträge[14] (für $a_1 \ldots a_{30}$) + 5 Einträge (für $b_1 \ldots b_5$) = 35 Einträge
Für jede weitere Iteration kommen bei der entsprechenden Migration 15 zusätzliche Einträge für die zuletzt ausgeführten Instanzen der Aktivitäten $c_1 \ldots c_5$, $d_1 \ldots d_5$ und $b_1 \ldots b_5$ hinzu. Bei der Iteration i müssen also $35+15\cdot(i-1)$ Einträge migriert werden. Damit ergibt sich als die Anzahl der bei der Migration M_2 insgesamt (für alle n Iterationen) zu übertragenden Historieneinträge:

$$\sum_{i=1}^{n} 35 + 15 \cdot (i-1) = 35 \cdot n + 15 \cdot \frac{(n-1) \cdot n}{2}$$

Werden die vorgestellten Optimierungsverfahren verwendet, so müssen bei der 1. Iteration ebenfalls 35 Historieneinträge übermittelt werden. Bei jeder weiteren Iteration werden aber nur die 10 neu hinzugekommenen Einträge für die Aktivitäten $d_1 \ldots d_5$ und $b_1 \ldots b_5$ migriert. Damit ergibt sich als Gesamtzahl der zu übertragenden Einträge: $35 + 10 \cdot (n-1)$
Zusätzlich muss bei jeder Migration die ID der letzten bekannten Aktivitäteninstanz (c_5) übertragen werden, was die Übertragung von insgesamt n IDs erfordert.

Da wir n = 10 Iterationen annehmen, müssen aufgrund der Optimierungen lediglich 125 Historieneinträge und 10 IDs anstelle von 1025 Einträgen übermittelt werden. Nehmen wir weiter an, dass für einen Ablaufhistorieneintrag 200 Byte erforderlich sind. Dies ist eher niedrig angesetzt, da ein solcher Eintrag außer der ID der zugehörigen Aktivitäteninstanz und WfMS-intern genutzter Information [RD98] noch weitere Daten, wie den Bearbeiter, die Start- und Endezeit und den Workflow-Server der Aktivität enthält. Nehmen wir ferner an, dass die ID eines solchen Eintrags 8 Byte lang ist. Dann müssen aufgrund der Optimierungen nur 25,08 kB anstelle von 205 kB übertragen werden. In diesem Beispiel ermöglichen die vorgestellten Verfahren also eine Reduktion der zu übertragenden Kontrolldaten um 88%. Die entsprechende Einsparung entsteht für jede einzelne Workflow-Instanz alleine bzgl. der Migrationen M_2 vom Server 2 zum Server 3.

Die *Migrationen von kleinen Datenelementen* (siehe Abschnitt 5.2) ist an die Übertragung von Historieneinträgen gekoppelt. Deshalb ergibt sich für sie dieselbe prozentuale Lastreduktion. Die Übertragung der IDs der letzten bekannten Historieneinträge ist bei dem Verfahren jedoch nicht nochmals notwendig, so dass sich sogar ein geringfügig besseres Ergebnis ergibt.

Zusätzlich entsteht eine Lastreduktion durch das in Abschnitt 5.3 vorgestellte Verfahren zur *Übertragung von großen Datenelementen* (z.B. ein Satz von Computertomographie-Bildern in einem Krankenhausprozess oder eine technische Zeichnung in einem verteilten Entwicklungsprozess). Für die nun durchgeführte Analyse nehmen wir an, dass in dem Gesamtprozess fünf große Datenelemente mit einer Größe von jeweils 5 MB verwendet werden. Von diesen werden drei Datenelemente für die Ausführung der hier betrachteten Aktivitäten $c_1 \ldots c_5$ benötigt. Bei Verwendung des vorgestellten Verfahrens werden in der ersten Iteration

[14] Bei der hier durchgeführten Analyse werden der Einfachheit halber die Einträge für das Starten und das Beenden einer Aktivitäteninstanz wie ein einziger (größerer) Eintrag behandelt.

nur drei Datenelemente angefordert, anstatt dass alle fünf übertragen werden. Dadurch ergibt sich eine Reduktion der Datenmenge von 25 MB auf 15 MB.[15] Nehmen wir nun an, dass in jeder Iteration zwei der drei von den Aktivitäten $c_1 \ldots c_5$ benötigten Datenelemente durch die Aktivitäten $b_1 \ldots b_5$ bzw. $d_1 \ldots d_5$ verändert werden. Dann müssen bei jeder weiteren Iteration die diesen beiden Datenelementen entsprechenden 10 MB übertragen werden. Eine konventionelle Realisierung der Migrationen würde auch bei jeder weiteren Iteration die Übertragung von 25 MB erforderlich machen. Insgesamt müssen aufgrund der optimierten Übertragung von großen Datenelementen also nur 105 MB anstelle von 250 MB transferiert werden. Dies entspricht einer Reduktion um 58%. Damit wurde gezeigt, dass die in den Abschnitten 4 und 5 vorgestellten Verfahren geeignet sind, die bei Migrationen übertragene Datenmenge signifikant zu reduzieren.

7 Diskussion

In dieser Arbeit verzichten wir aus Platzgründen auf einen ausführlichen Überblick über verteilte WfMS (siehe hierzu [BD99]). Stattdessen diskutieren wir, wie bzw. welche Workflow-Daten bei den anderen in der Workflow-Literatur diskutierten Verteilungsansätzen übertragen werden und welche Auswirkungen dies auf die entstehenden Migrationskosten hat. Auf zentrale WfMS (z.B. Panta Rhei [EG96], WASA [VW97], [DHL91]) und verteilte Ansätze, bei denen eine Workflow-Instanz stets nur von einem einzigen Workflow-Server kontrolliert wird (z.B. Exotica/Cluster [AKA+94], MOBILE [Jab97]), gehen wir nicht weiter ein, da hier keine Datenübertragung zwischen den Workflow-Servern stattfindet.

In der Literatur werden verteilte WfMS beschrieben, bei denen die Laufzeitdaten einer Workflow-Instanz bei Migrationen vollständig übertragen werden: So wird bei CodAlf und BPAFrame (siehe [SM96]) eine Workflow-Instanz als Objekt repräsentiert (inkl. internem Zustand, Datenelementen und der Definition des zugehörigen Workflow-Schemas), welches zwischen den Workflow-Servern migriert. Auch bei INCAS [BMR96] wurde ein solcher Objektmigrationsansatz gewählt, allerdings erfolgt die Workflow-Definition hier durch Regeln. Da zusätzlich zu den schon genannten Daten auch noch die alten Versionen der Datenelemente (vgl. Datenhistorie aus Abschnitt 5.1) im migrierten Objekt enthalten sind, erhöht sich die bei Migrationen zu übertragende Datenmenge weiter.

Es gibt aber auch Ansätze, die nicht nur bei Migrationen Daten übertragen: Bei Exotica/FMQM [AMG+95] werden die von einer Aktivitäteninstanz geschriebenen Datenelemente an diejenigen Systemkomponenten versendet, die Aktivitäteninstanzen kontrollieren, welche diese Datenelemente lesen (vgl. „Versenden von Datenelementen“ in Abschnitt 5.3.1). Die verteilte Workflow-Ausführung in MENTOR [WWK+97] basiert auf der Partitionierung von State- und Activitycharts. Um eine zum zentralen Fall äquivalente verteilte Ausführung garantieren zu können, müssen nach der Beendigung von parallel ausgeführten Aktivitäten Synchronisationsnachrichten und Datenelemente ausgetauscht werden. Da dies einen großen Kommunikationsaufwand erfordert, werden in [MWW+98] Verfahren vorgestellt, mit denen dieser Aufwand reduziert werden kann. Es wird also nicht das Ziel verfolgt, den Kommunikationsaufwand bei Migrationen zu reduzieren, sondern der Aufwand für die Synchronisation der Workflow-Server bei parallel ausgeführten Aktivitäten wird begrenzt. Eine solche Synchronisation der Workflow-Abarbeitung ist in ADEPT nicht notwendig, da die Bearbeitung paralleler Zweige unabhängig voneinander fortschreiten kann. In [GWWK00] wird, basierend auf warteschlangentheoretischen Untersuchungen, analysiert, welche Konfiguration für ein verteiltes WfMS am günstigsten ist. Hierbei werden auch Verfügbarkeitsfragestellungen einbezogen. Allerdings wird der Aspekt der Kommunikationskosten nicht betrachtet.

Einige Ansätze verwenden keine Partitionierung und Migrationen im eigentlichen Sinn, sondern starten Subprozesse auf einem entfernten Workflow-Server. Bei einer Erweiterung von MOBILE [SNS99] wird zur Ausführungszeit der Workflow-Instanzen entschieden, welcher Workflow-Server einen Sub-Workflow kontrollieren soll. Auch WIDE [CGP+96] erreicht Verteilung durch die entfernte Ausführung von Subprozessen. Diese Vorgehensweise hat den Vorteil, dass an einen Sub-Workflow nur die von ihm potenziell benötigten Daten übergeben werden müssen, anstatt dass alle Daten der Workflow-Instanz migriert werden. Dies führt zu einem ähnlichen Effekt wie bei den in dieser Arbeit vorgestellten Optimierungen. Eine ausschließlich auf Subprozessen basierende Verteilung wurde für ADEPT allerdings nicht gewählt, weil die Workflow-Kontrolle nach Beendigung jedes Sub-Workflows zum Server des Super-Workflows zurückkehren würde (auch wenn dies eigentlich nicht erwünscht ist) und die Veränderung einer Verteilung aufwendiger wäre (weil dazu eine andere Aufteilung in Subprozesse benötigt wird).

Bei WIDE [CGP+96] werden Datenelemente nicht direkt an andere Workflow-Server übergeben, sondern nur Referenzen auf sie. Diese Vorgehensweise ist häufig bei solchen Systemen zu finden, die wie WIDE auf einer objektorientierten Systeminfrastruktur (z.B. CORBA) basieren, da diese den ortstransparenten Zugriff auf die eigentlichen Datenelemente erlaubt. Bei METEOR$_2$ [DKM+97] steuert ein sog. „Taskmanager“ die Ausführung eines bestimmten Aktivitätentyps und signalisiert den Taskmanagern der Nachfolgeraktivitäten deren Beendigung. Zugriffe auf Datenelemente erfolgen ortstransparent durch CORBA. Dasselbe gilt für die Systeme METUFlow [GAC+97], MOKASSIN [GJS+99] und WASA$_2$ [Wes99, WHKS98], bei denen eine Aktivitäteninstanz jeweils durch ein CORBA-Objekt repräsentiert wird. Da bei all diesen Ansätzen lediglich Referenzen auf die Datenelemente migriert werden, wird zwar die Belastung des WfMS reduziert, aber nicht die des Kommunikationssystems. Ein großes Datenelement wird dann nicht nur einmal (durch eine Migration) in das Teilnetz transportiert, in dem es benötigt wird, sondern muss bei jedem einzelnen Zugriff eines Aktivitätenprogramms übertragen werden. Dies führt, insbesondere

[15] Die für die Optimierung zusätzlich notwendige Übertragung der drei IDs der Datenelemente fällt bei diesen Datenmengen nicht ins Gewicht. Ein zusätzlicher Vorteil ergibt sich bei der Optimierung jedoch dadurch, dass die Datenelemente von demjenigen Server angefordert werden können, zu dem die kostengünstigste Kommunikationsverbindung besteht.

in einer weiträumig verteilten Umgebung, zu einer höheren Belastung des Kommunikationssystems.

Eine Fragestellung, die mit der Migration von Zustandsinformation in einem verteilten WfMS verwandt ist, wird von CoAct [WK96] untersucht: Mehrere Personen verändern verteilt und parallel dasselbe Dokument, so dass mehrere gültige Änderungshistorien entstehen. Diese werden (wie die Ablaufhistorien in ADEPT) wieder zu einer einzigen Historie zusammengeführt (sog. *History Merge*). Ebenso wie bei ADEPT wird dabei die jeweilige Semantik der Historieneinträge ausgenutzt.

Zusammenfassend lässt sich feststellen, dass in der Literatur zahlreiche Ansätze für verteiltes Workflow-Management vorgestellt werden. Allerdings werden bei keinem von ihnen die bei Migrationen anfallenden Kommunikationskosten – wie bei den in diesem Beitrag vorgestellten Verfahren – minimiert. Die Ansätze ignorieren weitgehend die in einem verteilten WfMS anfallenden hohen Kommunikationskosten. Manche der Ansätze treffen aber Entwurfsentscheidungen, die auch Einfluss auf die entstehenden Kommunikationskosten haben: (1) Bei Migrationen werden nur Referenzen auf Datenelemente übergeben. Dies führt zu den in diesem Abschnitt schon diskutierten Nachteilen. (2) Bei Migrationen wird keine explizite Zustandsinformation übergeben (es werden lediglich die Nachfolgeraktivitäten über die Beendigung der Aktivitäteninstanz informiert). Dies führt dazu, dass keine Information über beendete Aktivitäten verfügbar ist, was die Realisierung fortschrittlicher Workflow-Konzepte, wie abhängige Bearbeiterzuordnungen und die Überwachung von Zeitbedingungen, beeinträchtigt.

8 Zusammenfassung

Durch die Verwendung von WfMS kann die Entwicklung von unternehmensweiten und -übergreifenden prozessorientierten Anwendungssystemen erheblich erleichtert werden, da der Programmcode einzelner Anwendungsfunktionen von der Prozesslogik getrennt wird. Dadurch wird die Entwicklung von solchen Anwendungssystemen mit vertretbarem Aufwand überhaupt erst möglich. Bei zahlreichen Anwendungen muss die Workflow-Steuerung verteilt realisiert werden, weil ein zentraler Workflow-Server überlastet wäre. Abgesehen davon lassen sich durch eine geeignete Verteilung der Workflow-Steuerung die Kommunikationskosten reduzieren, indem ein Workflow-Server „nahe“ bei den Bearbeitern der aktuellen Aktivität gewählt wird [BD97, BD00]. Allerdings entstehen durch die bei der verteilten Workflow-Ausführung notwendigen Migrationen gewisse Kommunikationskosten, die reduziert werden müssen, um ein effizientes verteiltes Workflow-Management zu ermöglichen.

In diesem Beitrag wurden Verfahren vorgestellt, mit denen das bei Migrationen zu übertragende Datenvolumen drastisch reduziert werden kann (vgl. Abschnitt 6). Mit diesen Verfahren wird sowohl die redundante Übertragung interner Zustandsinformation als auch der Parameterdaten von Aktivitätenprogrammen verhindert. Kleine Datenelemente werden anders behandelt als große, welche stets von demjenigen Workflow-Server bezogen werden, zu dem die beste Kommunikationsverbindung besteht. Die Verfahren sind für statische und für variable Serverzuordnungen gleichermaßen geeignet und berücksichtigen alle Konstrukte des ADEPT-Modells (z.B. bedingte und parallele Verzweigungen, Schleifen). Sie sollten deshalb auch auf andere Modelle übertragbar sein. Insbesondere im Zusammenhang mit Schleifen kann die Belastung des dem WfMS zugrunde liegenden Kommunikationssystems drastisch reduziert werden, weil mehrere Instanzen einer Aktivität vom selben Server kontrolliert werden, so dass bereits viel Information lokal vorhanden ist. Allerdings haben die vorgestellten Optimierungen auch ihren Preis: Die Anwendung der vorgestellten Verfahren erfordert zusätzlichen Rechenaufwand. Sie sollten deshalb nur in Umgebungen eingesetzt werden, in denen dieser Aufwand durch die Entlastung des Kommunikationssystems gerechtfertigt wird. Für weiträumig verteilte Anwendungen ist dies sehr häufig der Fall. Die vorgestellten Verfahren reduzieren die zu übertragende Datenmenge durch die Nutzung von Wissen über schon transferierte Daten. Dadurch lässt sich in einigen Fällen eine deutlich stärkere Reduktion der Datenmenge erreichen als mit klassischen Komprimierungsverfahren. Selbstverständlich kann mit diesen die zu übertragende Datenmenge aber noch weiter reduziert werden.

Verschiedene weitere Aspekte, wie das zugrunde gelegte Fehlermodell, die Art der Kommunikation oder das Transaktionsmanagement, haben ebenfalls Einfluss auf die bei der verteilten Workflow-Steuerung entstehenden Kommunikationskosten. Da diese Aspekte aber zu den hier betrachteten Fragestellungen weitgehend orthogonal sind, gehen wir nicht näher auf sie ein. Kommunikationskosten lassen sich auch reduzieren, indem Workflow-Schemata (bzgl. ihrer Graphstruktur oder ihrer Bearbeiterzuordnungen) verändert werden, oder indem die Position einzelner Benutzer im Kommunikationsnetzwerk verändert wird. Solche Modifikationen sind normalerweise aber nicht akzeptabel, da die Veränderung von (hoffentlich) optimierten Prozessen in der Regel negative Auswirkungen auf wichtige Größen wie Prozesskosten oder Durchlaufzeiten hat. Es ist wesentlich günstiger, die vorgestellten Optimierungen zu verwenden, da diese keine für den Benutzer sichtbaren Auswirkungen haben.

Die in diesem Beitrag beschriebenen Verfahren wurden in dem Prototypen ADEPT$_{workflow}$ [HRB+00] implementiert. ADEPT$_{workflow}$ wurde auf der CeBIT'2000, der EDBT'2000 und der BIS'2000 demonstriert. Durch die Implementierung konnte die Umsetzbarkeit der Verfahren gezeigt und ihr Zusammenspiel mit andern Konzepten (z.B. dynamischen Workflow-Änderungen) studiert werden. Anhand der tatsächlich bei Migrationen einer Workflow-Instanz übertragenen Datenmenge ist außerdem zu erkennen, dass die Belastung des Kommunikationssystems deutlich reduziert werden kann.

Danksagung. Wir danken Clemens Hensinger und Jochen Zeitler für die anregenden Diskussionen.

References

[AGK+96] G. Alonso, R. Günthör, M. Kamath, D. Agrawal, A. El Abbadi, C. Mohan: Exotica/FMDC: A Workflow Management System for Mobile and Disconnected Clients. Distributed and Parallel Databases, 4(3): 229–247 (1996)

[AKA+94] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör, 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, M. Kamath: Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management. In: Proc. IFIP Working Conf. on Information Systems for Decentralized Organisations, Trondheim, August 1995

[BD97] T. Bauer, P. Dadam: A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration. In: Proc. 2nd IFCIS Conf. on Cooperative Information Systems, pp. 99–108, Kiawah Island, SC, Juni 1997

[BD99] T. Bauer, P. Dadam: Verteilungsmodelle für Workflow-Management-Systeme – Klassifikation und Simulation. Informatik Forschung und Entwicklung, 14(4): 203–217 (1999)

[BD00] T. Bauer, P. Dadam: Efficient Distributed Workflow Management Based on Variable Server Assignments. In: Proc. 12th Conf. on Advanced Information Systems Engineering, pp. 94–109, Stockholm, Juni 2000

[BF99] E. Bertina, E. Ferrari: The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. ACM Trans. Inform. Syst. Sec., 2(1): 65–104 (1999)

[BMR96] D. Barbará, S. Mehrotra, 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, G. Sánchez: WIDE: Workflow Model and Architecture. CTIT Technical Report 96–19, University of Twente, 1996

[DHL91] U. Dayal, M. Hsu, R. Ladin: A Transactional Model for Long-Running Activities. In: Proc. 17th Int. Conf. on Very Large Data Bases, pp. 113–122, Barcelona, September 1991

[DKM+97] S. Das, K. Kochut, J. Miller, A. Sheth, 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

[DRK00] P. Dadam, M. Reichert, K. Kuhn: Clinical Workflows – The Killer Application for Process-oriented Information Systems? In: Proc. 4th Int. Conf. on Business Information Systems, pp. 36–59, Posen, April 2000

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

[GAC+97] E. Gokkoca, M. Altinel, I. Cingil, E.N. Tatbul, P. Koksal, A. Dogac: Design and Implementation of a Distributed Workflow Enactment Service. In: Proc. 2nd IFCIS Conf. on Cooperative Information Systems, pp. 89–98, Kiawah Island, SC, Juni 1997

[GJS+99] B. Gronemann, G. Joeris, S. Scheil, M. Steinfort, H. Wache: Supporting Cross-Organizational Engineering Processes by Distributed Collaborative Workflow Management - The MOKASSIN Approach. In: Proc. 2nd Symposium on Concurrent Multidisciplinary Engineering, 3rd Int. Conf. on Global Engineering Networking, Bremen, September 1999

[Gri97] M. Grimm: ADEPT$_{time}$: Temporale Aspekte in flexiblen Workflow-Management-Systemen. Diplomarbeit, Universität Ulm, Fakultät für Informatik, 1997

[GWWK00] M. Gillmann, J. Weissenfels, G. Weikum, A. Kraiss: Performance and Availability Assessment for the Configuration of Distributed Workflow Management Systems. In: Proc. 7th Int. Conf. on Extending Database Technology, pp. 183–201, Konstanz, März 2000

[HRB+00] C. Hensinger, M. Reichert, T. Bauer, T. Strzeletz, P. Dadam: ADEPT$_{workflow}$ - Advanced Workflow Technology for the Efficient Support of Adaptive, Enterprise-wide Processes. In: 7th Int. Conf. on Extending Database Technology, Software Demonstrations Track, pp. 29–30, Konstanz, März 2000

[Jab97] S. Jablonski: Architektur von Workflow-Management-Systemen. Informatik Forsch. Entw. Themenheft Workflow-Management, 12(2): 72–81 (1997)

[KAGM96] M. Kamath, G. Alonso, R. Günthör, C. Mohan: Providing High Availability in Very Large Workflow Management Systems. In: Proc. 5th Int. Conf. on Extending Database Technology, pp. 427–442, Avignon, März 1996

[KDB98] M. Klein, C. Dellaroca, A. Bernstein (Herausgeber): Workshop Towards Adaptive Workflow Systems, Conf. on Computer-Supported Cooperative Work, Seattle, WA, November 1998

[Kub98] M. Kubicek: Organisatorische Aspekte in flexiblen Workflow-Management-Systemen. Diplomarbeit, Universität Ulm, Fakultät für Informatik, 1998

[Ley97] F. Leymann: Transaktionsunterstützung für Workflows. Informatik Forsch. Entw. Themenheft Workflow-Management, 12(2): 82–90 (1997)

[LR00] F. Leymann, D. Roller: Production Workflow – Concepts and Techniques. Englewood Cliffs, NJ: Prentice Hall, 2000

[MO99] O. Marjanovic, M. Orlowska: On Modeling and Verification of Temporal Constraints in Production Workflows. Knowledge and Information Systems, 1(2): 157–192 (1999)

[MWW+98] P. Muth, D. Wodtke, J. Weißenfels, A. Kotz-Dittrich, G. Weikum: From Centralized Workflow Specification to Distributed Workflow Execution. J. Intell. Inform. Syst., Special Issue on Workflow Management Systems, 10(2): 159–184 (1998)

[RD98] M. Reichert, P. Dadam: ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. J. Intell. Inform. Syst., Special Issue on Workflow Management Systems, 10(2): 93–129 (1998)

[RS95] A. Reuter, F. Schwenkreis: ConTracts – A Low-Level Mechanism for Building General-Purpose Workflow Management-Systems. IEEE Computer Society, Bulletin of the Technical Committee on Data Engineering, 18(1): 4–10 (1995)

[Sch96] B. Schultheiß: Prozeßreengineering in klinischen Anwendungsumgebungen – Beispiele, Vorgehensmodelle, Werkzeuge. Diplomarbeit, Universität Ulm, Fakultät für Informatik, 1996

[SK97] A. Sheth, K.J. Kochut: Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems. In: Proc. NATO Advanced Study Institute on Workflow Management Systems and Interoperability, pp. 12–21, Istanbul, August 1997

[SM96] A. Schill, C. Mittasch: Workflow Management Systems on Top of OSF DCE and OMG CORBA. Distrib. Syst. Eng., 3(4): 250–262 (1996)

[SNS99] H. Schuster, J. Neeb, R. Schamburger: A Configuration Management Approach for Large Workflow Management Systems. In: Proc. Int. Joint Conf. on Work Activities Coordination and Collaboration, pp. 177–186, San Francisco, Februar 1999

[VW97] G. Vossen, M. Weske: The WASA Approach to Workflow Management for Scientific Applications. In: Proc. NATO Advanced Study Institute on Workflow Management Systems and Interoperability, pp. 145–164, Istanbul, August 1997

[Wes99] M. Weske: Workflow Management Through Distributed and Persistent CORBA Workflow Objects. In: Proc. 11th Int. Conf. on Advanced Information Systems Engineering, Heidelberg, 1999

[WHKS98] M. Weske, J. Hündling, D. Kuropka, H. Schuschel: Objektorientierter Entwurf eines flexiblen Workflow-Management-Systems. Informatik Forsch. Entw., 13(4): 179–195 (1998)

[WK96] J. Wäsch, W. Klas: History Merging as a Mechanism for Concurrency Control in Cooperative Environments. In: Proc. 6th Int. Workshop on Research Issues in Data Engineering – Interoperability of Nontraditional Database Systems, pp. 76–85, New Orleans, Februar 1996

[WMC99] Workflow Management Coalition: Terminology & Glossary, Document Number WFMC-TC-1011, Document Status – Issue 3.0, Februar 1999

[WWK+97] G. Weikum, D. Wodtke, A. Kotz-Dittrich, P. Muth, J. Weißenfels: Spezifikation, Verifikation und verteilte Ausführung von Workflows in MENTOR. Informatik Forsch. Entw., Themenheft Workflow-Management, 12(2): 61–71 (1997)

[Zei99] J. Zeitler: Integration von Verteilungskonzepten in ein adaptives Workflow-Management-System. Diplomarbeit, Universität Ulm, Fakultät für Informatik, 1999

Thomas Bauer studierte Informatik in Ulm. Seit seinem Diplom im Juli 1995 ist er wissenschaftlicher Mitarbeiter der Universität Ulm, Abteilung Datenbanken und Informationssysteme, wo er im Februar 2001 seine Promotion abschloss. Seine Interessen liegen in den Bereichen Workflow-Management und Skalierbarkeit.

Peter Dadam ist seit 1990 Professor für Informatik an der Universität Ulm und Leiter der Abteilung Datenbanken und Informationssysteme. Davor war er Leiter der Abteilung Advanced Information Management am Wissenschaftlichen Zentrum der IBM in Heidelberg. Seine aktuellen Arbeitsgebiete sind: Verteilte, kooperative Informationssysteme, Workflow-Management, Datenbanktechnologie und deren Anwendungen in anspruchsvollen Anwendungsgebieten.

Manfred Reichert studierte Wirtschaftsmathematik in Ulm. Er ist wissenschaftlicher Mitarbeiter an der Abteilung Datenbanken und Informationssysteme der Universität Ulm, wo er im Juli 2000 seine Promotion abschloss. Seine Interessen liegen in den Bereichen unternehmensweites Workflow-Management und adaptive Workflow-Systeme.