Ulmer Informatik-Berichte, Nr. 98-02, Januar 1998

Architekturen für skalierbare Workflow-Management-Systeme – Klassifikation und Analyse

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

Zusammenfassung

In unternehmensweiten Workflow-Management-Systemen (WfMS) kann die von der WF-Engine zu bewältigende Last sehr groß werden. Außerdem werden hohe Anforderungen an die Verfügbarkeit eines solchen Systems gestellt. Deshalb wurden in der Literatur zahlreiche Architekturen für skalierbare WfMSe vorgeschlagen, die auf unterschiedlichen Verteilungen der WF-Engine basieren. Diese Architekturen werden in dem Beitrag verglichen und klassifiziert. Des weiteren wird die Belastung der einzelnen Systemkomponenten analysiert und die Ansätze werden auf Schwachstellen hin untersucht. Die gewonnenen Erkenntnisse werden verwendet, um eine Entscheidungshilfe für die Auswahl einer geeigneten Architektur für bestimmte Anwendungstypen zu geben.

1 Einleitung

WfMSe ermöglichen die computergesteuerte Ausführung von Geschäftsprozessen in einer verteilten Systemumgebung. Aufgrund von wirtschaftlichen Zwängen, wie der Verschlankung von Unternehmen, ist das Interesse an solchen Systemen in den letzten Jahren stetig gewachsen. Ihr entscheidender Vorteil ist, daß sie helfen, große Anwendungssysteme überschaubarer zu gestalten. Dazu wird der applikationsspezifische Code der verwendeten Anwendungen von der Ablauflogik („Prozeßlogik“) getrennt. Anstelle eines großen monolithischen Programmpakets erhält man nun unabhängige Anwendungsbausteine, deren Ablauflogik in einer separaten Prozeßdefinition festgelegt wird. Diese bestimmt die Ausführungsreihenfolge und -bedingungen der einzelnen Anwendungen. Für die Ablaufdefinition bedient man sich i.d.R. einer graphischen Beschreibungssprache. Die Teilschritte werden hierbei in der Regel als Knoten dargestellt, während die Ausführungsreihenfolge (der „Kontrollfluß“) und ggf. auch der Datenfluß mittels gerichteter Kanten modelliert wird. Das WfMS sorgt dafür, daß nur Aktivitäten ausgeführt werden können, die der Ablauflogik zufolge zur Ausführung anstehen. Diese Teilschritte werden in die Arbeitslisten der autorisierten Bearbeiter eingefügt. Welche Bearbeiter zur Bearbeitung bestimmter Aufgaben autorisiert sind, wird meist durch ein Rollenkonzept festgelegt. Dabei ist es durchaus möglich, daß mehrere Benutzer ermittelt werden, die für die Bearbeitung einer bestimmten Aktivität geeignet sind.

Die kommerziell verfügbaren WfMSe erfüllen die Anforderungen, die durch unternehmensweite Anwendungen gestellt werden, zur Zeit noch nicht in ausreichendem Maße [GHS95]. Ein wichtiger Aspekt hierbei ist die Skalierbarkeit des WfMSs. Warum sie durch die heutigen zentralen Architekturen stark eingeschränkt ist, soll das folgende Zahlenbeispiel verdeutlichen. Betrachten wir z.B. die Schadensbearbeitung in einer Versicherung mit 150 Sachbearbeitern, die durchschnittlich je 5 min = 300 sec für eine Aktivität benötigen. Damit muß das System im Mittel 0,5 Teilschritte pro Sekunde ausführen. Alle Dokumente eines Vorgangs sollen (z.B. via Einscannen) in digitaler Form vorliegen. Pro Teilschritt fallen damit leicht Ein-/Ausgabedaten im Umfang von 2 MB (oder mehr[1]) an[2]. Bei 0,5 Teilschritten/sec müssen also 1 MB/sec (in bits: 8 Mbps) alleine an Nutzdaten bewegt werden, wodurch ein 10 Mbps-Ethernet bereits deutlich überfordert wäre.

Das Aktualisieren der Arbeitslisten der Benutzer erzeugt ebenfalls Last. Da 0,5 Schritte/sec bearbeitet werden, muß 1 mal pro Sekunde ein Eintrag in die Arbeitslisten eingefügt oder herausgenommen werden. Da jeder der 150 Benutzer hiervon betroffen sein kann, muß der WF-Server[3] 150 Bearbeiterbeschreibungen pro Sekunde auflösen, was für einen einzelnen Server und die zugehörige WF-Datenbank eine hohe Last bedeutet. Nimmt man weiter an, daß ein Benutzer alle 10 sec eine neue Arbeitsliste erhält, die ca. 1 kB umfaßt, so ergibt sich in diesem Beispiel ein Kommunikationsaufwand von 15 kB/sec. Dieser ist zwar in diesem Beispiel gegenüber dem Parametertransfer vernachlässigbar, bei größeren Benutzerzahlen und sehr kurzen Aktualisierungszeiten kann aber auch durch den Transport der Arbeitslisten ein beträchtlicher Aufwand entstehen. — Schon dieses recht konservative Beispiel zeigt, daß bei einem zentralen WfMS sowohl der WF-Server, wie auch dessen Teilnetz, zu einem Engpaß werden können. Sollte das WfMS sogar über ein WAN (Wide Area Network) weiträumig verteilt sein, so ist die Verringerung der Kommunikationslast noch wichtiger.

Skalierbarkeit wird nun erreicht, indem man Systemkomponenten repliziert und die Last so auf sie verteilt, daß keine Komponente überlastet ist. Die Art der Verteilung von Systemkomponenten und ihre Zuständigkeiten werden in diesem Artikel als Architektur eines WfMSs bezeichnet. Dies ist der in diesem Beitrag primär betrachtete Aspekt, anhand dessen die verschiedenen Ansätze verglichen und die Belastung der einzelnen Komponenten analysiert werden. Ebenfalls Teil der Architektur sind Aspekte, wie die Vorgehensweise bei der Aktualisierung der Arbeitslisten oder die Art des Datenflusses. Diese haben zwar keinen so starken Einfluß auf die Verteilung der Last, von ihnen hängt aber ab, wieviel Last insgesamt erzeugt wird. Deshalb werden sie im folgenden ebenfalls diskutiert. Nicht betrachtet werden Architekturaspekte, wie die Aufteilung eines Programms in Module oder die Schnittstellen zwischen Komponenten.

[1] Hier sind auch wesentlich größere Datenmengen denkbar, z.B. für Multimedia-Daten. Außerdem kann auch eine wesentlich höhere Benuterzahl errreicht werden. In [KAGM96, SK97] werden sogar Anwendungen beschrieben, bei denen die Zahl der Benutzer eines WfMSs bis auf einige zehntausend anwachsen kann oder mehrere zehntausend WF-Instanzen gleichzeitig im System sein können.

[2] In vielen Fällen wird das WfMS nur eine Referenz auf die Datenobjekte an die Anwendungsschicht übergeben. Die Auflösung dieser Referenz löst dann erst den eigentlichen Datentransport aus. Da die Gesamtmenge an bewegten Daten die gleiche ist, wollen wir im folgenden vereinfachend annehmen, daß das WfMS direkt die Datenobjekte transportiert.

[3] Als WF-Server (oder kurz Server) werden die Komponenten bezeichnet, die die WF-Engine bilden, also die Ausführung der Prozesse steuern. Die Komponenten, mit denen die Benutzer arbeiten, stellen die Clients dar. Diese erledigen Aufgaben wie die Darstellung der Arbeitslisten oder das Starten von Anwendungen.

Außer der Skalierbarkeit gibt es zahlreiche weitere relevante Aspekte in unternehmensweiten WfMSen. Aus Platzgründen werden sie hier nur kurz erwähnt. Ein für ein Unternehmen oft lebenswichtiger Aspekt ist die Verfügbarkeit seines Informationssystems. Die auch in anderen Bereichen oft verwendete Lösung ist der Einsatz von Backup-Servern, die beim Ausfall des Primär-Servers die Ausführung der Prozesse übernehmen [KAGM96]. Um die Konsistenz der unternehmensweit verteilten Daten gewährleisten zu können, benötigt das WfMS ein Transaktionskonzept. Die Vorschläge dazu reichen von der Kapselung eines oder einiger weniger Aktivitäten eines Workflows zu einer Transaktion [Ley97] bis hin zur Verwendung von erweiterten Transaktionsmodellen [Elm92]. Bei der Ausführung von Prozessen kommt es in Ausnahmefällen vor, daß vom vormodellierten Schema abgewichen werden muß. Dazu können z.B. Aktivitäten zusätzlich eingefügt oder ausgelassen werden. Das WfMS muß bei einer solchen Operation prüfen, ob die verschiedenen Korrektheitsaspekte (z.B. Versorgung der Parameter nach einer dynamischen Änderung) noch gewährleistet sind [RD98].

Der Rest dieses Papiers ist wie folgt aufgebaut: In Kapitel 2 werden grundlegende Konzepte für die Architektur von WfMSen vorgestellt und analysiert. In die dabei entstehende Klassifikation werden in Kapitel 3 kommerzielle Systeme und Vorschläge aus dem Bereich der Forschung eingeordnet. Diese Systeme werden beschrieben und auf Schwachstellen hin untersucht. Kapitel 4 faßt die gewonnen Erkenntnisse zusammen und bietet eine Entscheidungshilfe bei der Auswahl einer geeigneten Architektur für verschiedene Anwendungsszenarien.

2 Grundlagen

In erster Linie wird Skalierbarkeit und hohe Verfügbarkeit von WfMSen durch die Replikation von Systemkomponenten und ihre Verteilung auf mehrere Rechner und Teilnetze erreicht. Die dafür möglichen Realisierungsvarianten werden im folgenden beschrieben und analysiert. In Kapitel 3 werden ihnen dann konkrete Systeme zugeordnet. Des weiteren gibt es noch einzelne grundlegende Gesichtspunkte von WfMSen, die Einfluß auf die erzeugte Gesamtlast haben. Auch sie werden in diesem Kapitel beschrieben.

Die im folgenden betrachteten Konzepte sollen bezüglich ihres Leistungsverhaltens miteinander vergleichen werden. Dazu wird berechnet, wie sich die Gesamtsystemlast $Last_{Strg}$ für die Steuerung der Prozeßschritte und die Last für das Aktualisieren der Arbeitslisten $Last_{AL}$ in den einzelnen Komponenten niederschlägt. In einem unternehmensweiten WfMS können sowohl $Last_{Strg}$ als auch $Last_{AL}$ sehr groß werden. Damit keine Komponente überlastet wird, muß ihre Belastung unter die jeweils vorgegebene Maximal-Kapazität der Komponente gedrückt werden können. Dies ist z.B. dann gegeben, wenn die Last in einer Komponente $\frac{Last_{Strg}+Last_{AL}}{x}$ beträgt, wobei x eine konfigurierbare Größe sein muß, wie etwa die Anzahl der Server-Replikate im System. Ist diese Bedingung durch eine Architektur erfüllt, so kann diese als skalierbar bezeichnet werden.

2.1 Verteilung der Systemkomponenten

In diesem Abschnitt werden die verschiedenen Ansätze für die Replikation und Verteilung von Systemkomponenten verglichen und die bei ihnen auftretende Last analysiert. Bei die-

ser Analyse wird die Belastung der Systemkomponenten und des Netzwerks (soweit darüber Aussagen möglich sind) untersucht. In Kapitel 3 werden die in Abbildung 1 dargestellten Realisierungsvarianten noch weiter verfeinert und ihnen dann konkrete Systeme zugeordnet.

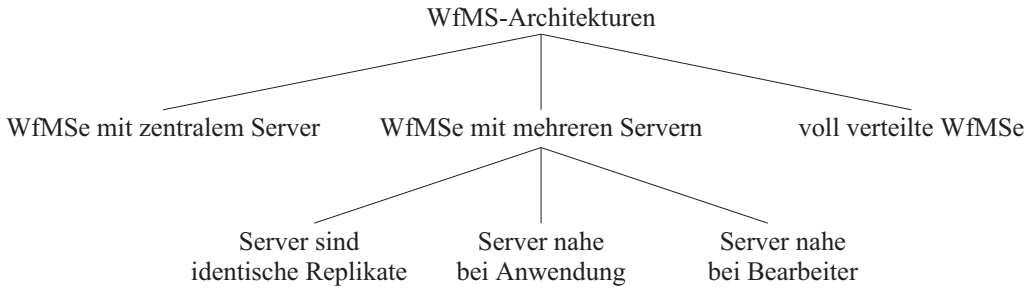

Abbildung 1 Realisierungsvarianten für die Verteilung von WfMSen

Die beiden Extreme, die existieren, sind Systeme mit nur einem zentralen Server und voll verteilte Systeme, bei denen jede Komponente Serveraufgaben wahrnimmt. Dazwischen liegen Systeme, die mehr als einen Server verwenden, aber trotzdem noch die Trennung zwischen Client und Server aufrechterhalten, so daß es weniger Server als Clients (Benutzerrechner) im System gibt.

2.1.1 WfMSe mit zentralem Server

Systeme der in Abbildung 2 dargestellten Kategorie verwenden eine zentrale Serverkomponente. Dies ist zumindest eine zentrale Datenbank mit nur einem DB-Server. Meist wird auch nur ein WF-Server verwendet, es gibt aber auch Systeme, bei denen sich mehrere WF-Server die zentrale Datenbank teilen.

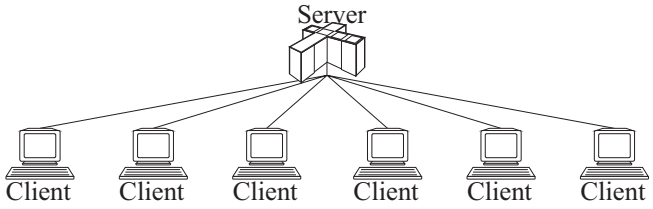

Abbildung 2 Architektur eines WfMSs mit zentralem Server

Die Leistungsfähigkeit eines solchen Systems ist i.w. durch den zentralen Server bestimmt bzw. beschränkt. Der Server hat die volle Last für die Ausführung der Aktivitäten und für die Aktualisierung der Arbeitslisten $Last_{Strg}+Last_{AL}$ zu tragen. Das Teilnetz des DB-Servers wird genauso stark belastet. Ein solches System ist deshalb nur für eine relativ kleine Anzahl von Benutzern (einige Dutzend) geeignet. Werden mehrere WF-Server verwendet (Anzahl: $\#Serv$), so ist ihre Last im besten Fall je $\frac{Last_{Strg}+Last_{AL}}{\#Serv}$, das zentrale DBMS bleibt aber weiterhin ein Flaschenhals.

Auch die Verwendung zentraler Hochleistungssysteme (Mehrprozessor-Systeme, Mehrrechner-DBMSe) löst die beschriebenen Probleme nicht. Bei einem lose gekoppelten System muß eine passende interne Verteilung gefunden werden, die den Synchronisationsaufwand klein hält. Dies führt zur selben Problemstellung, wie die Verteilung auf mehrere WF-Server. Wird ein eng gekoppeltes System verwendet, so beschränken die gemeinsamen Komponenten die Skalierbarkeit. Beide Varianten haben aber den Nachteil, daß das Teilnetz des (zentralen) Hochleistungssystems zum Flaschenhals wird und zu manchen Benutzern eine unnötig weit entfernte Kommunikation stattfinden muß.

2.1.2 WfMSe mit mehreren Servern

Systeme, bei denen der WF-Server mehrfach und auf verschiedene Maschinen verteilt vorhanden ist, fallen in diese Kategorie (Abbildung 3). Für die Verteilung der Server gibt es drei Ansätze. Durch die reine Replikation des WF-Servers ergeben sich identische Server, womit jeder Teilschritt von jedem beliebigen Server kontrolliert werden kann. Will man aus der Lokation des Servers Vorteile ziehen, so gibt es zwei Möglichkeiten. Man kann für die Steuerung eines Teilschrittes den Server verwenden, der sich auf dem Knoten der zugehörigen Anwendung befindet oder den, der sich bei den vorgesehenen Bearbeitern der Aktivität befindet. Diese Varianten werden im folgenden diskutiert.

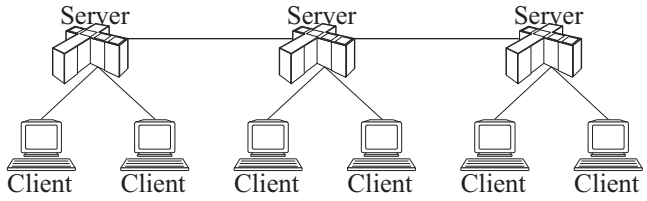

Abbildung 3 Architektur eines WfMSs mit mehreren Servern

2.1.2.1 Server sind identische Replikate

Beim ersten Ansatz werden identische Replikate der WF-Engine verwendet, die aus einer WF-Datenbank und einem WF-Server bestehen. Alle Prozeßtypen können von jedem dieser Cluster ausgeführt werden; eine Prozeßinstanz verbleibt in dem Cluster, in dem sie gestartet wurde. Die Cluster sind nicht an Geschäftsbereiche gebunden.

In der WF-Datenbank sind die replizierte Schemainformation, die Instanzen des Clusters und der diesen Cluster betreffende Teil der Arbeitslisten aller Benutzer gespeichert. Jeder Client muß eine Verbindung mit jedem Cluster aufbauen, da in jedem Cluster Aufträge für ihn vorhanden sein können.

Durch die geeignete Wahl des Clusters beim Prozeßstart (zufällig, zyklisch, lastabhängig) kann eine Lastbalancierung errreicht werden. Dadurch beträgt die Last für eine WF-Datenbank $\frac{Last_{Strg}}{\#Cluster}$, wenn $\#Cluster$ Cluster verwendet werden. Der Ausfall einer WF-Datenbank blockiert alle Instanzen des Clusters, die Instanzen anderer Cluster sind hiervon aber nicht

betroffen. Da die Benutzer normalerweise auch Instanzen anderer Cluster bearbeiten, können i.d.R. alle Benutzer, trotz des Ausfalls eines Clusters, zumindest eingeschränkt weiterarbeiten.

Für das Aktualisieren der Arbeitslisten stellen die Cluster einen Engpaß dar. Da in der WF-Datenbank Teile der Arbeitslisten aller Benutzer gespeichert sind, muß jeder Server alle Benutzer bedienen, woraus eine Last von $Last_{AL}$ pro Server resultiert. Die Last für das Aktualisieren der Arbeitslisten kann also mit diesem Ansatz nicht verteilt werden, so daß sehr große Benutzerzahlen nicht möglich sind.

Ein anderes Problem ergibt sich durch die Belastung des Netzwerks. Durch die Erhöhung der Anzahl der Teilnetze (auf die die Cluster verteilt werden) läßt sich zwar ihre Kommunikationslast senken, aber durch die zufällige Wahl des Clusters beim Instanzstart werden auch Cluster mit ungünstiger Lage gewählt. Dieses Problem ist bei weiträumig verteilten Systemen besonders gravierend. So kann die Bearbeitung eines Prozesses zwar geographisch beschränkt sein, aber durch die unglückliche Wahl des Clusters für eine Instanz ständige WAN-Kommunikation zur Steuerung notwendig werden.

2.1.2.2 Server nahe bei Anwendung

Durch die Verwendung mehrerer identischer Server (Cluster) läßt sich die Last für die Ausführung der Workflows auf diese Server verteilen, da die Instanzen gewöhnlich unabhängig voneinander laufen. Das Problem bei dieser reinen Replikation ist, daß die Cluster alle dieselben Benutzer haben, so daß jeder Server alle Benutzer bedienen muß. Nun ist die Situation in unternehmensweiten WfMSen aber meist nicht so, daß alle Benutzer dieselben Aufgaben erledigen, sondern gewisse Anwendungen oder Teilschritte sind einer bestimmten Benutzergruppe zugeordnet. Die dabei entstehenden Gruppen sind nicht unbedingt disjunkt, aber es ist möglich, ihre Existenz im Sinne einer Clusterbildung auszunutzen, um eine geeignete Lokation für die Server zu wählen. Da für jeden Server jetzt nur noch ein kleiner Teil der Benutzer relevant ist, muß er auch nicht mehr alle Arbeitslisten aktualisieren, was den Aufwand für diesen Vorgang erheblich reduziert. Bei der Verwendung von Polling (Abschnitt 2.2.1) ist allerdings Vorsicht geboten. Der Vorteil geht verloren, wenn die Clients weiterhin alle Server abfragen.

Bei diesem Konzept wird der Server dort plaziert, wo das Anwendungsprogramm der entsprechenden Aktivität läuft. Diese Variante wird vor allem von Systemen verwendet, die auf einer objektorientierten Infrastruktur basieren. Die Anwendung ist dabei als Objekt gekapselt, das an einem bestimmten Ort im verteilten System allokiert ist. Dieser Ort wird auch für den WF-Server des zugehörigen Teilschritts gewählt. Die Prozeßinstanz ist ein Objekt, das zu dem jeweiligen WF-Server migriert. Eine andere Möglichkeit ist, daß lediglich Objektreferenzen zwischen den Servern wandern und diese dann bei der Verwendung der zugehörigen Parameter entfernt auf die entsprechenden Objekte zugreifen.

Sind $\#Serv$ WF-Server vorhanden, so beträgt die Last für einen einzelnen Server-Knoten bei idealer Verteilung $\frac{Last_{Strg}+Last_{Migr}}{\#Serv}$. Die Migrationslast[4] $Last_{Migr}$ ist recht groß, da die

[4]Bei der Migration müssen alle von zukünftigen Prozeßschritten noch benötigten Instanzdaten zum neuen Server kopiert werden. Dies sind zum einen die Parameterdaten (die zusammen etliche MB umfassen können) und zum anderen die Zustandsinformation der Instanz (z.B. in Form einer Ablaufhistorie). Die konkrete Last hängt außer von der Größe dieser Datenmenge auch von der Häufigkeit der Migrationen ab.

Objekte alle Instanzdaten enthalten. Außerdem migriert das Instanzenobjekt bei jedem Teilschritt, selbst wenn er vom selben Bearbeiter wie der Vorgängerschritt ausgeführt wird, da gewöhnlich eine andere Anwendung verwendet wird. Falls entfernt auf die Daten zugegriffen wird, entfällt zwar $Last_{Migr}$, aber $Last_{Strg}$ ist wesentlich größer, da die Anwendungen mehrfach auf ein (evtl. weit) entferntes Objekt zugreifen. Ein generelles Problem des Ansatzes ist, daß die Verteilung der Last davon abhängt, wie häufig die einzelnen Anwendungen verwendet werden. Dies läßt sich nicht immer steuern. Außerdem hat die Anwendung einen fest vorgegebenen Ort, der (unter Umständen weit) vom Bearbeiter entfernt ist. Dadurch entstehen hohe Kommunikationskosten, da die gesamte Benutzerinteraktion über das Netzwerk abgewickelt werden muß.

Es ist schwierig, die Belastung der Server durch das Aktualisieren der Arbeitslisten zu bestimmen. Ein Server muß prinzipiell alle Benutzer bedienen, da die Verteilung nicht an Geschäftsbereichen (z.B. Abteilungen) orientiert ist, sondern an Anwendungen. Damit wäre die zu bewältigende Last $Last_{AL}$. Allerdings werden i.d.R. die meisten Benutzer nur wenige Anwendungen verwenden dürfen, da ihre Rolle sie auf bestimmte Tätigkeiten einschränkt. Die Anzahl dieser Anwendungen sei durch eine kleine Konstante k beschränkt, so daß für jeden Benutzer nur k Server relevant sind. Somit liegt die Last bei $\frac{Last_{AL} \cdot k}{\#Serv}$. Diese Annahme muß aber nicht gelten. Es sind Szenarien denkbar, in denen es zwar Abteilungen gibt, deren Mitarbeiter getrennte Prozesse bearbeiten, aber mit denselben Anwendungen. In einem solchen Fall wird jede Anwendung von jedem Benutzer verwendet, womit keine Skalierbarkeit mehr gegeben ist.

Dieser Ansatz geht von der Annahme aus, daß jede Anwendung einen eindeutig bestimmbaren Ort hat. Dies ist realistisch, wenn die Anwendung z.B. Datenbankzugriffe macht. Dann wird der Ort gewählt, an dem die Daten physisch gespeichert sind. Handelt es sich bei der Anwendung aber um ein überall verfügbares Programm, wie z.B. einen Editor, eine Textverarbeitung oder eine Eingabemaske, so wird durch die Anwendung kein Ort vorgegeben. Die Anwendung wird immer auf dem Rechner des (a priori unbekannten) Benutzers ausgeführt, der die Aktivität bearbeitet. Man könnte nun für den WF-Server jeden beliebigen Ort wählen, dies wäre aber bezüglich der Kommunikationskosten sehr ungünstig. Der nächste Ansatz löst dieses Problem.

2.1.2.3 Server nahe bei Bearbeiter

Beim diesem Ansatz wird ein WF-Server verwendet, der nahe bei den Bearbeitern liegt. Dies ist möglich, da angenommen werden kann, daß die meisten Aktivitäten von Benutzern ausgeführt werden, die alle derselben Organisationseinheit angehören. Damit ist deren Geschäftsbereich[5] eine gute Lokation für den WF-Server.

Die Konzepte dieser Kategorie können danach unterschieden werden, ob eine Prozeßinstanz den WF-Server wechseln kann, also eine Migration möglich ist, oder nicht. Eine solche Migration ist sinnvoll, wenn es Prozesse gibt, die nacheinander in verschiedenen Organisationseinheiten bearbeitet werden. Man denke an einen Prozeß, der zuerst die Verkaufsabteilung

[5]Ein Geschäftsbereich kann jede Art von organisatorischer Einheit (z.B. Abteilung, Zweigstelle o.ä.) sein. Allerdings wird in diesem Zusammenhang gefordert, daß der Geschäftsbereich auch geographisch beschränkt ist.

durchläuft und dann den Versand. Sind diese weit voneinander entfernt, so ist es wesentlich billiger, den gesamten Prozeß einmal zu migrieren, als die WAN-Verbindung für jeden weiteren Teilschritt zu verwenden. Ist keine Migration möglich, so wird von der Annahme ausgegangen, daß ein kompletter Prozeß weitgehend zu nur einem Geschäftsbereich gehört. Teilschritte, die in anderen Bereichen ausgeführt werden, werden von dem nicht optimalen Server gesteuert. Wird die Anzahl dieser Schritte zu groß, so kann $Last_{Strg}$ sehr groß werden.

Ist eine Migration von Prozeßinstanzen möglich, so kann jeder Teilschritt von dem für ihn optimalen Server kontrolliert werden. Allerdings gehen einige dieser Ansätze von der einschränkenden Annahme aus, daß alle potentiellen Bearbeiter eines Teilschritts diesem Geschäftsbereich angehören. Die Rollenauflösung erfolgt dann nur lokal für die Benutzer dieses WF-Servers, eine globale Rollenauflösung findet nicht statt.

Wir analysieren zuerst die Variante ohne Migration: Angenommen von einem Server existieren $\#Serv$ Replikate und die Last ist gleichmäßig zwischen ihnen verteilt. Da die Server für verschiedene Prozesse zuständig sind, deren Daten unabhängig voneinander sind, beträgt die Last für die Server und für die WF-Datenbank $\frac{Last_{Strg}}{\#Serv}$. Das Verteilen der Last ist etwas schwierig, da nur eine Zuordnung zwischen Prozeßtypen und Geschäftsbereichen möglich ist; wegen der fehlenden Migrationsmöglichkeit können Prozesse nur als Ganzes einem Server zugeordnet werden. Sollte ein Prozeßtyp alleine schon einen Server überlasten, gibt es keine geeignete Verteilung.

Im Fall mit Migration ist die Einzellast in Idealfall $\frac{Last_{Strg}+Last_{Migr}}{\#Serv}$. Es kommen zwar die Migrationskosten hinzu, aber $Last_{Strg}$ ist geringer, da durch die Migration für jede Aktivität ein geeigneter WF-Server ausgewählt werden kann.

Die Belastung der Server durch das Aktualisieren der Arbeitslisten ist unabhängig davon, ob Migrationen möglich sind. Es ist sinnvoll anzunehmen, daß ein Benutzer nicht in alle Prozeßtypen involviert ist. Deshalb ist jeder Server nur für einen Teil der Benutzer zuständig. Es wird wieder angenommen, daß für jeden Benutzer nur k Server relevant sind. Der Aufwand beträgt somit $\frac{Last_{AL} \cdot k}{\#Serv}$. Erfolgt die Rollenauflösung nur lokal, so ist die Last nur $\frac{Last_{AL}}{\#Serv}$. Allerdings hat diese Variante den Nachteil, daß wenn ein Server ausfällt, alle seine Benutzer blockiert sind, da sie ihre Aufträge nur von ihm erhalten. Außerdem sind manchmal Aktivitäten wünschenswert, die in mehreren Geschäftsbereichen bearbeitet werden können (z.B. Einholen einer Unterschrift von einem beliebigen Abteilungsleiter). Diese sind bei lokaler Rollenauflösung nicht modellierbar. Des weiteren schränkt das Rollenkonzept die Replikation der Server ein. Bei Überlastung eines Servers ist es nicht möglich, die Last auf zwei Server zu verteilen, da diese dann getrennte Geschäftsbereiche mit disjunkten Benutzern darstellen würden.

2.1.3 Voll verteilte WfMSe

Ein WF-Server und das zugehörige Teilnetz sind potentielle Engpässe eines Systems. Diese können vermeiden werden, indem man (wie in Abbildung 4 dargestellt) völlig auf Server verzichtet und die entsprechende Funktionalität in jedem Client realisiert. Die komplette WF-Instanz migriert dann nach Beendigung einer Aktivität zum Rechner des nächsten.

Dieser Ansatz führt zu einem hohen Aufwand beim Ermitteln des Zustandes einer Instanz,

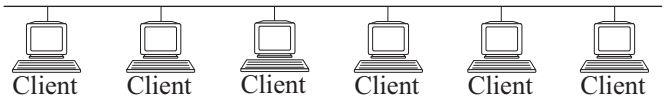

Abbildung 4 Architektur eines voll verteilten WfMSs

da diese Information nicht mehr in einem oder einigen wenigen Servern steckt, sondern über das ganze System verteilt ist. Des weiteren wird bei der Rollenauflösung eine verteilte Synchronisation nötig. Auch ist unklar, woher ein Client alle angemeldeten Benutzer mit einer passenden Rolle für den Nachfolgeschritt kennen soll. Dieses Problem kann dadurch umgangen werden, daß man auf die Rollenauflösung völlig verzichtet und jeder Aktivität einen eindeutigen Bearbeiter zuordnet. Dies stellt aber einen Verlust an Funktionalität bzw. Flexibilität dar, der nicht bei jeder Anwendung akzeptiert werden kann.

Die Last pro Knoten beträgt $\frac{Last_{Strg}+Last_{Migr}}{\#Ben}$. Da die Anzahl der Benutzer $\#Ben$ eines Systems viel größer ist als die Anzahl der Server $\#Serv$, ist dies ein viel besserer Wert, als bei den in Abschnitt 2.1.2 beschriebenen Systemen. Das ist auch notwendig, da die Arbeitsstationen keine leistungsstarken Maschinen, sondern nur die Rechner der Benutzer sind. Die Migrationslast ist bei diesem Ansatz besonders hoch, weil die Prozeßinstanz bei fast jedem Teilschritt migriert.

Der Aufwand je Client für das Aktualisieren der Arbeitslisten ist unabhängig von der Anzahl der Benutzer. Dies ist leicht einzusehen, wenn jeder Aktivität genau ein Bearbeiter zugeordnet ist. Dies ist aber auch dann der Fall, wenn eine globale Rollenauflösung durchgeführt wird. Die Häufigkeit, mit der ein bestimmter Benutzer die Bearbeitung von Aktivitäten beendet, so daß er die Arbeitslisten der Nachfolger aktualisieren muß, ist unabhängig von der Anzahl der (anderen) Benutzer im System. Sie hängt nur von der Zeit ab, die der Benutzer zum Bearbeiten einer Aktivität benötigt. Nimmt man nun an, daß i.d.R. wieder nur ein bestimmter Teil der Benutzer von einem Arbeitslisten-Update betroffen sind (nämlich die eines Geschäftsbereichs), so entsteht dabei ein konstanter Aufwand. Damit hat der Gesamtaufwand zum Aktualisieren von Arbeitslisten für jeden Benutzer eine konstante Größe. Auch die Belastung der Teilnetze stellt kein Problem dar, da es keine Flaschenhälse im Netzwerk gibt. Die Arbeitsstationen müssen allerdings auf ausreichend viele Teilnetze verteilt werden.

2.2 Arbeitslistenverwaltung

Für das Aktualisieren der Arbeitslisten gibt es prinzipiell zwei Möglichkeiten. Entweder fragt der WF-Client beim Server nach Änderungen in seiner Arbeitsliste nach (Polling) oder der Server benachrichtigt bei Änderungen die betroffenen Clients. Diese beiden Methoden werden im folgenden diskutiert.

2.2.1 Polling

Beim Polling wendet sich der Client an den Server, um seine aktuelle Arbeitsliste zu erhalten. Dies kann auf die explizite Anforderung des Benutzers hin geschehen, in festen Zeitabständen

oder ereignisorientiert (z.B. beim Beenden einer Aktivität). Dieses Verfahren wird z.B. im System WorkParty [Sie95] verwendet. Unter der Annahme, daß die $\#Ben$ Clients unabhängig voneinander pollen, z.B. mit der festen Frequenz $freq_{AL}$, ist die durch das Aktualisieren der Arbeitslisten erzeugte Last $Last_{AL} = freq_{AL} \cdot \#Ben$.

2.2.2 Server aktualisiert die Arbeitslisten

Beim Aktualisieren der Arbeitslisten durch den Server ist die Aufrufrichtung umgekehrt, d.h. Client und Server tauschen ihre Rollen. Beim Polling kann es vorkommen, daß ein Client eine Aktualisierung seiner Arbeitsliste wünscht, obwohl sie sich seit der letzten Übertragung nicht geändert hat[6]. Dies läßt sich nicht verhindern, da der Client nicht über die dazu notwendige Information verfügt. Im Gegensatz dazu verfügt der Server über diese Information. Er weiß, wann ein Eintrag in die Arbeitslisten eingefügt werden muß und wann er wieder entfernt werden kann. Diese Information kann folgendermaßen genutzt werden: der Server sendet einem Client nur dann eine neue Arbeitsliste, wenn sich diese geändert hat. Mit dieser Methode aktualisiert z.B. das System ProMInanD [Kar94] seine Arbeitslisten. Es ist auch möglich, maximale Verzögerungszeiten für das Erscheinen von Teilschritten in den Arbeitslisten der Benutzer zu garantieren. Das Aktualisieren der Arbeitslisten durch den Server ist also die geeignete Wahl, falls es Aktivitäten mit hohem Aktualitätsbedarf gibt.

Ein Problem dieses Verfahrens ist, daß es zu einer sehr hohen Last führt, falls bei jeder Änderung sofort alle betroffenen Arbeitslisten aktualisiert werden. Wenn dies aber eine gewisse Zeit verzögert wird und die Änderungen geblockt übertragen werden, so wird die Last auf $freq_{AL} \cdot \#Ben$ gedrückt, da es nun für Übertragungen zum selben Benutzer einen Mindestabstand gibt. Die Last ist also so groß wie beim periodischen Polling. Diese Optimierung ist in dem Sinne flexibel, daß für jeden Aktivitätentyp eine maximale Verzögerungszeit definiert werden kann, so daß Anforderungen an den Aktualitätsbedarf weiterhin erfüllt werden können. Im folgenden wird angenommen, daß der Aufwand für das Aktualisieren der Arbeitslisten $Last_{AL} = freq_{AL} \cdot \#Ben$ beträgt, da sich dies bei beiden Verfahren realisieren läßt.

2.3 Datenfluß

Zwischen den Aktivitäten eines Prozesses müssen Parameterdaten ausgetauscht werden. Das dafür verwendete Modell hat Auswirkungen auf den Umfang der zu transportierenden Daten. So werden mehrfach gespeicherte Daten unter Umständen bei einer Migration auch mehrfach übertragen oder Daten zu einer Aktivität transportiert, die gar nicht benötigt werden. Das Datenmodell hat noch andere Auswirkungen auf ein WfMS, z.B. auf die Modellierung. Auf diese Aspekte wird hier aber nur am Rande eingegangen, da sie die Skalierbarkeit nicht betreffen. Bei den meisten der in Kapitel 3 betrachteten Systeme ist nicht angegeben, welches Datenmodell verwendet wird und es sind auch verschiedene denkbar, allerdings mit unterschiedlich hohen Kosten.

[6] Dies ist insbesondere dann ein Problem, wenn das WfMS aus mehreren Servern besteht, von denen die Mehrzahl (fast) nie Aktivitäten für diesen Client hat, z.B. da sich der Client in einem anderen Geschäftsbereich befindet.

2.3.1 Objektmigrationsansatz

Elektronische Umlaufmappen (Electronic Circulation Folders) [KRW90] werden vor allem in dokumentenorientierten WfMSen wie ProMInanD [Kar94] verwendet. Dabei wird die komplette WF-Instanz zu dem Knoten, auf dem eine Aktivität ausgeführt werden soll, kopiert. Ein ECF enthält außer den Daten, die die Anwendungen benötigen (meist Dokumente), oft auch noch die komplette Schemainformation des Prozesses.

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

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

2.3.2 Input-/Output-Container

In FlowMark [IBM96] werden die Ein- und Ausgabeparameter von Aktivitäten aufeinander abgebildet, d.h. für jedes Eingabedatum eines Schrittes muß der Schritt und dessen Ausgabeparameter angegeben werden, von dem der Wert übernommen werden soll[7]. Auf diese Weise wird der Datenfluß explizit modelliert. Um nun in einem Schritt die Daten von schon abgeschlossenen Aktivitäten verwenden zu können, werden die Eingabecontainer aller Teilschritte physisch gespeichert. Die Daten sind also redundant vorhanden, falls sie von mehreren Teilschritten benötigt werden.

Der Gültigkeitsbereich von Variablen ist immer auf einen Teilschritt begrenzt, ihr Inhalt kann lediglich beim Schrittende anderen Variablen zukünftiger Schritte übergeben werden. Da deshalb zwei Aktivitäten nie eine gemeinsame Variable verwenden, können Synchronisationsprobleme nur dann auftreten, wenn der Datenfluß explizit so modelliert wurde, daß eine Variable ihren Input von mehreren Schritten bezieht. In diesem (leicht erkennbaren) Fall ist es die Aufgabe des WF-Designers, die gewünschte Synchronisation sicherzustellen. Ein Nachteil dieses Ansatzes ist, daß das Modellieren des kompletten Datenflusses sehr aufwendig ist. Des weiteren kann es notwendig werden, Daten durch eine Aktivität (d.h. durch das Anwendungsprogramm) durchzuschleussen, da für Prädikate zur Beschreibung von Verzweigungen nur die Daten aus dem Ausgabecontainer des unmittelbaren Vorgängerschrittes verwendet werden können.

[7] Es ist auch möglich hier mehrere Aktivitäten anzugeben. Dies ist notwendig, um bei Verzweigungen die Daten aus dem jeweils durchlaufenen Zweig übernehmen zu können. Werden mehrere Zweige parallel durchlaufen, so kann dies allerdings zu lost updates führen.

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

2.3.3 Globale Variablen

Das Konzept der prozeßglobalen Variablen wird von WorkParty [Sie95] verwendet. Für einen Prozeß können (wie die globalen Variablen in einem Programm) Variablen definiert werden. Für jede Aktivität kann nun angegeben werden, welche dieser Variablen sie liest bzw. schreibt.

Alle Variablen sind nur einfach vorhanden und zu einer Anwendung werden nur solche Daten transportiert, die wirklich benötigt werden. Bezüglich der Kommunikationskosten ist dieser Ansatz optimal. Allerdings können im Fall von parallelen Zweigen Zugriffskonflikte und lost updates auftreten. Diese lassen sich aber z.B. dadurch verhindern, daß die Variablen nur in einem der parallelen Zweige geschrieben werden können und in den anderen als read-only Parameter modelliert werden müssen.

3 Betrachtung konkreter Systeme

Entscheidend für die Skalierbarkeit und Verfügbarkeit eines WfMSs ist die Art der Verteilung seiner Komponenten. Deshalb haben wir im folgenden Systeme anhand des in Kapitel 2.1 eingeführten Schemas klassifiziert. Abbildung 5 zeigt die Systeme, die in diesem Kapitel beschrieben und analysiert werden. Aspekte, die unabhängig vom Leistungsverhalten sind, werden aus Platzgründen nur erwähnt. In Abschnitt 3.4 werden die betrachteten WfMSe, ihr Leistungsverhalten und ihre Besonderheiten zusammenfassend in tabellarischer Form aufgeführt.

3.1 Zentraler Server

Die meisten kommerziellen Systeme fallen in die in Abschnitt 2.1.1 beschriebene Kategorie mit einem zentralen Server. Einige dieser Systeme wurden so erweitert, daß eine Zusammenarbeit von mehreren Systemen desselben Herstellers möglich ist.

3.1.1 Prozeßabwicklung durch ein einzelnes WfMS mit zentralem Server

In diese Kategorie fallen viele **kommerzielle Systeme** wie WorkParty [Sie95], ProMInanD [Kar94] oder FlowMark [IBM96] bis Version 2.1. Sie verwenden einen zentralen WF-Server

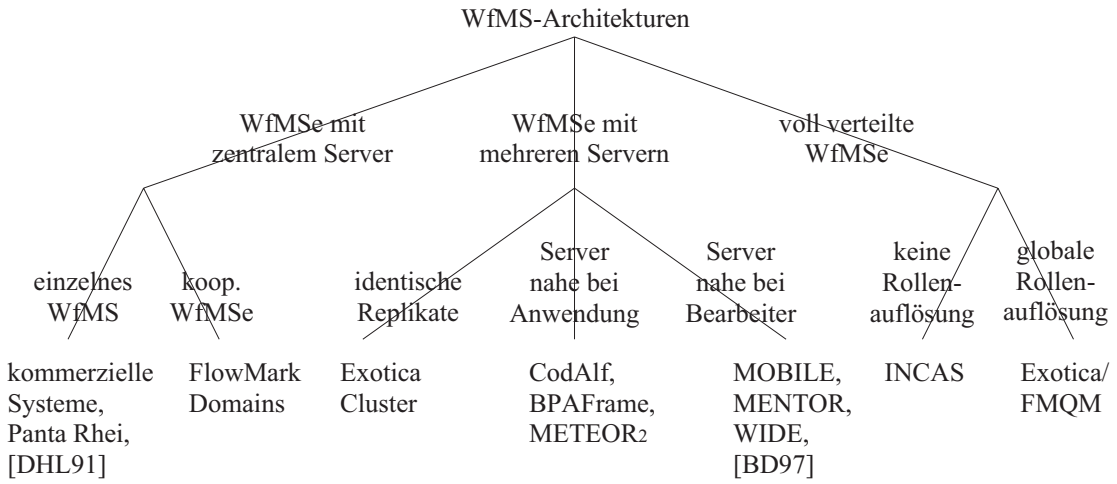

Abbildung 5 Klassifikation der WfMSe nach der Art der Verteilung ihrer Komponenten

oder zumindest eine zentrale WF-Datenbank (mit zentralem DB-Server), die zur Ausführung jedes Teilschritts benötigt wird. Auch einige Forschungsansätze (z.B. **Panta Rhei** [EG96], [**DHL91**]) setzen eine zentrale Kontrollinstanz voraus.

3.1.2 Prozeßabwicklung durch kooperierende WfMSe mit zentralen Servern

Um höhere Benutzerzahlen zu ermöglichen, haben einige Hersteller ihre Systeme so erweitert, daß mehrere Server zusammenarbeiten können. Dadurch entsteht eine Mischform aus einer zentralen Architektur und einer Architektur, bei der mehrere Server verwendet werden, die jeweils nahe bei den jeweiligen Bearbeitern angesiedelt sind. Ein Beispiel hierfür ist **FlowMark** [IBM96] ab Version 2.2. Es wird ermöglicht, einen Subprozeß in einem anderen FlowMark System (Local Domain) ausführen zu lassen. Dies wird verwendet, wenn ein Teil eines Prozesses von anderen Bearbeitern bzw. in anderen Geschäftsbereichen ausgeführt wird als der Rest. Dieser Teil wird dann als Subprozeß modelliert, mit der zusätzlichen Angabe in welchem System er ausgeführt werden soll. Es ist zu beachten, daß es sich bei den verschiedenen Local Domains um getrennte Systeme handelt, die lediglich kooperieren. Dies zeigt sich zum Beispiel daran, daß Benutzer, die in beiden Teilen des Prozesses angesprochen werden sollen, auch in beiden Systemen bekannt gemacht werden müssen.

Im Optimalfall wird die Last gleichmäßig auf alle $\#Serv$ Local Domains verteilt und beträgt damit wie bei einem WfMS mit mehreren Servern $\frac{Last_{Strg}}{\#Serv}$. Sind die Benutzer in jeweils k Domains aktiv und gleichmäßig auf sie verteilt, dann ist der Aufwand für das Aktualisieren der Arbeitslisten $\frac{Last_{AL} \cdot k}{\#Serv}$. Gehört aber ein Großteil der Teilschritte zu einem einzigen Geschäftsbereich, so kann die Last in diesem Local Domain bis auf $Last_{Strg} + Last_{AL}$ anwachsen. Das System bietet keine Unterstützung für das Zerlegen eines Domains, wie z.B. die Identifikation von Teilprozessen, die sich für eine entfernte Ausführung eignen. Der Aufwand für das Ändern der Verteilung ist erheblich, da zuerst entsprechende Subprozesse gebildet werden müssen, d.h. das WF-Modell verändert werden muß.

Ein solches System ist daher primär dann geeignet, wenn die Verteilung offensichtlich ist, z.B. weil getrennte Geschäftsbereiche verwendet werden, zwischen denen die Benutzer weitgehend disjunkt sind. Keiner dieser Geschäftsbereiche darf jedoch so groß werden, daß das lokale System überlastet ist. Besonders sinnvoll ist die Verteilung eines Prozesses, wenn Geschäftsbereiche weit voneinander entfernt sind, so daß die Weitverkehrsverbindung nur beim Starten des Subprozesses, nicht aber bei jedem seiner Teilschritte belastet wird.

3.2 WfMSe mit mehreren Servern

Im Gegensatz zu den eben beschriebenen kooperierenden Systemen mit je einem zentralen Server wurden die Systeme dieser Kategorie als verteilte WfMSe konzipiert. Leider verfügen die meisten von ihnen über keine Werkzeuge, die den Modellierer bei der Verteilung unterstützen und die zu erwartende Last analysieren. Auch wird bei den meisten Ansätzen die Belastung des Netzwerks nicht betrachtet, so daß Teilnetze überlastet werden können.

3.2.1 Server sind identische Replikate

Der **Exotica-Cluster**-Ansatz [AKA+94] verwendet, wie in Abbildung 6 dargestellt, mehrere identische Replikate des Servers, die sogenannten Cluster. Diese bestehen zwar wie in Abschnitt 2.1.2.1 beschrieben aus je einer WF-Datenbank, enthalten aber mehrere WF-Server. Ein Client muß sich mit nur einem WF-Server jedes Clusters verbinden, da durch die gemeinsame WF-Datenbank alle Server eines Clusters über alle notwendigen Informationen verfügen.

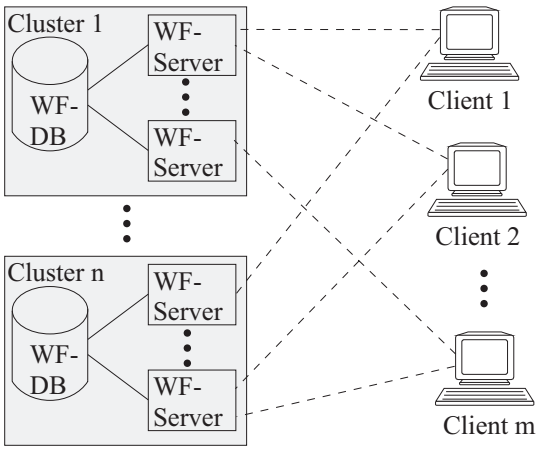

Abbildung 6 Systemarchitektur beim Exotica-Cluster-Ansatz

Die Verwendung mehrerer WF-Server in einem Cluster erhöht die Verfügbarkeit, da ein Client beim Ausfall eines WF-Servers einen anderen Server dieses Clusters verwenden kann. Der Ausfall der WF-Datenbank blockiert natürlich weiterhin den gesamten betroffenen Cluster.

Sind $\#Cluster$ und $\#Serv$ WF-Server je Cluster vorhanden, so beträgt die Last für einen Server $\frac{Last_{Strg}}{\#Cluster \cdot \#Serv}$. Da die Server eines Clusters alle Benutzer bedienen müssen, beträgt

die Last zum Aktualisieren der Arbeitslisten für jeden Server $\frac{Last_{AL}}{\#Serv}$. Die WF-Datenbank bildet hierbei weiterhin einen Flaschenhals.

3.2.2 Server nahe bei Anwendung

In diesem Abschnitt werden Systeme diskutiert, die auf dem im Abschnitt 2.1.2.2 beschriebenen Konzept basieren, bei dem der Ort des WF-Servers nahe dem Anwendungsobjekt gewählt wird.

In [SM96] werden die verwandten Systeme **CodAlf** und **BPAFrame** beschrieben, die auf unterschiedlicher objektorientierter Middleware basieren. Während CodAlf eine objektorientierte Erweiterung des OSF DCE [Sch93] verwendet, basiert BPAFrame auf CORBA [OMG95]. Wie schon erwähnt, wird jede Anwendung in einem Objekt gekapselt. Ein solches Objekt hat einen festen Ort, an dem sich auch der zugehörige WF-Server befindet, und wird als Runtime-System bezeichnet. Prozeßtypen werden als Objekttypen implementiert, die Prozeßinstanzen sind somit als Objekte realisiert. Sie enthalten außer den Anwendungsdaten auch noch die Prozeßbeschreibung, es handelt sich also um einen Objektmigrationsansatz. Diese mobilen Objekte migrieren zum Ort des Anwendungsobjektes der nächsten Aktivität. Dieser Ort wird über einer Trader ermittelt, der als Directory Service dient (also angibt, wo sich ein geeignetes Objekt befindet) und außerdem eine Lastverteilung vornimmt. Der erforderliche entfernte Zugriff auf Daten wird durch die verwendeten Middleware-Dienste realisiert. Dasselbe gilt für den Zugriff der Benutzer auf die entfernten Runtime-Systeme.

Wie schon erwähnt wurde, hängt bei diesem Ansatz die Last eines WF-Servers davon ab, wie häufig die zugehörige Anwendung verwendet wird. Falls es aber möglich ist, den Anwendungsserver zu replizieren, so sorgt der Trader für eine Verteilung der Last.

METEOR$_2$ [DKM+97, MSKW96, SK97] verwendet einen Ansatz, der dem eben beschriebenen sehr ähnlich ist. Er basiert ebenfalls auf CORBA. Allerdings ist die Ablaufbeschreibung nicht in einem mobilen Instanzenobjekt enthalten, sondern sie wird in ihre Teilschritte (jeweils mit Kanten zu vorangehenden und nachfolgenden Schritten) zerlegt. Aus dieser Information wird für jede Aktivität ein Teil des WF-Servers (Taskmanager genannt) erzeugt, der die Ausführung genau dieses Teilschritts steuert. Ein Taskmanager kann prinzipiell an einem beliebigen Ort allokiert werden, allerdings wird auch hier als geeignete Lokation der Ort der Anwendung vorgeschlagen. Im Gegensatz zu den beiden letzten Ansätzen migrieren keine Instanzenobjekte zwischen den Taskmanagern. Statt dessen werden beim Weiterschalten des Prozesses zum nächsten Schritt Referenzen auf die verwendeten Datenobjekte übergeben. Bei der Ausführung einer Aktivität muß dann auf die i.d.R. entfernt liegenden Datenobjekte zugegriffen werden. Schließlich wird in METEOR$_2$ der Ort der Anwendungen und der Taskmanager eindeutig festgelegt, so daß die Aufgabe des Traders entfällt. Dadurch ist aber auch keine dynamische Lastbalancierung möglich.

Der Ansatz betrachtet auch noch weitere Aspekte, wie das Recovery oder die häufige Operation des Erfragens des aktuellen Zustands einer Prozeßinstanz. Zu diesem Zweck gibt es einen zentralen Monitoring-Service, dem die Taskmanager Änderungen der Zustände melden. Außerdem werden ihm Referenzen auf verwendete Daten geschickt, um die Daten beim Recovery rekonstruieren zu können.

Außer der Last der WF-Server ist bei diesem Ansatz auch noch die Belastung des Monitoring-Services interessant. Da diese zentrale Komponente bei der Ausführung jedes Teilschrittes über Änderungen informiert werden muß, ist ihre Last $Last'_{Strg}$. Dabei ist $Last'_{Strg}$ allerdings viel kleiner als $Last_{Strg}$, da nur Referenzen auf Daten übertragen werden und nur einfache Lese- und Schreiboperationen durchgeführt werden. Aus diesem Grund ist diese zentrale Komponente bis zu einer sehr hohen Systemlast ausreichend. Etwas schlechter sieht die Analyse für das Teilnetz aus, in dem der Monitoring-Service angesiedelt ist. Es werden zwar nur kleine Pakete übertragen, dafür aber sehr viele. Dadurch wird der Zusatzaufwand, den das Kommunikationsprotokoll verursacht, sehr groß.

Der Monitoring-Service stellt auch ein Problem für die Verfügbarkeit dar. Da er für das Recovery notwendig ist, führt sein Ausfall zum Systemstillstand. Eine Replikation ist nicht vorgesehen, sie würde im Fall einer Netzpartitionierung auch nicht verhindern, daß der abgeschnittene Teil (u.U. ist das der größere Teil) blockiert ist.

3.2.3 Server nahe bei Bearbeiter

Es gibt mehrere Systeme, die den in Abschnitt 2.1.2.3 beschriebenen Ansatz verfolgen, bei dem versucht wird, den WF-Server nahe bei den potentiellen Bearbeitern zu allokieren. Diese lassen sich, wie in Tabelle 1 dargestellt, danach klassifizieren, ob ein Prozeß den Server wechseln kann. Der Zweck einer solchen Migration wurde schon erläutert. Eine weitere Klassifikation ist danach möglich, ob ein System die Verteilung durch Analysen und Vorschläge unterstützt, oder ob diese Aufgabe dem Augenmaß des Modellierers überlassen bleibt.

	ohne Migration	**mit Migration**
manuelle Verteilung	MOBILE	MENTOR WIDE
systemunterstützte Verteilung	—	[BD97]

Tabelle 1: Systeme, die die WF-Server in der Nähe der jeweiligen Bearbeiter allokieren

MOBILE [BHJ+96, HS96] verfolgt das Ziel, durch Serverreplikation und Datenpartitionierung die Last für die einzelnen Server zu minimieren und dadurch eine möglichst hohe Gesamtlast zu bewältigen. Dazu werden die Aufgaben eines WF-Servers in sogenannte Aspekte zerlegt, wie z.B. einen funktionalen Aspekt, den Kontollfluß und den Datenfluß. Jeder dieser Aspekte wird durch einen eigenen Server realisiert, die durch einen Kernel verbunden sind. Ist ein solcher Server überlastet, so wird er repliziert.

Das Synchronisationsproblem bei Änderung der Daten dieser Server wurde folgendermaßen gelöst: Statische Daten, wie z.B. Schemadaten werden repliziert. Die Server für verschiedene Aspekte haben keine gemeinsamen Daten, außer der statischen WF-Id; diese wird repliziert. Die anderen Daten lassen sich nach den Aspekten partitionieren. Die verschiedenen Replikate eines Servers für denselben Aspekt verwenden zwar dieselben dynamischen Instanzdaten, aber jeder Workflowtyp hat einen korrespondierenden Geschäftsbereich und damit einen fest

zugeordneten Server. Es werden also gesamte Prozesse einem WF-Server zugeordnet, eine Migration ist nicht vorgesehen. Entsprechend dieser Zuordnung lassen sich diese Daten partitionieren.

Die Aufteilung in $\#Asp$ Aspekte bringt bei einer optimalen Verteilung der Last eine Reduktion auf $\frac{Last_{Strg}}{\#Asp}$. Da $\#Asp$ aber eine kleine Konstante ([BHJ+96]: 5 bis 7) ist, die angibt, in wieviele logische Aspekte man die Funktionalität eines WF-Servers aufteilen will, führt dies zu keiner signifikanten Reduktion der Last. Auch ist eine gleichmäßige Verteilung der Last zwischen den Aspekte-Servern unrealistisch, da deren Aufgaben völlig unterschiedlichen Aufwand erfordern. Dazu kommt, daß für bestimmte Operationen (wie z.B. die dynamische Restrukturierung eines Ablaufgraphen) fast alle Aspekte benötigt werden, so daß durch die Verteilung auf mehrere Server ein zusätzlicher Kommunikations- und Synchronisationsaufwand entsteht. Der durch diese Methode zusätzlich erzielte Gewinn ist also fast vernachlässigbar.

Das **MENTOR**-System [WWWK96a, WWWK96b, WWK+97, MWW+98] basiert auf einigen Standardkomponenten, wie dem Transaktionsmonitor TUXEDO [UNI92], CORBA [OMG95] und Statemate, einem Werkzeug zur Modellierung und Ausführung von State-/Activitycharts. Dabei wird in einem Statechart der Kontrollfluß und in einem Activitychart der zugehörige Datenfluß und der funktionale Aspekt eines Workflows modelliert. Kernidee des Projektes ist es nun, die State-/Activitycharts so zu partitionieren, daß eine zum zentralen Fall äquivalente verteilte Ausführung entsteht und jeder Teilschritt in einem zuvor für ihn nach Geschäftsbereich festgelegten „Processing Entity“ ausgeführt wird. Der entsprechende Server ist somit nur für die ihm zugeordneten Teilschritte und für die seinem Processing Entity angehörigen Benutzer zuständig. Dies ist einer der Ansätze, die auf eine globale Rollenauflösung verzichten.

An den Zerlegungspunkten des State-/Activitycharts erfolgt ein Wechsel des Servers, d.h. die komplette Prozeßinstanz migriert zu einem anderen Server. Es werden nur globale Variablen verwendet, deren Änderungen durch einen in jedem Server vorhandenen Communication-Manager propagiert werden. Alle Kommunikationen werden durch ein 2PC geschützt, weshalb die verwendeten Anwendungen das XA-Protokoll unterstützen müssen. Als Anwendungen lassen sich auch Overview-Panels generieren. Dies sind zentrale Komponenten, mit denen sich der Zustand eines Prozesses verfolgen läßt. Ein Overview-Panel kann bei sehr hohen Benutzerzahlen zum Flaschenhals werden (vgl. $METEOR_2$ Monitoring-Service). Da das Overview-Panel aber nicht für das Recovery verwendet wird und somit nicht unbedingt notwendig ist, verschlechtert es die Verfügbarkeit nicht.

WIDE [CGP+96, CGS97] verfolgt einen ähnlichen Ansatz, allerdings ist die Skalierbarkeit in diesem Projekt nur ein Teilaspekt. Es wird auch ein Transaktionsmanagement auf verschiedenen logischen Ebenen (Nested Transactions für Teilschritte bzw. Sagas für Prozesse) angeboten und mit Hilfe von aktiven Regeln ist eine Ausnahmebehandlung möglich. Die Idee zur Verteilung ist analog zu MENTOR. Allerdings ist noch keine Migration vorgesehen, sondern die Daten verbleiben an einem Ort und es wird entfernt mit Hilfe von CORBA auf sie zugegriffen. Dies kann zu den schon erwähnten hohen Kosten bei mehrfachem weit entfernten Zugriff führen.

In [**BD97**] wird außer den Servern auch die Netzstruktur betrachtet, da auch Teilnetze durch zuviel Kommunikation überlastet werden können. Deshalb wird versucht, die Kommunikati-

onskosten zu minimieren, indem der Server für jede Aktivität optimal gewählt wird. Dazu wird ein Teilschritt meist von dem Server kontrolliert, in dessen Teilnetz sich die meisten Benutzern mit einer passenden Rolle befinden. Die Aktivität wird aber auch passenden Benutzern in anderen Teilnetzen angeboten, es findet also eine globale Rollenauflösung statt.

Um das System zu optimieren, werden Algorithmen verwendet, die zur Modellierungszeit laufen, also die Server nicht zusätzlich belasten. Die Algorithmen berechnen eine Verteilung durch die die Kommunikationskosten minimiert werden und analysieren die Last für die einzelnen Komponenten (WF-Server, Teilnetze und Gateways). Bei diesen Kosten wird die Kommunikation für das Aktualisieren der Arbeitslisten und für den Parametertransfer bei der Aktivitätenausführung berücksichtigt. Aber auch die Migrationskosten und die Kommunikation von Anwendungen mit externen Datenquellen werden mit einbezogen. Diese Algorithmen benötigen natürlich gewisse Informationen, die bei der Modellierung bereitgestellt werden müssen. Dies sind statistische Angaben, wie z.B. die Häufigkeit der Ausführung einer Instanz, die Belastbarkeit der Komponenten oder die durchschnittliche Größe der einzelnen Parameterdaten.

Die Last für WF-Server, WF-Datenbank und Teilnetz beträgt auch hier $\frac{Last_{Strg}+Last_{Migr}}{\#Serv}$. Die Migrationskosten sind dabei aber niedriger als bei MENTOR und WIDE, da nicht bei jedem Wechsel des Geschäftsbereichs migriert werden muß[8].

3.3 Voll verteilte WfMSe

Voll verteilte WfMSe, die völlig auf eine Rollenauflösung verzichten und damit einen Verlust an Funktionalität akzeptieren, bilden die erste Unterkategorie. Es gibt aber auch Beispiele für voll verteilte Systeme, die eine globale Rollenauflösung realisieren.

3.3.1 Systeme ohne Rollenauflösung

INCAS [BMR94] ist ein System, das ohne Server auskommt und auf eine Rollenauflösung verzichtet. Der Name kommt von einem INformation CArrier, der einer elektronischen Umlaufmappe entspricht und jeweils zu der Arbeitsstation des nächsten Schrittes migriert. Dieser INCA enthält den gewünschten Dienst, Regeln die den Daten- und Kontrollfluß beschreiben, die Daten der Instanz und Atomaritätsanforderungen. Auch die Arbeitsstationen verfügen über Regeln, die mit denen des INCA zusammen verwendet werden, um eine Aktivität auszuführen und den nächsten Schritt zu berechnen. Für diesen Nachfolgeschritt wird, ebenfalls anhand der Regeln, die zugehörige Arbeitsstation berechnet. Es findet keine Rollenauflösung statt, somit ist auch keine Synchronisation zwischen den potentiellen Nachfolgern notwendig. Bei jedem Weiterschalten zum nächsten Schritt wird der INCA verändert. Er wird allerdings nicht modifiziert, sondern es wird eine neue Version von ihm erzeugt, die zusammen mit der alten migriert. Regeln können sich damit auch auf alte Versionen von Daten beziehen.

Es wird ein verläßliches Kommunikationssystem verwendet, damit sich die Arbeitsstationen nicht um Sicherheitsaspekte beim Transport des INCAs kümmern müssen. Eine Besonderheit des Ansatzes ist, daß das gesamte System durch Regeln gesteuert wird. Die Regelmenge

[8] Es macht sicher keinen Sinn, wegen eines einzelnen Schrittes mit nur wenigen Bearbeitern und kleinen Parameterdaten, die komplette Prozeßinstanz zu einem anderen Server und zurück zu kopieren.

ist sogar dynamisch, d.h. bei der Ausführung einer Aktivität können Regeln erzeugt oder verändert werden. Auch die Transaktionssemantik der Aktivitäten wird mittels Regeln definiert. Allerdings ist eine große, verteilte und dazu noch dynamische Regelmenge schwer zu durchschauen.

Die durch die Migration verursachte Last ist bei diesem Ansatz besonders hoch, weil die Prozeßinstanz sehr groß ist, da sie die Ablaufbeschreibung und auch noch die alten Versionen der Daten enthält.

3.3.2 Globale Rollenauflösung

Exotica/FMQM [AMG+95] ist ebenfalls ein voll verteilter Ansatz. Auch hier wird ein sicheres Kommunikationssystem verwendet, nämlich Persistent-Queues. Allerdings wird der Ablauf nicht durch Regeln beschrieben, sondern wie in FlowMark durch einen Graphen mit Kontroll- und Datenkonnektoren. Dieser Graph wird so auf die Knoten verteilt, daß jeder Knoten diejenigen Teile des Graphen erhält, die Schritte mit den Rollen seiner Benutzer enthalten. Ist die Bearbeitung einer Aktivität beendet, so werden Nachrichten an alle Knoten geschickt, die für Schritte verantwortlich sind, zu denen vom aktuellen Schritt aus Kontroll- oder Datenkanten führen. Die Daten werden bei dieser Methode schrittweise verteilt, sie migrieren nicht mit der Prozeßinstanz.

Exotica/FMQM sieht nach Beendigung einer Aktivität eine globale Rollenauflösung vor. Damit kann eine Instanz nicht mehr einfach an den Knoten des nachfolgenden Schrittes gesendet werden, sondern dieser muß erst ermittelt werden. Dazu informiert der Vorgängerknoten alle potentiellen Nachfolger über den zu vergebenden Schritt, indem er die entsprechende Information in deren Message-Queues schreibt. Diese holen sich dann die Instanz aus dessen Ausgabequeue, wenn sie die nächste Aktivität ausführen wollen. Die Synchronisation erfolgt durch das Transaktionskonzept des Queueing-Systems. Dieses Verfahren führt zu einigen Schwierigkeiten: Durch die Auswertung der von einem Schritt ausgehenden Datenkonnektoren lassen sich lediglich die Schritte ermitteln, die diese Daten benötigen. Es ist aber noch nicht bekannt, an welchem Knoten diese Schritte später ausgeführt werden. Deshalb müssen die Daten zu allen potentiellen Ausführungsknoten transportiert werden.

Auch bei diesem Ansatz ist die Last größer als bei Ansätzen mit Servern, allerdings aus anderen Gründen als bei INCAS. Diese sind zum einen die zu mehreren potentiellen Nachfolgern transportierten Daten. Aber auch die Verwendung von persistenten Message-Queues kann zu einem Overhead führen, da viele Systeme keine Operationen ohne Transaktionsschutz zulassen und somit ein in manchen Fällen unnötiges 2PC erzwingen. Andere Systeme erlauben kein entferntes Lesen, so daß dies durch mehrere andere Operationen nachgebildet werden muß. Des weiteren entsteht durch die verteilte Rollenauflösung ein Synchronisationsaufwand.

3.4 Systemübersicht

Tabelle 2 gibt einen Überblick über die in diesem Bericht betrachteten Systeme. Er werden der Aufwand eines Servers zur Ausführung der Aktivitäten und zum Aktualisieren der Arbeitslisten verglichen und die Besonderheiten der einzelnen Systeme zusammengefaßt.

Klasse	System	Last pro Komponente		Bemerkungen
		Aktivitätenausf.	Arbeitsl.	
zentraler Server	kommerzielle Systeme	$Last_{Strg}$	$Last_{AL}$	Daten zentral gespeichert, meist nur 1 Server
	Panta Rhei	$Last_{Strg}$	$Last_{AL}$	zentrales aktives DBMS
	[DHL91]	$Last_{Strg}$	$Last_{AL}$	transaktionale Workflows
	FlowMark Domains	$\frac{Last_{Strg}}{\#Serv}$	$\frac{Last_{AL} \cdot k}{\#Serv}$	Subprozeß kann in anderem Domain ausgeführt werden
mehrere Server	Exotica Cluster	$\frac{Last_{Strg}}{\#Cluster}$	$Last_{AL}$	identische Cluster, können alle Prozeßtypen ausführen
	CodAlf, BPAFrame	$\frac{Last_{Strg}+Last_{Migr}}{\#Serv}$	$\frac{Last_{AL} \cdot k}{\#Serv}$	Anwendungen sind CORBA-Objekte, WF-Server bei Anwendung
	METEOR$_2$	$\frac{Last_{Strg}}{\#Serv}$	$\frac{Last_{AL} \cdot k}{\#Serv}$	CORBA-basiert, zentr. Monitoring-Serv.
	MOBILE	$\frac{Last_{Strg}}{\#Serv \cdot \#Asp}$	$\frac{Last_{AL} \cdot k}{\#Serv}$	keine Migration
	MENTOR	$\frac{Last_{Strg}+Last_{Migr}}{\#Serv}$	$\frac{Last_{AL}}{\#Serv}$	keine globale Rollenauflösung
	WIDE	$\frac{Last_{Strg}}{\#Serv}$	$\frac{Last_{AL}}{\#Serv}$	siehe MENTOR, keine Migration
	[BD97]	$\frac{Last_{Strg}+Last_{Migr}}{\#Serv}$	$\frac{Last_{AL} \cdot k}{\#Serv}$	Verteilung u. Lastanalyse durch System
voll verteilt	INCAS	$\frac{Last_{Strg}+Last_{Migr}}{\#Ben}$	konst.	keine Rollenauflösung
	Exotica/FMQM	$\frac{Last_{Strg}+Last_{Migr}}{\#Ben}$	konst.	verteilte globale Rollenauflösung

Tabelle 2: Vergleich der untersuchten WfMSe

4 Zusammenfassung und Ausblick

In diesem Beitrag wurden drei fundamentale Klassen von WfMS-Architekturen identifiziert. Dies sind zum einen Systeme mit einem zentralen Server. Da dieser einen potentiellen Engpaß darstellt, muß bei einer hohen Benutzerzahl ein entsprechend leistungsstarker und damit teurer Rechner verwendet werden. Die dafür aktuell zur Verfügung stehende Technologie bildet eine Obergrenze für die Leistungsfähigkeit des WfMSs. Weitere Klassen bilden Systeme mit mehreren Servern und Systeme ohne Server, bei denen die Ablaufsteuerung vollständig verteilt ist. Innerhalb dieser Klassen wurden jeweils noch Unterklassifikationen vorgenommen. Einige kommerzielle Systeme und Vorschläge aus dem Bereich der Forschung wurden bezüglich dieser Klassifikation eingeordnet und auf ihre Skalierbarkeit hin untersucht.

Um eine geeignete WfMS-Architektur auswählen zu können, ist es notwendig, die jeweilige Anwendung genau zu analysieren, um sie einem Anwendungsbereich zuordnen zu können. In Tabelle 3 wird für verschiedene Anwendungsbereiche eine geeignete Architektur vorgeschlagen. Diese Vorschläge sind als „Tendenzaussagen“ und nicht als absolute Feststellungen zu verstehen, da sich viele Anwendungen nicht eindeutig einem Bereich zuordnen lassen. Des

weiteren kann eine Anwendung auch noch weitere Eigenschaften haben, die die Wahl der Architektur beeinflussen (z.B. die Anforderung, daß sicherheitsrelevante Schritte von dem WF-Server der bearbeitenden Abteilung kontrolliert werden müssen).

Anwendungstyp	**empfohlene WfMS-Architektur**
wenige Benutzer, wenig Last	zentraler WF-Server (evtl. kooperierend)
Arbeitsschritt wird jeweils von einer bzw. wenigen Organisationseinheiten bearbeitet	Server nahe bei Bearbeiter
• gesamter Prozeß in selber Organisationseinheit	keine Migration
• alle Bearbeiter eines Arbeitsschritts sind in selber Organisationseinheit	nur lokale Rollenauflösung
Benutzer verwenden alle Arbeitsschritte, Anwendungen haben festen Ort	Server nahe bei Anwendung
Benutzer verwenden alle Arbeitsschritte und alle Anwendungen	identische Replikate des WF-Servers

Tabelle 3: Geeignete Architekturen für verschiedene Anwendungstypen

Die Auswahl einer Architektur ist recht einfach, falls eine Anwendung nur sehr wenige Benutzer hat, keine hohe Last erzeugt und sichergestellt ist, daß sich dies auch in Zukunft nicht ändert. Dann fällt die Wahl auf das einfachste Konzept, den zentralen Server. Im Fall von unternehmensweiten WfMSen ist die Architektur am vorteilhaftesten, bei der die WF-Server nahe bei den Bearbeitern allokiert werden. In diese Klasse fallen auch die meisten der betrachteten Systeme. Allerdings ist deren Verwendung nur möglich, wenn organisatorische Einheiten identifiziert werden können, denen jeweils bestimmte Prozeßschritte zugeordnet werden können. Ein solches System kann noch optimiert werden, falls die Prozesse immer fast vollständig einer Organisationseinheit zugeordnet werden können, oder falls immer alle potentiellen Bearbeiter eines Arbeitsschrittes derselben Organisationseinheit angehören. Ist keine Zuordnung zwischen Organisationseinheiten und Arbeitsschritten möglich, aber den Anwendungen ist (fast) immer ein fester Ort zugewiesen, so kann dieses Optimierungspotential genutzt werden, indem dieser Ort als Lokation für den WF-Server verwendet wird. Ist auch diese Bedingung nicht erfüllt, so bleibt nur noch die Möglichkeit identische Replikate des Servers zu verwenden. Dabei ist aber zu beachten, daß die Belastung eines Clusters durch das Aktualisieren der Arbeitslisten sehr groß werden kann. Die voll verteilte Architektur ist wegen des hohen Grades an Verteilung der Information und der daraus resultierenden Nachteile (siehe Abschnitt 2.1.3) normalerweise nicht empfehlenswert. Lediglich in Ausnahmefällen kommt diese Variante in Frage. Ein Beispiel hierfür könnte ein extrem großes System sein, bei dem jede Aktivität direkt einem Bearbeiter zugeordnet ist.

Schließlich bleibt zu bemerken, daß es noch ein großes Potential für die Entwicklung von Architekturen für unternehmensweite WfMSe gibt. Dabei ist die Skalierbarkeit und Verfügbarkeit ein wichtiger Aspekt. Es ist aber zu beachten, daß die Architektur auch Einfluß auf die Funktionalität eines Systems haben kann. Für die Performance ist es sicher von Vorteil,

die Prozeßbeschreibungen zu kompilieren, zu zerteilen und die von einem Knoten benötigen Fragmente hart in den Schedulern zu verdrahten (vgl. METEOR$_2$ [MSKW96]). Allerdings macht dies dynamische Modifikationen der Ablaufbeschreibung [RD98] nahezu unmöglich. Um beispielsweise feststellen zu können, welche Eingabeparameter durch die Modifikation einer WF-Instanz (z.B. Auslassen eines Schrittes) nicht mehr versorgt sind, muß der gesamte Prozeßgraph vorliegen und Änderungen sind nur möglich, wenn der Prozeßgraph interpretiert wird. Aber auch andere Aspekte, die in heutigen Systemen ebenfalls noch nicht realisiert sind, stellen gewisse Anforderungen an die Architektur. Beispiele hierfür sind das Zeitmanagement, die Verwendung von Backup-Servern oder die Integration mobiler Clients.

Teil unserer zukünftigen Arbeit wird sein, den qualitativen Vergleich der Konzepte durch quantitative Analysen zu ergänzen. Dazu soll das Leistungsverhalten der einzelnen Ansätze für verschiedene Anwendungsszenarien simuliert und die entstehende Last verglichen werden.

Literatur

[AKA+94] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör und C. Mohan: *Failure Handling in Large Scale Workflow Management Systems.* Technischer Bericht RJ9913, IBM Almaden Research Center, November 1994.

[AMG+95] G. Alonso, C. Mohan, R. Günthör, D. Agrawal, A. El Abbadi und M. Kamath: *Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management.* In: *Proceedings of the IFIP Working Conference on Information Systems for Decentralized Organisations*, Trondheim, August 1995.

[BD97] T. Bauer und P. Dadam: *A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration.* In: *Second IFCIS Conference on Cooperative Information Systems*, S. 99–108, Kiawah Island, SC, Juni 1997.

[BHJ+96] C. Bußler, P. Heinl, S. Jablonski, H. Schuster und K. Stein: *Architektur von unternehmensweit einsetzbaren Workflow-Management–Systemen.* In: *Proceedings of MobIS 96, Rundbrief des GI-Fachausschusses 5.2*, S. 73–77, Oktober 1996.

[BMR94] D. Barbará, S. Mehrotra und M. Rusinkiewicz: *INCAS: A Computational Model for Dynamic Workflows in Autonomous Distributed Environments.* Technischer Bericht, Matsushita Information Technology Laboratory, Princeton, NJ, Mai 1994.

[CGP+96] F. Casati, P. Grefen, B. Pernici, G. Pozzi und G. Sánchez: *WIDE: Workflow Model and Architecture.* Technischer Bericht CTIT 96-19, University of Twente, 1996.

[CGS97] S. Ceri, P. Grefen und G. Sánchez: *WIDE – A Distributed Architecture for Workflow Management.* In: *Seventh International Workshop on Research Issues in Data Engineering*, Birmingham, April 1997.

[DHL91] U. Dayal, M. Hsu und R. Ladin: *A Transactional Model for Long-Running Activities*. In: *Proceedings of the 17th International Conference on Very Large Data Bases*, S. 113–122, Barcelona, September 1991.

[DKM+97] S. Das, K. Kochut, J. Miller, A. Sheth und D. Worah: *ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR$_2$*. Technischer Bericht #UGA-CS-TR-97-001, Department of Computer Science, University of Georgia, Februar 1997.

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

[Elm92] A.K. Elmagarmid: *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, 1992.

[GHS95] D. Georgakopoulos, M. Hornick und A. Sheth: *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*. Distributed and Parallel Databases, 3(2):119–152, April 1995.

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

[IBM96] IBM: *FlowMark – Modeling Workflow, Version 2 Release 2, Document Number: SH19-8241-01*, 2. Auflage, Februar 1996.

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

[Kar94] B. Karbe: *Flexible Vorgangssteuerung mit ProMInanD*. In: U. Hasenkamp, S. Kirn, M. Syring (Herausgeber): *CSCW – Computer Supported Cooperative Work*, S. 117–133. Addison-Wesley, 1994.

[KRW90] B. Karbe, N. Ramsperger und P. Weiss: *Support of Cooperative Work by Electronic Circulation Folders*. In: *Conference on Office Information Systems, IEEE Computer Society*, S. 109–117, Cambridge, MA, 1990.

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

[MSKW96] J. A. Miller, A. P. Sheth, K. J. Kochut und X. Wang: *CORBA-Based Run-Time Architectures for Workflow Management Systems*. Journal of Database Management, Special Issue on Multidatabases, 7(1):16–27, 1996.

[MWW+98] P. Muth, D. Wodtke, J. Weißenfels, A. Kotz-Dittrich und G. Weikum: *From Centralized Workflow Specification to Distributed Workflow Execution*. Journal of Intelligent Information Systems, Special Issue on Workflow Management, 10(2), März 1998.

[OMG95] OMG: *Object Management Architecture Guide*. John Wiley & Sons, 3. Auflage, Juni 1995.

[RD98] M. Reichert und P. Dadam: *$ADEPT_{flex}$ – Supporting Dynamic Changes of Workflows Without Loosing Control*. Journal of Intelligent Information Systems, Special Issue on Workflow Management, 10(2), März 1998.

[Sch93] A. Schill: *DCE - Das OSF Distributed Environment, Einführung und Grundlagen*. Springer Verlag, 1993.

[Sie95] Siemens Nixdorf: *WorkParty – Benutzerhandbuch, Version 2.0*, August 1995.

[SK97] A. Sheth und K. J. Kochut: *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems*. In: *Proceedings of the NATO Advanced Study Institute on Workflow Management Systems and Interoperability*, S. 12–21, Istambul, August 1997.

[SM96] A. Schill und C. Mittasch: *Workflow Management Systems on Top of OSF DCE and OMG CORBA*. Distributed Systems Engineering, 3(4):250–262, Dezember 1996.

[UNI92] UNIX System Laboratories: *TUXEDO System Release 4.1 – Product Overview and Master Index*. Prentice Hall, 1992.

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

[WWWK96a] J. Weißenfels, D. Wodtke, G. Weikum und A. Kotz-Dittrich: *The Mentor Architecture for Enterprise-wide Workflow Management*. In: *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems*, S. 69–73, Athens, Mai 1996.

[WWWK96b] D. Wodtke, J. Weißenfels, G. Weikum und A. Kotz-Dittrich: *The Mentor Project: Steps Towards Enterprise-Wide Workflow Management*. In: *Proceedings of the 12th IEEE International Conference on Data Engineering*, S. 556–565, New Orleans, LA, März 1996.