Anwendungsspezifische Anforderungen an Workflow–Management–Systeme am Beispiel der Domäne Concurrent Engineering*

T. Beuter, P. Dadam
Universität Ulm

Zusammenfassung

Workflow–Management–Systeme (WFMS) erobern sich neue Anwendungsgebiete mit jeweils spezifischen Anforderungen an die Leistungsfähigkeit dieser Systeme. Neben den Anforderungsprofilen an WFMS aus verschiedenen Anwendungsgebieten werden hier besonders die Anforderungen aus dem Bereich des Concurrent Engineering aufgezeigt. Die jeweiligen anwendungsspezifischen Anforderungen führen nicht nur zu einer großen Zahl unterschiedlich mächtiger Modellierungskomponenten, sondern auch zu verschiedenen Alternativen für mögliche Laufzeitumgebungen der WFMS. Der Bericht zeigt die Probleme auf, die sich daraus für WFMS–Anwender und –Entwickler ergeben, und leitet praxisrelevante Fragestellungen ab, die aus wissenschaftlicher Sicht zu bearbeiten sind.

1 Einleitung

In vielen Geschäftsbereichen wird ein komplexer Vorgang (Geschäftsprozeß, Ablauf) nicht von einem Mitarbeiter allein, sondern in Zusammenarbeit mit Mitarbeitern der eigenen oder anderer Abteilungen und Unternehmen durchgeführt, die im Rahmen dieses Vorgangs Teilaufträge erfüllen. Beispiele hierfür sind Reiseplanung und –abrechnung, Untersuchungszyklen in Kliniken [36, 16], Kreditanträge und –anpassungen, Versicherungsantragsbearbeitung oder Fertigungsprozeßplanung und –steuerung [64]. Die Koordination dieser zum Teil komplexen und räumlich weit verteilten Abläufe wird von klassischen Informationssystemen (datenbankbasierten Anwendungen) nur sehr unzureichend unterstützt, da sie zur *statischen* Verwaltung von Informationen entwickelt wurden. Der *dynamische* Teil dieser Abläufe, wie das Zuteilen neuer Aufträge, der Austausch von Informationen zwischen Mitarbeitern oder die Festlegung und Überwachung zeitlicher Beschränkungen, erfolgt deshalb häufig informell zwischen den einzelnen beteiligten Personen und verursacht einen nicht unerheblichen Zusatzaufwand.

Will man mit konventionellen Programmiermethoden auch beim dynamischen Teil der Abläufe EDV–Unterstützung bieten, so führt dies zu sehr komplexen Anwendungsprogrammen: Der gesamte Informations– und Kontrollfluß, einschließlich die in verteilten Anwendungen sehr aufwendige Fehlerbehandlung, muß explizit ausprogrammiert werden. Hinzu kommt, daß diese „hart verdrahteten“ Abläufe nur schwer zu validieren, zu warten sowie an organisatorische oder funktionale Änderungen anzupassen sind.

*Diese Arbeit entstand im Rahmen eines von der Daimler-Benz-Forschung Ulm geförderten Forschungsprojekts.

WFMS [23, 30] bieten hier durch Trennung von Ablauflogik (*dynamisch*) und eigentlichem Anwendungscode (*statisch*) einen vielversprechenden Ansatz. Durch „Herausziehen“ der Ablauflogik aus dem Anwendungscode kann ein Ablauf schnell spezifiziert, über Analyse beziehungsweise Animation validiert und gegebenenfalls rasch an neue oder veränderte Anforderungen angepaßt werden.

Ein WFMS[1] besteht aus einer *Buildtime–Komponente* zur Modellierung und Validierung von Abläufen, sogenannten *Workflows*, und einer *Runtime–Komponente* für deren Ausführung (Laufzeitumgebung). Die Runtime–Komponente eines WFMS besteht wiederum aus einer oder mehreren *WF–Engines*, welche die eigentliche Ausführung der Workflows übernehmen. Die *Architektur* eines WFMS legt die Zahl und das Zusammenspiel der für eine Workflow–Ausführung benötigten WF–Engines, ihre funktionale Aufgliederung in Komponenten (Kontrollflußsteuerung, Organisationsverwaltung) sowie ihre physikalische Verteilung auf Rechnerknoten fest.

Aufgrund ihres generischen Ansatzes können WFMS im Prinzip zur Modellierung und Ausführung von Abläufen aus beliebigen Anwendungsgebieten eingesetzt werden. Neben „klassischen“ Bereichen, wie zum Beispiel der Büroautomatisierung [27, 63, 40] oder der Software–Entwicklung [1, 46], werden in letzter Zeit WFMS auch in anderen Gebieten eingesetzt. Beispiele hierfür sind medizinisch–organisatorische Abläufe [17, 16, 57, 43] oder Concurrent–Engineering–Anwendungen [48, 47, 2, 53, 45, 56, 35].

Durch diese breitgefächerten Einsatzgebiete entstehen jedoch zusätzliche *anwendungsspezifische Anforderungen* an WFMS, die im Rahmen dieses Berichts skizziert werden sollen (Abschnitt 3). Dabei wird neben Anforderungen aus anderen Anwendungsgebieten (Abschnitt 3.2) insbesondere auf Anforderungen aus dem Bereich des Concurrent Engineering (Abschnitt 3.1) eingegangen. Hierfür wird in Abschnitt 2 ein Beispiel beschrieben, das typisch für viele Abläufe des Concurrent Engineering ist. Abschnitt 4 beschreibt die Probleme, die sich aus den anwendungsspezifischen Anforderungen für Anwender und Entwickler von WFMS ergeben. Im darauffolgenden Abschnitt werden Herausforderungen an die Forschung formuliert, welche auf noch offene Fragestellungen hinweisen. Der Bericht endet mit Überlegungen bezüglich einer WFMS–Architektur, welche die zu Beginn aufgestellten Concurrent–Engineering–spezifischen Anforderungen umsetzen kann.

2 Anwendungsbeispiel: Concurrent Engineering

Wie es der Begriff *Concurrent Engineering* schon andeutet, werden komplexe Entwicklungsprozesse in der Regel *gemeinsam* von *mehreren* Personen durchgeführt, deren zum Teil *gleichzeitig* stattfindende Einzeltätigkeiten geeignet koordiniert werden müssen. Ein populäres Beispiel dafür ist die Konstruktion und der Bau von Fahrzeugen. Auch die Dauer eines Entwicklungsprozesses und die häufig anzutreffende räumliche Trennung der involvierten Personen sind Indizien dafür, daß sich der Einsatz von WFMS im Bereich des Concurrent Engineering lohnt.

Die prinzipielle Eignung von WFMS soll anhand des im folgenden beschriebenen vereinfachten Beispiels „Konstruktion und Fertigung von Leitungsbündeln“ [56, 35] gezeigt werden. Im Abschnitt 3.1 werden dann einige Anforderungen an WFMS formuliert, welche spezifisch für Concurrent–Engineering–Anwendungen sind.

[1] Im Bereich der WFMS existiert eine verwirrend hohe Zahl verschiedener Terminologien (siehe [67] für eine Gegenüberstellung der Begriffe). Die hier verwendeten Begriffe orientieren sich am Referenzmodell der *Workflow Management Coalition (WfMC)*, einem Standardisierungsgremium bestehend aus führenden Herstellern von WFMS [68].

Beim elektrischen Anlagenbau in Flugzeugen werden Einzelleitungen elektrischer Anlagen mit gleichem oder ähnlichem physikalischem Verlauf zu *Leitungsbündeln* zusammengefaßt.

Das Zusammenspiel der bei diesem Prozeß involvierten Einzelschritte läßt sich am besten über einen *anlagenbezogenen* und einen *bündelbezogenen* Ablauf beschreiben, zwischen denen Datenabhängigkeiten existieren (siehe Abbildung 1 a und 1 b)[2].

Zuerst wird von (verschiedenen) Mitarbeitern der Organisationseinheit *Systemdefinition* die prinzipielle Funktionsweise jeder im Flugzeug einzubauenden elektrischen Anlage durch jeweils einen *Wirkschaltplan (principal diagram)* beschrieben (CAD–Zeichnung, Teilschritt *„Wirkschaltplan erstellen“*). Da dieser Teilschritt für jede Anlage einmal aufgerufen wird, werden mehrere Instanzen des anlagenbezogenen Ablaufs gestartet. Der darauffolgende Teilschritt *„Wiring Diagram erstellen“* benötigt als Eingabeparameter VKE–Blockschaltbilder[3], so daß die entsprechenden anlagenbezogenen Abläufe solange blockiert sind, bis diese Schaltbilder vorliegen. Diese Daten werden im bündelbezogenen Ablauf durch den Teilschritt *„VKE–Blockschaltbild erstellen“* erzeugt (Interworkflow–Abhängigkeit). Um spätere Modifikationen an den VKE–Blockschaltbildern durch nachträglich eintreffende Wirkschaltpläne möglichst auszuschließen, wird in der Regel eine größere Zahl an Wirkschaltplänen angesammelt, bevor der Teilschritt *„VKE–Blockschaltbild erstellen“* im bündelbezogenen Ablauf durch einen Mitarbeiter aus dem Bereich *Konstruktion* bearbeitet wird.

Nach Fertigstellung der benötigten VKE–Blockschaltbilder werden im Teilschritt *„Wiring Diagram erstellen“* die Wirkschaltpläne ebenfalls im Bereich Konstruktion durch die in den Schaltbildern enthaltenen Informationen (Leitungsnummern, Zwischenstecker, ...) zu *Wiring Diagrams* erweitert.

Die automatischen Teilschritte *„Geräteliste erstellen“* beziehungsweise *„Leitungsliste erstellen“* extrahieren aus den einzelnen Wiring Diagrams (CAD–Zeichnungen) eine Liste der benötigten Geräte beziehungsweise Leitungen, die wiederum im bündelbezogenen Ablauf zu bündelspezifischen Listen zusammengefaßt werden (Organisationseinheit *Fertigungsunterlagen–Erstellung*). Die für die bündelspezifischen Leitungslisten benötigten Längenangaben werden im Teilschritt *„Leitungsbündel Schemazeichnung erstellen (VKM)“* (Bereich Konstruktion) anhand genauer Mockup[4]–Messungen festgelegt. Dieser Bearbeitungsschritt erfolgt meist parallel zur Erstellung der Wiring Diagrams. Die VKM–Schemazeichnungen dienen auch als Vorlage für die Konstruktion des Verlegebretts, auf dem dann im *Fertigungsbereich* das Leitungsbündel manuell erstellt wird.

3 Anwendungsspezifische Anforderungen an WFMS

Dieser Abschnitt beschreibt anwendungsspezifische Anforderungen aus dem Bereich des Concurrent Engineering (Abschnitt 3.1) sowie aus anderen Einsatzgebieten von WFMS (Abschnitt 3.2). Außerdem werden in Abschnitt 3.3 allgemeine Anforderungen an eine WFMS–Architektur formuliert.

[2] Auf die Bedeutung der unterschiedlichen Knotenformen in Abbildung 1 (4/6-eckig) wird später näher eingegangen.

[3] In einem VKE–Blockschaltbild werden die Einzelleitungen verschiedener elektrischer Anlagen zu (vorläufigen) Leitungsbündeln zusammengefaßt.

[4] Ein Mockup ist ein maßstabsgetreues Modell des Flugzeug(abschnitt)s.

Konstruktion und Fertigung von Leitungsbündeln

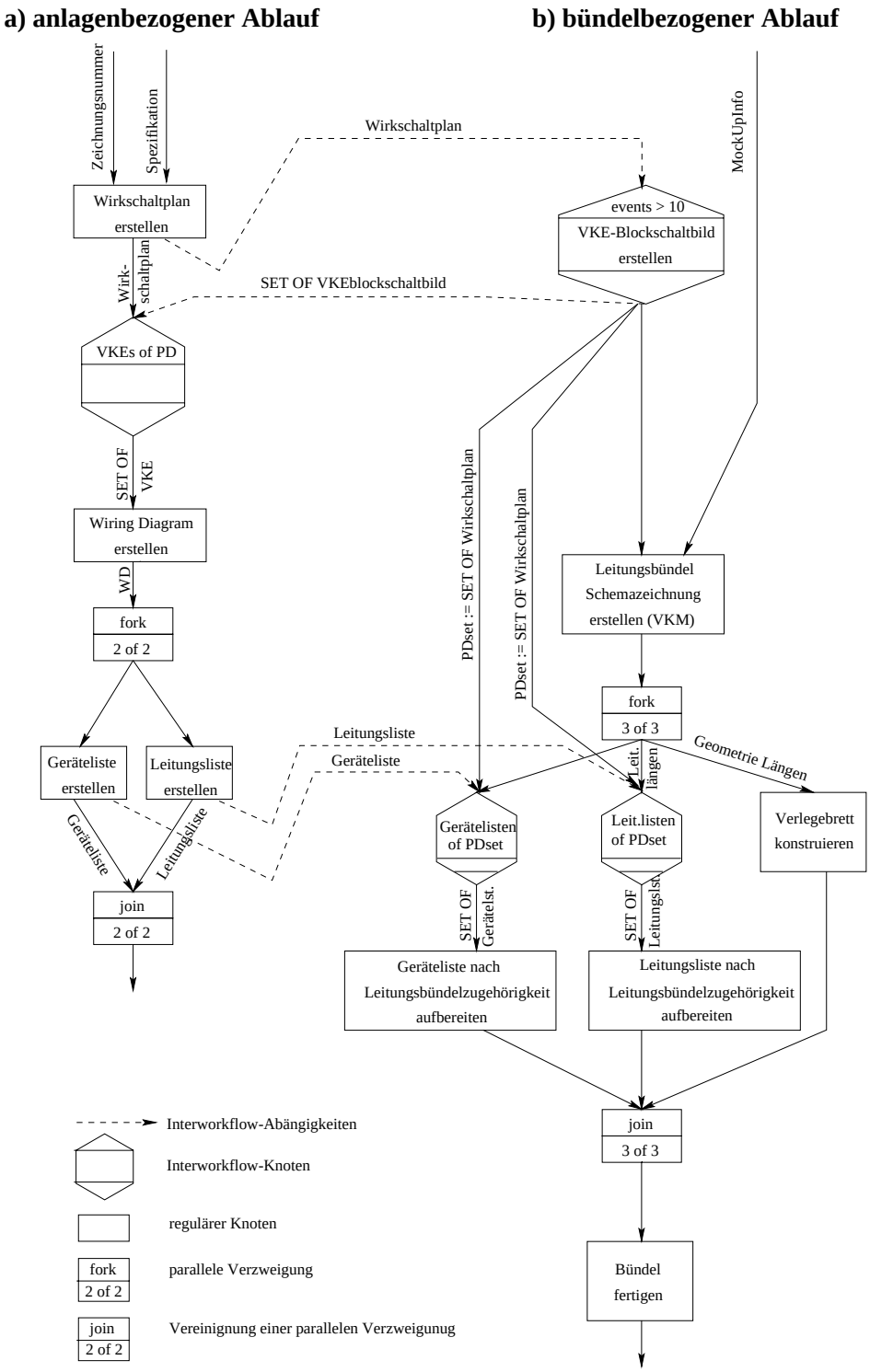


Abbildung 1: Beschreibung der Abläufe zur Konstruktion und Entwicklung von Leitungsbündeln

3.1 Concurrent–Engineering–spezifische Anforderungen

Das Beispiel aus Abschnitt 2 zeigt die folgenden anwendungsspezifischen Anforderungen, die typisch für viele Abläufe im Bereich des Concurrent Engineering sind:

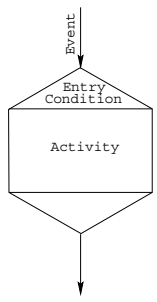
Wie aus den den bisher beschriebenen Abläufen ersichtlich, erfolgt die Konstruktion und Entwicklung eines Leitungsbündels stark sequentiell. Beispielsweise beginnt ein VKE–Blockschaltbild–Designer meist erst dann mit der Konstruktion eines neuen Leitungsbündels, wenn alle Wirkschaltpläne erstellt worden sind. Damit verzögert er viele anlagenbezogene Abläufe, da diese die Blockschaltbilder als Eingabedaten für Folgeschritte benötigen. Diese Verzögerungen sind unnötig, da ein Leitungsbündel nie Leitungen aus allen Anlagen enthält.

Zur Reduzierung der Konstruktionskosten und –zeiten sollte ein Blockschaltbild–Designer deshalb möglichst mit dem Eintreffen der ersten Wirkschaltpläne mit der Erstellung von Blockschaltbildern beginnen und diese für wartende Folgeschritte freigeben.

Damit besteht jedoch die Gefahr, daß der Designer beim Eintreffen weiterer Wirkschaltpläne die bereits fertiggestellten Blockschaltbilder modifizieren muß. Aus Konsistenzgründen müssen die Resultate von Folgeschritten, welche die „alten“ Blockschaltbilder direkt oder indirekt als Eingabedaten verwendet haben, rückgängig gemacht werden (kaskadierendes semantisches Rollback), und die Schritte erneut mit den aktuellen Daten wiederholt werden. Dies verursacht insbesondere bei interaktiven Schritten Mehrarbeit. Erschwerend kommt hinzu, daß die Bestimmung der von Datenänderungen betroffenen Teilschritte aufwendig und fehleranfällig ist, da sie häufig manuell erfolgt. Die Konsistenthaltung stellt deshalb in der Praxis ein schwerwiegendes Problem dar [35].

Durch den Einsatz eines WFMS kann trotz der skizzierten Probleme ein höherer Parallelitätsgrad erreicht werden, wenn das WFMS die „Aufräumungsarbeiten“ möglichst selbständig übernimmt: Dazu muß das WFMS eine *explizite Datenflußmodellierung* und *Protokollierung dieser Datenflüsse zur Laufzeit* anbieten, um dann alle von einer Datenmodifikation betroffen Folgeschritte automatisch bestimmen zu können. Ermöglicht das WFMS auch die *Modellierung von Interworkflow–Datenabhängigkeiten* (siehe unten und Abbildung 1), so können auch betroffene Folgeschritte anderer Abläufe erkannt werden. Ein geeignetes *Versionierungskonzept* [33, 53, 42], das auch verschiedene Qualitätsstufen [31, 53] unterstützt, kombiniert mit einem durch die Anwendung spezifizierten *Korrektheitsbegriff* [21, 18], ermöglicht außerdem eine (teil–) automatische Rücksetzung der „veralteten“ Bearbeitungsschritte.

Interworkflow–Datenabhängigkeiten lassen sich mittels *Interworkflow–Abhängigkeitsknoten (Interworkflow–Dependency–Knoten, ID-Knoten)* analog der untenstehenden Abbildung beschreiben, die als graphische Variante von ECA–Regeln angesehen werden können.



ID-Knoten wurden auch in Abbildung 1 zur Beschreibung der schon erwähnten Datenabhängigkeiten zwischen den *anlagenbezogenen* und den *bündelbezogenen* Abläufen verwendet. Ein ID–Knoten besteht aus drei Teilen: Die *Eingangsbedingung (Entry Condition)* ist ein boolescher Ausdruck über die Eingabeparameter des ID–Knotens und wird bei jedem eintreffenden *Ereignis (Event)* neu berechnet. Liefert sie „true“, so wird der mit *Activity* bezeichnete Teilschritt aktivierbar. Dieser Teilschritt entscheidet dann über die Fortsetzung dieses Workflows. Ein solcher Teilschritt muß sich dabei nicht von anderen im Workflow modellierten Aktivitäten unterscheiden. Insbesondere kann er beliebige Eingabe– und Ausgabeparameter besitzen sowie ein interaktiver oder automatischer Teilschritt sein. Es muß allerdings eine Abbildung seiner Rückgabewerte auf eine der Fortsetzungsaktionen

möglich sein. Als Fortsetzungsaktionen sind denkbar:

- **continue:** Das neu eingetroffene Ereignis ermöglicht ein Fortsetzen dieses Workflows.
- **reevaluate:** Das Ereignis reicht (noch) nicht zum Fortsetzen dieses Workflows aus.
- **regainControl:** Die Teilschritte, die bei früheren Fortsetzungen dieses Ablaufs bearbeitet wurden, müssen zurückgesetzt werden, weil sie auf veralteten Daten beruhen. Dies gilt auch für alle Teilschritte von "interworkflow–abhängigen" Abläufen.

Der Abschnitt 6 diskutiert verschiedene Architektur–Ansätze, wie die hier aufgestellten Concurrent–Engineering–spezifischen Anforderungen umgesetzt werden können.

3.2 Anwendungsspezifische Anforderungen aus anderen Bereichen

Auch in vielen anderen Einsatzgebieten existiert eine Reihe von anwendungsspezifischen Anforderungen, die sich sowohl auf die Modellierungsebene als auch auf die WFMS–Architektur auswirken. Im einzelnen werden skizziert: benötigte Strukturierungskonstrukte für den Kontrollfluß, Flexibilität in Ausnahmefällen, Datenflußmodellierung, Korrektheitsbedingungen sowie Zeitbeschränkungen. Manche dieser Anforderungen, wie zum Beispiel *Strukturierungskonstrukte für den Kontrollfluß* oder *die Einhaltung von Korrektheitsbedingungen* sind in vielen Anwendungsbereichen (Banken, Versicherungen, medizinisch–organisatorischer Bereich, Fertigung) zu finden. Anforderungen wie *Flexibilität* (medizinisch–organisatorische Abläufe, Decision Support) oder *Zeitbeschränkungen* (medizinisch–organisatorische Vorgänge, Fertigungsprozesse) werden dagegen nur in einzelnen Bereichen gestellt.

Bei *schwach strukturierten* Prozessen erfolgt die Koordination und letztendlich auch die Spezifikation der Abläufe durch die Anwender zur Laufzeit (Ad-hoc Workflows [23]). Bei diesen *relativ kurzdauernden* und nur *wenige Teilschritte* umfassenden Workflows ist nur eine *kleine Gruppe von Personen* beteiligt. Ad-hoc Workflows werden in der gleichen Form meist *nicht wiederholt*. Der Datenaustausch zwischen den einzelnen Teilschritten erfolgt häufig über den Austausch von *anwendungsspezifisch strukturierten Dokumenten*. Durch die fehlende (Vor–) Modellierung von Ad-hoc Workflows und die benutzergesteuerte Ablaufkoordination wird weder eine Ablaufmodellierungskomponente noch eine systemseitige Steuerung des Kontroll– und Datenflusses benötigt. Es genügen *Komponenten zur Modellierung der Organisationsstruktur* sowie *benutzerfreundliche Mechanismen für den Informationsaustausch* und *für die dynamische Auswahl* der nächsten Bearbeiter, wie sie zum Beispiel von Groupware–Systemen [29] geboten werden.

Stark strukturierte, sich häufig wiederholende Abläufe werden dagegen *vor* ihrer Ausführung modelliert. Abhängig von der Komplexität der Strukturierung muß deshalb die Buildtime–Komponente des WFMS *verschiedene und zum Teil recht komplexe Kontrollkonstrukte* (Parallelität n–aus–m–Verzweigungen, Schleifen, Reihen, ... [30, 43]) anbieten. Die *automatische Koordination und Überwachung* des vormodellierten Workflows zur Laufzeit ist bei diesen Workflow–Typen eine der Aufgaben des WFMS. Eine benutzerinitiierte Änderung des modellierten Ablaufs zur Laufzeit wird dagegen hier weniger gefordert beziehungsweise teilweise sogar explizit verboten (Beispiel: Überspringen/Auslassen des Gegenzeichnens einer weiteren Person bei der Kreditvergabe).

Dagegen ist bei *medizinisch–organisatorischen Anwendungen* diese *Flexibilität zur Laufzeit* eine existentielle Anforderung für die Einsatzfähigkeit eines WFMS [36, 57, 43]. Das WFMS muß es — in Ausnahmefällen — ermöglichen, Schritte auszulassen, zu vertauschen oder zu einem früheren Bearbeitungszustand im Ablauf zurückzukehren. Hierbei kann man zwei Arten von Ausnahmen unterscheiden: *vorhersehbare*, und damit vormodellierbare, Ausnahmen sowie *nicht*

vorhersehbare Ausnahmen. Die Modellierung der vorhersehbaren Ausnahmefälle kann prinzipiell zusammen mit der Modellierung des „Normalablaufs“ sowie mit den gleichen Konstrukten erfolgen. Um die Modellierungskomplexität zu reduzieren [30] und um auch zur Laufzeit zwischen Ausnahme– und Normalfall unterscheiden zu können [16], ist jedoch eine *Modellierung in mehreren Sichten* beziehungsweise mittels *unterscheidbaren Konstrukten* sinnvoll. Bei beiden Arten von Ausnahmen ist es Aufgabe des Systems, auf das Fehlen von Eingabedaten geeignet zu reagieren und die Datenkonsistenz zu sichern [16]. In manchen Anwendungsgebieten, so zum Beispiel bei medizinisch–organisatorischen Abläufen, werden außerdem Möglichkeiten zur Beschreibung von *Interworkflow–Abhängigkeiten* gefordert [37].

Bei sehr *datenintensiven Anwendungen* (Bilder, Video–Sequenzen, CAD–Zeichnungen) oder bei *weit verteilten* Abläufen (large workflows [44]) sind zusätzlich Konstrukte zur *expliziten Datenflußmodellierung* erforderlich, um nicht unnötig Daten zwischen den einzelnen Bearbeitungsschritten austauschen zu müssen (need–to–know Prinzip, [32, 52]).

Langdauernde Abläufe mit hohen Konsistenzansprüchen benötigen *Konstrukte zur Beschreibung von Korrektheitsbedingungen* [13, 39] und eine entsprechende Laufzeitumgebung zur Realisierung der modellierten Konsistenzanforderungen (*transactional workflows*) [18, 21]. Bei eng gekoppelten oder integrierten Systemen erfolgt die Einhaltung der geforderten Korrektheitsbedingungen in der Regel mit herkömmlichen Methoden (Two–Phase–Commit, Persistent Queues, TP–Monitore) [26]. Sollen auch autonome Teilschritte mit eventuell eigener persistenter Datenhaltung eingebunden werden, so ist die Gewährleistung von Konsistenzbedingungen mit erheblich höherem Aufwand verbunden (Kompensationsschritte, ...) [23, 39, 13].

Zeitbeschränkte Abläufe benötigen Konstrukte, um *relative* (minimaler/maximaler zeitlicher Abstand zwischen Teilschritten) und *absolute* (Start am ..., Ende bis ...) *Zeitgrenzen* modellieren zu können.

3.3 Allgemeine Anforderungen an eine WFMS–Architektur

Die Eignung eines WFMS für ein Anwendungsgebiet hängt nicht nur von der Modellierungsmächtigkeit seiner Buildtime–Komponente, sondern auch von den Eigenschaften seiner System–Architektur (Systemeigenschaften) ab:

- *Fehlertoleranz: Unternehmenskritische* Workflows [23] benötigen eine Systemverfügbarkeit von möglichst 100 %. Zentralistische WFMS–Architekturen [27, 40, 63, 20, 62] mit ihrem „single point of failure“ sind hierfür nur in Verbindung mit zusätzlichen Maßnahmen (Backup–Server [44]) geeignet. Dagegen stellen *Ad-hoc* Workflows aufgrund ihrer relativ kurzen Dauer sowie ihrer Übersichtlichkeit (wenige Teilschritte und beteiligte Personen) in der Regel nicht so hohe Anforderungen an Verfügbarkeit und Konsistenz.
- *Skalierbarkeit:* Da die Zahl der gleichzeitig aktiven Workflows im voraus nur sehr schwer abzuschätzen ist, sollte die Koexistenz mehrerer WF–Engines möglich sein, die jeweils einen (disjunkten) Teil der Workflows bearbeiten [44, 5, 6, 59, 55]. Insbesondere bei weit verteilten Workflows [44, 5] muß die Ausführungskontrolle von einer WF–Engine zu einer anderen weitergereicht werden können, wenn die zweite Engine für die als nächstes zu aktivierenden Teilschritte besser geeignet ist (z. B. lokale Ausführung möglich). Diese Fähigkeit zur Migration der Ausführungskontrolle muß auch zwischen WF–Engines verschiedener WFMS–Hersteller möglich sein (Interoperabilität [68]).

- *Antwortzeiten:* Bei Interaktionen mit dem WFMS zur Laufzeit werden in der Regel Antwortzeiten von weniger als 1 Sekunde gefordert. Die Antwortzeiten werden dabei wesentlich vom Synchronisationsaufwand bestimmt[5], der wiederum von zwei Aspekten abhängt:
  - Zahl der gleichzeitig aktionsberechtigten Personen: Bei strikten Synchronisationsanforderungen muß jede Bewegung im Ablaufgraphen mit allen (vorher und nachher) aktionsberechtigten Personen abgestimmt werden. Oftmals wird es jedoch genügen, (Zustands–) Änderungen im Ablauf zeitverzögert, eventuell erst auf Anforderung, an aktionsberechtigte Personen weiterzuleiten. Diese Personen besitzen dann nicht immer die aktuellsten Informationen über einen Prozeß.
  - Zahl der beteiligten Workflow–Engines: Erfolgt aus Fehlertoleranzgründen die Ablaufkoordination von mehreren Workflow–Engines gemeinsam, so benötigen sie zusätzliche Synchronisationsmechanismen [11].
- *API–Funktionen:* Über die vom Laufzeitsystem angebotene Schnittstelle kann ein Anwender Einfluß auf die Durchführung von Workflows nehmen (API–Funktionen). Diese Schnittstelle muß mindestens das Starten von Abläufen beziehungsweise von darin enthaltenen (interaktiven) Teilschritten umfassen. Häufig werden jedoch auch Funktionen zum Stoppen, Unterbrechen und Zurücksetzen von Prozessen sowie verschiedene Anfrage– und Überwachungsoperationen (Status, History, Monitoring eines Workflows) gefordert. *Kontrollorientierte* WFMS [58], die das Aufrufen externer Teilschritte ermöglichen, benötigen zusätzliche API–Funktionen zur Interaktion mit diesen externen Programmbausteinen [16].

Diese Systemeigenschaften bilden zusammen mit den geforderten Modellierungskonstrukten das anwendungsspezifische *Anforderungsprofil*.

4 Berücksichtigung anwendungsspezifischer Anforderungen

Aufgrund der oben skizzierten unterschiedlichen Anforderungsprofile ist bisher keines der auf dem Markt befindlichen beziehungsweise in der wissenschaftlichen Literatur vorgeschlagenen WFMS in allen Anwendungsgebieten (gleich gut) geeignet. Die heutige Situation ist vielmehr durch eine Vielzahl an die jeweiligen Anwendungsbereiche angepaßten WFMS mit unterschiedlichen *Funktionalitäten* und *Systemeigenschaften* gekennzeichnet [60, 58]. Das Hinzukommen weiterer Anwendungsgebiete und damit zusätzlicher Anforderungsprofile wird diese Situation eher noch verschärfen, weshalb ein in allen Anwendungsbereichen gleich gut geeignetes *„General–Purpose–WFMS“* auch in Zukunft mehr Wunschdenken als Realität bleiben wird. Die Unterschiede in den WFMS beschränken sich dabei nicht nur auf die Buildtime–Komponente, sondern führen häufig auch zu *unterschiedlichen WFMS–Architekturen* mit jeweils spezifischen Vor– und Nachteilen (z.B. bzgl. Skalierbarkeit, Fehlertoleranz). Anwender und Entwickler von WFMS stehen deshalb vor den folgenden zwei Problemen:

1. Welche Laufzeitumgebung ist bei *gegebenem* Anforderungsprofil am besten geeignet? Wie bestimmt man diese „optimale“ WFMS–Architektur? Anhand welcher Kriterien kann man verschiedene Laufzeitumgebungen (quantitativ) vergleichen?

   Diese Fragen stellen sich WFMS–Entwickler, wenn sie die Architektur eines vorhandenen WFMS restrukturieren beziehungsweise ein neues WFMS entwerfen möchten.

[5] Bei sehr datenintensiven Abläufen beeinflußt auch die Bereitstellung der notwendigen Eingabedaten für einen Teilschritt das Antwortzeitverhalten (siehe Abschnitt 6.1).

Für WFMS–Anwender ist eine Antwort auf diese Frage interessant, wenn sie bei der Festlegung des Anforderungsprofils die Folgen für die Workflow–Ausführungen abschätzen wollen: Welche Konsequenzen auf die Komplexität der Workflow–Ausführung hat die Hinzunahme eines weiteren Modellierungskonstrukts oder einer zusätzlichen Systemeigenschaft? Kann durch den Verzicht auf einen Teil der Anforderungen eine einfacher realisierbare Laufzeit–Umgebung verwendet werden?

2. Welche Klasse von Workflows (bzgl. Modellierungsmächtigkeit, Skalierbarkeit, Konsistenz, Antwortverhalten, ...) kann eine *gegebene* WFMS–Architektur (maximal) unterstützen?

   Für Anwender stellt sich diese Frage insbesonders dann, wenn ein schon im Einsatz befindliches WFMS weitere oder modifizierte Abläufe unterstützen soll. Neben der Frage, ob sich die weiteren Abläufe geeignet modellieren lassen, muß insbesondere möglichst früh entschieden werden können, ob die gegebene WFMS–Architektur die zusätzlichen Workflows zur Laufzeit bewältigen kann.

   Ein WFMS–Anbieter steht vor diesem Problem, wenn sein WFMS um zusätzliche Modellierungskonstrukte oder Systemeigenschaften erweitert werden soll, um zum Beispiel für weitere Anwendungsgebiete eingesetzt zu werden [30]. Er muß dann möglichst früh prüfen, ob beziehungsweise welche Modifikationen an der WFMS–Architektur notwendig sind.

Vergleichende Veröffentlichungen von WFMS beschränken sich in der Regel auf eine Beurteilung der jeweiligen Buildtime–Komponenten [60, 58, 23]. Damit sind Rückschlüsse auf die Laufzeitumgebung, wie sie zur Beantwortung der obigen Fragen zwingend erforderlich sind, nicht möglich. Deshalb ist beispielsweise ein Anwender, hat er sich bezüglich der gewünschten Modellierungsmächtigkeit festgelegt, auf aufwendige Vergleichstest angewiesen, um aus der Menge „gleich–mächtiger“ WFMS dasjenige mit der geeignetesten Laufzeitsystem–Architektur herauszufinden. Der Aufwand resultiert im wesentlichen daraus, daß *repräsentative* Workflows in mehreren WFMS vollständig modelliert und unter realen Bedingungen ausgeführt werden müssen. Damit kann man sich in der Praxis immer nur auf den Vergleich weniger Systeme beschränken.

5 Herausforderungen an die Forschung

Aus den oben genannten Gründen sind deshalb Untersuchungen wünschenswert, welche die praxisrelevanten WF–Klassen durch Festlegung der Modellierungskonstrukte und Systemeigenschaften beschreiben *und* für diese die „optimale“ WFMS–Architektur bestimmen. Die Suche nach der optimalen WFMS–Architektur reduziert sich damit auf die Bestimmung der geeigneten WF–Klasse, welche die geforderten Modellierungskonstrukte und Systemeigenschaften umfaßt (Abbildung des geforderten Anforderungsprofils auf das Anforderungsprofil der WF–Klasse). Auch der durch Frage 2 geforderte „Rückweg“ wäre dadurch möglich: Bei gegebener WFMS–Architektur können alle WF–Klassen und damit die Menge der möglichen Konstrukte und Systemeigenschaften bestimmt werden, die für diese Laufzeitumgebung geeignet sind.

Unseres Wissens existieren bisher jedoch noch keine Arbeiten, welche die Abhängigkeiten zwischen Modellierungsmächtigkeit und Systemeigenschaften auf der einen Seite sowie der dazu möglichen WFMS–Architekturen auf der anderen Seite untersucht haben.

Als Ergebnis einer wissenschaftlich fundierten Untersuchung dieser Abhängigkeiten sind folgende Lösungen denkbar:

Es existiert die General–Purpose–WFMS–Architektur, die optimal für alle WF–Klassen und damit für alle Anwendungsgebiete ist. Aufgrund der sich zum Teil widersprechenden Anforderungsprofilen ist dieses Ergebnis nicht zu erwarten.

Es ist wahrscheinlicher, daß für eine (hoffentlich) größere Zahl von WF–Klassen eine gemeinsame WFMS–Architektur gefunden werden kann, die sehr viele Anwendungsgebiete abdeckt. Da ein Multi–Purpose–WFMS nicht alle Anforderungsprofile erfüllen kann, werden aus einer solchen Untersuchung mehrere Multi–Purpose-WFMS resultieren, die jeweils für andere Anwendungsbereiche konzipiert sind. Multi–Purpose–WFMS–Architekturen werden allerdings immer einen mehr oder weniger guten Kompromiß für das jeweilige konkrete Einsatzgebiet darstellen.

Deshalb wird eine solche Untersuchung auch „Nischen–WFMS“ mit speziell für das jeweilige Anwendungsgebiet zugeschnittenen Architekturen hervorbringen.[6] Nischen–WFMS sind sinnvoll:

- bei *„einfachen“ Anwendungen* (z. B. einfach strukturierte Abläufe mit kurzen Ausführungszeiten, vgl. Abschnitt 3.2), die nur wenige Modellierungsmöglichkeiten und nur geringe Systemunterstützung zur Laufzeit benötigen. Der Einsatz eines weit mächtigeren Multi–Purpose–WFMS wäre hier viel zu aufwendig.
- bei *sehr anwendungsspezifischen und schwierig umsetzbaren Anforderungen* an ein WFMS. Eine hochspezialisierte und darauf abgestimmte WFMS–Architektur ist hier dann sogar die einzige Möglichkeit, die gestellten Anforderungen umzusetzen. WFMS–Architekturen, die für *large workflows* (siehe Abschnitt 3.2) geeignet sein müssen, können beispielsweise eine solche Spezialisierung erfordern.

Bei der Suche nach geeigneten WFMS–Architekturen für die jeweiligen WF–Klassen ist ein Vorgehen nach folgenden Schritten sinnvoll:

1. *Festlegung von praxisrelevanten WF–Klassen*
   Eine WF–Klasse besteht aus einer *maximalen Menge* von Modellierungskonstrukten und Systemeigenschaften. Maximalität bedeutet in diesem Zusammenhang, daß durch die Hinzunahme eines weiteren Modellierungskonstrukts beziehungsweise einer zusätzlichen Systemeigenschaft eine grundlegende Veränderung der System–Architektur erforderlich wird. Eine praxisorientierte Klasseneinteilung wird dabei die verschiedenen anwendungsspezifischen Anforderungen berücksichtigen.
2. *Erarbeitung verschiedener WFMS–Architektur–Alternativen für die WF–Klassen*
   Die Beschreibung einer WFMS–Architektur muß neben der physikalischen Verteilung der WF–Engines und ihrer funktionaler Komponenten im Netzwerk auch ihre Art der Zusammenarbeit (autonom, kooperativ) sowohl im fehlerfreien Fall als auch während Knoten– und Netzausfällen beinhalten. Zusätzlich muß auch die Einbindung externer Anwendungen (Klienten/Teilschrittprogramme) festgelegt werden, wobei hier insbesondere bezüglich der Konsistenzanforderungen der externen Anwendungen verschiedene Qualitätsstufen vorstellbar sind. Für die WF–Klasse „Concurrent Engineering“ wird im folgenden Abschnitt überlegt, wie sich die in Abschnitt 3.1 aufgestellten Anforderungen umsetzen lassen.
3. *Entwicklung eines Kriterienkatalogs zur systematischen und quantitativen Beurteilung verschiedener Laufzeitumgebungen für eine WF–Klasse*
   Der Kriterienkatalog muß mindestens die in Abschnitt 3.3 beschriebenen Systemeigenschaften umfassen. Quantitative Analysen sind unter anderem bei den Kriterien „Fehlertoleranz“ [10] und „Antwortzeiten“ (z. B. über Warteschlangenanalyse [54, 9, 61, 49, 8, 12],

[6] Diese Nischen–Systeme sind auch in anderen Bereichen der DV–Welt zu finden. Man denke hierbei nur an Echtzeitbetriebssysteme, die nur für besonders zeitkritische Anwendungen benötigt werden, an Single–User–Betriebssysteme für einfache Einzelplatzsysteme oder an Hochleistungstransaktionssysteme (z. B. Transaction Processing Facility (TPF) von IBM), die optimiert auf maximalen Transaktionsdurchsatz sind.

zeitbehaftete Petri–Netze [51, 9, 49, 34, 8, 65] oder stochastischer Graphen [9, 28]) möglich. Eine „Antwortzeiten"–Analyse kann dabei für jede vom Laufzeitsystem angebotene API–Funktion getrennt erfolgen. Hiermit lassen sich dann beim Vergleich verschiedener WFMS–Architekturen (siehe Punkt 4) die Häufigkeiten der Funktionsaufrufe mit berücksichtigen.

4. *Bestimmung der optimalen Laufzeitumgebungen mittels des Kriterienkatalogs*
   Da die optimale WFMS–Architektur immer einen Kompromiß zwischen den verschiedenen Anforderungen an eine Klasse darstellen wird, können je nach Priorisierung dieser Anforderungen unterschiedliche optimale WFMS–Architekturen resultieren. Alternativ ist auch denkbar, daß eine Änderung der Priorisierung zu weiteren WF–Klassen führt, wobei dann die Gefahr der „WF–Klassen–Explosion" besteht.

Zu beachten ist hierbei, daß sich die Mächtigkeit einer WF–Klasse (Punkt 1) und die optimale Laufzeitumgebung (Punkt 4) gegenseitig beeinflussen. Deshalb stellt die Bestimmung der Klassifikationsgrenzen einen iterativen Prozeß dar. Auch kann es für die Bildung von Multi–Purpose–WFMS sinnvoll sein, WF–Klassen mit ähnlichen Architekturen zusammenzufassen, um das Einsatzgebiet des WFMS zu vergrößern.

6 Überlegungen zu einer Concurrent–Engineering–geeigneten WFMS–Architektur

Dieser Abschnitt enthält Überlegungen zu einer WFMS–Architektur, die den Concurrent–Engineering–spezifischen Anforderungen aus Abschnitt 3.1 genügt. Dabei werden aus den Anforderungen resultierende Probleme zur Laufzeit skizziert und mögliche Lösungsansätze beschrieben. Außerdem wird dieser Abschnitt aufzeigen, in welchen Bereichen noch weiterer Forschungsbedarf besteht.

6.1 Zeitgerechte Bereitstellung der benötigten Eingabedaten

Eine bei interaktiven Teilschritten zeitkritische Aufgabe eines WFMS ist die Bereitstellung der notwendigen Eingabedaten für die einzelnen Bearbeitungsschritte. Entscheidet sich ein Anwender, einen ihm in seiner Worklist angebotenen Teilschritt (in der Regel durch „Anklicken" des entsprechenden Items) durchzuführen, so müssen diesem Teilschritt *innerhalb weniger Sekunden* die von ihm benötigten Daten zur Verfügung gestellt werden. Diese Daten–Bereitstellung ist insbesondere bei Concurrent–Engineering–Abläufen aus den folgenden Gründen problematisch:

1. Die bereitzustellenden Datenmengen sind in der Regel groß.
2. Die Daten weisen unterschiedliche Formate auf.
3. Die Daten müssen teilweise über große Entfernungen transferiert werden, da bei typischen Concurrent–Engineering–Abläufen häufig Mitarbeiter aus anderen, räumlich weit verteilten Abteilungen, Niederlassungen oder kooperierenden Firmen beteiligt sind (large workflows [44]).[7]
4. Die Eingabedaten eines Teilschritts müssen an verschiedenen Orten bereitgestellt werden, da der Schritt oftmals mehreren Mitgliedern (eines Teams) zeitgleich zur Durchführung angeboten wird.

[7] Beispielsweise erfolgt die Airbus–Entwicklung über mehrere europäische Länder verteilt.

Unter Verwendung eines geeigneten Kommunikationsstandards zum Datenaustausch läßt sich die Datenkonvertierungsproblematik (Punkt 2) weitgehend lösen. Für den im Concurrent–Engineering–Bereich benötigten Austausch von Produktdaten eignet sich hierbei insbesondere die von der ISO entwickelte Norm STEP (STandard for the Exchange of Product model data) [41], die bei technischen Anwendungen eine immer größere Verbreitung findet.

Für die Speicherung der großen Datenmengen (Punkt 1) gibt es für ein WFMS drei prinzipielle Möglichkeiten: beim Erzeuger der Daten, bei den potentiellen Verbraucher–Teilschritten[8] oder an einer unabhängigen Zwischenstelle. Die Berücksichtigung der Punkte 3 und 4 deckt jedoch bei jedem dieser Ansätze Schwachstellen auf:

Bei einer Speicherung auf dem Knoten des *Erzeuger–Teilschritts* beziehungsweise an einer *Zwischenstelle* lassen sich tolerierbare Zugriffszeiten nur dann realisieren, wenn eine *schnelle* und *ausfallsichere* Verbindung zwischen den Daten und den *entfernten* Verbraucher–Teilschritten gewährleistet werden kann. Diese Anforderung wird bei den heutigen Netzinfrastrukturen in der Regel nur im LAN–Bereich beziehungsweise bei einigen Hochleistungs–MAN [14] erfüllt. Bei einer größeren geographischen Verteilung der an einem Ablauf beteiligten Personen (Punkt 3) stoßen diese Ansätze an ihre Grenzen. Die Speicherung an einer zentralen Zwischenstelle birgt zusätzlich die Gefahr eines Flaschenhalses.

Zur Vermeidung dieser Verfügbarkeits– und Zugriffszeitenprobleme bietet sich bei einer größeren örtlichen Verteilung eine Speicherung der Daten auf allen Verbraucherknoten an (*Daten–Prefetching*). Verbraucherknoten sind alle Knoten, auf denen Teilschritte, die diese Daten als Eingabeparameter benötigen, potentiell ausgeführt werden können. Dadurch kann die Datenübertragung erfolgen, bevor die entsprechenden Teilschritte ihren potentiellen Bearbeitern angeboten und damit ausgeführt werden können [52, 32]. Diese Übertragung ist deshalb *nicht* zeitkritisch. Problematisch bei dieser Lösung ist jedoch, daß die Daten auf viele Knoten kopiert werden müssen, da diese Rechner alle für die Durchführung von Folgeschritten in Frage kommen. Dies ist immer dann der Fall, wenn ein Folgeschritt mehreren Anwendern angeboten wird (Punkt 4) und/oder wenn es mehrere verschiedene Folgeschritte gibt (vergleiche Flexibilität in Abschnitt 3.2). Eine große Zahl von potentiellen Anwendern erhöht somit die Zahl der zu versendenden Datenkopien und reduziert damit die Praxistauglichkeit dieser Lösung. Die aktuelle Zahl der geeigneten Anwender steht dabei erst zur Laufzeit fest, da sie von der Anzahl der gerade im WFMS angemeldeten Anwender für die jeweilige Rolle abhängt.

Um die skizzierten Nachteile der verschiedenen Ansätze abzuschwächen, müssen die bisherigen Modelle sicherlich noch verfeinert und mit mehr Anwendungssemantik versehen werden. Hierbei sind unter anderem folgende Verbesserungen auf ihre Eignung zu untersuchen:

1. *Daten–Clusterung:* Geographisch benachbarte Bearbeiter teilen sich eine Datenkopie. Ein Cluster besteht aus allen Rechnerknoten, die über ein schnelles und zuverlässiges Netzwerk (LAN/MAN) verbunden sind. Jeder Cluster erhält dann eine Datenkopie. Damit hängt die Zahl der zu versendenden Kopien von der Anzahl der Cluster und nicht von der im allgemeinen größeren Zahl der potentiellen Benutzer ab (Lokalitätsprinzip). Dieser hybride Ansatz ermöglicht eine Datenverfügbarkeit und Zugriffszeiten, die vergleichbar zur Lösung der Datenspeicherung auf *allen* Verbraucherknoten sind, ohne den Nachteil dieser Lösung, den hohen Kopieraufwand, im vollen Umfang zu übernehmen. Durch eine entsprechende Wahl der Clustergröße sind dabei die Speicherung an einer Zwischenstelle (Cluster umfaßt *alle* Knoten) und die Speicherung auf allen Verbraucherknoten (jeder Verbraucherknoten

[8] Mit Verbraucher–Teilschritten werden Schritte bezeichnet, die diese Daten als Eingabeparameter benötigen. Ein Verbrauchen im eigentlichen Sinn findet natürlich nicht statt.

bildet einen eigenen Cluster) als Spezialfälle enthalten.

2. *Reservierung von Teilschritt–Durchführungen:* Den potentiellen Bearbeitern eines Teilschritts wird als weitere Operation eine *Reservierungsfunktion* angeboten. Durch das Anstoßen dieser Operation verpflichtet sich der Bearbeiter, die Tätigkeit zu übernehmen. Der Aufruf des Teilschritts zur Durchführung der entsprechenden Tätigkeit erfolgt jedoch erst zu einem späteren Zeitpunkt. Damit kann auf ein bei vielen potentiellen Bearbeitern kostspieliges Prefetching der Daten verzichtet werden, da in der Regel — entsprechende Arbeitsorganisation der Benutzer vorausgesetzt — zwischen Reservierung und Durchführung eines Teilschritts genügend Zeit zur Datenübertragung bleibt. Auch kurze Verbindungsausfälle können dadurch toleriert werden. Nach der Datenübertragung kann dann der Mitarbeiter autark den Teilschritt durchführen. Dieses Vorgehen eignet sich deshalb auch für die Bearbeitung eines Teilschritts auf *mobilen Rechnern*, die nur zeitweise mit dem WFMS verbunden sind [3]. Eine Reservierung sollte jedoch zeitlich begrenzt werden, um den Teilschritt nach Ablauf dieser Frist anderen Bearbeitern erneut zur Durchführung anzubieten. Dadurch wird einer unnötig langen Verzögerung eines Workflows vorgebeugt, wenn der Mitarbeiter, der einen Teilschritt reserviert hat, daran gehindert wird, den Schritt fristgerecht durchzuführen (z.B. wegen Krankheit, Rechnerausfall). Andererseits sind hiermit Inkonsistenzen durch Mehrfachbearbeitungen möglich, wenn aufgrund von Verbindungsunterbrechungen die Beendigungsmeldung eines Teilschritts nicht an das WFMS übermittelt werden kann. *Replikationsverfahren* [11] können solche Inkonsistenzen erkennen und gegebenenfalls automatisch auflösen. Bei einer automatischen Inkonsistenzauflösung ist auch der Einsatz der Konzepte aus dem Bereich *erweiterter Transaktionsmodelle* (z.B.: Kompensation [21], acceptable state set [38]) sinnvoll. Jedoch müssen alle diese Vorschläge bezüglich ihrer Anwendbarkeit für den Concurrent–Engineering–Bereich untersucht und entsprechend angepaßt werden.
3. *Zeitverzögerte Versorgung mit Eingabeparametern:* Die Durchführung von Teilschritten im Concurrent–Engineering–Bereich (z.B. *„VKE–Blockschaltbild erstellen“*, siehe Abschnitt 2) ist häufig eine längere Tätigkeit, die beim Start nicht sofort alle Eingabedaten benötigt. Entsprechende Schlüsselworte (`immediate, deferrable`) bei der Deklaration der Eingabeparameter eines Teilschritts ermöglichen eine Reduzierung der Eingabedaten, die beim Start des Teilschritts vom WFMS zur Verfügung gestellt werden müssen.
4. *Datenversionierung:* Ist es aus Anwendungssicht tolerierbar, daß ein Teilschritt beim Aufruf mit einer veralteten Datenversion versorgt wird, so kann auch damit eine Reduzierung des benötigten Datenvolumens erreicht werden. Auch dieses anwendungsspezifische Wissen kann über geeignete Schlüsselattribute dem WFMS mitgeteilt werden. Hier stellt sich die Frage, welche Klasse von Teilschritten welche Höhe der dadurch möglichen Inkonsistenzen toleriert. Welches Korrektheitskriterium eignet sich deshalb zur Beschreibung der erlaubten Inkonsistenzen (z.B.: ϵ–Serialisierbarkeit [50])?

Bei allen skizzierten Verbesserungsvorschlägen muß untersucht werden, welche *„Stellschrauben“* (z.B.: Anzahl und Größe der Cluster, Zeitraum einer Reservierung, Höhe der erlaubten Inkonsistenzen) in der Praxis sinnvoll sind und welche durch Ablauf–Modellierer oder Bearbeiter modifiziert werden dürfen.

6.2 Überwachung der Interworkflow–Abhängigkeiten

Interworkflow–Abhängigkeiten, wie sie in Abschnitt 3.1 beschrieben worden sind, ergeben sich erst zur Laufzeit. Eine systemseitige Unterstützung dieser Interworkflow–Abhängigkeiten kann deshalb nur über eine allen Workflow–Instanzen bekannte *Vermittlerstelle*, beispielsweise über einen allgemein zugänglichen *aktiven Daten–Pool* (`shared data pool`, siehe auch Abbildung 2) erfolgen. Im wesentlichen besteht die Aufgabe der Vermittlerstelle darin, datenerzeugende Workflow–Instanzen und datenverbrauchende Workflow–Instanzen über ihre Interworkflow–Abhängigkeiten zu informieren. Dazu müssen in der Vermittlerstelle jedoch nicht die auszutauschenden Datenobjekte selbst, sondern nur Informationen über die auszutauschenden Objekte, sogenannte *Metadaten*, gespeichert werden. Der eigentliche Datenaustausch kann dadurch weiterhin ohne Zwischenspeicherung in der Vermittlerstelle direkt zwischen den involvierten Workflow–Instanzen erfolgen.

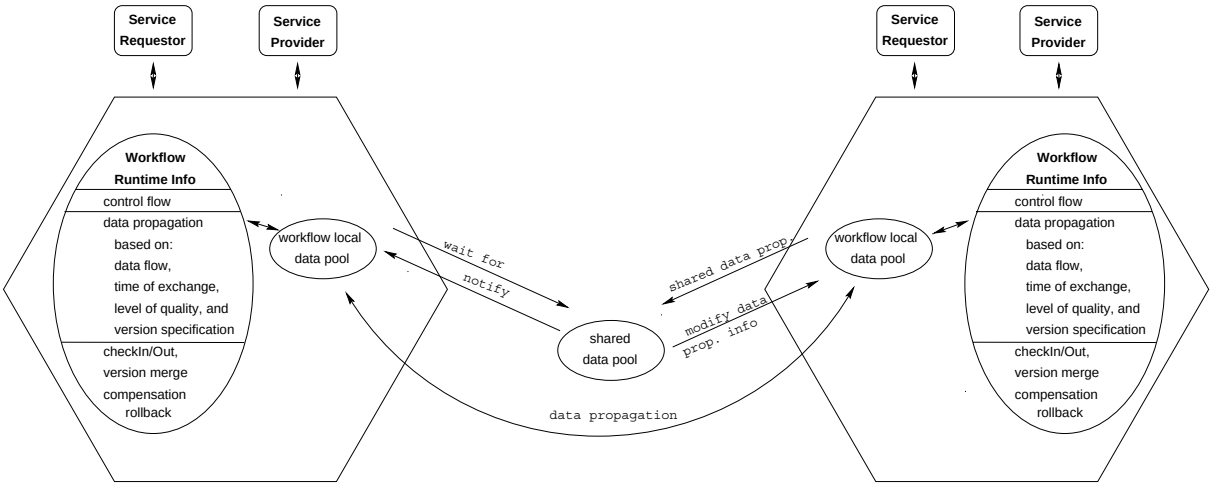


Abbildung 2: Überwachung der Interworkflow–Abhängigkeiten

Im einzelnen muß eine solche Vermittlerstelle die folgenden Operationen anbieten:

Mit der Operation `wait for` kann eine Workflow–Instanz ihr Interesse an bestimmten Datenobjekten anmelden. Treffen die entsprechenden Metadaten bei der Vermittlerstelle ein (Operation `shared data propagation`), so werden von ihr die darauf wartenden Workflow–Instanzen benachrichtigt (Operation `notify`) und die entsprechenden datenerzeugenden Workflows beauftragt, diese Daten auch an die wartenden Instanzen weiterzuleiten (Operation `modify data propagation info`).

Der Vorteil dieses *publish–and–subscribe*–Prinzips [66, 24] liegt in der Entkoppelung von Datenerzeuger und –verbraucher. Damit ist die Bestimmung, an welche Workflow–Instanzen die Daten weitergeleitet werden müssen, nicht Aufgabe des „Erzeuger–Workflows“, sondern erfolgt durch die Vermittlerstelle.

Bei diesem zentralistischen Lösungsansatz muß sicherlich noch untersucht werden, ob er sich auch für Interworkflow–Abhängigkeiten zwischen geographisch weit verteilten Workflow–Instanzen eignet (vergleiche Kritikpunkte an zentralistischen Lösungen bei weit verteilten Workflows in [4]). Im Gegensatz zu einer zentralen Verwaltung der Datenflüsse *innerhalb* eines weit verteilten Workflows ist ein zentraler Ansatz bei *Interworkflow*–Datenabhängigkeiten jedoch aus drei Gründen weniger kritisch:

- Die Zahl von Interworkflow–Datenabhängigkeiten ist geringer als die Datenabhängigkeiten von Teilschritten innerhalb eines Workflows.
- Über die Vermittlerstelle werden nur Informationen *über* die auszutauschenden Daten (Metadaten) und nicht die Daten selbst ausgetauscht. Die auszutauschende Datenmenge ist dadurch weit geringer.
- Der Datenaustausch kann erfolgen, bevor den Anwendern der aufgrund von Interworkflow–Abhängigkeiten blockierte Schritt angeboten wird (vergleiche Abschnitt 6.1).

6.3 Einbindung autonomer Teilschritt–Programme

Bei Workflows im Concurrent–Engineering–Bereich werden Teilschritte häufig mit bereits früher entwickelten Anwendungssystemen (z.B.: CAx–Systeme) verknüpft. Diese Anwendungssysteme werden meist als autonome Stand–alone–Applikationen entwickelt. Eine Integration dieser autonomen Teilschritt–Programme in ein übergeordnetes externes System, wie zum Beispiel ein WFMS, ist deshalb meist nicht vorgesehen.

Dies erschwert die Realisierung der in Abschnitt 3.1 geforderten automatisch durchzuführenden Aufräumungsarbeiten, weil aufgrund der fehlenden Integrationsmöglichkeiten ein atomares Zurücksetzen der vom WFMS verwalteten Daten *und* der (persistenten) Daten der entsprechenden Teilschritt–Programme verhindert wird (keine *globale Transaktion* möglich, welche „WFMS–Daten“ und „Teilschritt–Programm–Daten“ umfaßt) [7]. Das Zurücksetzen der Teilschritt–Programm–Daten erfolgt deshalb üblicherweise durch den Aufruf eines entsprechenden Kompensationschritts[9], der unabhängig zu den WFMS–internen Aufräumungsarbeiten erfolgt. Bei Auftreten von Kommunikations– oder Knotenfehlern sind damit Inkonsistenzen möglich. Beispielsweise können die WFMS–Daten schon auf einen früheren Bearbeitungszustand zurückgesetzt worden sein, der Aufruf des Kompensationsschritts zum Zurücksetzen der Teilschritt–Programm–Daten schlug dagegen fehl.

Um die daraus resultierenden Inkonsistenzen zu verhindern, muß es dem WFMS ermöglicht werden, den konkreten Bearbeitungszustands eines Teilschritts (kompensiert, erfolgreich beendet, ...) beim zugeordneten Anwendungssystem zu erfragen, wozu das entsprechende Schritt–Programm jedoch einen Teil seiner Autonomie aufgeben muß. Ein alternativer Ansatz besteht in einer *idemponenten* Realisierung der Kompensationsschritte, damit auch mehrmaliges Aufrufen der Kompensation keine weiteren Inkonsistenzen verursacht [6].

Wann sich welche Alternative besser eignet, hängt im wesentlichen von den Eigenschaften des jeweiligen Teilschrittprogramms ab. Diese Eigenschaften können stark unterschiedlich sein: Bereitschaft zum Two–Phase–Commit, Abfragemöglichkeiten bestimmter interner Bearbeitungszustände, Möglichkeit zum Abbrechen eines Auftrags, usw. Sind diese dem WFMS bekannt, so können sie zur Konsistenzsicherung geeignet ausgenützt werden. Dazu müssen jedoch Kriterien erarbeitet werden, nach denen die verschiedenen Teilschrittprogramme entsprechend ihrer Eigenschaften eingeteilt werden können.

[9] Kompensation ist auch aufgrund der langen Workflow–Bearbeitungsdauer notwendig [22], da diese eine Konsistenzsicherung mittels (Zwei–Phasen–) Sperren über den gesamten Zeitraum aus Effizienzgründen unmöglich macht.

7 Fazit

WFMS erobern sich neue Anwendungsgebiete mit jeweils spezifischen Anforderungen an die Mächtigkeit dieser Systeme. Neben den Anforderungsprofilen aus unterschiedlichen Anwendungsgebieten wurden hier die spezifischen Anforderungen aus dem Bereich des Concurrent Engineering exemplarisch aufgezeigt und die daraus resultierenden Folgen für eine geeignete WFMS–Architektur beschrieben. Hierbei wurde deutlich, daß bei der Realisierung einer Concurrent–Engineering–geeigneten WFMS–Architektur noch viele offene Probleme existieren, die weiterer Forschung bedürfen.

Allgemein läßt sich sagen, daß die Berücksichtigung anwendungsspezifischer Anforderungen nicht nur zu einer großen Zahl unterschiedlich mächtiger Buildtime–Komponenten führt, sondern auch zu verschiedenen Alternativen für mögliche Laufzeitumgebungen (Runtime–Komponenten).

Bisher fehlen jedoch fundierte Untersuchungen, wie die geeignete WFMS–Architektur für ein vom Anwendungsbereich vorgegebenes Anforderungsprofil auszusehen hat beziehungsweise anhand welcher Kriterien verschiedene alternative WFMS–Architekturen verglichen werden können. Neben einem Kriterienkatalog zur Beurteilung von WFMS–Architekturalternativen müssen solche Untersuchungen eine Klassifikation verschiedener Laufzeitumgebungen liefern. Jeder WF–Klasse (Menge von Konstrukten und Systemeigenschaften) sollen dabei eine oder mehrere WFMS–Architekturen zugeordnet sein, die für diese Klasse am besten geeignet sind.

Der praktische Nutzen solcher Untersuchungen wäre vielfältig: Ein WFMS–Anwender ist bei der Auswahl eines geeigneten WFMS nicht mehr auf aufwendige Vergleichstests angewiesen, sondern kann mit Hilfe der Kriterien die unterschiedlichen WFMS–Architekturen vergleichen und so das am besten geeignete WFMS bestimmen. Durch Abbilden seiner Anforderungen auf eine geeignete WF–Klasse ist er außerdem in der Lage, den Realisierungsaufwand und die Ausführungskomplexität des benötigten WFMS abzuschätzen. Entwickler von WFMS können einerseits anhand der Klassifikation die optimale Architektur für eine vorgegebene Menge von Anforderungsprofilen bestimmen und anderseits feststellen, um welche Konstrukte und Systemeigenschaften ihr bisheriges WFMS erweitert werden kann, ohne die Laufzeitumgebung grundlegend verändern zu müssen.

Danksagung

Diese Arbeit entstand im Rahmen eines Kooperationsprojekts mit dem Daimler-Benz-Forschungszentrum Ulm, Abteilung Produktionsinformatik (F3P). Wir möchten uns hiermit insbesondere beim Leiter dieser Abteilung, Herrn Dr. Haban, sowie bei seinen Mitarbeitern Herrn Ortiz Bernal und Herrn Feltes für die gute und immer kooperative Zusammenarbeit bedanken. Bei der Entstehung dieses Berichts gebührt Herrn Feltes unserer besonderer Dank. Seine Gesprächsbereitschaft und fachliche Kompetenz hat die Beschreibung der hier skizzierten Abläufe und Probleme aus der Anwendungsdomäne „Concurrent Engineering“ ermöglicht.

Auch wollen wir uns an dieser Stelle bei den Teilnehmern des im April 1995 stattgefunden „PEFF–Workshops *Concurrent Engineering*“, insbesondere bei Herrn Dr. Krammer und Herrn Vilsmeier, Daimler–Benz Aerospace Ottobrunn, bedanken. Die dort stattgefundenen Vorträge und Gespräche haben uns einen tieferen Einblick in die in der Praxis auftretenden Probleme der Flugzeugentwicklung bei der Daimler–Benz Aerospace gegeben.

Außerdem danken wir Herrn Heinlein und Herrn Reichert, Universität Ulm, für ihre wertvollen Hinweise und Ratschläge bei der Entstehung dieser Ausarbeitung.

Literatur

[1] R. Admomeit, W. Deiters, B. Holtkamp, F. Schülke, and H. Weber. K/2r: A kernel for the ESF software factory support environment. In *Proc. 2nd Int'l Conf. on Systems Integration*, pages 325–336, Morristown, New Jersey, 1992.

[2] D. Agrawal, J. L. Bruno, A. E. Abbadi, and V. Krishnaswamy. Managing concurrent activities in collaborative environments. In CoopIS [15], pages 99–110.

[3] G. Alonso, R. Günthör, M. Kamath, D. Agrawal, A. E. Abbadi, and C. Mohan. Exotica/FMDC: Handling disconnected clients in a workflow management system. In CoopIS [15], pages 99–110.

[4] G. Alonso, M. Kamath, D. Agrawal, A. E. Abbadi, R. Günthör, and C. Mohan. Failure handling in large scale workflow management systems. Research Report RJ9913, IBM, Nov. 1994.

[5] G. Alonso, C. Mohan, R. Günthör, D. Agrawal, A. E. Abbadi, and M. Kamath. Exotica/FMQM: A persistent message-based architecture for distributed workflow management. In *Proc. IFIP WG8.1 Working Conference on Information Systems for Decentralized Organizations*, Trondheim, Aug. 1995.

[6] D. Barbará, S. Mehrotra, and M. Rusinkiewicz. INCAS: Managing dynamic workflows in distributed environments. Technical report, Matsushita Information Technology Labratory, Princeton N. Y, May 1994.

[7] T. Bauer. Realisierung einer Kommunikationsinfrastruktur für sichere, verteilte Anwendungen. Master's thesis, Universität Ulm, Fakultät für Informatik, 1995.

[8] F. Bause and M. Sczittnicj. Design von Modellierungstools zur Leistungsbewertung: HIT, MACOM QPN–Tool. *it + ti, Informationstechnik und Technische Informatik*, 37(3):34–40, June 1995.

[9] H. Beilner. Werkzeuge zur modellgestützten Leistungsbewertung. *it + ti, Informationstechnik und Technische Informatik*, 37(3):5–9, June 1995.

[10] F. Belli, K. Echtle, and W. Görke. Methoden und Modelle zur Fehlertoleranz. *GI Informatik Spektrum*, 19:68–81, 1986.

[11] T. Beuter and P. Dadam. Prinzipien der Replikationskontrolle in verteilten Systemen. Ulmer Informatik Berichte 95–11, Universität Ulm, Nov. 1995.

[12] G. Bolch, M. Roessler, and R. Zimmer. Leistungsbewertung mit PEPSY–QNS and MOSES. *it + ti, Informationstechnik und Technische Informatik*, 37(3), June 1995.

[13] Y. Breitbart, A. Deacon, H.-J. Schek, A. Sheth, and G. Weikum. Merging application–centric and data–centric approaches to support transaction–oriented multi–system workflows. *ACM SIGMOD Record*, 22(3):23–30, Sept. 1993.

[14] B. Butscher, L. Henckel, and T. Luckenbach. Die Kommunikationsplattform BERKOM für multimediale Anwendungen. *GI Informatik Spektrum*, 14:261–269, 1991.

[15] *Proc. 3rd Int. Conf. on Cooperative Information Systems*, Vienna, Austria, May 1995.

[16] P. Dadam, K. Kuhn, M. Reichert, T. Beuter, and M. Nathe. ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen. In GI [25], pages 677–686.

[17] U. Dayal, M. Hsu, and R. Ladin. A transactional model for long–running activities. In *Proc. 17th Int'l Conf. on Very Large Data Bases (VLDB)*, pages 113–122, Barcelona, Spain, Sept. 1991. Morgan Kaufmann.

[18] Special issue on workflow and extended transaction systems. *IEEE Data Engineering Bulletin*, 16(2), June 1993.

[19] *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*, Valencia, Spain, 1992. Springer.

[20] J. Eder and W. Liebhart. The workflow activity model WAMO. In CoopIS [15], pages 87–98.

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

[22] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem. Modeling long–running activities as nested sagas. *IEEE Data Engineering Bulletin*, 14(1), Mar. 1991.

[23] D. Georgakopoulos, M. F. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.

[24] R. Gerstner. Ereignisse als Basis einer workfloworientierten Architektur von Anwendungssystemen. In *Proc. GI Fachtagung MobIS'95*, Bamberg, Germany, Oct. 1995.

[25] *Proc. GI–Jahrestagung*, Zürich, Sept. 1995. Springer.

[26] J. Gray and A. Reuter. Transaction processing monitors: An overview. In *Transaction Processing: Concepts and Techniques*, chapter 5, pages 239–290. Morgan Kaufmann, 1993.

[27] V. Gruhn. Entwicklung von Informationssystemen in der LION–Entwicklungsumgebung. In G. Scheschonk and W. Reisig, editors, *Petri–Netze im Einsatz für Entwurf und Entwicklung von Informationssystemen*. Springer, 1993.

[28] F. Hartleb, P. Dauphin, R. Klar, A. Quick, and M. Siegle. Modellierung und Meßunterstützung mit dem Werkzeug PEPP. *it + ti, Informationstechnik und Technische Informatik*, 37(3):41–47, June 1995.

[29] H. Hartstock, M. Kleppel, R. Kossow, M. Tusche, and J. Wege. *Lotus Notes, Das Kompendium, Einführung, Arbeitsbuch, Nachschlagewerk*. Markt & Technik Buch– und Software Verlag, 1995.

[30] S. Jablonski. *Workflow–Management–Systeme, Modellierung und Architektur*. Thomson, 1995.

[31] S. Jablonski, S. Barthel, T. Kirsche, T. Rödinger, H. Schuster, and H. Wedekind. Datenbankunterstützung für kooperative Gruppenarbeit. *Informationstechnik und Technische Informatik*, 35(1):34–44, 1993.

[32] S. Jablonski, T. Ruf, and H. Wedekind. Implementation of a distributed data management system for technical applications– a feasibility study. *Information Systems*, 15(2):247–256, 1990.

[33] R. D. Katz. Toward a unified framework for version modeling in engineering databases. *ACM Computing Surveys*, 22(4):375–408, Dec. 1990.

[34] C. Kelling, R. German, A. Zimmermann, and G. Hommel. TimeNET — ein Werkzeug zur Modellierung mit zeiterweiterten Petri–Netzen. *it + ti, Informationstechnik und Technische Informatik*, 37(3):21–27, June 1995.

[35] Krammer, Altmeyer, and Holz. PEFF: Prozeßnetzwerk Entwicklung und Fertigung im Flugzeugbau, Ergebnisbericht Nr. 2, IST–Prozeßerfassung und Schwachstellen. Technical report, Daimler–Benz Aerospace, Daimler–Benz Forschung und Technik, Produktionsinformatik (F3P), Sept. 1993.

[36] K. Kuhn, M. Reichert, and P. Dadam. Unterstützung der klinischen Kooperation durch Workflow-Management-Systeme: Anforderungen, Probleme, Perspektiven. In H. J. Trampisch and S. Lange, editors, *40. Jahrestagung der GMDS: Medizinische Forschung — Ärztliches Handeln*, pages 437–441, München, 1995.

[37] K. Kuhn, M. Reichert, and P. Dadam. Using workflow management systems in clinical environments, a critical analysis. Submitted for publication, 1996.

[38] Y. Leu. Composing multidatabase applications using flexible transactions. *IEEE Data Engineering Bulletin*, 14(1), Mar. 1991.

[39] F. Leymann. Supporting business transactions via partial backward recovery in workflow management systems. In *Proc. GI–Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (BTW)*, Dresden, Germany, Mar. 1995. Springer, Informatik aktuell.

[40] F. Leymann and W. Altenhuber. Managing business processes as an information resource. *IBM Syst. J.*, 33(2):326–348, 1994.

[41] H. Lührsen. ISO–Norm STEP/EXPRESS — Einsatz objektorientierter Technologie in der Produktdatenverwaltung. In *Innovative Softwaretechnologien: Neue Wege mit objektorientierten Methoden und Client/Server–Architekturen*, pages C614.01–C614.15. S. Jänichen, 1994.

[42] D. R. McCarthy and S. K. Sarin. Workflow and transactions in InConcert. In DEB [18], pages 53–56.

[43] J. Meyer. Anforderungen an zukünftige Workflow–Management–Systeme: Flexibilisierung, Ausnahmebehandlung und Dynamisierung — Erörterung am Beispiel medizinisch–organisatorischer Abläufe. Master's thesis, Universität Ulm, Fakultät für Informatik, 1996.

[44] C. Mohan, G. Alonso, R. Günthör, and M. Kamath. Exotica: a research perspective on workflow management systems. *IEEE Data Engineering Bulletin*, 18(1):19–26, Mar. 1995.

[45] J. Mylopoulos and R. Motschnig-Pirtrik. Partitioning information bases with contexts. In CoopIS [15], pages 44–54.

[46] A. Oberweis. INCOME/STAR: Methodology and tools for the development of distributed information systems. *Information Systems*, 19(8):643–660, Dec. 1994.

[47] R. Ortiz and P. Dadam. The concurrency model: Activating an engineering database through an integrated product and process data model. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*, London, UK, Sept. 1995. Springer.

[48] R. Ortiz and P. Dadam. Towards the boundary of concurrency. In *Proc. of the Int. Conference on Concurrent Engineering*, McLain, Virginia, USA, Aug. 1995.

[49] R. Pooley. Performance Analysis Tools in Europe. *it + ti, Informationstechnik und Technische Informatik*, 37(3):10–16, June 1995.

[50] C. Pu and A. Leff. Replica control in distributed systems: An asynchonous approach. *ACM SIGMOD Record*, 20(2):377–386, June 1991.

[51] C. V. Ramamoorthy and G. S. Ho. Performance evaluation of asynchronous concurrent systems using petri nets. *IEEE Trans. on Software Engineering*, SE-6(5):440–449, Sept. 1980.

[52] B. Reinwald and H. Wedekind. Automation of control and data flow in distributed application systems. In DEXA [19], pages 475–481.

[53] N. Ritter, B. Mitschang, T. Härder, and U. Nink. Unterstützung der Ablaufsteuerung in Entwurfsumgebungen durch Versionierung und Konfigurierung. In *Proc. STAK'94, Softwaretechnik in Automatisierung und Kommunikation — Datenbanken unter Realzeit– und technischen Entwicklungsanforderungen*, pages 135–169, 1994.

[54] T. G. Robertazzi. *Computer Networks and Systems: Queueing Theory and Performance Evaluation.* Springer, 1995.

[55] A. Schill and A. Malhotra. Language and distributed system support for complex organizational services. In P. de Jong, editor, *Conference on Organizational Computing Systems*, pages 1–15, Atlanta, Georgia, Nov. 1991. ACM press.

[56] P. Schneider and M. Feltes. PEFF: Prozeßnetzwerk Entwicklung und Fertigung im Flugzeugbau, Zwischenbericht zur Betrachtung der Prozeßkette „Konstruktion und Fertigung eines Leitungsbündels". Technical report, Daimler–Benz Forschung und Technik, Produktionsinformatik (F3P), Nov. 1993.

[57] B. Schultheiß. Prozeßreengineering in klinischen Anwendungsumgebungen, Beispiele, Vorgehensmodelle, Werkzeuge. Master's thesis, Universität Ulm, Fakultät für Informatik, 1996.

[58] W. Schulze and M. Böhm. Klassifikation von Vorgangsverwaltungssystemen. In Vossen and Becker [64], chapter 16, pages 279–293.

[59] H. Schuster, S. Jablonski, T. Kirsche, and C. Bussler. A client/server architecture for distributed workflow management systems. In *Proceedings of the International Conference on Parallel and Distributed Information Systems*, Austin, Texas, Sept. 1994. extended version.

[60] K. Schwab. Koordinationsmodelle und Softwarearchitekturen als Basis für die Auswahl und Spezialisierung von Workflow–Management–Systemen. In Vossen and Becker [64], chapter 17, pages 295–317.

[61] C. U. Smith. The Evolution of Performance Analysis Tools. *it + ti, Informationstechnik und Technische Informatik*, 37(3):17–20, June 1995.

[62] WorkParty V2.0, Produktinformationen. Technical report, Siemens Nixdorf Informationssysteme AG, Oct. 1995.

[63] P. Vogel and R. Erfle. Backtracking office procedure. In DEXA [19], pages 506–511.

[64] G. Vossen and J. Becker, editors. *Geschäftsprozeßmodellierung und Workflow–Management.* Thomson, 1996.

[65] H. Wabnig and G. Haring. PAPS — Werkzeuge zur Leistungsvorhersage von parallelen Systemen. *it + ti, Informationstechnik und Technische Informatik*, 37(3):48–53, June 1995.

[66] H. Wächter, F. J. Fritz, A. Berthold, B. Drittler, H. Eckert, R. Gerstner, R. Götzinger, R. Krane, A. Schaeff, C. Schlögel, and R. Weber. Modellierung und Ausführung flexibler Geschäftsprozesse mit SAP Business Workflow 3.0. In GI [25], pages 197–204.

[67] Glossary — workflow management coalition specification. Technical report, Workflow Management Coalition, Dec. 1994.

[68] The workflow reference model. TC00-1003, Workflow Management Coalition, Dec. 1994.