Mentor: Entwurf einer Workflow–Management–Umgebung basierend auf State– und Activitycharts

Dirk Wodtke[1], **Angelika Kotz Dittrich**[2], **Peter Muth**[1], **Markus Sinnwell**[1], **Gerhard Weikum**[1]

[1] Universität des Saarlandes
Fachbereich Informatik
Postfach 15 11 50, D–66041 Saarbrücken
E–Mail: {wodtke, muth, sinn, weikum}@cs.uni–sb.de

[2] Schweizerische Bankgesellschaft
UBILAB
Postfach 2336, CH–8033 Zürich
E–Mail: angelika@ubilab.ubs.ch

Zusammenfassung

Der Aspekt der Steuerung von Arbeitsabläufen rückt heute in Unternehmen - gerade auch im Dienstleistungsbereich - immer mehr in den Vordergrund. Eine möglichst optimale Strukturierung und Abwicklung von Unternehmensprozessen spielt, etwa im Rahmen des Business Process Reengineering, für die Wirtschaftlichkeit eine große Rolle. Die aus einzelnen Arbeitsschritten zusammengesetzten Arbeitsabläufe werden auch als "Workflows" bezeichnet. Bei der computerunterstützten Abwicklung von Workflows sollen die einzelnen Arbeitsschritte mittels beliebiger Datenbanksysteme und/oder anderer Systemkomponenten (Dokumentenarchive, Applikationsprogramme etc.) ausgeführt werden; die Workflow–Management–Umgebung soll die Infrastruktur zur koordinierten und fehlertoleranten Ausführung in einer unternehmensweit verteilten und hochgradig heterogenen Informationssystemlandschaft bereitstellen. Dieser Artikel stellt das Projekt *Mentor* vor, in dem eine Workflow–Management–Umgebung entwickelt wird, bei der Workflows mit Hilfe von State– und Activitycharts spezifiziert, ausgeführt, überwacht und gesteuert werden. Die von David Harel entwickelte Spezifikationsmethode der State– und Activitycharts kombiniert die Einfachheit und mathematische Rigorosität von Automatenmodellen mit Möglichkeiten zur Visualisierung von Spezifikationen und hat gleichzeitig eine mit Prädikat–Transitions–Netzen vergleichbare Ausdrucksmächtigkeit. Ausgehend von dem kommerziellen Werkzeug Statemate, mit dem State– und Activitycharts entworfen und simuliert werden können, soll eine komplette Spezifikations– und Laufzeitumgebung für unternehmensweite Workflows entwickelt werden. Diese Umgebung soll verschiedene Middleware–Komponenten wie TP–Monitore und Dienste einer verteilten Programmierumgebung (z.B. OMG CORBA und COSS) integrieren und Workflow–Spezifikationen darauf abbilden.

1 Einführung

Durch den Einsatz von Datenbanksystemen konnten in vielen Dienstleistungsunternehmen in der Vergangenheit große Rationalisierungspotentiale ausgeschöpft werden. Zusätzliche Rationalisierungspotentiale lassen sich erschließen, indem Arbeitsabläufe in einem Unternehmen, die auf die Dienste dieser Datenbanksysteme zugreifen, (teil–)automatisiert werden. Typische Beispiele für solche Arbeitsabläufe sind die Bearbeitung eines Kreditantrages in einem Kreditinstitut, die Bearbeitung eines Schadensfalls in einer Versicherung und die Ablaufplanung und –überwachung eines stationären Krankenhausaufenthalts. Die Bearbeitung eines Kreditantrags beispielsweise beinhaltet u.a. die Prüfung von Firmenkrediten mit entsprechenden Bonitätsprüfungen und Risikoabschätzungen bezüglich des Kreditnehmers sowie seiner finanziellen Verflechtungen mit

anderen nationalen und internationalen Unternehmen. Beispielsweise kann es bei der Vergabe eines Kredits an die Firma X von Bedeutung sein, ob bereits an eine Tochtergesellschaft von X ein hoher Kredit vergeben wurde, ob X zu einer Holding–Gesellschaft gehört, für die bereits ein hohes Engagement der Bank existiert, usw. Die zur Entscheidungsvorbereitung notwendigen Arbeitsabläufe erstrecken sich über verschiedene Geschäftsarten (Kredite, Wertpapiere, Devisen, usw.) und verschiedene Niederlassungen, nicht selten sogar auf internationaler Ebene.

Ein *Workflow (Arbeitsablauf)* ist die koordinierte Ausführung einer Menge zusammengehöriger *Arbeitsschritte* in einer verteilten Arbeitsumgebung [Jab93, Rei93, MC94, RS94]. Die einzelnen Arbeitsschritte eines Workflows werden von spezifischen *Ausführungsorganen* bearbeitet; dies können menschliche Sachbearbeiter und Entscheidungsträger wie auch Computersysteme sein oder eine Kombination davon. In der Literatur werden Ausführungsorgane auch als *Ressourcen* bezeichnet. Ausführungsorgane übernehmen bezüglich der Arbeitszuteilung sogenannte *Rollen*; Ausführungsorgane in derselben Rolle sind austauschbar (z.B. verschiedene Sachbearbeiter mit denselben Fähigkeiten und Kompetenzen). Wichtigen Anwendungen in großen Unternehmen liegen häufig hochgradig heterogene Informationssysteme als Ausführungsorgane in einer weitläufig verteilten und damit stark fehleranfälligen Umgebung zugrunde. Die Gesamtheit der organisatorischen und computergestützten Maßnahmen zur Spezifikation, Verifikation, Ausführung, Überwachung und Steuerung von Workflows wird als *Workflow–Management* bezeichnet. Ein *Workflow–Management–System* umfaßt die Gesamtheit aller für das Workflow–Management benötigten Systemkomponenten.

Dieser Artikel stellt das Projekt *Mentor* (Middleware for Enterprise–wide Workflow Management) vor, in dem eine umfassende Workflow–Management–Umgebung entwickelt wird, bei der Workflows mit Hilfe von State– und Activitycharts spezifiziert, ausgeführt, überwacht und gesteuert werden. State– und Activitycharts wurden von David Harel als Spezifikationsmethode für prozeßorientierte, sogenannte reaktive Systeme, entwickelt [Ha87, Ha88, Ha90]. Zur Entwicklung, Visualisierung und Simulation von Spezifikationen wird ein kommerzielles Werkzeug *Statemate* [Ha90, i–Log91] angeboten. State– und Activitycharts wurden primär für die Spezifikation technischer Steuerungssysteme (z.B. in Automobilen oder Flugzeugen) entwickelt und für diesen Anwendungsbereich erfolgreich eingesetzt. Ihre Verwendung für Workflow–Management wird unseres Wissens erstmals in diesem Artikel diskutiert.

1.1 Beispiel einer Workflow–Spezifikation

In Abbildung 1 ist ein Beispiel für einen Workflow wiedergegeben, der mit einem Activitychart und einem Statechart modelliert ist. Das Beispiel zeigt den Workflow “Begutachtung eines Papiers bei einer wissenschaftlichen Zeitschrift”.

Der obere Teil der Abbildung enthält das Activitychart *Zeitschrift_AC*, das aus den vier Activities *Einreichung_A*, *Begutachtung_A*, *Entscheidung_A* und *Mitteilung_A*, dem durch die Pfeile dargestellten Datenfluß und einem Verweis auf das Statechart *Zeitschrift_SC* besteht. Das Statechart *Zeitschrift_SC* ist im unteren Teil der Abbildung wiedergegeben und beschreibt das Verhalten des Activitycharts, insbesondere den Kontrollfluß zwischen den Activities. Die Rechtecke mit den abgerundeten Ecken in *Zeitschrift_SC* stellen Zustände dar. Die Pfeile geben den Kontrollfluß an und werden als Transitionen bezeichnet. Sind zwei Zustände durch eine Transition mit der Transitionsbeschriftung *E[C]/A* verbunden, werden beim Eintreten des Ereignisses *E,* sofern Bedingung *C* erfüllt ist, der Ausgangszustand der Transition verlassen, ihr Zielzustand betreten und die Aktion *A* ausgeführt. Transitionen haben somit dieselbe Mächtigkeit wie ECA–Regeln, wie sie bei aktiven Datenbanken verfolgt werden [WD94], sind aber in übersichtlicher Weise in die Gesamtspezifikation eingebettet und vermeiden damit die Probleme der mangelnden Überschaubarkeit und Beherrschbarkeit umfangreicher unstrukturierter Mengen von ECA–Regeln.

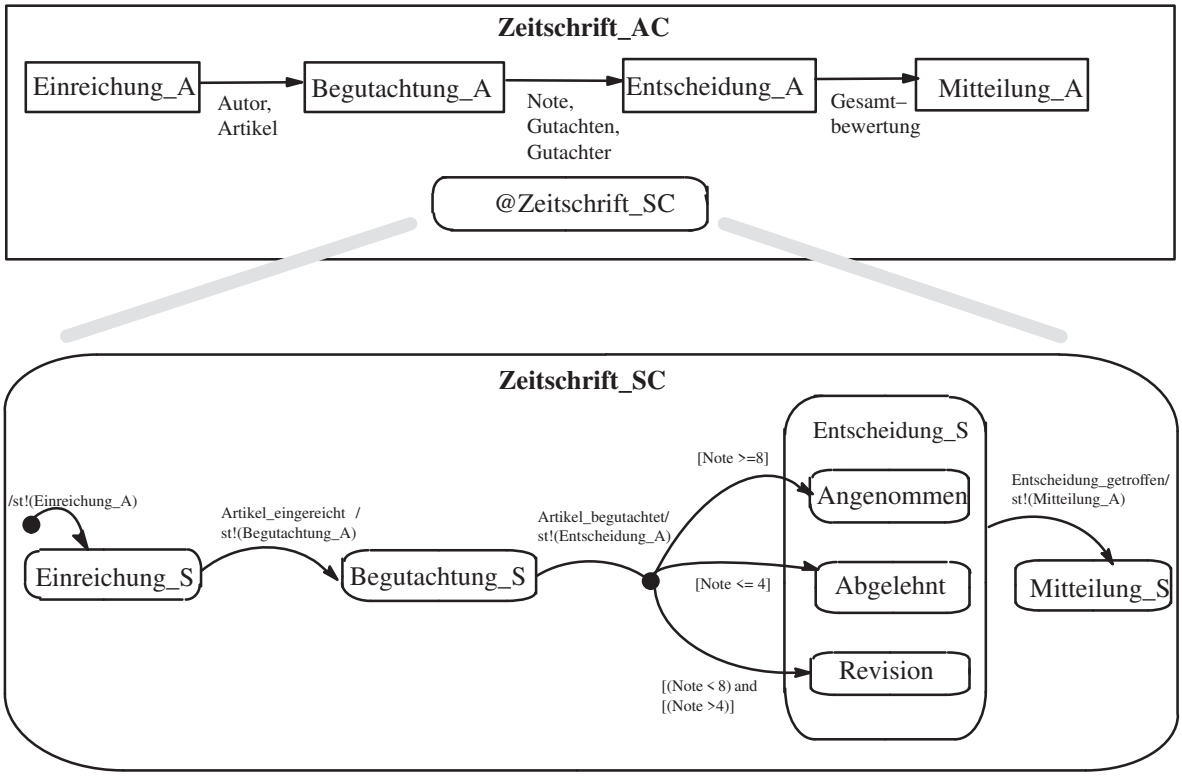

Abbildung 1: Activity– und Statechart zur Modellierung des Begutachtungsprozesses bei einer wissenschaftlichen Zeitschrift

Zu Beginn der Ausführung des Workflows wird die Activity *Einreichung_A* durch die Anweisung *st!(Einreichung_A)* gestartet und der Zustand *Einreichung_S* betreten. Nach der Einreichung eines Papiers (Ereignis *Artikel_eingereicht* wurde generiert) wird die Activity *Begutachtung_A* gestartet und der nächste Zustand *Begutachtung_S* betreten. Liegen die nötigen Gutachten vor, d.h. wurde das Ereignis *Artikel_begutachtet* generiert, wird die Activity *Entscheidung_A* gestartet. In Abhängigkeit von der von den Gutachtern vergebenen *Note* wird einer der Zustände *Angenommen, Abgelehnt* oder *Revision* betreten. Ist die Entscheidung über ein Papier getroffen, wird die Activity *Mitteilung_A* gestartet. Dieser – natürlich etwas übersimplifizierte – Workflow beinhaltet zwei verschiedene Ausführungsorgane, den Editor und den bzw. die Gutachter. (Im Falle elektronischer Einreichungen könnte man die Autoren sogar noch als drittes Ausführungsorgan hinzunehmen.) In Abschnitt 4.1 wird gezeigt, wie der Zustand *Begutachtung_S* verfeinert werden kann.

1.2 Warum State– und Activitycharts?

Die Verwendung von State– und Activitycharts hat sich vor allem in technischen Anwendungen (siehe z.B. [Sta94]) sehr bewährt; ihre Verwendung für Workflow–Management ist dagegen Neuland. Diese Spezifikationsmethode kombiniert die Einfachheit und mathematische Rigorosität von Automatenmodellen [HPSS87] mit weitreichenden Möglichkeiten zur Visualisierung [Ha88, Ha90] und hat gleichzeitig eine mit Prädikat–Transitions–Netzen vergleichbare Ausdrucksmächtigkeit.

Im Gegensatz zu den meisten bisher entwickelten oder vorgeschlagenen Workflow–Management–Umgebungen, die auf Skriptsprachen oder Konzepten aktiver Datenbanksysteme wie ECA–Regeln basieren, ist mit State– und Activitycharts eine theoretisch wohlfundierte Basis gegeben. Von fundamentaler Bedeutung ist außerdem die durch Schachtelung von Zuständen gege-

bene Möglichkeit, Spezifikationen zu modularisieren – zur schrittweisen Verfeinerung im Entwurfsprozeß oder zur Komposition existierender Workflows zu übergeordneten Worfklows –, wohingegen Skripts oder ECA–Regeln ohne weitere Strukturierung ab einem gewissen Umfang unübersichtlich oder gar undurchschaubar werden. Auch bei Petrinetzen und ihren zahlreichen Varianten sind die Unterstützung der Verfeinerung und Komposition von Spezifikationen tendenziell kritische Punkte (siehe z.B. [Ha87, Fu93]).

Gegenüber Ansätzen, die auf Petrinetz–Varianten basieren, zeichnen sich State– und Activitycharts ferner durch die Möglichkeit aus, beliebige, bereits existierende Subsysteme als Activities in die Spezifikation und Simulation einzubeziehen. Diese "offene Architektur" ist eine entscheidende Voraussetzung für die Verwendung von State– und Activitycharts als Basis der echten Ausführung von Workflows in einer hochgradig verteilten und heterogenen Systemlandschaft. Petrinetz–basierte Systeme erwarten dagegen typischerweise, daß sämtliche Anwendungsteile komplett als Petrinetz spezifiziert sind, und beschränken sich dementsprechend praktisch ausschließlich auf die Spezifikation und Simulation sowie bestenfalls die prototypische Ausführung auf einem Rechner. (Eine Ausnahme, die auch die Anbindung externer Software an eine Petrinetz–Simulationsumgebung vorsieht, ist das in der Entwicklung befindliche INCOME/STAR [OSS94].)

1.3 Forschungsziel und Beitrag dieses Artikels

Dieser Artikel untersucht das Nutzpotential von State– und Activitycharts für Workflow–Management generell und stellt das Mentor–Projekt im besonderen vor. Ziel des Mentor–Projekts ist die Verbindung von State– und Activitycharts als Spezifikationsumgebung mit einer Laufzeitumgebung, die verschiedene sogenannte Middleware–Komponenten integriert und um zusätzliche Infrastruktursoftware zur Ausführung, Überwachung und Steuerung von Workflows erweitert. Mentor sieht als wichtigste Middleware–Komponenten einen TP–Monitor [GR93, Ob94] als Ausgangsbasis zur Erreichung von Fehlertoleranz und eine verteilte Programmierumgebung wie OMG CORBA und COSS [OMG92, OMG94] zur Bewältigung der Heterogenität vor.

Die Ausführung von Workflows soll direkt aus der Spezifikationsumgebung heraus – auf der Basis von State– und Activitycharts – erfolgen. Dazu soll die offene Architektur des Werkzeugs Statemate ausgenutzt werden, bei der Activities beliebigen C–Code und Subsystemaufrufe beinhalten können. Auf diese Weise läßt sich eine Kontrollflußsteuerung auf der Ebene von Statecharts mit Aufrufen beispielsweise eines TP–Monitors koppeln. Während der Workflow–Ausführung sollen umgekehrt die Visualisierungsmöglichkeiten von Statemate auch für die Überwachung von Workflows ausgenutzt werden. Beispielsweise sollen bereits ausgeführte und laufende Arbeitsschritte jederzeit abfragbar und konform zur Spezifikationsumgebung visuell darstellbar sein. Eine derartige Gesamtarchitektur für verteilte, heterogene Workflow–Management–Umgebungen, die alle Ebenen der Spezifikation, Validierung, Ausführung, Überwachung und Steuerung durchgängig überdeckt, ist unseres Wissens neu und hebt sich von bisherigen Arbeiten im Bereich des Workflow–Managements ab.

Im folgenden Abschnitt 2 wird der potentielle Nutzen von Workflow–Management–Systemen diskutiert, und es werden existierende kommerzielle Produkte und Forschungsprojekte im Bereich Workflow–Management kurz charakterisiert. In Abschnitt 3 wird die Architektur unseres Ansatzes für ein Workflow–Management–System vorgestellt. In Abschnitt 4 wird die Spezifikation von Workflows mit State– und Activitycharts anhand eines Anwendungsbeispiels genauer vorgestellt. In Abschnitt 5 folgt eine Diskussion der bei der Ausführung und Überwachung von Workflows zu bewältigenden Probleme.

2 Potential und Stand der Technik von Workflow–Management–Systemen

2.1 Potentieller Nutzen

Der mit der Verbreitung verteilter Systeme einhergehende Einsatz unterschiedlicher Anwendungsprogramme in heterogenen Systemumgebungen zwingt zur Integration der bestehenden Hardware– und Softwarekomponenten. Damit wird es möglich, Arbeitsabläufe in ihrer Gesamtheit stärker zu automatisieren, als dies mit Stand–alone–Ansätzen erzielbar ist. Gelingt es, die Anforderungen, die an solche integrierten Ansätze hinsichtlich Bewältigung der Heterogenität, Fehlertoleranz und Konsistenz gestellt werden, zu erfüllen, läßt sich die Effektivität von Arbeitsabläufen erhöhen, indem zum Beispiel auf die mehrfache Eingabe gleicher Daten in verschiedenen Applikationen verzichtet werden kann. Belegtransporte können eliminiert werden, wenn der Datenfluß automatisiert wird. Als Folge der kürzeren Bearbeitungszeiten der einzelnen Arbeitsschritte und der verkürzten Laufzeiten zwischen den beteiligten Subsystemen ist eine geringere Turnaround–Zeit von Kundenaufträgen zu erwarten.

Positiv für die Unternehmensleitung ist, daß die Überwachung von Geschäftsvorgängen erleichtert und die Transparenz der Arbeitsabläufe für alle Beteiligten erhöht wird. Die Analyse von Arbeitsabläufen (z.B. zur Identifikation von zeitlichen Engpaßstellen) sollte einfacher und besser möglich sein. Die integrierte Sicht auf Arbeitsabläufe erlaubt ferner eine Verbesserung der Informationsmöglichkeit über den Bearbeitungszustand von Arbeitsabläufen. Denkbar ist die Nutzung dieses Informationspotentials auch zum Zweck automatisierter Terminüberwachung. Weisungskonformität kann garantiert werden, wobei das Workflow–Management–System trotz der weitgehenden Standardisierung von Arbeitsschritten und einer Entlastung von Routinetätigkeiten zusätzlich die Möglichkeit kontrollierter, d.h. in der Auswirkung auf den Workflow begrenzter Ausnahmebehandlungen vorsehen sollte. Schließlich sollten Anpassungen von Arbeitsabläufen im Hinblick auf veränderte Marktsituationen oder individuelle Kundenwünsche einfacher und schneller als bisher möglich sein.

Die genannten Anforderungen können als ein Baustein moderner Managementkonzepte im Kontext von Lean Management, Total Quality Management und Kundenorientierung angesehen werden. Dennoch stellt ein Workflow–Management–System kein Allheilmittel für jede Art von Unternehmensproblemen dar, sondern allein ein technisches Infrastrukturmittel, dem eine wichtige Unterstützungsfunktion zukommt.

2.2 Stand der Technik

Das Gebiet des Workflow–Managements ist eng verwandt mit dem Thema CSCW (Computer–Supported Cooperative Work). CSCW zielt jedoch stärker auf ungeplante, spontane Interaktion in einem Arbeitsteam ab, wohingegen Workflow–Management sich primär mit bereits weitgehend strukturierten und vor allem häufig wiederkehrenden Arbeitsabläufen befaßt. Innerhalb der ”Routinearbeitsabläufe” sollte Workflow–Management jedoch auch kontrollierte Ausnahmen und manuelle Eingriffsmöglichkeiten zulassen – bis hin zu ausgesprochenen “Ad–hoc”–Workflows. Enge Parallelen gibt es auch mit dem Gebiet des CIM (Computer–Integrated Manufacturing), welches sich allerdings auf Abläufe in Fertigungsunternehmen konzentriert. Workflow–Management wird gelegentlich auch als das “CIM der Dienstleistungsunternehmen” bezeichnet.

Aus Datenbanksicht knüpft Workflow–Management vor allem an die Gebiete der erweiterten Transaktionsmodelle und der aktiven Datenbanksysteme an. In den folgenden Unterabschnitten wird der Stand der Technik auf dem Gebiet des Workflow–Management näher erörtert.

2.2.1 Kommerzielle Produkte

Zur Unterstützung von Workflow–Management werden eine Reihe von Produkten angeboten wie z.B. Lotus Notes, Staffware, FlowMark, ARIS Toolset usw. (siehe z.B. [McC93, LA94, Sch94]). Diese Produkte sind primär für den lokalen Office–Bereich konzipiert und für abteilungsübergreifende, u.U. unternehmensweite Arbeitsabläufe kaum geeignet. Sie sind überwiegend nur in homogenen Umgebungen einsetzbar, bei denen typischerweise alle relevanten Daten auf einem zentralen Server gehalten werden. Heterogenität wird nur auf der Netzwerk– und Betriebssystemebene erlaubt (so daß z.B. eine Koexistenz von DOS–PCs und Unix–Clients möglich ist), nicht jedoch auf der Ebene der Ausführungsorgane (z.B. verschiedene Datenhaltungssysteme). Die Produkte bieten kaum Fehlertoleranzmaßnahmen; sie verwenden z.B. einfache E–Mail und sind weit entfernt von den Fehlertoleranzgarantien, die man etwa vom OLTP–Bereich kennt. Obwohl für einige Einzelprobleme Lösungen in Produkten existieren, gibt es kein Produkt, das alle notwendigen Eigenschaften geeignet kombiniert.

2.2.2 Forschungsprojekte

Workflow–Management wurde in der “Challenges Session” der SIGMOD–Konferenz 1993 zu einem besonders herausfordernden Forschungsgebiet erklärt [Da93] (vgl. auch [De94, Me94]). Dementsprechend gibt es in der Datenbankszene eine stattliche Anzahl von aktuellen Forschungsprojekten zu diesem Thema (siehe u.a. [Bi94, BMR94, Be93, Br93, DHL91, EN93, GGS93 GHKM94, Hsu93, Jab93, KUW94, Ober94, RS94, SK94, ST94, WR92, Wei93]). Eines der ersten und das vermutlich am weitesten fortgeschrittene Projekt ist das ConTracts–Projekt an der Universität Stuttgart [WR92, RSW92, Schw93]. Im Vordergrund dieses Projekts steht die konsequente Erweiterung transaktionsorientierter Mechanismen zur fehlertoleranten Ausführung beliebiger verteilter Abläufe. Insbesondere wurde Wert gelegt auf die saubere Trennung zwischen Ressourcenmanagern und Transaktionsmanagern im Sinne des X/Open–Standards XA und auf die fehlertolerante Verwaltung von Verarbeitungskontexten über Transaktionsgrenzen hinweg. Die Spezifikation von ConTracts beruht auf einer speziellen Skriptsprache, die intern in Prädikat–Transitions–Netze übersetzt wird. Die Tauglichkeit dieser Sprache zur Spezifikation komplexer, unternehmensweiter Workflows und zur Anbindung beliebiger Ausführungsorgane ist nicht weitergehend untersucht worden. An dieser Stelle versprechen wir uns von der Verwendung von State– und Activitycharts ganz entscheidenden Fortschritt gegenüber dem – ansonsten sicherlich wegweisenden – ConTract–Projekt.

Verteilte Abläufe generell werden in einer Vielzahl von Projekten untersucht, insbesondere im Umfeld objektorientierter Systeme (z.B. [CBHR93, Mu93, Sch93, OMG92, OMG94]). Dabei werden allerdings Aspekte der fehlertoleranten Ausführung entweder ausgeklammert oder beschränken sich auf relativ primitive Basismechanismen wie z.B. Prozeßgruppen oder (geschlossen) geschachtelte Transaktionen. Die Spezifikation verteilter Abläufe ist ebenfalls ein breites Forschungsgebiet, bei dem insbesondere Petrinetzvarianten zum Einsatz kommen (z.B. [EN93, OSS94]); die Spezifikation sogenannter “object lifecycles” ist eine spezielle Variante dieser Richtung (z.B. [KS91, Saa93]). Diese Forschungslinie behandelt jedoch die Ausführung von Spezifikationen eher am Rande, ganz zu schweigen von verteilten Ausführungsumgebungen.

3 Architektur von Mentor

Dieser Abschnitt beschreibt die Architektur des Workflow–Management–Systems Mentor. State– und Activitycharts spielen in Mentor sowohl während der Spezifikation von Workflows als auch während deren Ausführung und Steuerung eine zentrale Rolle. Sie stellen für jede dieser

Aufgaben die Schnittstelle zum Benutzer dar. Abbildung 2 zeigt die Systemumgebung von Mentor, bestehend aus Workstations, die von Workflow–Designern, Sachbearbeitern und Mitarbeitern mit Überwachungsfunktion bedient werden.

Die Ausführung der einzelnen Arbeitsschritte übernehmen Mentor-Server, die die Verbindung zu den Ausführungsorganen (z.B. Datenbanksysteme) schaffen und den Kontroll– und Datenfluß zwischen den Arbeitsschritten steuern. In der Regel erstreckt sich ein Workflow über mehrere Server und Workstations, d.h. die Arbeitsschritte des Workflows werden von mehreren Mentor–Servern ausgeführt.

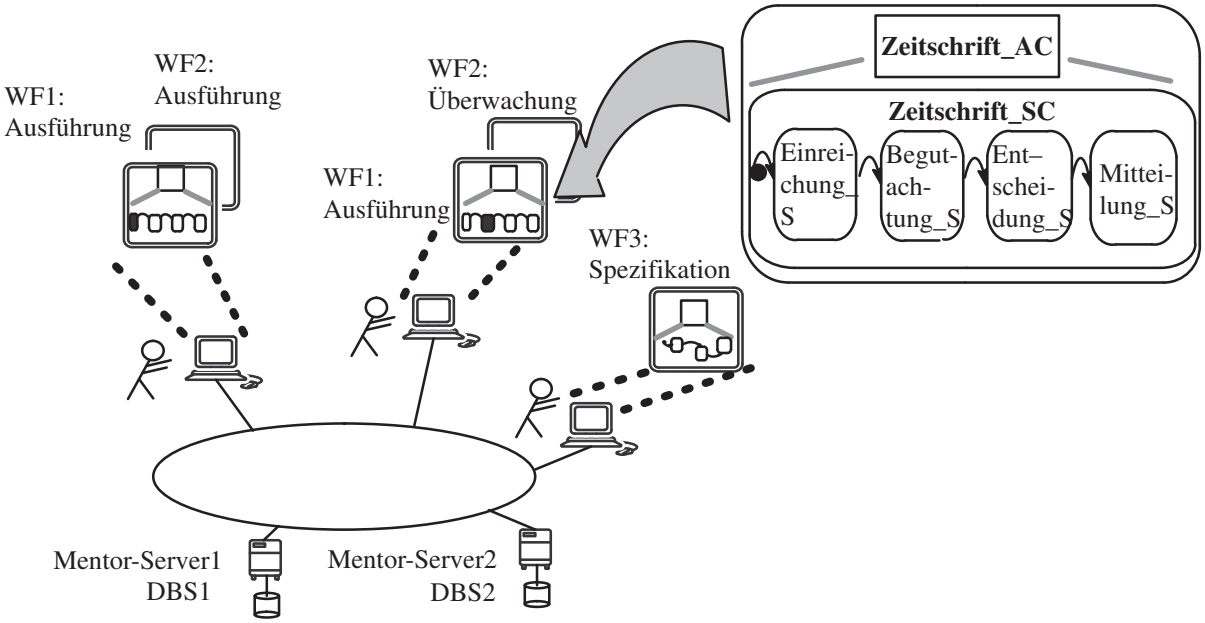

Abbildung 2: Systemumgebung des Workflow–Management–Systems Mentor

Jeder Workstation können zu jedem Zeitpunkt beliebig viele Workflows zugeordnet werden, und ein Workflow kann die Bearbeitung auf beliebig vielen Workstations erfordern. Im Beispiel erfordert die Ausführung des Workflows *WF1* die Bearbeitung durch die ersten beiden Sachbearbeiter. Der Workflow *WF2* kann allein vom ersten Sachbearbeiter bearbeitet werden, Sachbearbeiter zwei übernimmt hier eine Überwachungsfunktion, z.B. um die Effizienz des Ablaufs späterer Ausführungen von *WF2* zu verbessern. Workflow *WF3* wird gerade spezifiziert.

Jeder der drei Mitarbeiter bewegt sich dabei in der selben abstrakten Welt: State– und Activitycharts. Die State– und Activitycharts, die Struktur und Ablauf der Workflows festlegen, sind stilisiert innerhalb der abgerundeten Quadrate wiedergegeben. Innerhalb dieser Quadrate stellt jeweils das große Quadrat ein Activitychart dar, dem das Statechart, das aus den durch Pfeile miteinander verbundenen Quadraten besteht, zugeordnet ist. Ist ein Zustand abgedunkelt, so weist dies auf seine Aktivierung hin. *WF1* wird also zuerst auf der ersten Workstation ausgeführt, nach dem Schalten der Transition zum zweiten Zustand verlagert sich die Ausführung auf die zweite Workstation. Daten– und Kontrollfluß eines Workflows werden mittels Activitycharts und Statecharts vor Ausführung des Workflows spezifiziert, sollen jedoch auch während der Ausführung eines Workflows für kontrollierte Ausnahmen und manuelle Eingriffsmöglichkeiten dynamisch änderbar sein.

Abbildung 3 zeigt die Modularchitektur von Mentor. Das für State- und Activitycharts angebotene Werkzeug Statemate bildet die oberste Schicht und ist auf jeder Workstation lokal verfügbar. Im Sinne der gewünschten durchgängigen Umgebung für Spezifikation, Ausführung und Über-

wachung von Workflows verwaltet diese Komponente die Daten- und Kontrollflußdefinitionen eines Workflows und visualisiert die Ausführung.

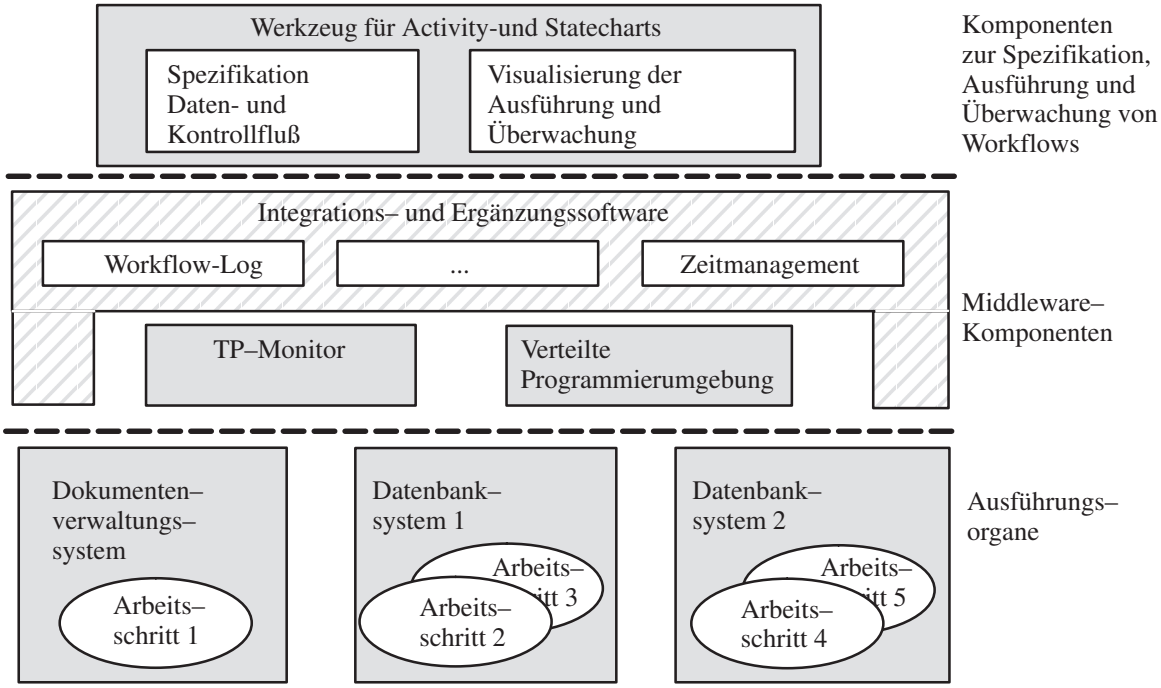

Abbildung 3: Gesamtarchitektur von Mentor

Als Middleware-Komponenten sieht Mentor einen TP–Monitor, eine verteilte Programmierumgebung und einen "Rahmen" von Integrations- und Ergänzungssoftware vor, der die Middleware–Komponenten untereinander verbindet und die Anschlüsse an die State- und Activitycharts nach oben bzw. die Datenverwaltungssysteme nach unten bereitstellt. Die Komponenten der Middleware stehen auf jedem Mentor-Server zur Verfügung. Wir streben eine minimale Menge dieser Komponenten in Mentor an, wobei der notwendige Funktionsumfang noch zu untersuchen ist. Daraus ergeben sich auch die Kriterien für eine mögliche Replikation der Datenstrukturen der Middleware-Komponenten und der Workflow-Spezifikationen auf den Mentor-Servern.

Ein TP–Monitor wird für fehlertolerante Ausführungen benötigt, insbesondere für verteilte Transaktionen und transaktionsgeschützte Nachrichten ("recoverable message queues"). Die verteilte Programmierumgebung (z.B. OMG CORBA und COSS [OMG92, OMG94]) stellt Dienste wie Remote Procedure Call, Naming Service, entfernten Methodenaufruf und Authentifizierungsdienste für heterogene Systeme bereit. Datenbanksysteme und Dokumentenverwaltungssysteme (z.B. Archivierungssysteme für Textdokumente) stellen wichtige Arten von Ausführungsorganen dar.

4 Spezifikation mit State– und Activitycharts

In diesem Abschnitt werden die wesentlichen Konzepte von State– und Activitycharts kurz vorgestellt, die im zweiten Teil des Abschnitts anhand des bereits eingeführten Beispiels erläutert werden.

4.1 Konzepte von Statecharts und Activitycharts

Eine Verhaltensbeschreibung eines Systems muß komplexe Folgen von Ereignissen, Aktionen und Bedingungen berücksichtigen, die häufig in Kombination mit gewissen zeitlichen Restrik-

tionen auftreten. Eine brauchbare Modellierungsmethode muß modular, hierarchisch und gut strukturiert sein und muß Konstrukte enthalten, mit denen parallele Abläufe dargestellt werden können. Diese Anforderungen werden vom Formalismus der State– und Activitycharts erfüllt. Die Grundidee von State- und Activitycharts basiert auf der Darstellung mittels visueller Formalismen. Diese erlauben ein in hohem Maße anschauliches Vorgehen beim Entwurf, wobei durch eine formale Semantik [HPSS87] eine präzise, kompakte und eindeutige Spezifikation möglich wird.

Mit Hilfe von Activitycharts kann ein System aus funktionaler Sicht modelliert werden. Konkret bedeutet dies eine Untergliederung des zu modellierenden Systems in Funktionen (*Activities*) und Datenflüsse zwischen den Activities. Statecharts beschreiben ein System aus Verhaltenssicht. Sie definieren zeitabhängige Vorgänge, die durch Zustandsübergänge modelliert werden. Zusätzlich zu den Darstellungsmitteln der Zustandsübergangs–Diagramme endlicher Automaten bieten Statecharts Konstrukte für die hierarchische Anordnung von Zuständen, für die Darstellung orthogonaler (paralleler) Zustände und für die Kommunikation zwischen Zuständen mittels Ereignissen. In Abbildung 1 sind zum Beispiel die drei Zustände *Angenommen, Abgelehnt* und *Revision* Unterzustände des Zustands *Entscheidung_S*.

Die wesentlichen Eigenschaften von Activitycharts und Statecharts sollen anhand des Beispiels in Abbildung 4 erläutert werden. Das Activitychart *A_AC* besteht aus den beiden Activities *Activity1* und *Activity2* und einem Verweis auf ein Statechart *A_SC*, das den Kontrollfluß zwischen den Activities beschreibt. Außerdem ist der Datenfluß innerhalb des Activitycharts angegeben. Mit Activities kann C–Code assoziiert werden, durch den die Funktionalität der Activity genau ausgedrückt ist. Das Statechart ist im unteren Teil der Abbildung angegeben und besteht aus mehreren hierarchisch aufgebauten Zuständen und den sie verbindenden Transitionen. Beispielsweise besagt die Beschriftung der Transition von *ABB* nach *ABC*: Wenn Zustand *ABB* aktiviert ist, *Activity1* beendet wird und gleichzeitig Bedingung *C1* erfüllt ist, dann schaltet die Transition und die *Activity2* wird gestartet. Im Beispiel ist die Namensgebung der Zustände entsprechend der hierarchischen Einordnung gewählt.

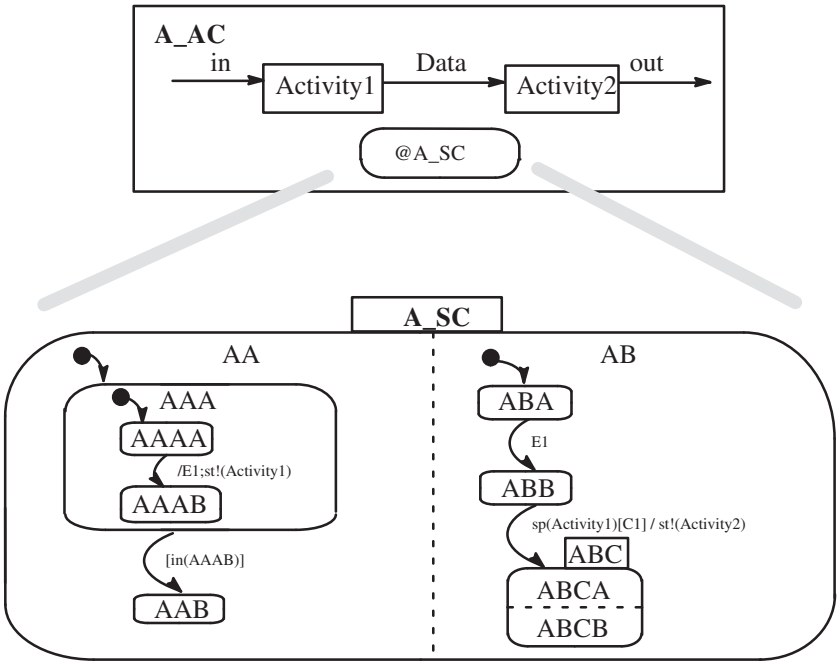

Abbildung 4: Beispiel für ein Activitychart und ein Statechart

Beim Starten des Activitycharts *A_AC* wird automatisch das Statechart *A_SC* aktiviert, d.h. daß mittels der Transitionen ohne Startzustand (die von den kleinen schwarzen Kreisen ausgehenden *Default–Transitionen*) die Zustände *AAA*, *AAAA* und *ABA* aktiviert werden. Bemerkenswert ist

hierbei, daß die Zustände *AAA* und *ABA*, die Unterzustände der orthogonalen Zustände *AA* bzw. *AB* sind, gleichzeitig aktiviert werden.

Im nächsten Schritt schaltet die Transition, die die Zustände *AAAA* und *AAAB* verbindet. Sie löst die Generierung des Ereignisses *E1* aus und startet *Activity1* durch die Aktion *start!(Activity1)* (abgekürzt *st!(Activity1)*). Durch die Generierung des Ereignisses *E1* kann im darauffolgenden Schritt die Transition von *ABA* nach *ABB* schalten. Gleichzeitig schaltet die Transition von *AAA* nach *AAB*, da die Schaltbedingung *in(AAAB)* (d.h. Aktivierung des Zustands *AAAB*) im vorhergehenden Schritt erfüllt war. Das Verlassen des Zustandes *AAA* bewirkt auch eine Deaktivierung aller Unterzustände, in diesem Fall des Zustandes *AAAB*. Terminiert *Activity1* zu einem späteren Zeitpunkt, wird automatisch das Ereignis *stopped(Activity1)* (abgekürzt *sp(Activity1)*) generiert. Als Folge kann die Transition von *ABB* nach *ABC* schalten, wenn die Bedingung *C1* gleichzeitig erfüllt ist. Zeitgleich mit dem Schalten der Transition werden *Activity2* gestartet und der Zustand *ABC* mit seinen orthogonalen Komponenten *ABCA* und *ABCB* aktiviert.

Anhand des Beispiels wird verdeutlicht, wie die Verbindung zwischen Activitycharts und Statecharts die funktionale Sichtweise und die Verhaltenssicht zu einer Gesamtsicht auf ein Modell vereint. Parallelität wird in Statecharts durch orthogonale Zustände dargestellt, Tiefe durch die hierarchische Anordnung von Zuständen und Kommunikation durch Generieren von Ereignissen und Warten auf Ereignisse, auch über die Grenzen von Activities hinweg.

4.2 Anwendungsbeispiel

Ein wesentliches Konzept der Entwurfsmethodik mit State– und Activitycharts besteht in der Unterstützung eines Top–Down–Entwurfsverfahrens, indem bereits entworfene State– und Activitycharts verfeinert werden können. Zum Beispiel lassen sich die Activity *Begutachtung_A* und der Zustand *Begutachtung_S* aus dem Activitychart *Zeitschrift_AC* bzw. dem Statechart *Zeitschrift_SC* aus Abbildung 1 zu dem Activitychart bzw. Statechart in Abbildung 5 verfeinern.

Die Begutachtung eines Papiers geschieht gemäß dem Activitychart durch drei Gutachter, deren Begutachtungstätigkeit in drei Activities *Begutachtung1_A* bis *Begutachtung3_A* modelliert ist. Die zugehörigen Verhaltensbeschreibungen befinden sich in drei orthogonalen Zuständen *Gutachter1* bis *Gutachter3* im Statechart *Begutachtung_SC*. Die darin enthaltenen Zustände *Gutachten_anfertigen* sind dabei Instanzen eines weiteren, gewissermaßen generischen Statecharts. In diesem Statechart wird die Activity *Begutachtung_A* für den jeweiligen Gutachter (*st!(Begutachtung_A)*) gestartet. Ist die Begutachtung beendet, wird das Ereignis *Gutachten_vorgelegt* generiert. Der Begutachtungsprozeß sei hier bereits nach dem Eingang von zwei Gutachten beendet. Da nicht von vornherein bekannt ist, welche beiden Gutachter die Begutachtung als erste beenden werden, wird in einem orthogonalen Zustand die Zahl der angefertigten Gutachten mittels der Zählvariablen *Zahl_Gutachten* protokolliert. Sind zwei Gutachten erstellt worden, ist die Schaltbedingung der Transition von *Start_Begutachtung* nach *Stop_Begutachtung* in einem weiteren orthogonalen Zustand erfüllt und die Transition kann schalten, wobei gleichzeitig die Durchschnittsnote der beiden Gutachten berechnet und das Ereignis *Artikel_begutachtet* generiert werden.

Bei der Modellierung des Verhaltens eines Systems werden häufig explizite Zeitangaben benötigt. Zum Beispiel könnten nach Ablauf einer bestimmten Zeit bestimmte Aktionen oder Zustandsübergänge ausgeführt werden. Der Statechart–Formalismus erlaubt hierfür zum einen die Spezifikation zeitorientierter Ereignisse mittels der Anweisung *timeout (E, n)* (abgekürzt *tm(E, n)*). Diese Anweisung bewirkt das Auslösen eines Ereignisses *n* Zeiteinheiten nach dem Auftreten des Ereignisses *E*. Im Beispiel aus Abbildung 5 wird 100 Zeiteinheiten nach dem Betreten des Zustands *Start* das Ereignis *Frist_naht* generiert, falls die Begutachtung zu diesem Zeitpunkt

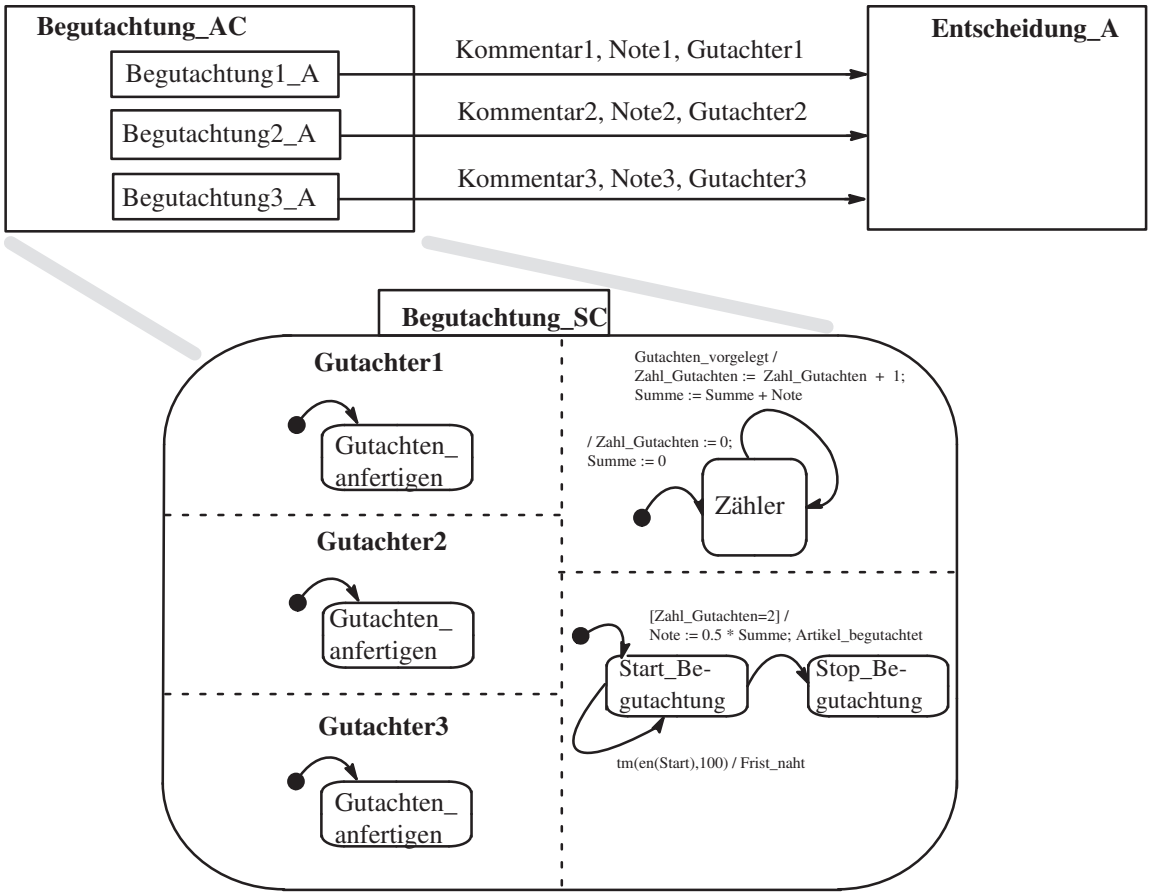

Abbildung 5: Verfeinerung der Activity *Begutachtung_A* und des Zustands *Begutachtung_S* aus Abbildung 1

noch nicht beendet worden ist. Die Generierung dieses Ereignisses könnte zum Beispiel in einer weiteren Verfeinerung des Modells dazu benutzt werden, Aktionen wie eine Beschleunigung der Begutachtung oder eine Begutachtung durch einen weiteren Gutachter auszulösen.

Die zweite Modellierungsmöglichkeit für explizite Zeitangaben sind zeitabhängige Aktionen, die mittels der Aktion *schedule!(E, n)* (abgekürzt *sc!(E, n)*) angegeben werden können. Mit dieser Anweisung wird das Ereignis *E n* Zeiteinheiten nach der Ausführung dieser Anweisung generiert. Im Beispiel ließe sich mit dem Aufruf dieser Anweisung beim Start von *Zeitschrift_AC* das Ereignis *Frist_erreicht* zu einem festgelegten Zeitpunkt generieren, und zwar unabhängig davon, welcher Zustand zu diesem Zeitpunkt aktiviert ist.

5 Ausführung und Überwachung von Workflows

Dieser Abschnitt diskutiert die bei der Ausführung und Überwachung von Workflows zu lösenden Probleme und zeigt einige Lösungsansätze auf. Wie bereits in den vorangegangenen Abschnitten diskutiert, ist es sinnvoll, für diese beiden Bereiche dasselbe Visualisierungskonzept zu benutzen wie für die Spezifikation von Workflows. State- und Activitycharts leisten dazu in Simulationen, z.B. von Schaltkreisen, gute Dienste. Ein Workflow-Management-System stellt jedoch weit höhere Anforderungen an Ausführung und Überwachung als eine Simulationsumgebung. Es gibt im allgemeinen keine zentrale Instanz, die alle Arbeitsschritte einer Workflow–Spezifikation oder gar alle Workflow–Spezifikationen kennt, und es sollte auch keine zentrale Instanz an der Ausführung oder Überwachung aller Workflows beteiligt sein müssen. Andernfalls ist das System nicht skalierbar. Skalierbarkeit ist aber eine zentrale Anforderung in einer

dynamisch wachsenden Workflow-Umgebung. Das resultierende verteilte System muß hohe Anforderungen bezüglich der Fehlertoleranz erfüllen.

5.1 Ausführung von Workflows

Zur Ausführung eines Workflows zählen Aspekte des Kontrollflusses, des Datenflusses zwischen Arbeitsschritten, der Fehlertoleranz von Daten– und Kontrollfluß und der dynamischen Modifikation von Workflows.

Datenbanksysteme bieten mit dem Transaktionskonzept die Basis für eine fehlertolerante Ausführung von Arbeitsschritten. Eine Verbindung mehrerer Arbeitsschritte innerhalb eines Datenbanksystems oder auch zwischen verschiedenen Systemen wird jedoch nicht unterstützt; allenfalls könnte man verschiedene Arbeitsschritte zu einer verteilten Transaktion zusammenfassen. Damit lassen sich jedoch keinesfalls komplexere *Kontrollflußabhängigkeiten* zwischen Arbeitsschritten realisieren, wie beispielsweise die folgende: "Bei Eingang von zwei von drei Gutachten wird die Begutachtung beendet". Einen ersten, aber wesentlichen Schritt zur Lösung dieses Problems stellen TP-Monitore dar. Sie können beispielsweise die Ausführung eines Arbeitsschritts mit dem Versenden einer Nachricht zu einer atomaren Einheit kombinieren, die genau einmal ausgeführt wird. Auf diese Weise kann das Ende der Begutachtung eindeutig bestimmt werden, ohne in der Schicht der Integrations- und Ergänzungssoftware aufwendige Algorithmen implementieren zu müssen. Wir kommen darauf in der Diskussion unseres Beispielszenarios in Abschnitt 5.3 zurück.

Nicht weniger komplex ist die Realisierung des *Datenflusses* zwischen Arbeitsschritten. Zunächst stellt sich das Problem der Heterogenität beim Datenfluß zwischen Arbeitsschritten, die von verschiedenen Datenverwaltungssystemen ausgeführt werden. Um nicht bei n Systemen n^2 Konverter zu benötigen, bedient man sich eines gemeinsamen Standardformates. Standards wie CORBA [OMG92, OMG94] verdecken darüber hinaus die Verwendung unterschiedlicher Implementierungssprachen; eine C++ Methode kann beispielsweise eine COBOL-Routine aufrufen. Zum Austausch komplexer Strukturen wie z.B. Hypertexte oder multimediale Dokumente müssen diese Daten im verwendeten Standardformat darstellbar sein und als Parameter den aufgerufenen Methoden in den Arbeitsschritten übergeben werden können. Dies beinhaltet z.B. die Darstellung und Übergabe von Objektreferenzen als Teil eines Hypertextes.

Nicht jeder Workflow kann im Rahmen seiner Spezifikation zu Ende geführt werden. Äußere Einflüsse können eine Ausnahmebehandlung oder eine dynamische Modifikation nötig machen. Eine Ausnahmebehandlung kann etwa durch außerplanmäßiges Eingreifen eines Vorgesetzten entstehen. Es soll möglich sein, einen Workflow in einem bestimmten Zustand anzuhalten und dann später – zum Beispiel nach dem Treffen intellektueller Entscheidungen – in einem anderen Zustand fortzusetzen. Dynamische Modifikationen werden z.B. bei kurzfristigen Gesetzesänderungen nötig, die Eingriffe in Spezifikationen von laufenden Workflows erfordern. Die Häufigkeit von Ausnahmebehandlungen und dynamischen Änderungen ist anwendungsabhängig. Da in solchen Fällen jedoch Ablaufeigenschaften des Workflows vom System nicht mehr garantiert werden können, sollten Ausnahmebehandlungen und dynamische Änderungen selten bleiben.

Die Verbindung der Middleware–Komponenten mit State- und Activitycharts geschieht durch Programmcode, der den Activities zugeordnet wird. Beim Starten einer Activity wird dieser Code automatisch ausgeführt. Der Programmcode enthält die Anweisungen, die die Dienste der Middleware–Komponenten aufrufen. Konkret bedeutet dies, daß bei der Aktivierung einer Activity A (z.B. mit dem Kommando *st!(A)*) der mit der Activity A assoziierte Programmcode gestartet wird, der wiederum die Dienste der Middleware–Komponenten aufruft.

5.2 Überwachung von Workflows

Die Möglichkeit zur Überwachung von Workflows ist eine wesentliche Anforderung an ein Workflow-Management-System, da sie die Kontrolle über die nun als Workflow implementierten innerbetrieblichen Abläufe ermöglicht. Zur Überwachung der Ausführung von Workflows zählen das Protokollieren der ausgeführten Arbeitsschritte, das Stellen von Anfragen an die entstehende Workflow-Historie, die Berücksichtigung von Zeitaspekten bei der Ausführung sowie Aspekte der Arbeitsverteilung

In Mentor sollen den an einer Workflow–Ausführung beteiligten Sachbearbeitern und sonstigen Personen mit Überwachungsfunktionen dieselben Visualisierungsmöglichkeiten über State- und Activitycharts zur Verfügung gestellt werden, die bei der Spezifikation vorhanden sind. Dazu gehören die graphische Darstellung der verwendeten State- und Activitycharts, die Identifizierung des aktuellen Zustandes, die Trennung in erledigte, gerade bearbeitete und in Zukunft zu bearbeitende Activities, einzuhaltende Fristen und sich evtl. ergebende Wartebeziehungen auf externe Events, z.B. den Abschluß einer für die eigene Arbeit notwendigen Activity eines anderen Sachbearbeiters. Ein überwachender Mitarbeiter muß Zugriff auf die entsprechenden Daten aller von ihm überwachten Mitarbeiter haben. Zusätzlich stehen ihm abgeleitete Informationen zur Verfügung, wie z.B. kritische Pfade für Fristen oder Urlaubsdaten zur Einteilung von Mitarbeitern.

Grundlage für alle Arten der Überwachung sind die bei der Ausführung des Workflows aufgezeichneten Protokolldaten. Dabei werden – auch dynamisch – Anfragen auf historischen Daten durchgeführt. Wir können nicht erwarten, daß die beteiligten Datenverwaltungssysteme die Protokollierung der für uns relevanten Daten der Historie selbst vornehmen. Darüber hinaus wären beispielsweise Einträge aus Logdateien von Datenbanksystemen für unsere Zwecke wertlos, da der Bezug zu den semantisch komplexen Spezifikationen der Workflows nicht hergestellt werden kann. Mentor sieht daher Protokollierungsfunktionen in der Schicht der Integrations- und Ergänzungssoftware vor.

Durch eine laufende Überwachung der Ausführung von Workflows können Informationen über die Lastverteilung gewonnen werden, so daß bei Unbalanciertheit die Ausführungsorgane gleichmäßig mit Arbeit versorgt werden können. Dies kann sowohl für die verwendeten Systemressourcen wie z.B. Workstations als auch für die beteiligten Sachbearbeiter geschehen. Auf der Ebene der Systemressourcen kann die Balancierung weitgehend verdeckt erfolgen. Ergeben sich beispielsweise bei einem Sachbearbeiter zu lange Ausführungszeiten für seine Arbeitsschritte, können diese automatisch zu anderen Rechnern migriert werden. Das Workflow-Management-System entscheidet dies selbständig und für den Sachbearbeiter transparent, oder es gibt dem Systemadministrator zumindest Hilfestellung bezüglich des Verhältnisses von Nutzen durch Verwendung weiterer Rechner und den Kosten der notwendigen Verlagerung von Daten dorthin.

Eine Lastbalancierung auf der Ebene der Sachbearbeiter könnte beispielsweise durch einen allen Sachbearbeitern derselben Rolle gemeinsamen “Eingangspostkorb” unterstützt werden. Darüber hinausgehende Lastverteilungsmaßnahmen können dagegen auf dieser Ebene nur durch explizit erlassene Richtlinien oder durch persönliche Absprachen erfolgen. Ist ein Sachbearbeiter mit der Menge der auszuführenden Arbeitsschritte überfordert, können Arbeitsschritte zu anderen Sachbearbeitern verlagert werden. Das Workflow-Management-System sorgt für die Bereitstellung der erforderlichen State- und Activitycharts und Daten beim übernehmenden Sachbearbeiter.

Durch das Einbeziehen von Zeitaspekten in das Workflow–Management können weitere Verlagerungen von Arbeitsschritten nötig werden. Drohende Terminüberschreitungen können erkannt und vermieden werden, indem vor Überschreitung eines Termins neue Ressourcen alloziert werden oder eine notwendige Lastbalancierung durchgeführt wird. Werden Sachbearbeiter dem in Terminschwierigkeiten befindlichen Workflow neu zugeordnet, muß die Arbeit an anderen

Workflows langsamer ablaufen oder ganz ruhen. Die nun entstehenden neuen Terminverschiebungen müssen vom Workflow-Management-System erkannt und den Vorgesetzten zur Kenntnis gebracht werden.

5.3 Anwendungsbeispiel

In diesem Abschnitt werden die obigen Überlegungen zur Ausführung und Überwachung von Workflows in Mentor anhand eines Beispiels veranschaulicht. Wir legen wieder das Beispiel des Begutachtungsprozesses aus Abschnitt 4.2 zugrunde, konzentrieren uns aber nun auf das Zusammenwirken der Systemkomponenten aus Abbildung 3 zur Laufzeit.

Der Begutachtungsprozeß beginnt mit dem Eintreffen des Papiers. Der Editor registriert das Papier und bestimmt drei Gutachter. Er möchte kompetente Gutachter, die aber nicht gleichzeitig stark überlastet sind. Der Editor stellt also eine Anfrage an seine persönliche Datenbank, die ihm passende Gutachter liefert zusammen mit der Zahl der Papiere, die der jeweilige Gutachter für den Editor schon begutachtet hat. Die Middleware-Komponenten haben die Aufgabe, die Verbindung von den State- und Activitycharts des Editors zu seiner Datenbank herzustellen. Dazu werden C-Funktionen verwendet, die direkt mit Activities assoziiert werden. Im Beispiel aus Abbildung 1 ist dies beispielsweise der Code für die Activity *Einreichung_A*, die beim Schalten der Transition zum Zustand *Einreichung_S* aufgerufen wird. Dieser Code beinhaltet u.a. die Aufrufe an das lokale Datenbanksystem und an das dem Dialog mit dem Editor zugrundeliegende GUI. Die Verbindung zwischen der Activity *Einreichung_A* und der C–Funktion *einreichung* wird folgendermaßen hergestellt; die dabei aufgerufene Funktion `sc_connect_task` gehört zum Laufzeitsystem von Statemate.

```
sc_connect_task (”Einreichung_A”, einreichung);
```

Die C–Funktion *einreichung* kann wiederum mit Hilfe von Statemate–Funktionen wie folgt auf die Datenflußelemente des Workflows Bezug nehmen, wobei *autor_name* eine beliebige Variable der C–Funktion sein kann.

```
sc_set_string_data_item (”Autor”, autor_name);
```

Für das Aussuchen der Gutachter sind keine speziellen Funktionen der Middleware nötig. Im Fall eines Fehlers bei der Ausführung der beschriebenen Datenbankanfrage sollte sie wiederholt werden, bis ein Ergebnis vorliegt, oder bekannt wird, daß z. B. durch einen Ausfall einer Systemkomponente eine Wiederholung (zu diesem Zeitpunkt) nicht durchführbar ist. Da keine Änderungen in der Datenbank vorgenommen werden, ist dies problemlos möglich; bezüglich Fehlertoleranz liegen keine besonderen Anforderungen vor.

Nach der Auswahl der Gutachter werden mit dem Übergang in den Zustand *Begutachtung_S* die drei Activities *Begutachtung1_A* bis *Begutachtung3_A* gestartet und jeweils der Zustand *Gutachten_anfertigen* betreten (siehe Abb. 5). Diese Activities werden auf dem Rechner des Editors initiiert, müssen aber letztlich auf den Rechnern der jeweiligen Gutachter ablaufen. Wir nehmen an, daß jeder Gutachter für die Erstellung seines Gutachtens eine eigene Workstation benutzt und auf seine persönliche Datenbank wie auch auf öffentliche Datenbanken zugreift, um weitere Informationen zum Wissensgebiet des Papiers zu erhalten. Jetzt kommen wir nicht mehr mit einfachen C–Funktionen zum Ansprechen der Datenverwaltungssysteme aus. Die Papiere müssen an die entfernten Workstations der Gutachter geschickt werden. Dies wird durch den Einsatz des TP–Monitors oder einer verteilten Programmierumgebung in der Middleware realisiert: Die entfernten Systeme werden mittels Remote–Procedure–Call (RPC) angesprochen. RPC–Mechanismen erlauben allerdings meist nicht den Austausch komplexer Datenstrukturen, z.B. strukturierter Dokumente. Soll das eingereichte Papier in einer anderen Form als ASCII oder Postscript an

die Gutachter verteilt werden (es könnte sich beispielsweise um einen Hypertext handeln), so sind zusätzliche Komponenten der Integrations- und Ergänzungssoftware nötig, etwa zur Konvertierung des Dokuments in ein geeignetes Datenaustauschformat.

Nach Fertigstellung von mindestens zwei Gutachten soll der Begutachtungsprozeß beendet werden. Die Implementierung dieser Forderung stellt die bisher größten Anforderungen an die Middleware. Die Gutachten liegen verteilt in den Datenbanken der Gutachter vor und werden – wieder mittels RPC und eventueller Konvertierungsfunktionen – an den Editor gesendet. Dort werden die eintreffenden Gutachten gezählt (vgl. Abb. 5). Hier ist eine "Exactly–once"–Semantik des Versendens der Gutachten gefordert. Geht ein Gutachten verloren, wird die nötige Zahl von zwei Gutachten entweder gar nicht oder zumindest erst verspätet erreicht. Wird ein Gutachten mehrfach gesendet, wird es doppelt gezählt und der Begutachtungsprozeß wird fälschlicherweise schon beim Vorliegen eines Gutachtens beendet.

Da die Activities *Begutachtung1_A* bis *Begutachtung3_A* aus Sicht des Editors auf entfernten Rechnern laufen, ihre Beendigung und die (bis zu dreimalige) Generierung des Ereignisses *Gutachten_vorgelegt* aber beim Editor bekannt gemacht werden muß, werden wieder die Kommunikationsdienste des TP–Monitors benötigt. Stellt ein Gutachter sein Gutachten fertig, wird vom TP-Monitor garantiert, daß der Editor genau eine Nachricht über die beendete Begutachtung erhält (in Form des Ereignisses *Gutachten_vorgelegt)*. Dies ist die Voraussetzung dafür, daß der *Zähler* des Editors korrekt erhöht wird und sichergestellt ist, daß alle Einzelgutachten berücksichtigt werden. Abbildung 6 verdeutlicht, wie die Information über die Beendigung der Begutachtung unter Zuhilfenahme eines TP–Monitors vom Rechner eines Gutachters zum Rechner des Editors gelangt. Die drei Funktionsaufrufe `Ereignis_generieren`, `Ext_Ereignis_generieren` und `DB_Änderung` werden dabei zu einer atomaren Einheit geklammert.

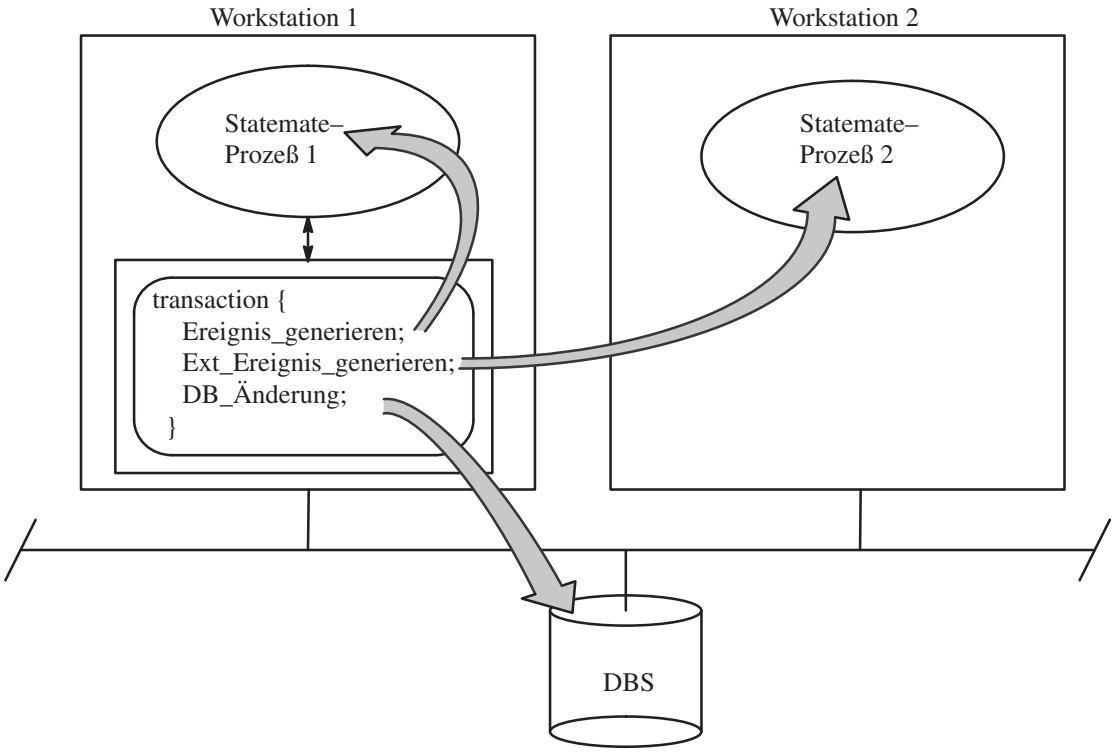

Abbildung 6: Transaktionsgeschützte Prozeßkommunikation mittels TP–Monitor

Der folgende Ausschnitt aus dem C–Code der Activity *Begutachtung1_A* (und analog *Begutachtung2_A* und *Begutachtung3_A*) zeigt exemplarisch eine mögliche Implementierung mit den Sprachmitteln von Encina [Enc93].

```
void Begutachtung1()
{
        ...
        EXEC SQL BEGIN DECLARE SECTION;
                ...
        EXEC SQL END DECLARE SECTION;
        ...
        Gutachten_editieren();
        ...
        sc_set_int_data_item (”Note”, meine_note);
        ...
        transaction{
                DB_Änderung();
                Ereignis_generieren(”Gutachten_vorgelegt”);
                Ext_Ereignis_generieren(”Gutachten_vorgelegt”);
        }
}
```

In diesem Codestück werden die drei Funktionsaufrufe `DB_Änderung`, `Ereignis_generieren` und `Ext_Ereignis_generieren` zu einer Transaktion zusammengefaßt. Diese Funktionen werden in der Encina–spezifischen “Transaction Interface Definition Language” wie folgt deklariert. Das Schlüsselwort “transactional” bedeutet, daß Aufrufe der deklarierten Funktionen innerhalb einer Transaktion transaktionsgeschützt sein sollen.

```
...
interface ...
{
        [transactional] void DB_Änderung ();
        [transactional] void Ereignis_generieren ([in] char *Ereignisname);
        [transactional] void Ext_Ereignis_generieren ([in] char *Ereignisname);
}
...
```

Die ersten beiden der drei Funktionen werden lokal – auf dem Rechner des Gutachters – ausgeführt; der Code dafür sieht folgendermaßen aus.

```
void DB_Änderung()
{
        EXEC SQL
                UPDATE Meine_Gutachten
                SET Note = :Note
                WHERE ...;
}


void Ereignis_generieren(char *Ereignisname)
{
                sc_do_action(Ereignisname);
}
```

Dabei beinhaltet `DB_Änderung` Aufrufe an das lokale Datenbanksystem, und `Ereignis_generieren` ruft die Funktion `sc_do_action` des lokalen Statemate–Prozesses zur Ereignissignalisierung auf.

Die dritte Funktion, `Ext_Ereignis_generieren`, soll auf dem Rechner des Editors entfernt aufgerufen werden und dem dort laufenden Statemate–Prozeß das Ereignis *Gutachten_vorgelegt* signalisieren. Der TP–Monitor setzt den Aufruf dieser Funktion automatisch in einen entsprechenden RPC um, so daß auf dem Rechner des Editors letztlich die folgende Funktion als Teil der verteilten Transaktion ausgeführt wird.

```
void Ext_Ereignis_generieren(char *Ereignisname)
{
        sc_do_action(Ereignisname);
}
```

Auf diese Weise werden also die Zustandsinformationen der beteiligten Statemate–Prozesse beim Gutachter bzw. den Gutachtern und beim Editor synchron aktualisiert, und zwar dank des TP–Monitors auf atomare Art und Weise. Dies ist eine notwendige Voraussetzung für die fehlertolerante Abarbeitung des spezifizierten Workflows, sicher aber noch nicht ausreichend. Im Fehlerfall genügt es nämlich noch nicht, die Effekte einer unvollständigen verteilten Transaktion zurückzusetzen; zusätzlich muß die Transaktion automatisch neu gestartet werden, und zwar so oft, bis sie erstmals erfolgreich beendet wird. Erweiterungen zur Gewährleistung einer solchen "Exactly–once"–Ausführungssemantik sollen in der Ergänzungsschicht von Mentor – aufbauend auf derartigen Diensten eines TP–Monitors – realisiert werden.

Weitere Anforderungen an die Middleware ergeben sich, wenn man zeitliche Beziehungen innerhalb oder zwischen Workflows betrachtet. In unserem Beispiel gibt es eine Frist, nach der alle Einzelgutachten vorliegen müssen. In einem angemessenen Zeitraum vor Erreichen dieser Frist, sollen Erinnerungsschreiben an die verspäteten Gutachter verschickt werden (Ereignis *Frist_naht* in Abb. 5). Dies erfordert eine Schedulingkomponente in der Middleware, die ihre Informationen – wie die anderen Komponenten auch – aus den Definitionen der State- und Activitycharts erhält. Liefert mehr als ein Gutachter gar kein Gutachten, muß mindestens ein weiteres Gutachten von einem zusätzlichen Gutachter eingeholt werden. Die entsprechende Aktivität wird verlagert. Dies zu ermöglichen, ist eine wesentliche Anforderung an ein Workflow-Management-System. In großen Unternehmen müssen ständig Arbeitsschritte von einem Sachbearbeiter an einen anderen delegiert werden, z.B. bei Krankheit. Dabei wird man üblicherweise die vor der Delegation vorliegenden Teilergebnisse mitgeben wollen, um Doppelarbeit zu vermeiden. In der Integrations- und Ergänzungsschicht der Middleware sind entsprechende Mechanismen vorzusehen, die selbst wieder andere Komponenten der Middleware benutzten, um die Delegation durchzuführen. Beispielsweise benötigt man auch dafür eine "Exactly–once"–Semantik, damit nicht später mehrere oder gar kein Sachbearbeiter den betreffenden Arbeitsschritt fortführen.

Die zur Überwachung des Workflows vorgesehene Protokollierungskomponente von Mentor erlaubt Anfragen an die Historie der Workflows. Beispielsweise könnte der Editor in unserem Beispiel wissen wollen, wie oft ein bestimmter Gutachter in früheren Fällen die Frist überschritten hat, oder ein Gutachter könnte in Erfahrung bringen wollen, ob die anderen Gutachter bereits mit ihren Gutachten fertig sind (ohne die Identität der Gutachter preiszugeben). Derartige Möglichkeiten würden hoffentlich zu einem zügigeren Begutachtungsprozeß beitragen.

6 Zusammenfassung und Ausblick

In diesem Artikel haben wir einen neuartigen Ansatz für Workflow–Management vorgestellt, der auf State– und Activitycharts beruht. Diese Spezifikationsmethode hat sich für die Modellierung technischer Steuerungssysteme sehr bewährt; ihre Verwendung für unternehmensweite Workflows wurde unseres Wissens zuvor noch nicht erwogen. Da wir mit dem Mentor–Projekt in diesem Sinne Neuland betreten, liegen natürlicherweise noch keine tiefgehenden Ergebnisse über den gewählten Ansatz vor. Unsere hier vorgestellten Modellierungsüberlegungen weisen jedoch auf ein sehr gutes Potential der verwendeten Spezifikationsmethode hin. Im Vergleich zu anderen Methoden zur Workflow–Spezifikation – Skriptsprachen, ECA–Regeln, Petrinetze oder ähnliches – konnten insbesondere die hervorragenden, intuitiv leicht nachvollziehbaren Möglichkeiten zur Verfeinerung und Komposition von Workflows demonstriert werden. Diese Eigenschaf-

ten sind für die Spezifikation komplexer Workflows und die Wiederverwendung existierender Arbeitsabläufe in übergeordneten, unternehmensweiten Workflows von entscheidender Bedeutung.

Über die reine Spezifikation von Workflows hinaus verfolgen wir im Mentor–Projekt das Anliegen, Spezifikationen direkt in der zugrundeliegenden verteilten und heterogenen Systemlandschaft ausführen zu können. Zur Ausführung, Überwachung und Steuerung von Workflows wollen wir die Spezifikationsumgebung Statemate mit geeigneten Middleware–Komponenten koppeln, insbesondere mit einem TP–Monitor. Die grundsätzliche Vorgehensweise für eine solche Kopplung und die damit verbundenen Möglichkeiten haben wir in diesem Artikel demonstriert. Dabei wurde die der State– und Activitychart–Methode innewohnende "offene Architektur" ausgenutzt, indem die spezifizierten Activities um Aufrufe des TP–Monitors und anderer externer Subsysteme erweitert wurden. Zur Realisierung der projektierten vollständigen Workflow–Management–Umgebung ist freilich noch ein gewaltiges Stück Forschungsarbeit zu leisten. Dieses Ziel wird in einem gemeinsamen Projekt der Universität des Saarlandes, des UBILAB der Schweizerischen Bankgesellschaft und der ETH Zürich verfolgt. Im Vordergrund stehen dabei naturgemäß Anwendungen aus dem Bankbereich, beispielsweise die Abwicklung von Kreditanträgen. Das erst vor kurzem begonnene Mentor–Projekt steckt sicher noch in den Kinderschuhen, ist aber nach unserer Überzeugung auf dem richtigen Weg zu ausgereifteren Resultaten.

Literaturverzeichnis

[Bi94] A. Biliris, S. Dar, N. Gehani, H.V. Jagadish, K. Ramamritham, ASSET: A System for Supporting Extended Transactions, ACM SIGMOD Conference, 1994

[BMR94] D. Barbara, S. Mehrotra, M. Rusinkiewicz, INCAS: A Computation Model for Dynamic Workflows in Autonomous Distributed Environments, Technical Report, Matsushita Information Technology Laboratory, Princeton, 1994

[Be93] P.A. Bernstein, Middleware: An Architecture for Distributed System Services, Technical Report, Digital Corporation, Cambridge Research Laboratory, 1993

[Br93] Y. Breitbart, A. Deacon, H.–J. Schek, A. Sheth, G. Weikum, Merging Application–centric and Data–centric Approaches to Support Transaction–oriented Multi–system Workflows, ACM SIGMOD Record Vol.22 No.3, September 1993

[CBHR93] V.J. Cahill, R. Balter, N. Harris, X. Rousset de Pina (Editors), The Comandos Distributed Application Platform, Springer–Verlag, 1993

[DHL91] U. Dayal, M. Hsu, R. Ladin, A Transactional Model for Long–Running Activities, VLDB Conference, 1991

[Da93] U. Dayal, H. Garcia–Molina, M. Hsu, B. Kao, M.–C. Shan, Third Generation TP Monitors: A Database Challenge, ACM SIGMOD Conference, 1993

[De94] P.J. Denning, The Fifteenth Level, Keynote Address, ACM SIGMETRICS Conference, 1994

[Enc93] An Introduction to Programming the Encina Monitor, Transarc Corporation, 1993

[EN93] C.A. Ellis, G.J. Nutt, Modeling and Enactment of Workflow Systems, Invited Paper, 14th International Conference on Application and Theory of Petri Nets, 1993

[Fu93] U. Furbach, Formal Specification Methods for Reactive Systems, Journal of Systems Software Vol. 21, pp. 129–139, 1993

[GHKM94] D. Georgakopoulos, M. Hornick, P. Krychniak, F. Manola, Specification and Management of Extended Transactions in a Programmable Transaction Environment, IEEE Data Engineering Conference, Houston, 1994

[GGS93] M. Gesmann, A. Grasnickel, H. Schoening, A Remote Cooperation System Supporting Interoperability in Heterogeneous Environments, IEEE International Workshop on Research Issues in Data Engineering: Interoperability of Multidatabase Systems, Vienna, 1993

[GR93] J. Gray, A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann, 1993

[Ha87] D. Harel, Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming Vol.8, 1987, pp. 231–274

[Ha88] D. Harel, On Visual Formalisms, Communications of the ACM Vol.31 No.5, 1988

[Ha90] D. Harel et al., STATEMATE: A Working Environment for the Development of Complex Reactive Systems, IEEE Transactions on Software Engineering Vol.16 No.4, April 1990

[HPSS87] D. Harel, A. Pnueli, J.P. Schmidt, R. Sherman, On the Formal Semantics of Statecharts, 2nd IEEE Symposium on Logic in Computer Science, 1987

[Hsu93] M. Hsu (Editor), IEEE Data Engineering Bulletin Vol.16 No.2, June 1993, Special Issue on Workflow and Extended Transaction Systems

[i–Log91] i–Logix Inc., Languages of STATEMATE, in: Documentation for the Statemate System, 1991

[Jab93] S. Jablonski, Aktivitäten–Management–Systeme: Ziele, Grundlagen und Lösungen, Manuskript, 1993

[KS91] G. Kappel, M. Schrefl, Object/Behavior Diagrams, IEEE Data Engineering Conference, 1991

[KUW95] St. Kirn, R. Unland, U. Wanka, MAMBA: Automatic Customization of Computerized Business Processes, Information Systems Vol.19 No.8, 1994

[LA94] F. Leymann, W. Altenhuber, Managing Business Processes as an Information Resource, IBM Systems Journal Vol.33 No.2, 1994

[McC93] T. McCusker, Workflow takes on the enterprise, Datamation, Dec 1, 1993

[MC94] T.W. Malone, K. Crowston, The Interdisciplinary Study of Coordination, ACM Computing Surveys Vol.26 No.1, March 1994

[Me94] W.P. Melling, Enterprise Information Architectures – They're Finally Changing, Invited Industrial Plenary Talk, ACM SIGMOD International Conference on Management of Data, Minneapolis, 1994

[Mu93] S. Mullender (Editor), Distributed Systems, ACM Press, 2nd Edition, 1993

[Ob94] R. Obermarck (Editor), IEEE Data Engineering Bulletin Vol.17 No.1, March 1994, Special Issue on TP Monitors and Distributed Transaction Management

[Ober94] A. Oberweis, Workflow Management in Software Engineering Projects, 2nd International Conference on Concurrent Engineering and Electronic Design Automation, 1994

[OSS94] A. Oberweis, G. Scherrer, W. Stucky, INCOME/STAR: Methodology and Tools for the Development of Distributed Information Systems, Information Systems Vol.19 No.8, 1994

[OMG92] Object Management Group, The Common Object Request Broker: Architecture and Specification, 1992

[OMG94] Object Management Group, The Common Object Services Specification, Volume 1, 1994

[Rei93] B. Reinwald, Workflow–Management in verteilten Systemen, Teubner–Verlag, 1993

[RSW92] A. Reuter, F. Schwenkreis, H. Wächter, Zuverlässige Abwicklung großer verteilter Anwendungen mit ConTracts – Architektur einer Prototypimplementierung, in: R. Bayer, T. Härder, P. Lockemann (Hsrg.), Objektbanken für Experten, Springer–Verlag, 1992

[RS94] M. Rusinkiewicz, A. Sheth, Specification and Execution of Transactional Workflows, in: W. Kim (Editor), Modern Database Systems: The Object Model, Interoperability, and Beyond, ACM Press, 1994

[Saa93] G. Saake, Objektorientierte Spezifikation von Informationssystemen, Teubner–Verlag, 1993

[SK94] A. Sheth, N. Krishnakumar, Specification of Workflows with Heterogeneous Tasks in METEOR, 20th VLDB Conference, Poster Paper Collection, 1994

[Sta94] 2. Deutsches Anwenderforum für STATEMATE/Express V–HDL, Berner & Mattner GmbH, 1994

[ST94] B. Salzberg, D. Tombroff, DSDT: Durable Scripts Containing Database Transactions, Manuscript, submitted for publication, 1994

[Sch94] A.–W. Scheer, ARIS Toolset: a Software Product is Born, Information Systems Vol.19 No.8, 1994

[Schi93] A. Schill (Editor), DCE – The OSF Distributed Computing Environment, Client/Server Model and Beyond, International DCE Workshop, Springer, 1993

[Schw93] F. Schwenkreis, APRICOTS – A Prototype Implementation of a ConTract System – Management of the Control Flow and the Communication System, 12th Symposium on Reliable Distributed Systems, 1993

[WR92] H. Wächter, A. Reuter, The ConTract Model, in: A.K. Elmagarmid (Editor), Database Transaction Models for Advanced Applications, Morgan Kaufmann, 1992

[Wei93] G. Weikum, Extending Transaction Management to Capture More Consistency with Better Performance, Invited Paper, 9th French Database Conference, Toulouse, September 1993

[WD94] J. Widom, U. Dayal (Editors), A Guide To Active Databases, Morgan Kaufmann, 1994