

Objektorientierter Entwurf eines flexiblen Workflow-Management-Systems

Mathias Weske, Jens Hündling, Dominik Kuroopka, Hilmar Schuschel

Lehrstuhl für Informatik, Universität Münster
Steinfurter Straße 107, D-48149 Münster
{weske,hundlin,kuroopka,schusch}@helios.uni-muenster.de

Zusammenfassung. Dieser Beitrag enthält die wesentlichen konzeptionellen Überlegungen zur Entwicklung eines flexiblen Workflow-Management-Systems. Ausgehend von einer Diskussion möglicher Modellierungsalternativen wird unter Verwendung objektorientierter Modellierungstechniken ein Workflow-Meta-Schema entwickelt, das die relevanten Entitäten klassifiziert und ihre Beziehungen abbildet. Anhand eines Beispiels werden die getroffenen Designentscheidungen ausführlich begründet. Schließlich werden Zustände und Zustandsübergänge von Workflow-Instanzen beschrieben, und es wird diskutiert, wie dynamische Veränderungen von Workflows zur Laufzeit durchgeführt werden können, um laufende Workflows dynamisch an veränderte Umweltbedingungen anzupassen.

Schlüsselwörter: Flexibles Workflow-Management, Workflow-Meta-Schema, Dynamische Veränderungen, Objektmodellierung, Systemdesign

Abstract: This contribution discusses the conceptual design of a flexible workflow management system. After discussing a set of modeling alternatives, a workflow meta schema is developed, using object oriented design principles. We describe how workflows are modelled and executed. Key design decisions are validated using a sample workflow model and a sample workflow instance based on that model. We propose a state transition diagram for workflow instances, and we discuss how dynamic modifications of workflow models can be conducted to adapt running workflow instances to changes in the workflow environment.

Keywords: flexible workflow management, workflow meta schema, dynamic changes, object modeling, system design

ACM CR Classification: D.2; C.2.4; H.4; H.5.3

1 Einführung

Workflow-Management-Systeme sind komplexe Software-Systeme, die zur Modellierung und kontrollierten

Ausführung von Anwendungsprozessen in unterschiedlichen Umgebungen eingesetzt werden können [8, 11, 25]. Standen in der ersten Phase des Einsatzes von Workflow-Technologie kommerzielle Büroanwendungen mit einer fest vordefinierten Prozeßstruktur im Vordergrund [14], die auf homogenen Software-Plattformen ausgeführt wurden, so finden Workflow-Techniken in zunehmendem Maß in naturwissenschaftlichen [21, 28] und technischen [15] Umgebungen Verwendung, wo die Prozesse meist weniger stark strukturiert sind [30] und wo es sich typischerweise um heterogene Hard- und Software-Umgebungen handelt. In diesem Beitrag werden die wesentlichen konzeptionellen Überlegungen beim objektorientierten Design eines flexiblen Workflow-Management-Systems entwickelt. Wir verwenden einen objektorientierten Ansatz, um die Verwendung des Systems in heterogen verteilten Umgebungen zu erleichtern; wir sprechen von einem flexiblen Workflow-Management-System, da dynamische Modifikationen unterstützt werden, die es ermöglichen, Workflow-Instanzen zur Ausführungszeit an veränderte Umweltbedingungen anzupassen.

Die objektorientierte Modellierung besitzt eine Reihe von Eigenschaften, die sie gerade im Workflow-Kontext attraktiv macht. Zum einen handelt es sich bei Workflow-Management-Systemen um komplexe Software-Systeme, deren Entwurf von den Abstraktionsmöglichkeiten objektorientierter Analysemethoden profitieren kann. Darüber hinaus stehen mit einem objektorientierten Design Möglichkeiten offen, verteilte Objektsysteme als technische Infrastruktur einer Implementierung zu verwenden, etwa die unter der Regie der Object Management Group (OMG) spezifizierte CORBA-Architektur (Common Object Request Broker Architecture) [17, 27]. In diesem Zusammenhang sind Aktivitäten der OMG zu nennen, die auf die Definition einer Workflow-Facility [18] sowie einer Business-Object-Facility [4] abzielen. Mit der Workflow-Facility sollen workflow-spezifische Dienste definiert werden, um sie im Rahmen der CORBA-Architektur Anwendungen zur Verfügung zu stellen. Business-Objekte stellen spezielle Strukturen und Funktionen in gekapselter Form bereit, die über CORBA-Schnittstellen abgerufen werden können. Mit einem CORBA-basierten

Workflow-Management-System wird die Einbindung von Business-Objekten in Workflow-Anwendungen aufgrund der gemeinsamen Schnittstellen signifikant erleichtert. Die Workflow-Facility und die Business-Object-Facility befinden sich derzeit im Spezifikationsprozeß; erste Begutachtungen der Einreichungen zur Workflow-Facility haben eine Reihe von Defiziten, u.a. bei der Objektmodellierung herausgearbeitet [23], von denen die wichtigsten weiter unten skizziert werden.

Die in diesem Beitrag vorgestellten Ansätze und Konzepte wurden im Rahmen des WASA-Projekts [1, 29] an der Universität Münster erarbeitet, welches sich mit der Unterstützung flexibler Workflows [28, 30] beschäftigt; sie beschreiben die Design-Entscheidungen bei der Entwicklung von WASA₂, der prototypischen Realisierung eines flexiblen, CORBA-basierten Workflow-Management-Systems.

Dieser Beitrag ist wie folgt gegliedert. In Abschnitt 2 werden Workflow-Grundlagen vorgestellt; Abschnitt 3 präsentiert das Workflow-Meta-Schema des WASA₂-Systems und begründet die getroffenen Modellierungsentscheidungen anhand einer Diskussion von Modellierungsalternativen sowie eines Beispiel-Workflows. Abschnitt 4 beschäftigt sich mit der Modellierung dynamischer Aspekte, wobei zunächst Workflow-Ausführungen beschrieben werden, bevor das Verhalten von Workflow-Instanzen durch Zustandsübergangsdiagramme untersucht wird. In Abschnitt 5 diskutieren wir, durch welche Eigenschaften des Workflow-Meta-Schemas das zur Flexibilisierung wichtige Konzept der dynamischen Modifikation realisiert wird. Abschnitt 6 betrachtet Aspekte von Sicherheit und Fehlertoleranz bei Workflow-Ausführungen. Eine Diskussion verwandter Arbeiten und ein Ausblick schließen diesen Beitrag.

2 Workflow-Grundlagen

Um das Ziel der kontrollierten Ausführung von Anwendungsprozessen zu erreichen, benötigen Workflow-Management-Systeme eine von ihnen zu verarbeitende Spezifikation dieser Prozesse; solche Spezifikationen werden als Workflow-Schemata bezeichnet [12]. Zur Spezifikation von Workflow-Schemata werden Workflow-Sprachen verwendet [32]. Wir verwenden im weiteren Verlauf dieses Beitrags eine graph-basierte Workflow-Sprache, bei der Workflow-Schemata als geschachtelte, gerichtete Graphen repräsentiert werden [30]. Die Schachtelung wird zur Modularisierung von Workflow-Schemata verwendet, mit den Zielen der Wiederverwendbarkeit und der Komplexitätsreduktion. Die Hierarchisierung von Workflows wird dadurch realisiert, daß komplexe und atomare Workflow-Schemata zur Verfügung stehen. Während komplexe Workflow-Schemata aus einer Menge von (komplexen oder atomaren) Workflow-Schemata und deren Beziehungen bestehen, sind atomare Workflow-Schemata nicht weiter verfeinert; sie werden meist durch Applikationsprogramme realisiert, deren Ausführung durch das Workflow-Management-System angestoßen wird. Findet keine Benutzerinteraktion bei der Ausführung eines solchen Applikationsprogramms

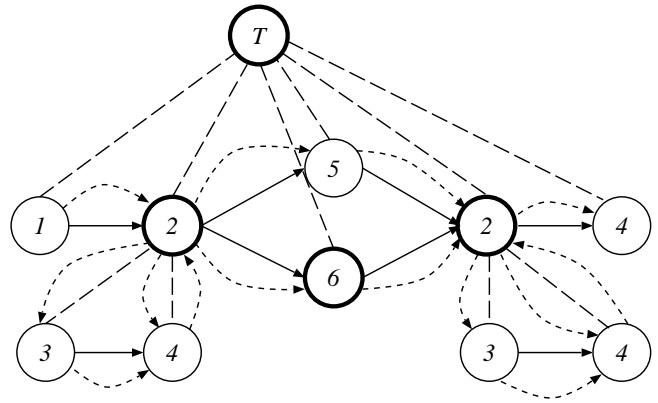


Abb. 1. Komplexes Workflow-Schema T , mit Kontroll- und Datenflußbeziehungen zwischen Sub-Workflow-Schemata.

statt, so handelt es sich um einen automatischen Workflow, sonst um einen manuellen Workflow.

Ein durch ein komplexes Workflow-Schema charakterisierter Anwendungsprozeß wird aufgrund seiner relativen Position zu den anderen Workflows auch als Toplevel-Workflow bezeichnet; er besteht aus einer Menge von Sub-Workflows. Diese Sub-Workflows können eine Reihe von Beziehungen zueinander besitzen, etwa Datenfluß- und Kontrollflußbeziehungen. Unter Datenfluß verstehen wir die systemkontrollierte Weitergabe von Daten, die durch Workflows generiert oder verändert werden. Kontrollfluß beschreibt die relative Ausführungsreihenfolge von Sub-Workflows innerhalb eines komplexen Workflows. Um Datenfluß geeignet modellieren zu können, besitzt jeder Workflow eine Menge von Eingabeparametern und eine Menge von Ausgabeparametern. Durch Verbindung eines Ausgabeparameters eines Workflows i mit einem Eingabeparameter eines Workflows j wird ein Datenfluß von i nach j modelliert.

Abbildung 1 zeigt ein Workflow-Schema T , bei dem Sub-Workflow-Schemata sowie Kontroll- und Datenflußbeziehungen zwischen diesen angegeben sind. T besitzt komplexe und atomare Sub-Workflow-Schemata; komplexe Workflow-Schemata werden durch Fettdruck repräsentiert. Gerichtete Kanten zwischen Sub-Workflow-Schemata geben Kontrollflußbeziehungen an; Datenflußbeziehungen werden durch gestrichelte, gerichtete Kanten dargestellt; die Zuordnung von komplexen Workflows zu den Sub-Workflows wird durch ungerichtete, gestrichelte Kanten repräsentiert. Aus Gründen der Übersichtlichkeit wird auf die Darstellung von Parametern verzichtet und Datenflußbeziehungen zwischen Workflow-Schemata und nicht zwischen Parametern von Workflow-Schemata eingezeichnet. Komplexe Workflow-Schemata können durch Definition ihrer Sub-Workflow-Schemata und deren Beziehungen verfeinert sein, wie dies bei dem komplexen Workflow-Schema 2 dargestellt ist. In Abbildung 1 wird die interne Struktur des komplexen Workflow-Schemas 6 nicht dargestellt; diese wird bei der Diskussion von Flexibilitätseigenschaften in Abschnitt 5 genauer betrachtet.

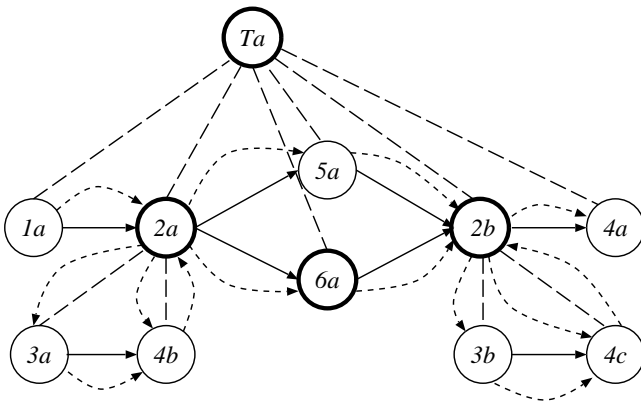


Abb. 2. Komplexe Workflow-Instanz Ta , basierend auf Workflow-Schema T .

Wir weisen darauf hin, daß Workflow-Schemata eine Reihe weiterer Komponenten besitzen, die später ausführlich beschrieben werden; dazu gehören

- Ein- und Ausgabeparameter, mit deren Hilfe Datenfluß realisiert werden kann,
- Startbedingungen, die zur Laufzeit ausgewertet werden, um zu entscheiden, ob ein Sub-Workflow ausgeführt werden soll oder nicht,
- Rollen und Agenten, die den Organisationsaspekt abdecken [32], sowie
- Datentypen von Parametern und Anwendungsdaten.

Um die Wiederverwendung von Workflow-Schemata zu unterstützen, können Workflow-Schemata in mehreren komplexen Workflow-Schemata als Sub-Workflow-Schemata vorkommen. Darüber hinaus können bestimmte Sub-Workflow-Schemata in einem komplexen Workflow-Schema mehrfach vorkommen; so kommt das Workflow-Schema 2 in Abbildung 1 zweimal als Sub-Workflow-Schema von T vor. Unterschiedliche Vorkommen eines Workflow-Schemas in verschiedenen Workflow-Schemata oder in einem komplexen Workflow-Schema besitzen in der Regel unterschiedliche Eigenschaften bezüglich ihrer Einbettung in den jeweiligen Kontext. So können die Vorkommen unterschiedliche Startbedingungen und/oder Kontroll- und Datenflußbeziehungen zu anderen Sub-Workflow-Schemata besitzen. Aus diesem Grund sollte bei der Modellierung von Workflows ein hohes Maß an Modularität zur Verfügung gestellt werden.

Wie später genauer dargestellt wird, repräsentieren wir die Struktur von komplexen Workflow-Schemata durch Beziehungen zwischen einem komplexen Workflow-Schema und seinen Sub-Workflow-Schemata. Diese Beziehungen werden dazu verwendet, spezielle Eigenschaften des Workflow-Schemas im Hinblick auf seine aktuelle Verwendung zu definieren. Auf diese Weise ist es möglich, daß das Workflow-Schema 2 mehrfach als Sub-Workflow-Schema von T auftritt, wobei jeweils unterschiedliche Startbedingungen sowie unterschiedliche Kontroll- und Datenflußbedingungen definiert sein können.

Abbildung 2 zeigt eine Workflow-Instanz, basierend auf Workflow-Schema T . Eine Workflow-Instanz bezeich-

net die systeminterne Repräsentation eines konkreten Anwendungsprozesses, der durch ein Workflow-Management-System unter Verwendung eines Workflow-Schemas kontrolliert wird. Beim Start einer Workflow-Instanz wird das entsprechende komplexe Workflow-Schema dazu verwendet, für jedes Sub-Workflow-Schema eine entsprechende Workflow-Instanz zu erzeugen. Wie die Erzeugung genau abläuft, wird später erklärt; an dieser Stelle ist wichtig, daß für jedes Auftreten eines Sub-Workflow-Schemas in einem komplexen Workflow genau eine Sub-Workflow-Instanz erzeugt wird. Im obigen Beispiel werden daher zunächst die komplexe Workflow-Instanz Ta und ihre unmittelbaren Sub-Workflow-Instanzen $1a$, $2a$, $5a$, $6a$, $2b$, $4a$ erzeugt.

Neben diesen Unterschieden bestehen auch eine Reihe von Gemeinsamkeiten zwischen Workflow-Schemata und Workflow-Instanzen. Die wohl größte Gemeinsamkeit besteht in der analogen Struktur von Schema und Instanz. Die Beziehungen des komplexen Workflow-Schemas T zu seinen Sub-Workflow-Schemata spiegelt sich wider in der Beziehung der komplexen Workflow-Instanz Ta zu ihren Sub-Workflow-Instanzen. Wie bereits oben erwähnt, besteht ein Workflow-Schema aus einer Reihe weiterer Komponenten, wie etwa Ein- und Ausgabeparameter, Rolleninformationen und Startbedingungen. Bei der genauen Untersuchung der Instanz-Ebene fällt dabei auf, daß die entsprechenden Beziehungen auch für diese Komponenten auf Schema- und Instanz-Ebene analog auftreten, so daß insgesamt eine große Ähnlichkeit in der Struktur von Workflow-Schemata und -Instanzen besteht.

Natürlich besitzen Workflow-Schemata und Workflow-Instanzen unterschiedliches Verhalten. So kann man eine Workflow-Instanz starten, anhalten, einem Agenten zur Ausführung zuweisen, während diese Funktionen für Workflow-Schemata nicht zur Verfügung stehen. Jedoch gibt es auch gemeinsame Funktionen, etwa die Suche nach den (durch Kontrollflußbeziehungen definierten) Nachfolgern eines Sub-Workflows. Diese Eigenschaften werden im folgenden Abschnitt mit ihren Auswirkungen auf die objektorientierte Modellierung der logischen Struktur unseres Workflow-Management-Systems genauer untersucht.

3 Objektorientierte Modellierung von Workflow-Meta-Schemata

In diesem Abschnitt wird das in $WASA_2$ verwendete Workflow-Meta-Schema entwickelt. Da wir objektorientierte Techniken zur Implementierung des Systems verwenden, liegt eine objektorientierte Modellierung des Workflow-Meta-Schemas nahe. Aus Sicht der Software-Entwicklung ist das Workflow-Meta-Schema durch ein Objektmodell beschrieben, welches die Struktur von Objekten in einem System und ihre Beziehungen untereinander abbildet. Im allgemeinen wird dabei zwischen einem Analyse-Objektmodell und einem Design-Objektmodell unterschieden [22]. Bei der Entwicklung eines Software-Systems wird meist zunächst ein Analyse-Objektmodell erstellt, welches die logische Struktur des zu

entwickelnden Systems beschreibt, ohne bereits festzulegen, wie die Implementierung des Systems erfolgen soll. Durch Ergänzung des Modells um Implementierungsdetails wird ein Design-Objektmodell erstellt. Dient das Analyse-Objektmodell primär dem Verständnis der Anwendung, so liegt der Schwerpunkt der Betrachtungen beim Übergang zum Design-Objektmodell auf einer angemessenen und effizienten Implementierung des Systems. In diesem Beitrag konzentrieren wir unsere Betrachtungen auf das Analyse-Objektmodell; das entsprechende Design-Objektmodell ist in [33] beschrieben. Da dieses Analysemodell den Aufbau und die Struktur von Workflow-Schemata beschreibt, stellt es ein Workflow-Meta-Schema dar. Zunächst werden unterschiedliche Modellierungsalternativen diskutiert, und die von uns gewählte Modellierungsalternative wird begründet. Nach der Vorstellung des WASA₂-Workflow-Meta-Schemas werden die getroffenen Modellierungsentscheidungen anhand eines Beispiels veranschaulicht, das ein komplexes Workflow-Schema und eine auf diesem basierende Workflow-Instanz mit den involvierten Objekten und ihren Beziehungen beschreibt.

3.1 Modellierungsalternativen

Bei der Modellierung komplexer Gegenstandsbereiche gibt es stets eine Reihe unterschiedlicher Modellierungsalternativen, die sich in ihrer Angemessenheit im Hinblick auf das Modellierungsziel unterscheiden. Nachdem im vorhergehenden Abschnitt erklärt wurde, wie Workflow-Schemata und Workflow-Instanzen strukturiert sind, werden nun mögliche Modellierungsalternativen vorgestellt.

3.1.1 Workflow-Schemata als Klassen

Eine erste, naheliegende Modellierungsalternative sieht vor, Workflow-Schemata als Klassen zu repräsentieren und Workflow-Instanzen als Objekte dieser Klassen. Da komplexe Workflows aus einer Menge von Sub-Workflows und Beziehungen zwischen diesen bestehen, besitzt eine entsprechende Workflow-Klasse Aggregationsbeziehungen zu einer Menge von Klassen, die ihre Sub-Workflow-Schemata repräsentieren. Wird nun eine Workflow-Instanz erzeugt, so werden Objekte der komplexen Workflow-Schema-Klasse sowie Objekte aller Sub-Workflow-Schema-Klassen erzeugt. Aus dieser Form der Modellierung ergeben sich eine Reihe von Nachteilen:

- Werden spezielle Workflow-Schemata in unterschiedlichen komplexen Workflow-Schemata als Sub-Workflows verwendet, so entsteht eine komplexe Klassenstruktur. Falls nun Workflow-Instanzen erzeugt werden, so sind diese aufgrund der Klassenstruktur mehreren komplexen Workflows zugeordnet. Dies ist jedoch aus Sicht der Anwendung nicht korrekt, da jede Workflow-Instanz höchstens einer komplexen Workflow-Instanz zugeordnet sein kann. (Toplevel-Workflows sind keiner Workflow-Instanz zugeordnet.)

Damit beschreibt diese Modellierungsalternative nicht die durch die Anwendungsumgebung gegebenen Zusammenhänge.

- Bei dieser Form der Modellierung werden Workflow-Schemata durch Klassen dargestellt, so daß die Neudefinition eines Workflow-Schemas einer Struktur-Veränderung entspricht. Da Struktur-Veränderungen im allgemeinen schwerer handhabbar als Werteveränderungen sind, erscheint diese Modellierungsalternative gerade im Hinblick auf dynamische Modifikationen wenig angebracht.
- Falls es zu einer Modifikation eines Workflow-Schemas kommt, so entspricht dies ebenfalls einer Struktur-Veränderung. In diesem Fall sind komplexe Operationen notwendig, um die Workflow-Instanzen der veränderten Workflow-Schema-Klasse an die neue Struktur anzupassen. Oft ist es nicht sinnvoll oder möglich, alle Workflow-Instanzen an ein verändertes Schema anzupassen. Darüber hinaus sind Anpassungsoperationen während der Laufzeit oft nicht möglich, so daß das System zunächst angehalten werden muß, bevor die Anpassungen erfolgen können. Dieses Verhalten ist – gerade im Hinblick auf Flexibilitätseigenschaften – für Workflow-Management-Systeme im allgemeinen nicht wünschenswert.

Diese Probleme führen zur Entwicklung eines zweiten Ansatzes, bei dem Workflow-Schemata und Workflow-Instanzen als Objekte zweier generischer Klassen modelliert werden.

3.1.2 Generische Workflow-Klassen

Die Grundidee dieses Ansatzes besteht darin, alle Workflow-Schemata in einer generischen Workflow-Schema-Klasse und alle Workflow-Instanzen in einer generischen Workflow-Instanz-Klasse zu repräsentieren. Die Zuordnung von Workflow-Instanzen zu Workflow-Schemata kann im Objektmodell durch eine Beziehung zwischen den entsprechenden Klassen dargestellt werden.

Bei dieser Modellierungsalternative entspricht die Definition eines Workflow-Schemas nicht mehr einer Strukturveränderung, sondern einer Werteveränderung. Damit sind Veränderungen von Workflow-Schemata leichter durchzuführen und sogar Veränderungen zur Laufzeit von Workflow-Instanzen möglich, wie später genauer untersucht wird. Die Wiederverwendbarkeit von Workflow-Schemata als Sub-Workflow-Schemata unterschiedlicher komplexer Workflow-Schemata ist bei dieser Form der Modellierung einfacher möglich, da diese Zuordnung durch Beziehungen zwischen Objekten und nicht durch Klassenzugehörigkeiten abgebildet werden. Bei diesem generellen Ansatz gibt es eine Reihe von Modellierungsmöglichkeiten, die sich in den Beziehungen zwischen den beiden Klassen unterscheiden; von diesen Möglichkeiten werden nun zwei wesentliche untersucht:

Modellierung mit Einfachvererbung Um die Gemeinsamkeiten zwischen Workflow-Schemata und Workflow-Instanzen möglichst redundanzfrei zu modellieren, werden

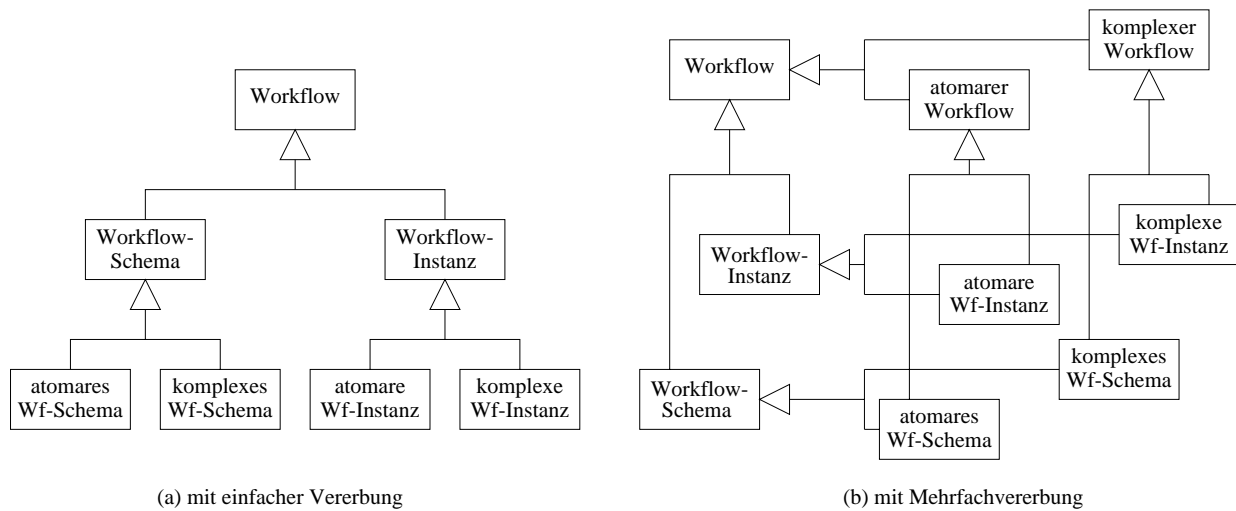


Abb. 3. Ansätze für mögliche Modellierungsalternativen für Workflow-Meta-Schemata; ohne (a) bzw. mit Mehrfachvererbung (b).

diese Gemeinsamkeiten in einer Klasse *Workflow* zusammengefaßt; die Klassen *Workflow-Schema* und *Workflow-Instanz* sind dann Spezialisierungen der Klasse *Workflow*. Wie in Abschnitt 2 erläutert, sind *Workflow-Schemata* und *Workflow-Instanzen* entweder atomar oder komplex. Struktur und Verhalten von atomaren und komplexen Workflows unterscheiden sich voneinander, da komplexe Workflows aus einer Menge von Sub-Workflows bestehen, die unter Beachtung von Bedingungen ausgeführt werden. Die speziellen Eigenschaften komplexer bzw. atomarer Workflows werden durch Spezialisierungsbeziehungen repräsentiert. Da *Workflow-Schemata* und *Workflow-Instanzen* in unterschiedlichen Klassen verwaltet werden, besitzt jede dieser Klassen die entsprechenden Sub-Klassen für atomare bzw. komplexe *Workflow-Schemata* bzw. *Workflow-Instanzen*. Das daraus resultierende Klassendiagramm ist unter Verwendung der Unified Modeling Language (UML) [19], insbesondere der Generalisierungsbeziehungen, in Abbildung 3(a) dargestellt. (Auf weitere Eigenschaften von UML wird bei der Vorstellung des *WASA₂-Workflow-Meta-Schemas* in Abschnitt 3.2 eingegangen.) Bei diesem Klassendiagramm wird auf Mehrfachvererbung verzichtet, was eine redundante Beschreibung der Gemeinsamkeiten von atomaren und komplexen Workflows erforderlich macht.

Modellierung mit Mehrfachvererbung Eine Lösung bietet hier eine Modellierung über multiple Vererbung. Dabei besitzt die *Workflow*-Klasse zwei orthogonale Spezialisierungsbeziehungen, einmal bezüglich *Workflow-Instanz* bzw. *-Schema* und zum zweiten bezüglich komplexer bzw. atomarer Strukturierung. Auf diese Weise erbt etwa die Klasse *'atomares Wf-Schema'* von *'atomarer Workflow'* und von *'Workflow-Schema'*, während die Klasse *'komplexe Wf-Instanz'* von *'Workflow-Instanz'* und von *'komplexer Workflow'* erbt; das entsprechende Klassendiagramm ist in Abbildung 3(b) dargestellt. Diese Art der Modellierung erweist sich bei näherer Betrachtung

aber ebenfalls als problematisch, da sich die Spezialisierungsbeziehungen nicht auf die *Workflow*-Klassen beschränken, sondern auch zwischen vielen zu den *Workflow*-Klassen in Beziehung stehenden Klassen auftreten, etwa Rollen, Agenten oder Parameter (die in Abb. 3 nicht dargestellt sind). Die Vielzahl der Mehrfachvererbungen und der Spezialisierungsbeziehungen, die bei dieser Art der Modellierung auftreten, machen dieses Objektmodell unübersichtlich und tragen nur wenig zum Verständnis der logischen Struktur des Problems bei. Daher haben wir uns für einen Ansatz entschieden, der ohne Mehrfachvererbung auskommt und auch weitgehend redundanzfrei ist.

Der WASA₂-Ansatz Aufgrund der bei den oben skizzierten Modellierungsalternativen aufgetretenen Problemen haben wir uns entschieden, *Workflow-Schemata* und *Workflow-Instanzen* in einer gemeinsamen Klasse als *Workflow*-Objekte zu repräsentieren. Die Unterschiede zwischen Schema und Instanzen werden durch Zustände repräsentiert, die *Workflow*-Objekte annehmen können. Durch diese Modellierungsentscheidung wird das *Workflow-Meta-Schema* stark vereinfacht, ohne daß eine redundante Beschreibung der Gemeinsamkeiten von *Workflow-Schemata* und *Workflow-Instanzen* notwendig wird. Dieses *Workflow-Meta-Schema* wird nun ausführlich dargestellt und anhand eines Beispiel-*Workflow-Schemas* und einer von diesem Schema erzeugten *Workflow-Instanz* beschrieben.

3.2 *WASA₂-Workflow-Meta-Schema*

Im *Workflow-Meta-Schema* repräsentieren wir *Workflow-Schemata* und *Workflow-Instanzen* gemeinsam als *Workflow*-Objekte in einer Klasse *Workflow* und unterscheiden lediglich zwischen den Zuständen *Schema* und *Instanz*. (Ein *Workflow*-Objekt im Zustand *Instanz* wird auch als *Workflow-Instanz* bezeichnet; analog wird ein

Eine wichtige Eigenschaft von Workflow-Schemata ist die Abbildung von Datenfluß; hierzu werden Datenkonnektoren verwendet. Wir unterscheiden horizontalen und vertikalen Datenfluß, wobei ein horizontaler Datenfluß zwei Sub-Workflows eines gemeinsamen komplexen Workflows miteinander verbindet, während ein vertikaler Datenfluß zwischen einem Super-Workflow und seinen Sub-Workflows bestehen kann. Dabei ist ein Datenfluß von dem Super-Workflow zu seinen Sub-Workflows sowie ein Datenfluß in entgegengesetzter Richtung möglich. Diese beiden Formen des Datenflusses werden in Abschnitt 3.4 anhand eines Beispiels genauer erläutert. Bei horizontalem Datenfluß zwischen Workflow-Schemata i und j eines Super-Workflows k muß beachtet werden, daß in beiden Workflow-Sub-Workflow-Beziehungen k als Super-Workflow erscheint (*constraint c1a*). Beim vertikalen Datenfluß ist zu beachten, daß entweder ein Eingabeparameter eines Super-Workflows mit einem Eingabeparameter eines Sub-Workflows oder ein Ausgabeparameter eines Sub-Workflows mit einem Ausgabeparameter des Super-Workflows verbunden ist (*constraint c1b*). Durch diese Modellierung ist es möglich, Workflow-Schemata in verschiedenen komplexen Workflow-Schemata wieder zu verwenden, wobei die jeweils modellierten Datenflüsse sich ausschließlich auf die jeweilige Verwendung des Workflow-Schemas im Rahmen eines komplexen Workflow-Schemas beziehen.

Neben Datenfluß spielt Kontrollfluß eine wichtige Rolle bei der Workflow-Modellierung. Ein Kontrollkonnektor stellt eine binäre Beziehung zwischen zwei Workflow-Sub-Workflow-Beziehungen dar und legt mögliche Ausführungsreihenfolgen der zugehörigen Sub-Workflows fest. Startbedingungen werden bei der Workflow-Modellierung benötigt, um explizit anzugeben, unter welchen Bedingungen ein Workflow ausgeführt werden kann. In unserem Modell kann einem Workflow im Kontext eines komplexen Workflows eine Startbedingung zugeordnet werden. Dies wird analog zum Kontrollkonnektor durch die Beziehung zwischen der Klasse `Start Condition` und der Klasse `WF-SubWF Relationship` zum Ausdruck gebracht.

Bei Parametern wird durch eine Generalisierungsbeziehung zwischen Input- und Output-Parametern unterschieden. Die Klasse `Input Parameter` besitzt im Gegensatz zur Klasse `Output Parameter` eine Aggregationsbeziehung zur Klasse `Start Condition`. Damit wird es bei der Implementierung der Klasse `Start Condition` möglich, die Eingabeparameter eines Workflows zu verwenden, um zur Laufzeit anhand übergebener Werte die Startbedingung auszuwerten.

Um die Beziehung zwischen Workflow-Schemata und Workflow-Instanzen zu verwalten, besitzt jede Workflow-Instanz eine *instance of*-Beziehung zu einem Workflow-Schema, nämlich zu dem Schema, von dem sie instantiiert wurde. Diese Beziehung ist notwendig, um Veränderungen eines Schemas, falls gewünscht, auf die davon abgeleiteten Workflow-Instanzen übertragen zu können.

Zur Laufzeit von Workflows werden im Rahmen der Rollenauflösung Agenten zur Ausführung konkreter Aktivitäten ausgewählt. Dabei sind jedem Workflow eine

oder mehrere Rollen zugeordnet. Es kann sich dabei um eine in bestimmter Weise qualifizierte Person handeln oder auch um ein Anwendungsprogramm. Zwischen Rollen und Agenten besteht eine $n:m$ -Beziehung, die festlegt, welche Rollen von welchen Agenten übernommen werden können.

Das Rollenkonzept wird durch die Klassen `Role` und `Agent` realisiert. Rollen sind Workflow-Schemata zugeordnet und abstrahieren von konkreten Agenten. Beispielsweise werden einem Workflow `Kunde_Benachrichtigen` nicht konkrete Mitarbeiter zugeordnet, sondern die Rolle `Sachbearbeiter`. Dies hat den Vorteil, daß Workflow-Schemata unabhängig von den momentan zur Verfügung stehenden Mitarbeitern formuliert werden können. Während bei atomaren Workflows eine zugeordnete Rolle zum Ausdruck bringt, welcher Agent den Workflow bearbeiten kann, drückt die Rolle eines komplexen Workflows die Verantwortlichkeit für diesen Workflow aus. Die Beziehung zwischen den Klassen `Agent` und `Workflow` existiert nur für Workflow-Instanzen und beschreibt, welcher Agent eine konkrete Instanz bearbeitet.

3.3 Abstraktionsebenen des Workflow-Meta-Schemas

In der Softwaretechnik unterscheidet man bei dem Design und der Entwicklung von Software-Systemen meist vier Ebenen, die sich in ihrem Abstraktionsgrad voneinander unterscheiden. Die oberste Ebene bildet das Meta-Meta-Schema, mit dem auf der nächsten Ebene die Elemente der Meta-Schema-Ebene beschrieben werden können. Auf der dritten Ebene befinden sich Schemata, gefolgt von der Anwendungsschicht auf der untersten Ebene. Im Datenbankentwurf etwa entspricht die Meta-Meta-Ebene der Beschreibung von speziellen Sprachen, mit denen Meta-Schemata beschrieben werden können. Eine solche Sprache sind etwa Entity-Relationship-Diagramme, die auf der Meta-Ebene einzuordnen sind. Auf der Schema-Ebene wird die logische Struktur der Daten in Form eines konkreten ER-Diagramms dargestellt; der konkrete, in einer Datenbank gespeicherte Datenbestand entspricht der Instanz- oder Anwendungsebene.

Diese Ebenen sind auch bei dem oben vorgestellten Workflow-Meta-Schema vorhanden: Auf der Meta-Meta-Ebene befindet sich die Sprache UML, mit der das Meta-Schema, d.h., das Workflow-Meta-Schema beschrieben wird. Instanzen des Workflow-Meta-Schemas sind Workflow-Schemata, die der Schema-Ebene in der Softwaretechnik entsprechen. Sie sind als Elemente der Workflow-Klasse Instanzen des Workflow-Schemas und reflektieren daher unmittelbar die übliche Ebenenarchitektur. Auf der nächsten Abstraktionsebene befinden sich Workflow-Instanzen, deren Struktur von Workflow-Schemata beschrieben wird. Allerdings werden Workflow-Instanzen – wie oben ausführlich begründet – in unserem Workflow-Meta-Schema ebenfalls als Objekte der Workflow-Klasse repräsentiert. Damit liegt aus logischer Sicht die übliche Vier-Ebenen-Architektur auch bei unserer Modellierung vor; aufgrund der speziellen Eigenschaften des Modellierungsgegenstands werden Workflow-Schemata und Workflow-Instanzen in unserem Ansatz aus den oben

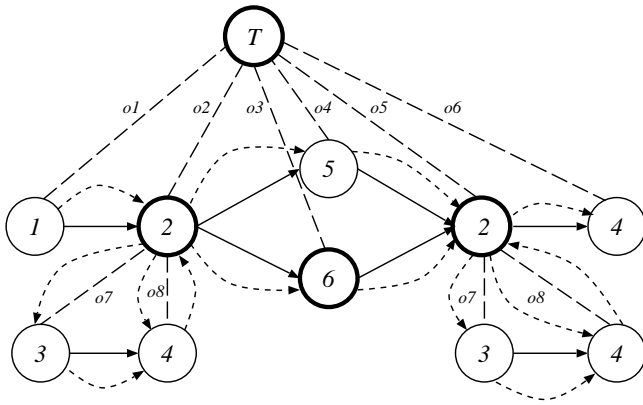


Abb. 5. Workflow-Schema, mit bezeichneten WF-SubWF-Relationship-Objekten.

genannten Gründen in eine Klasse des Workflow-Meta-Schemas integriert und dort durch unterschiedliche Zustände voneinander abgegrenzt.

3.4 Beispiel-Workflow

In diesem Abschnitt wollen wir anhand eines Beispiel-Workflows die wesentlichen Konzepte des im vorhergehenden Abschnitt entwickelten WASA₂-Workflow-Meta-Schemas verdeutlichen.

Abbildung 5 zeigt den obigen Beispiel-Workflow, wobei zusätzlich Bezeichner für WF-SubWF-Relationship-Objekte eingezeichnet sind. Wie oben erklärt, repräsentieren diese Objekte die Verwendung eines Workflows als Sub-Workflow eines komplexen Workflows. So wird etwa die Verwendung von 1 als Sub-Workflow von T durch das Objekt $o1$ repräsentiert. Die beiden Vorkommen von 2 als Sub-Workflow-Schema von T werden durch die Objekte $o2$ und $o5$ dargestellt; da es sich bei 2 jeweils um dasselbe Workflow-Schema handelt, werden die Sub-Workflow-Schemata von 2 jeweils durch die Objekte $o7$ bzw. $o8$ repräsentiert.

Um Datenfluß darzustellen, wird nun der komplexe Workflow 2 genauer untersucht. Abbildung 6 zeigt Kontrollfluß zwischen seinen Sub-Workflow-Schemata sowie detailliert die an horizontalem bzw. vertikalem Datenfluß beteiligten Workflow-Schemata bzw. die beteiligten Parameter und deren Beziehungen. Kontrollfluß wird, wie oben diskutiert, durch Paare von WF-SubWF-Relationship-Objekten repräsentiert. In Abbildung 6 wird der Kontrollfluß von 3 nach 4 durch $(o7, o8)$ definiert, wobei $o7$ die Beziehung zwischen dem komplexen Workflow-Schema 2 und 3 und $o8$ die Beziehung zwischen 2 und 4 repräsentiert.

Vertikaler Datenfluß verbindet je einen Eingabeparameter des komplexen Workflows 2 mit je einem Eingabeparameter der Sub-Workflows 3 und 4 sowie einen Ausgabe-Parameter des Workflows 4 mit einem Ausgabe-Parameter von Workflow 2. Horizontaler Datenfluß verbindet den Ausgabeparameter b von Workflow 3 mit dem Eingabeparameter c von Workflow 4. Jeder vertikale Datenfluß wird im Workflow-Meta-Schema durch eine Be-

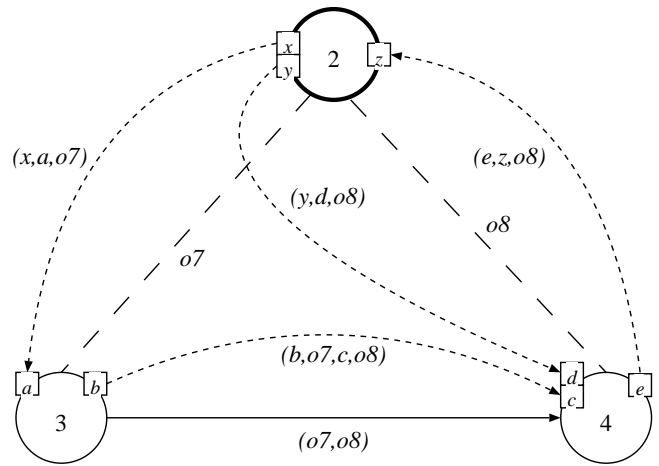


Abb. 6. Komplexes Workflow-Schema mit Parametern und horizontalem sowie vertikalem Datenfluß.

ziehung zwischen drei Objekten repräsentiert, nämlich durch die beiden beteiligten Parameter sowie durch ein WF-SubWF-Relationship-Objekt. Wie in Abbildung 6 dargestellt, wird der vertikale Datenfluß von dem Eingabeparameter x des komplexen Workflows 2 zu dem Eingabeparameter a von Workflow 3 durch das Objekt $(x, a, o7)$ repräsentiert, wobei $o7$ das entsprechende WF-SubWF-Relationship-Objekt darstellt.

Jeder horizontale Datenfluß ist durch eine Beziehung zwischen vier Objekten charakterisiert: Neben den beiden involvierten Parametern sind jeweils zwei Objekte der WF-SubWF-Relationship-Klasse notwendig. So ist der in Abbildung 6 dargestellte horizontale Datenfluß von Workflow 3 nach Workflow 4 durch eine Beziehung zwischen den Objekten $b, o7, c, o8$ repräsentiert, wobei b und c die Parameter von Workflow 3 bzw. Workflow 4 und $o7$ und $o8$ die entsprechenden WF-SubWF-Relationship-Objekte sind. Dieses Beispiel veranschaulicht die Modellierung von horizontalem Datenfluß durch eine vierstellige Beziehung und die Modellierung von vertikalem Datenfluß durch eine dreistellige Beziehung.

Mit dieser Modellierung ist es möglich, beliebige Datenflußbeziehungen zwischen Sub-Workflows eines komplexen Workflows (horizontaler Datenfluß) bzw. zwischen einem komplexen Workflow und seinen Sub-Workflows (vertikaler Datenfluß) zu spezifizieren. Um jedoch sicherzustellen, daß zu übergebende Daten bei Workflow-Ausführungen rechtzeitig erzeugt werden, erlauben wir einen Datenfluß von einem Sub-Workflow i zu einem Sub-Workflow j nur dann, wenn i vor j ausgeführt wird, d.h., wenn es einen Pfad von Kontrollkonnektoren von i nach j gibt. Datenflußbeziehungen gelten stets nur im Kontext eines komplexen Workflows. Mögliche Auftreten der Sub-Workflows in anderen komplexen Workflows besitzen im allgemeinen unterschiedliche Datenflußbeziehungen. Damit ist auch bezüglich der Modellierung von Datenfluß ein hohes Maß an Modularität gegeben, so daß die Wiederverwendbarkeit von Workflow-Schemata in unterschiedlichen komplexen Workflows

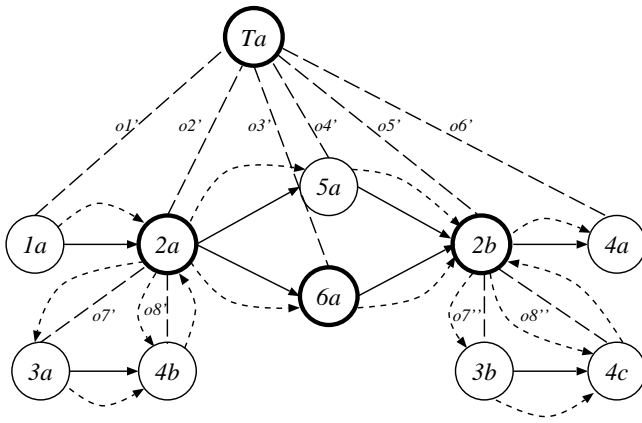


Abb. 7. Workflow-Instanz-Objekt, mit bezeichneten WF-SubWF-Relationship-Objekten.

auch dann möglich ist, wenn sie verschiedene Datenflußbeziehungen besitzen.

4 Modellierung dynamischer Aspekte

Um die dynamischen Aspekte des Systems zu beschreiben, werden Workflow-Ausführungen zunächst informal betrachtet; anschließend wird das dynamische Verhalten von Workflow-Instanzen unter Verwendung von Zustandsübergangsdiagrammen spezifiziert.

4.1 Ausführung von Workflow-Instanzen

Zur Beschreibung der Ausführung von Workflow-Instanzen wird eine Workflow-Instanz Ta betrachtet, die auf dem im vorhergehenden Abschnitt diskutierten Toplevel-Workflow-Schema T basiert; diese Workflow-Instanz ist in Abbildung 7 dargestellt.

Die Ausführung der Workflow-Instanz beginnt mit einer Instantiierung des Toplevel-Workflows Ta . Dabei werden die entsprechenden Instanzen der zweiten Ebene erzeugt; die Erzeugung erfolgt nicht durch eine zentrale Workflow-Engine, sondern sie wird durch das Workflow-Instanz-Objekt Ta vorgenommen. Die initiierten Sub-Workflow-Instanzen sind in Abbildung 7 mit $1a, 2a, 5a, 6a, 2b, 4a$ bezeichnet. Für jedes Auftreten eines Workflow-Schemas in T wird daher ein Workflow-Instanz-Objekt erzeugt. Das mehrfache Auftreten des Workflow-Schemas 2 im komplexen Workflow-Schema wurde (auf Schema-Ebene) durch mehrere WF-SubWF-Relationship-Objekte dargestellt; das Workflow-Schema 2 war jedoch lediglich einmal vorhanden. Nun wird für jedes Auftreten des Workflow-Schemas 2 im Toplevel-Workflow-Schema T ein Workflow-Instanz-Objekt erzeugt, d.h., es werden die Workflow-Instanzen $2a$ und $2b$ erzeugt.

Die Ausführung von Ta erfolgt verteilt, d.h., nicht nur die Erzeugung der Workflows, sondern auch die Kontrolle der Ausführung erfolgt durch Workflow-Instanzen; diese sorgen selbst für eine Ausführung, die mit den im Workflow-Schema spezifizierten Bedingungen

übereinstimmt; dies erfolgt – wie bei objektorientierten Systemen üblich – durch Senden von Nachrichten zwischen Objekten. Eine Voraussetzung hierfür ist, daß jedes Objekt alle Objekte kennt, mit denen es Kontroll- oder Datenflußbeziehungen besitzt; diese Voraussetzung ist bei dem in Abschnitt 3 vorgestellten Workflow-Meta-Schema erfüllt.

Im Beispiel stellt Ta zunächst fest, daß $1a$ der einzige Sub-Workflow ohne eingehende Kontrollkonnektoren ist; daher sendet Ta eine Nachricht an $1a$. Falls die Startbedingung von $1a$ nach wahr ausgewertet wird, kann dieser Workflow gestartet werden. Nach seiner Termination sendet $1a$ eine Nachricht an den nachfolgenden Workflow $2a$ (die Identität dieser Workflow-Instanz ist durch das Kontrollkonnektor-Objekt ($o1', o2'$) zugreifbar.). Trifft diese Nachricht ein, so wird die Startbedingung von $2a$ ausgewertet und $2a$ gestartet. Da es sich bei $2a$ um einen komplexen Workflow handelt, werden nun die Sub-Workflow-Instanzen $3a$ und $4b$ initiiert. Dies erfolgt – wie bereits die Erzeugung der anderen Workflow-Instanzen – nicht durch eine zentrale Workflow-Engine, sondern durch eine komplexe Workflow-Instanz, hier durch $2a$. Nun werden $3a$ und $4b$ ausgeführt, und $4b$ sendet eine Nachricht an $2a$, um diese Workflow-Instanz über die Termination ihrer Sub-Workflows zu informieren.

Aus dieser Beschreibung folgt, daß Workflow-Instanzen ebenenweise erzeugt werden, und zwar nur genau dann, wenn sie auch benötigt werden; dies soll nun genauer erläutert werden. Nachdem $2a$ die entsprechende Nachricht von $4b$ erhalten hat, sendet diese Workflow-Instanz Nachrichten an ihre Nachfolger $5a$ und $6a$. Diese werten ihre jeweiligen Startbedingungen aus. Nehmen wir an, die Startbedingung von $5a$ wird nach wahr ausgewertet und die Startbedingung von $6a$ wird nach falsch ausgewertet, so wird $5a$ ausgeführt, während $6a$ nicht ausgeführt wird. Da es sich bei $6a$ um einen komplexen Workflow handelt, werden seine Sub-Workflows nicht erzeugt. Damit werden jeweils nur diejenigen Workflows erzeugt, die bei einer konkreten Ausführung benötigt werden. Um $2b$ davon zu informieren, daß $6a$ nicht ausgeführt wird, setzen wir *dead-path-elimination* [14] ein. Nach der Termination von $5a$ kann $2b$ seine Startbedingung auswerten und die Ausführung des Workflows fortsetzen. Dabei werden die Sub-Workflows von $2b$ erzeugt und anschließend autonom ausgeführt, bevor die Ausführung des Toplevel-Workflows mit der Sub-Workflow-Instanz $4a$ fortgesetzt wird. Nach der Termination von $4a$ sendet diese Workflow-Instanz eine Nachricht an Ta , so daß bei Empfang dieser Nachricht der Toplevel-Workflow beendet werden kann.

4.2 Zustände und Zustandsübergänge

Die Beschreibung des dynamischen Verhaltens von Workflow-Instanzen erfolgt durch explizite Modellierung von Zuständen und Zustandsübergängen. Im allgemeinen besitzen Objekte einen Zustand und kommunizieren über Nachrichten miteinander, und ein Objekt reagiert auf Ereignisse in Abhängigkeit von seinem Zustand. In diesem Abschnitt beschreiben wir die Zustände und

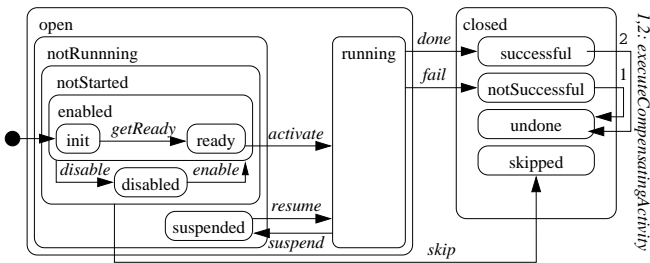


Abb. 8. Generisches Zustandsübergangsdiagramm

Zustandsübergänge, die ein Workflow-Instanz-Objekt innerhalb seines Lebenszyklus besitzen bzw. erfahren kann.

Der Zustand einer Workflow-Instanz kann im Rahmen der kontrollierten Ausführung von dem Workflow-Management-System sowie durch aufgerufene Applikationsprogramme oder durch Benutzerinteraktionen geändert werden. Workflow-Instanzen wurden bereits in Abschnitt 3.2 beschrieben; sie existieren in verschiedenen Arten, die sich auch in ihrem Verhalten, d.h. in den Zuständen und Zustandsübergängen, unterscheiden. Zur Darstellung werden Zustandsübergangsdiagramme verwendet, in denen Zustände durch beschriftete Rechtecke und Zustandsübergänge durch markierte Pfeile dargestellt werden (siehe Abb. 8). Zustände können geschachtelt sein; sie können dann in der Punkt-Schreibweise eindeutig spezifiziert werden. So kann z.B. der Unterzustand *running* im Zustand *open* durch *open.running* eindeutig spezifiziert werden. Wenn die Bezeichnung der Unterzustände eindeutig ist, verzichten wir auf die Punkt-Schreibweise.

Der allgemeine Ablauf von Workflow-Instanzen wird in Abbildung 8 dargestellt; dieses generische Zustandsübergangsdiagramm beschreibt insbesondere das Verhalten von komplexen und atomaren manuellen Workflow-Instanzen. Dieses Diagramm wird als Grundlage für die weitere Diskussion verwendet. Wird eine Workflow-Instanz kreiert, so nimmt sie den Zustand *open* an, den sie erst bei ihrer Beendigung verläßt. Einer der Unterzustände ist *notRunning*, in dem sich eine Workflow-Instanz befindet, wenn sie nicht an der Ausführung eines Workflows beteiligt ist. Das bedeutet, daß sie entweder noch nicht gestartet worden ist (Zustand *notStarted*) oder derzeit unterbrochen ist (*suspended*).

Der Zustand *init* stellt den Startzustand von Workflow-Instanzen dar. Dies wird in einem Zustandsübergangsdiagramm durch einen Pfeil mit einem Kreis am Ausgangspunkt dargestellt. In dem *init*-Zustand werden die Vorbereitungen für die Ausführung der Workflow-Instanz getroffen. Sind die Vorbereitungen abgeschlossen, so kann durch den *getReady*-Übergang zum Zustand *ready* gewechselt werden. Durch die *disable*- bzw. *enable*-Übergänge können Workflow-Instanzen temporär deaktiviert werden. Von *ready* aus kann die Workflow-Instanz aktiviert werden, wobei der *activate*-Übergang zum Zustand *running* vollzogen wird. Der Zustand *suspended* kann vom *running*-Zustand aus betreten werden. Er drückt aus, daß die betreffende Workflow-Instanz unterbrochen worden ist. Der Zustand kann durch den *re-*

sume-Übergang zum Zustand *open.running* wieder verlassen werden.

Das temporäre Unterbrechen von Workflow-Instanzen ist insbesondere für komplexe Workflow-Instanzen von großer Bedeutung. Wechselt ein komplexer Workflow von *running* in den Zustand *suspended*, so kann dies für die Sub-Workflows je nach Implementierung unterschiedliche Auswirkungen haben:

- Alle Sub-Workflows gehen in den Zustand *suspended* über, werden also ebenfalls unterbrochen. Dabei ist zu beachten, daß der Workflow komplett angehalten wird, was u.U. mit erheblichem Aufwand verbunden ist, da das Stoppen durch alle Ebenen der Workflow-Hierarchie propagiert werden muß.
- Die Sub-Workflows laufen zunächst unverändert weiter, allerdings werden nach deren Beendigung keine weiteren Sub-Workflows gestartet. Dies ist einfach zu realisieren, da Workflows ihre Sub-Workflows selbst erzeugen und der Kontroll- und Datenfluß von den Sub-Workflows gesteuert wird.
- Die aktiven Sub-Workflows laufen weiter. Sind alle Sub-Workflows beendet, bleibt der Super-Workflow so lange unvollendet, wie er sich im *suspended*-Zustand befindet. In diesem Fall wird lediglich die Beendigung des komplexen Workflows ausgesetzt, die Sub-Workflows können ihre Tätigkeit ungehindert fortsetzen und beenden.

Im Zustand *open.running* wird die eigentliche Tätigkeit der Workflow-Instanz verrichtet. Von diesem Zustand aus kann bei der Beendigung der Workflow-Instanz zu *closed* gewechselt werden. Letzteres kann durch die Zustandsübergänge *done* bzw. *fail* erfolgen, was einer erfolgreichen bzw. nicht erfolgreichen Ausführung der Workflow-Instanz entspricht.

Beendete Workflow-Instanzen befinden sich im *closed*-Zustand. Erreicht eine Workflow-Instanz diesen Zustand, so verläßt sie ihn nicht wieder. Dieser Zustand beinhaltet mehrere Unterzustände, die wir jetzt näher beschreiben wollen. Bei einer erfolgreichen Beendigung der Aktivität wird der Zustand *successful* betreten. Scheitert die Ausführung der Aktivität, so wird der Zustand *notSuccessful* betreten. Diese beiden Zustände können durch den *executeCompensatingActivity*-Übergang zum Zustand *undone* wieder verlassen werden. Wir weisen darauf hin, daß die Möglichkeit der Kompensation von Workflows nicht generell gegeben ist, sondern von der Verfügbarkeit entsprechender kompensierender Aktivitäten abhängig ist, die den bereits ausgeführten Workflow semantisch ungeschehen machen [13]. Workflows können auch übersprungen werden, was durch den neu eingeführten Zustand *skipped* ausgedrückt wird. Eine solche Situation fällt in den Bereich der dynamischen Modifikation [30]. Eine Workflow-Instanz kann nur dann übersprungen werden, wenn sie noch nicht gestartet wurde, d.h., solange sie sich in einem *notStarted*-Zustand befindet.

Bei einer automatischen Workflow-Instanz handelt es sich um eine atomare Aktivität, die im allgemeinen von einem Anwendungsprogramm ausgeführt wird. Wir

verzichten auf eine Abbildung des entsprechenden Zustandsübergangsdiagramms und beschreiben stattdessen die Unterschiede zu der oben diskutierten, generischen Variante.

Der Zustand *init* stellt den Startzustand einer automatischen Workflow-Instanz dar. In diesem Zustand werden die Vorbereitungen für die Ausführung der Workflow-Instanz getroffen. Dies beschränkt sich auf das Einlesen der Input-Daten. Der Zustand *suspended* kann von allen Unterzuständen von *running* aus betreten werden und drückt aus, daß die betreffende Workflow-Instanz unterbrochen worden ist; bei einer automatischen Workflow-Instanz wird dann das laufende Programm unterbrochen. Der Zustand kann wieder (nach *running*) verlassen werden, wobei das Programm fortgesetzt wird. Im Zustand *running* wird die eigentliche Aktivität der Workflow-Instanz ausgeführt.

Im allgemeinen wird unterschieden zwischen transaktionalen und nicht-transaktionalen automatischen Workflow-Instanzen. Falls das ausführende Software-System transaktionale Zusicherungen über die Ausführung liefern kann, so handelt es sich um transaktionale automatische Workflows, sonst um nicht-transaktionale [13]. Beispiele für transaktionale Applikationen sind etwa Datenbank-Programme; bei einem Zugriff auf einen Web-Server handelt es sich demgegenüber um eine Applikation, die keine Zusicherungen über die Korrektheit ihrer Ausführung machen kann; der entsprechende automatische Workflow ist daher nicht-transaktional. Diese Eigenschaften von Applikationsprogrammen werden durch die Zustände *transactional* bzw. *nonTransactional* als Unterzustände von *running* repräsentiert.

Der Zustand *transactional* kann auf zwei Arten verlassen werden: entweder mit *commit* zum Zustand *committed* oder mit *abort* zum Zustand *aborted*, was einer erfolgreichen bzw. nicht erfolgreichen Ausführung eines transaktionalen Workflows entspricht. Da die Konsistenz des Datenbestands nach einem Abort durch die transaktionale Applikation sichergestellt wird, ist in diesem Fall keine kompensierende Aktivität notwendig. Vom Zustand *nonTransactional* steht ein Übergang zum Zustand *done* für die erfolgreiche Ausführung der Aktivität und ein Übergang zum Zustand *failed* bei einer gescheiterten Ausführung zur Verfügung. Die Zustände *committed* und *done* werden zu *successful*, die Zustände *aborted* und *failed* werden zu *notSuccessful* zusammengefaßt.

Die Zustände und Zustandsübergänge der obigen Workflow-Instanz werden in Abbildung 9 anhand eines Zeitdiagramms dargestellt. In diesem Zeitdiagramm erscheint für jedes Objekt eine Linie, durch die sein Zustand repräsentiert wird; es werden lediglich die Zustände *init*, *running* und *closed* unterschieden. In Abbildung 9 schreitet die Zeit *t* von links nach rechts fort; wie im objektorientierten Kontext üblich, kommunizieren Objekte durch Nachrichten, die als Pfeile dargestellt sind. Jeder komplexe Workflow erzeugt bei seinem Start seine unmittelbaren Sub-Workflow-Objekte, die dann den Zustand *init* besitzen. Nach dem Empfang einer *getReady*-Nachricht gehen sie in den *running*-Zustand über, wobei der Zustandsübergang von *ready* nach *running* im Zeitdiagramm nicht explizit dargestellt wird. Nach ihrer

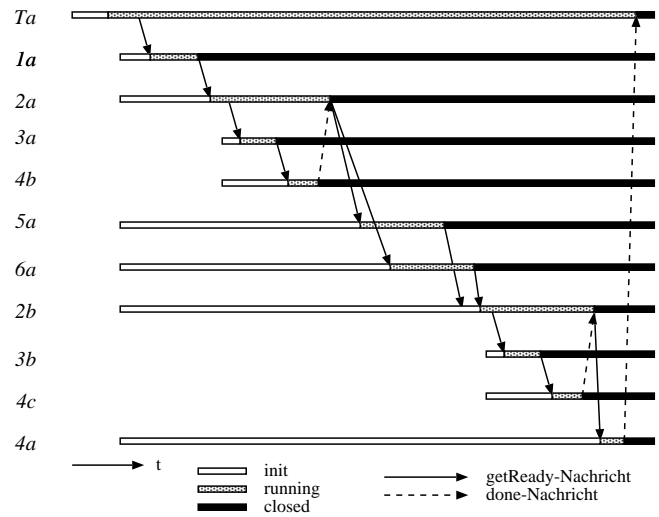


Abb. 9. Zeitdiagramm der Ausführung des komplexen Workflows aus Abbildung 7.

Termination erreichen Workflow-Instanzen den *closed*-Zustand; dann senden sie entweder eine *getReady*-Nachricht an ihren jeweiligen Nachfolge-Workflow oder eine *done*-Nachricht an ihren Super-Workflow. Um die Abbildung übersichtlich zu gestalten, wurden Zustände, die nicht notwendigerweise betreten werden (z.B. *disable*, *suspend*), nicht betrachtet.

5 Flexibilitätseigenschaft: Unterstützung dynamischer Modifikationen

Bei der Modellierung von Workflows haben wir bereits auf die modulare Struktur von Workflow-Schemata und ihrer Komponenten hingewiesen, die insbesondere die Wiederverwendbarkeit von Workflow-Schemata in unterschiedlichen Kontexten und die leichte Konfigurierbarkeit von Workflow-Schemata erlaubt. In diesem Abschnitt soll diskutiert werden, welche weiteren Eigenschaften zur Flexibilisierung von Workflow-Management-Systemen durch den hier vorgestellten Ansatz ermöglicht werden.

Der Flexibilitäts-Begriff befindet sich in der Workflow-Forschung derzeit in der Diskussion; es scheint jedoch anerkannt zu sein, daß dynamische Modifikationen, d.h., die Möglichkeit zur Veränderung aktiver Workflow-Instanzen zur Laufzeit, um auf veränderte Umweltbedingungen flexible reagieren zu können, eine wichtige Form von Flexibilität darstellt [20, 10, 30]. Beispiele für solche Veränderungen sind etwa veränderte Marktsituationen oder neue gesetzliche Regelungen, die Auswirkungen auf die Ablauforganisation besitzen. Die Möglichkeit, Veränderungen zur Laufzeit zu erlauben, ist besonders dann wichtig, wenn aktive Workflow-Instanzen lange laufen (etwa über Monate oder Jahre hinweg) oder wenn Veränderungen schnell durchgeführt werden sollen, etwa um durch eine rasche Anpassung aktiver Workflow-Instanzen Wettbewerbsvorteile realisieren zu können. Nun

wird diskutiert, wie dynamische Modifikationen von unserem System unterstützt werden und welche Rolle dabei das vorgestellte Workflow-Meta-Schema spielt.

5.1 Realisierung dynamischer Modifikationen

In diesem Abschnitt wird untersucht, wie die Auswirkungen von dynamischen Modifikationen auf aktive oder zukünftige Workflow-Instanzen beschrieben werden können.

Dazu betrachten wir eine Situation, in der ein Workflow-Schema T und aktive Workflow-Instanzen Ta , Tb und Tc dieses Workflow-Schemas vorhanden sind, d.h., es gibt jeweils eine instance-of-Beziehung zwischen T und Ta , Tb bzw. Tc . Um eine dynamische Modifikation zu beschreiben, nehmen wir an, daß das Workflow-Schema T während der Laufzeit dieser Instanzen modifiziert wird, etwa um den im Workflow-Schema definierten Ablauf effizienter oder kundenorientierter zu gestalten. Hierzu wird zunächst eine neue Version von T erzeugt, indem das ursprüngliche Modell kopiert und entsprechend den Wünschen modifiziert wird (ein Beispiel für konkrete dynamische Modifikationen wird im weiteren Verlauf dieses Abschnitts gegeben). Die erzeugte neue Version des Workflow-Schemas wird mit T' bezeichnet.

Nachdem die Änderungen in der neuen Schemaversion vorgenommen wurden, muß entschieden werden, welche Instanzen an das neue Schema angepaßt werden sollen, bzw., wann eine solche Anpassung überhaupt möglich ist. Letzteres wird anhand von Konsistenzregeln überprüft. Hierbei ist jeweils entscheidend, wie weit die Abarbeitung der anzupassenden Workflow-Instanz zum Zeitpunkt der Anpassung bereits fortgeschritten ist. Im allgemeinen kann eine Workflow-Instanz an eine Modifikation angepaßt werden, wenn die modifizierten Teile des Schemas ihre Ausführung noch nicht begonnen haben. Nachdem die Instanzen identifiziert wurden, für die die Konsistenzregeln erfüllt sind, werden diese durch Verändern der entsprechenden instance-of-Beziehungen in ihrer Struktur der neuen Schemaversion angepaßt, und ihre Ausführung wird fortgesetzt. Nehmen wir an, die Bearbeitung von Tc ist bereits fortgeschritten und lediglich Ta und Tb können an das veränderte Workflow-Schema angepaßt werden, so werden die dynamischen Modifikationen von Ta und Tb dadurch realisiert, daß sie jetzt eine instance-of-Beziehung mit T' unterhalten.

Dieser Ansatz erlaubt auch, den Gültigkeitsbereich einer dynamischen Modifikation frei festzulegen, d.h., eine beliebige Auswahl der Instanzen zuzulassen, die angepaßt werden. Damit wird unter Verwendung der modularen Struktur von Workflow-Schemata und Workflow-Instanzen und durch die Verwaltung von Beziehungen zwischen Workflow-Schema und den kontrollierten Workflow-Instanzen (instance-of-Beziehung) ermöglicht, dynamische Modifikationen durchzuführen und ihre Auswirkung auf aktive bzw. zukünftige Workflow-Instanzen zu kontrollieren. Ist etwa eine Anpassung der Workflow-Instanzen Ta und Tb an das modifizierte Workflow-Schema T' möglich, entscheidet der Benutzer jedoch, lediglich Ta anhand von T' auszuführen, so wird lediglich Ta

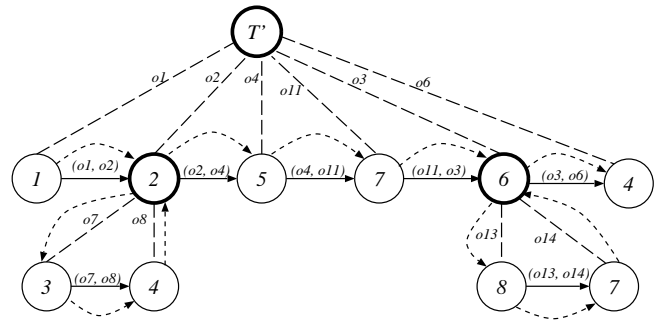


Abb. 10. Workflow-Schema, dynamisch verändert

als instance-of T' eingetragen; Tb und Tc sind weiterhin instance-of T .

5.2 Beispiel für dynamische Modifikationen

Konkrete dynamische Modifikationen werden nun anhand eines Beispiels gezeigt. Dabei wird davon ausgegangen, daß die Notwendigkeit zu einer dynamischen Veränderung bei der Ausführung einer konkreten Workflow-Instanz auftritt. Dieses Beispiel basiert auf dem obigen Workflow-Schema T und einer Workflow-Instanz Ta .

Angenommen, Ta beginnt (wie in Abschnitt 3.4 beschrieben) mit den Sub-Workflows $1a$ und $2a$. Nun wird festgestellt, daß anschließend nicht $5a$ und $6a$ nebenläufig auszuführen sind (wie im Workflow-Schema T vorgesehen), sondern zunächst $5a$ und anschließend ein Workflow $7a$ auszuführen ist. Sodann sollen $6a$ und $4a$ ausgeführt werden. Diese dynamische Veränderung ist auf Ebene des Workflow-Schemas in Abbildung 10 dargestellt. Dabei wird eine neue Version des Workflow-Schemas T erzeugt, die mit T' bezeichnet ist. Diese unterscheidet sich von T wie folgt: Workflow-Schema 7 wird durch WF-SubWF-Relationship-Objekt $o11$ als Sub-Workflow von T' eingetragen, und die neue Position von Workflow-Schema 6 wird durch die entsprechenden Kontrollkonnektor-Objekte ($o11, o3$) und ($o3, o6$) repräsentiert. Durch die Kontrollkonnektor-Objekte ($o4, o11$), ($o11, o3$) und ($o3, o6$) wird 7 als Nachfolger von 5 und als Vorgänger von 6 sowie 6 als Vorgänger von 4 definiert; die nun nicht mehr notwendigen WF-SubWF-Relationship-Objekte werden mit ihren adjazenten Kontroll- und Datenkonnektor-Objekten gelöscht, um zu spezifizieren, daß sie nicht mehr Sub-Workflow des Toplevel-Workflows sind. Man beachte, daß es jetzt Workflow-Schemata T und T' gibt, so daß aktive Workflow-Instanzen auch weiterhin nach T ausgeführt werden können.

Nehmen wir an, die dynamischen Veränderungen haben vor Beendigung von Workflow-Instanz $2a$ stattgefunden, so wird die Workflow-Instanz nun an das veränderte Workflow-Schema T' angepaßt, wie im vorhergehenden Abschnitt beschrieben. Nach der Veränderung wird zunächst die zusätzliche Workflow-Instanz $7a$ erzeugt und durch das WF-SubWF-Relationship-Objekt $o11'$ als Sub-Workflow von Ta' definiert; dabei nehmen wir an, daß $6a$ bereits erzeugt wurde. Nun werden die Kontrollflußbedingungen durch die Objekte ($o4', o11'$), ($o11', o3'$)

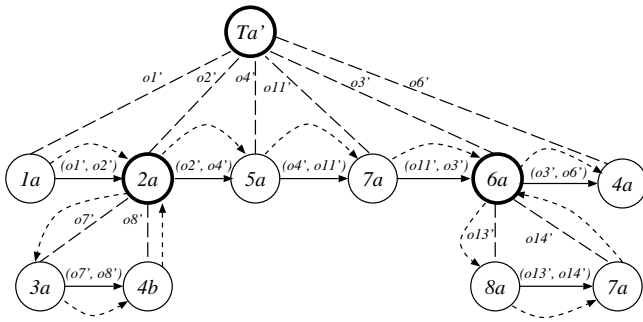


Abb. 11. Komplexer Toplevel-Workflow, dynamisch verändert

und $(o3', o6')$ in den Kontext des Toplevel-Workflows eingebunden und ggf. bereits erzeugte und nun nicht mehr benötigte Kontroll- und Datenflußkonnector-Objekte gelöscht. Der weitere Verlauf der Workflow-Ausführung richtet sich nun nach dem Workflow-Schema T' , so daß die Ausführung von $5a$, $7a$, $6a$ (inklusive seiner Sub-Workflows) und $4a$ die dynamisch modifizierte komplexe Workflow-Instanz schließlich beendet.

5.3 Unterstützung durch das Workflow-Meta-Schema

Nachdem beispielhaft gezeigt wurde, wie dynamische Veränderungen von Workflow-Instanzen zur Laufzeit vorgenommen werden können und welche Workflow-relevanten Objekte an der Durchführung einer solchen Veränderung beteiligt sind, wird nun untersucht, auf welche Weise dynamische Veränderungen durch das WASA₂-Workflow-Meta-Schema unterstützt werden, d.h., welche Eigenschaften des Workflow-Meta-Schemas dynamische Modifikationen ermöglichen.

- Bei der Verwendung von generischen Workflow-Klassen ist zur dynamischen Modifikation keine Veränderung von Klassenstrukturen notwendig, wie das bei der in Abschnitt 3.1.1 diskutierten Modellierungsalternative durch konkrete Workflow-Klassen der Fall wäre. Daher bildet der generische Ansatz eine wichtige Voraussetzung zur Realisierung dieser Form von Flexibilität.
- Die Struktur eines komplexen Workflows wird durch spezielle Beziehungs-Objekte repräsentiert. Durch Verändern dieser Objekte ist es (wie im obigen Beispiel gezeigt) möglich, zur Laufzeit die Struktur von Workflow-Schemata und Workflow-Instanzen zu verändern, um dynamische Modifikationen zu realisieren.
- Workflow-Schemata und Workflow-Instanzen werden in einer gemeinsamen Workflow-Klasse repräsentiert; eine instance-of-Beziehung gibt für jede Workflow-Instanz an, unter Verwendung welches Workflow-Schemas sie kontrolliert wird. Dynamische Modifikationen sind möglich, da sich diese Zuordnung zur Laufzeit verändern kann.
- Die Kapselung von Struktur und Verhalten erlaubt es, Workflow-Instanzen als komplexe Objekte zu betrachten, deren interne Struktur nach außen hin nicht

sichtbar ist. Damit kann bei einer dynamischen Modifikation innerhalb eines Sub-Workflows sichergestellt werden, daß keine Auswirkungen auf den entsprechenden Super-Workflow bestehen und seine Ausführung unter Verwendung des modifizierten Sub-Workflows fortgesetzt werden kann.

- Da die dynamische Modifikation von Workflow-Instanzen nicht direkt, sondern durch Anpassung an modifizierte Schemaversionen erfolgt, kann die Reichweite der Modifikation auf eine beliebige Teilmenge der Instanzen des ursprünglichen Schemas ausgedehnt werden. Hierzu werden die im Workflow-Meta-Schema modellierten version-of- und instance-of-Beziehungen benötigt. Mit diesem Mechanismus ist es möglich zu kontrollieren, welche aktiven oder zukünftigen Workflow-Instanzen von einer dynamischen Modifikation betroffen sein sollen.
- In dem Workflow-Meta-Schema ist es vorgesehen, daß Workflow-Instanzen als Objekte modelliert werden, die ihre Ausführung selbstständig kontrollieren. Dies bedeutet insbesondere, daß die Sub-Workflows eines komplexen Workflows durch Nachrichten miteinander kommunizieren, um den im Workflow-Schema definierten Kontroll- und Datenflußbedingungen zu genügen. Aufgrund dieser Funktionalität von Workflow-Instanzen ist es einfach möglich, die Struktur von komplexen Workflows durch Verändern von Objektreferenzen zu modifizieren, was im obigen Beispiel veranschaulicht wurde.

Eine weitere Möglichkeit zur dynamischen Modifikation wird nun unter dem Stichwort "Selbstmodifikation" kurz diskutiert. Da im Rahmen des Datenflusses beliebige Objekte zwischen Workflows übergeben werden können, ist auch die Übergabe eines Workflow-Schemas als Input-Parameter eines Workflows möglich. Damit kann durch Verwendung eines Workflow-Modellierungswerkzeugs, das im Rahmen eines Workflows aufgerufen wird, zur Laufzeit des Workflows ein neues Workflow-Schema erstellt und im nachfolgenden Teil des Workflows verwendet werden. Eine derartige "Selbstmodifikation" ist insbesondere dann sinnvoll, wenn sich die zur vollständigen Modellierung des Schemas notwendigen Informationen erst während der Ausführung des Workflows ergeben. Ist der Zeitpunkt bekannt, an dem die benötigten Informationen vorliegen werden, kann an dieser Stelle ein Workflow zur Selbstmodifikation des Workflows eingesetzt werden. Hierdurch wird das Workflow-Management-System zum gegebenen Zeitpunkt die vollständige Modellierung des Schemas durch einen Benutzer veranlassen.

6 Fehlersicherheit

Ein wichtiges Kriterium bei dem Design und der Entwicklung von Workflow-Management-Systemen ist die Behandlung von Fehlerfällen [6]. Wir unterscheiden generell drei Arten von Fehlern: (i) Situationen in der Anwendungsumgebung, die im Workflow-Schema nicht spezifiziert sind, (ii) Anwendungsfehler (Fehler in Applikationsprogrammen) und (iii) Systemfehler (Verlust

des Hauptspeicherinhalts). Eine Möglichkeit, mit den unter (i) zusammengefaßten Ausnahmesituationen umzugehen, wurde mit dynamischen Modifikationen im vorhergehenden Abschnitt bereits ausführlich diskutiert. Im weiteren Verlauf dieses Abschnittes soll kurz beschrieben werden, durch welche Maßnahmen Fehlersicherheit für die Klassen (ii) und (iii) in unserem Ansatz unterstützt werden kann. Darunter verstehen wir die Eigenschaft eines Workflow-Management-Systems, Fehler in Anwendungsprogrammen zu tolerieren und durch geeignete Maßnahmen für die Korrektheit der Workflow-Ausführung zu sorgen (ii) bzw. mit einem Systemausfall, bei dem Hauptspeicherinhalte verlorengehen, fehlertolerant umzugehen (iii).

6.1 Anwendungsfehler

Das Systemverhalten bei Fehlern in Anwendungsprogrammen wird von Eigenschaften atomarer Workflows beeinflusst. Handelt es sich um einen transaktionalen atomaren Workflow (siehe Abschnitt 4.2), so sorgt das verwendete Applikationssystem für ein ordnungsgemäßes Abbrechen des Programms und ein Zurücksetzen eventuell veränderter Werte. Falls es sich um nicht-transaktionale Workflows handelt, so können entweder vordefinierte Rücksetzungsaktivitäten ausgeführt werden, oder es ist auf das Wissen von Benutzern zurückzugreifen, um die Effekte von teilweise bearbeiteten, nicht-transaktionalen Workflows zu eliminieren.

Mit dem Workflow-Meta-Schema ist auch die Modellierung von mächtigen Konstrukten zur Unterstützung von Fehlertoleranz möglich, etwa von Kontrollsphären [13]. Dabei bezeichnet eine Kontrollsphäre eine Menge von Sub-Workflows, die idealerweise alle erfolgreich beendet werden. Falls dies nicht möglich ist, so müssen alle Sub-Workflows der Kontrollsphäre, die bereits erfolgreich beendet wurden, durch vordefinierte Kompensations- bzw. Undo-Sub-Workflows semantisch rückgängig gemacht werden. Dies wird in unserem Ansatz dadurch unterstützt, daß für atomare Workflows kompensierende Workflows definiert werden können; dies wird durch die Beziehung *undoes* zwischen atomaren Workflow-Schemata abgebildet. Durch die in Abschnitt 4.2 vorgestellte explizite Verwaltung von Zuständen von Workflow-Instanzen ist es möglich, Informationen über den Erfolg einer Workflow-Instanz für die Realisierung von Kontrollsphären zu verwenden. Insbesondere kann bei einer nicht erfolgreichen Termination eines Workflows einer Kontrollsphäre anhand der Zustände der anderen Workflow-Instanzen dieser Sphäre über die nachfolgend durchzuführenden Aktionen entschieden und so das Konzept der Kontrollsphäre unterstützt werden.

6.2 Systemfehler

Um auf die Behandlung von Systemfehlern eingehen zu können, stellen wir nun kurz den in WASA₂ verwendeten Implementierungsansatz vor. Die objektorientierte Modellierung des Workflow-Meta-Schemas ermöglicht es,

für jede Klasse des Workflow-Meta-Schemas ein CORBA-Interface unter Verwendung der CORBA-IDL (Interface Definition Language) [17] zu definieren. Unsere Implementierung verwendet CORBA Common Object Services, um beispielsweise Objekte dynamisch zu erzeugen, sie zu verschieben, Beziehungen zwischen Objekten abzubilden und Objekte persistent zu speichern.

Wir untersuchen nun, wie unsere Implementierung mit dem Ausfall von einzelnen Rechnern der CORBA-Umgebung umgeht, bzw. welche Maßnahmen getroffen werden, um diese Situation fehlertolerant zu bewältigen. Verfahren, um mit dem Verlust von Hauptspeicherinhalten umzugehen, werden im Datenbank-Kontext unter dem Begriff der Persistenz diskutiert [9]. Während sich dort die Datenbank nach einem Wiederanlaufen des Systems in einem konsistenten Zustand befinden muß, ist eine entsprechende Forderung im Workflow-Kontext, daß die Zustände der beim Systemausfall aktiven Workflow-Instanzen wieder hergestellt werden können und die Workflow-Instanzen ihre Ausführung wie im Workflow-Schema beschrieben fortsetzen können. Um diese Funktionalität zur Verfügung zu stellen, sind Workflow-Ausführungsdaten (etwa unter Verwendung des CORBA Persistency Service) dauerhaft (d.h. im nicht-flüchtigen Bereich) zu speichern. Zu den persistent gespeicherten Daten gehören verwendete Workflow-Schemata, Zustände von Workflow-Instanzen (z.B. *init*, *running*, *suspended*, *committed*) ebenso wie Inhalte der Ein- und Ausgabeparameter. Das in Abschnitt 3 vorgestellte Workflow-Meta-Schema spezifiziert in einheitlicher Weise die persistent zu speichernden Objekte. Darüber hinaus werden technische Aspekte, wie etwa die Inhalte von Arbeitslisten, über die Workflow-Anwender Informationen des Systems zur Verfügung gestellt bekommen, persistent gespeichert. (Da der Aufbau und die Struktur von Arbeitslisten nicht Teil der logischen Struktur eines Workflow-Management-Systems ist, werden sie im Workflow-Meta-Schema nicht betrachtet.)

Der von uns in Anlehnung an die OMG-Spezifikation implementierte Persistency Service bietet die Funktionalität, Objektattribute im Dateisystem beziehungsweise in einer oder mehreren Datenbanken persistent zu speichern; zudem werden Objekte, die sich zum Zeitpunkt einer Anfrage nicht im Hauptspeicher befinden, automatisch und für den Benutzer transparent geladen. Der Trading Object Service ermöglicht die Registrierung von Objekten. Eine solche Registrierung bleibt persistent und kann von anderen Objekten oder Applikationen über diesen Dienst angefragt werden, auch wenn das Objekt selbst aus dem flüchtigen Speicher entfernt wurde. In diesem Fall wird das Objekt durch den Persistency Service geladen; dies erfolgt transparent für das Objekt, welches den Dienst angefragt hatte.

Aus diesem Grund müssen nach einem Systemausfall und anschließendem Wiederanlaufen ausgefallener Systemkomponenten für "statische" Objekte des Workflow-Meta-Schemas keine besonderen Vorkehrungen getroffen werden. Unter statischen Objekten sind alle Objekte gemeint, die nicht der Klasse Workflow angehören sowie Objekte der Klasse Workflow, die sich nicht im Zustand *running* befinden. Diese Objekte werden auto-

matisch zum Zeitpunkt einer Anfrage vom Persistency Service geladen und stehen somit nach dem Neustart des Systems automatisch wieder zur Verfügung.

Objekte der Klasse Workflow, die sich im Zustand *running* befinden, werden als "dynamische" Objekte bezeichnet; bei ihnen reicht das einfache Laden über den Persistency Service nicht aus. Sie müssen demgegenüber unmittelbar nach dem Wiederanlaufen des Systems ihre Arbeit wieder aufnehmen und sorgen anschließend durch Zugriffe auf weitere, statische Objekte für die korrekte Fortsetzung der Workflow-Ausführung. Aus diesem Grund sind auch Workflow-Instanzen im Zustand *running* im Trading Object Service registriert. Fährt ein Rechner hoch, dann erfragt dieser alle Workflow-Instanzen vom Trading Object Service, die sich auf dem betreffenden Rechner befinden und im Zustand *running* sind. Diese werden dann geladen und können ihre Ausführung an der Stelle fortsetzen, an der zuvor der Systemfehler aufgetreten war. Dabei wird die Eigenschaft verwendet, daß Objekte generell persistent sind und sich jeweils der aktuellen Zustand eines Objekts in der Datenbank befinden.

Da CORBA die Verteilung von Objekten in heterogenen Umgebungen unterstützt, kann jedes Workflow-Objekt auf einem beliebigen Rechner ausgeführt werden. Wie in Abschnitt 4.1 erklärt, gibt es in unserem Ansatz keine zentrale Workflow-Engine, die eine Menge von Workflow-Instanzen kontrolliert. Stattdessen kommunizieren Workflow-Instanzen miteinander, um die im Workflow-Schema definierten Eigenschaften für die konkrete Ausführung sicherzustellen. Dabei können die Workflow-Instanzen auf unterschiedlichen Rechnern ausgeführt werden; die Kommunikation erfolgt durch den Object Request Broker. Durch diesen Ansatz erreichen wir ein hohes Maß an Toleranz gegenüber Ausfällen einzelner Rechnersysteme. Nehmen wir an, daß das Rechnersystem eines komplexen Workflows zu einem Zeitpunkt ausfällt, zu dem seine Sub-Workflows bereits instanziiert wurden und ihre Berechnungen begonnen haben. Da – wie oben beschrieben – die Ausführung eines komplexen Workflows durch seine Sub-Workflows autonom kontrolliert wird, laufen diese bei Ausfall des Super-Workflows zunächst unbehindert weiter. Wenn der letzte Sub-Workflow beendet ist und dieser dem Super-Workflow eine Nachricht sendet, so wird der Super-Workflow unter Verwendung des CORBA Persistency Service neu geladen, so daß dieser seine Ausführung fortsetzen kann.

Bei der Reaktivierung einer aktiven Workflow-Instanz muß beachtet werden, ob es sich um einen transaktionalen oder um einen nicht-transaktionalen Workflow handelt. Für transaktionale Workflows und für Workflows ohne externe Datenzugriffe reicht es aus, diese neu zu starten. Nicht transaktionale Workflows müssen über eine kompensierende Aktivität zurückgesetzt werden, sofern dies vorgesehen ist. Anderenfalls muß eine verantwortliche Person darüber informiert werden, daß eine Workflow-Instanz während der Ausführung gescheitert ist und ein automatisches Wiederanlaufen nicht möglich ist.

7 Verwandte Arbeiten

In der Forschung und Entwicklung im Bereich Workflow-Management gibt es derzeit eine Reihe von Schwerpunkten, von denen Flexibilisierung und dynamische Veränderungen [10, 20, 29, 30, 7] sowie objektorientierte Ansätze [26, 5] und die OMG-Aktivitäten bei der Definition einer Workflow-Facility [18] für unsere Arbeiten besonders relevant sind.

Wir sind zu Beginn kurz auf die Ausschreibung zur Spezifikation der OMG Workflow Facility eingegangen [18], auf die es eine Reihe von Einreichungen namhafter Firmen aus dem Workflow-Bereich gibt [16, 3]. Diese Vorschläge sind in der Regel aus bereits bestehenden Workflow-Management-Systemen entstanden, die meist nicht unter Verwendung von objektorientierten Techniken modelliert oder entwickelt wurden. Aus diesem Grund ist es auch nicht verwunderlich, daß diese Einreichungen eine Reihe von Mängeln aufweisen, die u.a. in [23] aufgezeigt werden. Besonders auffällig ist, daß kaum "objektorientiert" vorgegangen wird, sondern oft bereits bestehende relationale bzw. prozedurale Strukturen in Objekte umgewandelt werden. Wesentliche Kritikpunkte werden nun anhand der Einreichung *jFlow* [3] zusammenfassend dargestellt:

Zunächst fällt auf, daß oft implementierungstechnische Strukturen als Objekte modelliert werden. So werden zum Beispiel Iterator-Objekte spezifiziert, die für die logische Struktur eines Workflow-Management-Systems nicht relevant sind, sondern die technische Realisierung von Datenströmen beschreiben. Mit dieser Modellierung werden bereits bei der objektorientierten Analyse implementierungstechnische Details vorweggenommen. Weiterhin zeichnet sich objektorientiertes Vorgehen stets dadurch aus, daß Daten und Funktionalität von Objekten gemeinsam beschrieben werden. Dieses Prinzip wird jedoch bei der *jFlow*-Einreichung durch die getrennte Spezifikation einer Workflow-Engine (in [3] als *Workflow Manager Object* bezeichnet) und Workflow-Instanzen (*Execution Elements*) verletzt: Die Daten werden auf Seiten der Workflow-Instanzen modelliert, während die Funktionalität der Workflow-Instanzen bei der Workflow-Engine beschrieben wird. Ein weiterer Schwachpunkt des *jFlow*-Ansatzes ist die Schnittstelle der *Process Definition*, mit der sich lediglich Workflow-Instanzen erzeugen und manipulieren lassen. Es wird nicht beschrieben, wie Prozesse definiert und verwaltet werden. Insgesamt weist dieser Vorschlag aus objektorientierter Sicht eine Reihe von Defiziten auf, die eine systemtechnische Umsetzung des in [3] beschriebenen Designs mit CORBA-Technologie erschweren.

Im Bereich objektorientierte Workflow-Management-Systeme sind die Projekte Meteor₂ [26], METUFlow [5] und WorCOS [24] zu nennen. In Meteor₂ werden Workflow-Spezifikationen in ausführbaren Workflow-Code übersetzt, der dann in einer verteilten Umgebung ausgeführt werden kann; Arbeiten zur Flexibilisierung und dynamischen Veränderung wurden im Rahmen dieses Projekts bislang nicht veröffentlicht. Das METUFlow-System [5] erlaubt die Spezifikation von Workflows durch Definition spezieller Ausführungseigenschaften, die un-

ter Verwendung des ACTA-Formalismus [2] spezifiziert werden. Das Hauptaugenmerk dieses Projekts liegt auf einer verteilten Ausführung von Workflows unter Verwendung des Guard-Konzepts [5]; Flexibilitäts-Aspekte werden im Rahmen dieses Projekts derzeit nicht untersucht. Das WorCOS Projekt beschäftigt sich mit der Entwicklung eines Workflow-Management-Systems auf der Basis einer CORBA-Architektur. Die Schwerpunkte dieses Projekts liegen auf der Wiederverwendbarkeit von Workflow-Schemata sowie der Einbindung von Business-Objekten in Workflow-Anwendungen [24].

In [20] wird mit ADEPT eine Workflow-Sprache vorgestellt, in der Workflow-Schemata als symmetrische Graphen definiert werden können. Es gibt spezielle Knotenarten, etwa für AND- und OR-Verzweigungen. Basierend auf dieser Workflow-Sprache wird mit ADEPT_{flex} ein Modell beschrieben, im Rahmen dessen dynamische Modifikationsoperationen (etwa Einfügen von Aktivitäten) auf kontrollierte Weise durchgeführt werden können. Dies bedeutet, daß ein modifiziertes Workflow-Schema durch Umformungsoperationen in ein äquivalentes Workflow-Schema überführt wird, das den durch die Workflow-Sprache definierten Regeln (u.a. Symmetrie) entspricht. Während sich ADEPT_{flex} derzeit im wesentlichen auf konzeptioneller Ebene mit dynamischen Modifikationen beschäftigt, schlägt unser Beitrag einen konkreten objektorientierten Entwurf eines Workflow-Management-Systems zur Unterstützung dynamischer Modifikationen vor. Ein weiterer Ansatz zur Flexibilisierung stellt [10] dar. In dieser Arbeit werden spezielle Petri-Netze zur Workflow-Modellierung verwendet. Es werden einfache dynamische Modifikationen betrachtet, die es erlauben, spezielle Netz-Teile beim Start eines Workflows undefiniert zu lassen, um sie zur Laufzeit durch Auswahl aus einer Menge von vorhandenen bzw. durch Definition neuer Teil-Prozesse zu definieren; dieser Vorgang wird mit "später Modellierung" bezeichnet; sie ist im Prototypen CORMAN implementiert [10].

8 Zusammenfassung und Ausblick

Zum Design und zur Implementierung komplexer Software-Systeme ist eine konsistente und angemessene Modellierung des Systems und seiner angestrebten Funktionalität unerlässlich. In diesem Beitrag haben wir ein Workflow-Meta-Schema eines flexiblen Workflow-Management-Systems vorgestellt, das die wesentlichen Entitäten klassifiziert und ihre Beziehungen zu anderen Entitäten darstellt. Workflow-Instanzen werden von komplexen Workflows zur Laufzeit dezentral erzeugt; sie nehmen während ihrer Laufzeit eine Reihe von Zuständen an, die sie von ihrer Instantiierung über ihren Start bis hin zu ihrer Termination führen. Dieses dynamische Verhalten von Workflow-Instanzen wurde unter Verwendung von geschachtelten Zustandsübergangsdiagrammen spezifiziert, wobei insbesondere festgelegt wurde, welche Methoden in welchen Zuständen zur Verfügung stehen und welche Zustandsübergänge sie bewirken. Anhand eines Beispiels wurden dynamische Modifikationsoperatio-

nen mit ihren Auswirkungen auf Workflow-Objekte beschrieben.

Ausgehend von den in diesem Beitrag beschriebenen Konzepten implementieren wir derzeit prototypisch eine zweite Version des Workflow-Management-Systems WASA, dessen generische Architektur in [1] beschrieben wurde. Die bei der Entwicklung eines ersten WASA-Prototypen gemachten Erfahrungen und weitere, bereits entwickelte Konzepte [30, 31] sowie zu vertiefende Konzepte – etwa im Bereich von Fehlersicherheit – werden Eingang in den WASA₂-Prototypen finden.

Danksagung: Wir danken den anonymen Gutachtern für die konstruktive Kritik.

Literatur

1. Bauzer Medeiros, C., Vossen, G., Weske, M.: *WASA: A Workflow-based Architecture to Support Scientific Database Applications (Extended Abstract)*. In Proc. 6th DEXA Conference, London, Lect. Notes Comput. Sci., Vol 978, pp 574–583. Berlin: Springer 1995
2. Chrysanthis, P.K., Ramamritham, K.: *ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behaviour*. In Proceedings of the 1990 ACM SIGMOD Conference on Management of Data, 194–203 (1990)
3. CoCreate Software, Cententus, CSE Systems, Data Access Technologies, Digital Equipment Corp., DSTC, EDS, FileNet Corp., Fujitsu Ltd., Hitachi Ltd., Genesis Development Corp., IBM Corp., ICL Enterprises, NIIP Consortium, Oracle Corp., Plexus - Division of BankTec, Siemens Nixdorf Informationssysteme, SSA, Xerox: *BODTF-RFP 2 Submission Workflow Management Facility (jFlow)*. OMG Document bom/98-06-07 (1998) (available from www.omg.org)
4. Electronic Data Systems, Data Access Technologies, Genesis Development Corp., NIIP, System Software Associates, Inc.: *BODTF-RFP 1: Combined Business Object Facility*. OMG Document bom/98-05-03 (1998) (available from www.omg.org)
5. Dogac, A., Gokkoca, E., Arpinar, S., Koksall, P., Cingil, I., Arpinar, B., Tatbul, N., Karagoz, P., Halici, U., Altinel, M.: *Design and Implementation of a Distributed Workflow Management System: METUFlow*. NATO ASI Workshop, Istanbul, August 12–21 (1997). Berlin: Springer ASI NATO Series
6. Eder, J., Liebhart, W.: *Workflow Recovery*. In Proc. First IFCIS International Conference on Cooperative Information Systems CoopIS'96. IEEE Computer Society Press 1996, 124–134
7. Ellis, C., Keddara, K., Rozenberg, G.: *Dynamic Change Within Workflow Systems*. In Proc. Conference on Organizational Computing Systems (COOCS), Milpitas, CA 1995, 10–22
8. Georgakopoulos, D., Hornick, M., Sheth, A.: *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*. Distributed and Parallel Databases, 3:119–153 (1995)
9. Gray, J., Reuter, A.: *Transaction Processing: Concepts and Techniques*. San Mateo, CA: Morgan Kaufmann 1993
10. Hagemeyer, J., Herrmann, T., Just-Hahn, K., Striemer, R.: *Flexibilität bei Workflow-Management-Systemen*. Software-Ergonomie '97, 179–190, Dresden, März 3–6 (1997)
11. Jablonski, S., Bußler, C.: *Workflow-Management: Modeling Concepts, Architecture and Implementation*. London: International Thomson Computer Press 1996
12. Jablonski, S., Böhm, M., Schulze, W. (Hrsg.): *Workflow-Management: Entwicklungen von Anwendungen und Systemen*. Heidelberg: dpunkt 1997
13. Leymann, F.: *Transaktionsunterstützung für Workflows*. Informatik Forschung und Entwicklung 12, 82–90 (1997)

14. Leymann, F., Altenhuber, W.: *Managing Business Processes as an Information Resource*. IBM Systems Journal 33, 326–347 (1994)
15. McClatchey, R., LeGoff, J.-M., Baker, N., Harris, W., Kovacs, Z.: *A Distributed Workflow and Product Data Management Application for the Construction of Large Scale Scientific Apparatus*. NATO ASI Workshop, Istanbul, August 12–21 (1997). Berlin: Springer ASI NATO Series
16. Nortel: *Submission Workflow Management Facility Specification*. (Supported by: University of Newcastle upon Tyne) OMG Document bom/98-03-01 (1998) (available from www.omg.org)
17. OMG: *CorbaServices: Common Object Services Specification* (available from www.omg.org)
18. OMG: *Workflow Management Facility: Request for Proposals*. OMG Document cf/97-05-06 1997 (available from www.omg.org)
19. Rational Software et al.: *Unified Modeling Language – UML Notation Guide. Version 1.1*, September 1997 (available from www.rational.com/uml)
20. Reichert, M., Dadam, P.: *Supporting Dynamic Changes of Workflows Without Losing Control*. Journal of Intelligent Information Systems, Special Issue on Workflow and Process Management, Vol. 10, No. 2 (1998)
21. Reuß, T., Vossen, G., Weske, M.: *Modeling Samples Processing in Laboratory Environments as Scientific Workflows*. Proc. 8th International Workshop on Database and Expert Systems Applications 1997, Toulouse, France. IEEE Computer Society Press, 49–55
22. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenzen, W.: *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice Hall 1991
23. Schulze, W.: *Evaluation of the Submissions to the Workflow Management Facility RFP*. OMG Document bom/97-09-02 (1997)
24. Schulze, W.: *Objektorientierte Implementierungstechniken für Workflow-Management-Systeme in OMA-konformen Architekturen*. In: Jablonski, S., Böhm, M., Schulze, W. (Hrsg.): *Workflow-Management: Entwicklungen von Anwendungen und Systemen*. Heidelberg: dpunkt 1997
25. Sheth, A., Georgakopoulos, D., Joosten, S.M.M., Rusinkiewicz, M., Scacchi, W., Wileden, J., Wolf, A.: *Report from the NSF Workshop on Workflow and Process Automation in Information Systems*. Technical Report UGA-CS-TR-96-003 University of Georgia, Athens 1996
26. Sheth, A., Kochut, K.J.: *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems*. NATO ASI Workshop, Istanbul, August 12–21, 1997. Berlin: Springer ASI NATO Series
27. Siegel, J.: *Corba – Fundamentals and Programming*. New York: John Wiley 1996
28. Vossen, G., Weske, M.: *The WASA Approach to Workflow Management for Scientific Applications*. NATO ASI Workshop, Istanbul, August 12–21, 1997. Berlin: Springer ASI NATO Series
29. Vossen, G., Weske, M., Wittkowski, G.: *Dynamic Workflow Management on the Web*. Technical Report Angewandte Mathematik und Informatik 24/96-I, Universität Münster 1996
30. Weske, M.: *Flexible Modeling and Execution of Workflow Activities*. In Proceedings of 31st Hawai'i International Conference on System Sciences, Software Technology Track (Vol VII), 713–722. IEEE Computer Society Press 1998
31. Weske, M.: *State-based Modeling of Flexible Workflow Executions in Distributed Environments*. Proc. Third Biennial World Conference on Integrated Design and Process Technology (IDPT), Volume 2: Issues and Applications of Database Technology, 94–101. Society for Design and Process Science 1998
32. Weske, M., Vossen, G.: *Workflow Languages*. In: P. Bernus, K. Mertins, G. Schmidt (Editors): *Handbook on Architectures of Information Systems*, Berlin: Springer 1998
33. Weske, M., Kuroepka, D., Hündling, J., Schuschel, H.: *Konzeption eines flexiblen Workflow-Management-Systems für Corba-Architekturen*. Technical Report Angewandte Mathematik und Informatik 18/97-I, Universität Münster 1997

This article was processed by the author using the L^AT_EX style file *cljour2* from Springer-Verlag.