



UNIVERSITÄT HAMBURG

FACHBEREICH WIRTSCHAFTSWISSENSCHAFTEN

INSTITUT FÜR WIRTSCHAFTSINFORMATIK

## Seminar zur Wirtschaftsinformatik

Prof. Dr. Stefan Voß

Thema: 1

Workflow-Muster

Betreuer: Herr Fink

Abgabe: 18.12.2003

Vorgelegt von:

Nathalie Jorzik

Matr.-Nr. 5300790

Studiengang

Wirtschaftsmathematik

Fachsemester 7

nathalie\_jorzik@gmx.de

Christine Kleingarn

Matr.-Nr. 5293190

Studiengang

Wirtschaftsmathematik

Fachsemester 7

christine\_kleingarn@gmx.de



# Inhaltsverzeichnis

<b>ABBILDUNGSVERZEICHNIS</b>	<b>I</b>
<b>TABELLENVERZEICHNIS</b>	<b>III</b>
<b>1 EINLEITUNG</b>	<b>1</b>
1.1 PROBLEMSTELLUNG	1
1.2 ZIELSETZUNG	1
1.3 GANG DER UNTERSUCHUNG	1
<b>2 WORKFLOW-MUSTER</b>	<b>1</b>
2.1 GRUNDLEGENDE KONTROLLFLUßMUSTER	1
2.2 ERWEITERTE BRANCHING- UND SYNCHRONISATIONSMUSTER	3
2.3 STRUKTURMUSTER	7
2.4 MUSTER MIT MULTIPLLEN INSTANZEN	8
2.5 ZUSTANDSBASIERTE MUSTER	9
2.6 STORNIERUNGSMUSTER	14
<b>3 WORKFLOW-MANAGEMENT-SYSTEME</b>	<b>15</b>
3.1 EINFÜHRUNG	15
3.2 ERGEBNISSE	15
<b>4 SCHLUSSBETRACHTUNG</b>	<b>16</b>
<b>ANHANG</b>	<b>17</b>
<b>LITERATURVERZEICHNIS</b>	<b>19</b>

## Abbildungsverzeichnis

Abbildung 1: Design-Muster für multi-choice.....	4
Abbildung 2: merge.....	4
Abbildung 3: merge.....	5
Abbildung 4: Typische Anwendung von Multi-merge-Mustern.....	6
Abbildung 5: Anwendung von arbitrary cycles.....	7
Abbildung 6: Transformation zur Blockstruktur.....	7
Abbildung 7: Unterschied zwischen einem impliziten und expliziten XOR-Split.....	10
Abbildung 8: Möglichkeiten der „deferred choice“ Implementierung.....	11
Abbildung 9: Verschachtelte Ausführung von A, B und C.....	12
Abbildung 10: Mögliche Anwendung vom „milestone“ Muster.....	13

## **Tabellenverzeichnis**

Tabelle 1: Vergleich zwischen den Management-Systemen.....17

## **1. Einleitung**

In unserer Seminararbeit beschäftigen wir uns mit Workflow-Mustern mit Bezug auf Workflow-Management-Systeme. Dabei stützen wir uns auf eine Arbeit von W.M.P van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski und A.P.Barros [van der Aalst et al. (2000)], welche das gleiche Thema behandelt.

### **1.1 Problemstellung**

Die vielen verschiedenen aktuellen Workflow-Management-Systeme, teilweise nachzuschlagen z.B. in [van der Aalst (1998)], [van der Aalst / van Hee (2002)] oder [Ellis / Nut (1993)], stellen verschiedene Anforderungen an die Architektur des Workflows. Diese Unterschiede wirken sich auf die Anwendbarkeit und die Aussagekraft der Systeme aus.

### **1.2 Zielsetzung**

Indem wir Workflow-Muster und ihre Anwendungen untersuchen, wollen wir nützliche routing-Konstruktionen finden, um den Anforderungen der Systeme gerecht zu werden und außerdem das Ausmaß dieser Anforderungen aufzeigen. Die von den Autoren gefundenen Muster wurden schon von verschiedenen Autoren genutzt, um neue Workflow-Sprachen zu entwickeln, z.B. in [Lavana (2000)], [Hirschall (2001)].

### **1.3 Gang der Untersuchung**

Wir werden 20 Workflow-Muster erläutern, angefangen bei den einfachsten bis hin zu komplizierten Konstruktionen. Dabei geben wir bei den grundlegenden Mustern eine kurze Beschreibung an, deuten auf Synonyme hin und geben Beispiele.

Zur Erklärung der komplexeren Muster beschreiben wir außerdem Umsetzungsprobleme und Anwendungsstrategien. Letztendlich gehen wir auch noch kurz auf die Workflow-Management-Systeme ein, da zwischen ihnen und den Mustern ein enger Zusammenhang besteht.

## **2. Workflow-Muster**

### **2.1 grundlegende Kontrollfluß-Muster**

In diesem Abschnitt stellen wir die elementarsten Muster vor. Sie werden von allen Workflow-Management-Systemen unterstützt. Die Definitionen entsprechen denen der elementaren Kontrollflußkonzepte der Workflow Management Coalition bei [WFM (1999)].

**Muster 1: Sequence (Reihung)**

**Beschreibung** Ein Vorgang in einem Workflow-Prozess ist nur durchführbar, wenn zuvor ein anderer durchgeführt wurde.

**Synonyme** Sequential routing, serial routing

**Beispiel:** Eine Versicherungsforderung wird geprüft nachdem der Fall des Klienten aufgenommen wurde.

***Implementierung***

Das Reihenfolgen-Muster wird benutzt, um aufeinander folgende Schritte in einem Workflow-Prozess zu modellieren

**Muster 2: paralleler Split**

**Beschreibung** An einem Knoten in einem Workflow-Prozess spaltet sich ein einzelner Kontrollpfad auf in mehrere Pfade, die parallel weiter laufen können.

**Synonyme** AND-Split, parallel routing

**Beispiel:** Nachdem die Versicherungsforderung registriert wurde, starten zwei Prozesse parallel: zum einen wird die laufende Versicherung des Kunden geprüft, zum anderen wird der Schaden festgestellt.

***Implementierung***

Es gibt zwei grundlegende Ansätze: Ein expliziter AND-Split ist ein Knoten mit mehreren ausgehenden Transitionen, die aktiviert werden, sobald der Knoten aktiviert ist. Bei einem impliziten AND-Split hat jede ausgehende Transition zugehörige Bedingungen. Für eine parallele Schaltung müssen die Bedingungen der Transitionen also übereinstimmen.

**Muster 3: Synchronisierung**

**Beschreibung** mehrere parallele Pfade fließen an einem Knoten zusammen. Dabei wird angenommen, dass jeder ankommende Zweig durchgeführt wurde.

**Synonyme** AND-join, Rendezvous, Synchronizer

**Beispiel:** Die Versicherungsforderung wird geprüft nachdem der Versicherungsstand des Kunden geprüft wurde und der Schaden festgestellt wurde.

***Implementierung***

Wie bei Muster 2 gibt es auch hier zwei grundlegende Ansätze: explizite und implizite AND-joins.

**Muster 4: Exclusive Choice**

**Beschreibung** Ein Knoten in einem Workflow-Prozess, an dem einer von mehreren Pfaden gewählt wird.

**Synonyme** XOR-Split, switch, selection

**Beispiel:** Nach *Schadensbeurteilung* erfolgt entweder *Zahlung* oder *Kunden\_informieren*.

***Implementierung***

Wie bei Muster 2 gibt es mehrere Strategien. Einige Systeme unterstützen das Muster direkt, bei anderen Systemen muss der Designer die Exklusivität der Wahl durch eine exklusive Transitionsbedingung simulieren.

### **Muster 5: Simple Merge**

**Beschreibung** Ein Knoten, bei dem zwei oder mehr Zweige ohne Synchronisation zusammenfließen. Dabei wird nur eine der Alternativen durchgeführt.

**Synonyme** XOR-Join, asynchroner Join, merge

**Beispiel:** *Forderung\_ablegen* erfolgt entweder nach *Zahlung* oder nach *Kunden\_informieren*.

### **Implementierung**

Nehmen wir an, dass es keine parallele Durchführung mehrerer Pfade gibt. Dann gibt es in allen Systemen Konstruktionen, um dieses Muster anzuwenden. Einigen Systemen verlangen einen bestimmten Grad an Strukturiertheit, um zu garantieren, dass nur eine Alternative zur Zeit aktiv sein kann. In anderen Systemen muss der Anwender selbst die parallele Ausführung verhindern.

## **2.2 Erweiterte Branching- und Synchronisationsmuster**

Nun betrachten wir erweiterte Muster für Verzweigungen und Synchronisationen. Im Gegensatz zu den eben vorgestellten Mustern werden diese nicht grundsätzlich von allen Workflow-Systemen unterstützt. Dennoch sind sie in den Standard-Geschäftsprozessen weit verbreitet.

Muster 4 geht davon aus, dass genau eine der Alternativen eintritt. Manchmal ist es jedoch sinnvoll, mehrere Alternativen gleichzeitig zuzulassen. Dafür gibt es *multi-choice*.

### **Muster 6: Multi-Choice**

**Beschreibung** Ein Knoten im Workflow-Prozess, an dem ein oder mehrere Zweige gewählt werden können.

**Synonyme** Conditional routing, selection, OR-Split

**Beispiel:** Nach *Schadensfeststellung* wird entweder *Feuerwehr\_informieren* oder *Versicherung\_informieren* durchgeführt, oder beides.

**Problem** In vielen Systemen kann man Bedingungen an die Transitionen knüpfen. Dann kann das multi-choice-Muster direkt angewandt werden. Jedoch gibt es auch Systeme, bei denen das nicht geht. Diese bieten nur die reinen AND-Splits und XOR-Splits an.

### **Implementierung**

- Bei Systemen mit Transitionen mit Bedingungen wählt der Anwender einfach für jede Transition die gewünschte Bedingung. Bei Systemen, die nur AND-Splits und XOR-Splits kennen, kann multi-choice als Kombination der beiden Muster dargestellt und angewandt werden. Jeder Zweig erhält ein XOR, was den Zweig entweder aktiviert oder stilllegt. Dann werden alle XORs mit einem AND-Split aktiviert.

- Ähnlich dem ersten Konzept erhält man eine Lösung durch Vertauschung der Reihenfolge des parallelen Splits und der exklusiven Wahl. Jedem Zweig, der parallel aktiviert wird, wird ein AND-Split vorgeschaltet. Allen AND-Splits geht ein XOR-Split voraus, der den entsprechenden AND-Split aktiviert (Siehe Abb.1).

Bemerkung: Es gibt einen Zielkonflikt zwischen dem multi-choice in Workflow A und in Workflow C in Abbildung 1. Die in A dargestellte Lösung ist kompakter und damit anwenderfreundlicher. Jedoch ist eine automatische Überprüfung des Workflows nicht möglich, wenn die Abhängigkeit der Transitionsbedingungen nicht bekannt ist.

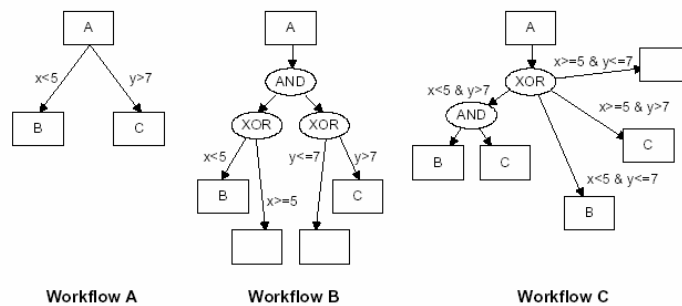


Abbildung 1: Design-Muster für multi-choice.

Bei neueren Workflow-Systemen ist das multi-choice-Muster einfach zu handhaben. Leider ist die korrespondierende Merge-Konstruktion (wird im nächsten Muster vorgestellt) nicht so einfach umzusetzen. Sie soll parallele Zweige synchronisieren und alternative Zweige verflechten. Die Schwierigkeit besteht darin, zu unterscheiden, wann was gemacht werden soll. Abbildung 2 zeigt ein Beispiel: nach A soll entweder B oder C, oder B und C, oder weder B noch C aktiviert werden.

Synchronisation könnte zum deadlock führen, Verflechtung könnte D mehrfach aktivieren.

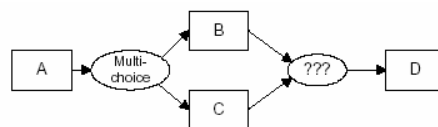


Abbildung 2: merge.

### **Muster 7: Synchronizing Merge**

**Beschreibung** ein Knoten im Workflow-Prozess, an dem mehrere Pfade zu einem Strang zusammenlaufen. Sind es mehrere Pfade, müssen die aktiven synchronisiert werden.

Ist es nur einer, teilen sich die alternativen Zweige ohne Synchronisation. Das Muster setzt voraus, dass ein aktivierter Pfad während der Verflechtung nicht noch einmal aktiviert werden kann.

**Synonyme** Synchronisierender Join

**Beispiel:** Entweder nach *Schadensfeststellung* oder nach *Feuerwehr\_informieren* oder nach beiden wird *Bericht\_vorlegen* ausgeführt (genau einmal!).

**Problem** Das Muster muss entscheiden, wann synchronisiert und wann verflochten wird.

### **Implementierung**

In den meisten Systemen ist die Anwendung dieses Musters nicht direkt möglich. Eine andere Lösung besteht darin, explizite OR-Splits, die mehr als eine ausgehende Transition haben, zu vermeiden und dafür AND- und XOR-Splits zu kombinieren (siehe Muster 6).

Die nächsten beiden Muster können angewendet werden, wenn die bei Muster 5 (Simple merge) gemachten Annahmen nicht gegeben sind, wenn es also auch Verflechtungen mit mehreren parallel ankommenden Zweigen gibt. Ein Beispiel dafür ist in Abbildung 3 gegeben. Mit dem einfachen Synchronisationsmuster wird D genau dann gestartet, wenn B und C gestartet wurden. Jedoch ist es manchmal nützlich, dass D startet, nachdem B oder C aktiviert sind. Solch ein Muster wird *Diskriminator* genannt.

In anderen Situationen ist es brauchbar, dass D *zweimal* startet, nach B und nach C. So ein Muster bezeichnet man als *multi-merge*.

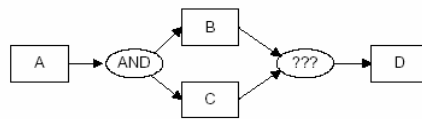


Abbildung 3: merge.

### **Muster 8: Multi-Merge**

**Beschreibung** Ein Knoten in einem Workflow-Prozess, an dem zwei oder mehr Zweige ohne Synchronisation zusammenlaufen. Dabei löst jeder ankommende aktive Zweig die nachfolgende Aktion erneut aus.

**Beispiel:** Manchmal haben mehrere Zweige ein gemeinsames Ende. Anstatt es für jeden Zweig einzeln zu konstruieren, kann ein multi-merge verwendet werden.

**Problem** Die von vielen Systemen angebotene Standard-merge-Konstruktion liefert für dieses Muster meist schlechte Ergebnisse.

### **Implementierung**

- Es gibt 3 merge-Konstruktionen, die direkt angewendet werden können.
- Ist das multi-merge nicht Teil einer Schleife, dann wird in Systemen, die nur eine Gelegenheit für einen Vorgang zulassen, der Vorgang im Modell wiederholt (siehe z.B. Abb. 4). Gehört es zu einer Schleife, dann ist die Anzahl der Gelegenheiten eines Vorgangs nach einem multi-merge während der Laufzeit nicht bekannt. Typische Lösungen für das Problem sind in Muster 14 und 15 enthalten.

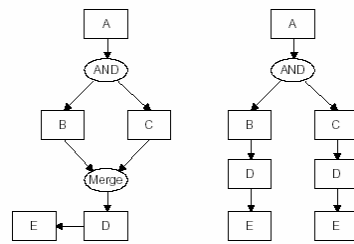


Abbildung 4: Typische Anwendung von multi-merge-Mustern.

### **Muster 9: (Diskriminator)**

**Beschreibung** Ein Punkt im Workflow-Prozeß, der auf einen ausgeführten ankommenden Zweig wartet, um den nachfolgenden Vorgang auszulösen. Von da an wartet der Diskriminator auf alle restlichen Zweige, um sie zu ignorieren. Sind alle ankommenden Zweige ausgelöst worden, kann der Diskriminator von neuem ausgelöst werden (wichtig für Schleifen!).

**Beispiel:** Um Zeit zu sparen wird eine komplexe Suche über das Internet an 2 verschiedene Datenbanken geschickt. Das schnellere Ergebnis soll den Prozess weiterführen, das zweite wird dann ignoriert.

**Problem** Bei den wenigsten Systemen ist der Diskriminator direkt anwendbar. Wie beim multi-merge-Muster angemerkt, wird durch die Standard-merge-Konstruktion in einigen Systemen der zweite Vorgang einer Aktivität nicht ausgelöst, wenn der erste aktiv ist. Jedoch wenn der erste Vorgang endet bevor neu gestartet wird, dann wird auch der zweite aktiviert.

### ***Implementierung***

- In den meisten Systemen kann der Diskriminator nicht direkt angewendet werden. Besonders für Schleifen ist er schwierig zu konstruieren und wird daher in seiner normalen Form kaum angewandt.
- In einem System hat ein normaler Join die gleiche Wirkung.
- In anderen gibt es eine spezielle Konstruktion: Endet einer der ankommenden Pfade, so wird der nachfolgende Vorgang ausgeführt und der Diskriminator schaltet von „ready“ auf „waiting“. Nachdem alle ankommenden Zweige geendet haben schaltet er wieder auf „ready“. Dies bietet eine direkte Anwendung, funktioniert aber nicht in einer Schleife.
- Einige Systeme benutzen *Custom Triggers*, individuelle Auslöser, z.B. für mehrere ankommende Transitionen. Bedingungen werden definiert, nach denen der Vorgang gestartet wird. Damit lassen sich Diskriminator-ähnliche Gebilde modellieren. Jedoch sind sie kompliziert und schwer nachzuvollziehen.

In [Casati et al. (1998)] sind noch weitere Anwendungsmöglichkeiten und Abweichungen beschrieben.

## 2.3 Strukturmuster

Verschiedene Systeme machen verschiedene Beschränkungen für ihre Workflow-Modelle. Dadurch wird die Anwendungsvielfalt eingeschränkt. Entweder muss der Designer von Anfang an diese Restriktionen beachten, oder er modelliert unabhängig davon und transformiert die Ergebnisse später. Von Interesse dabei ist die Tauglichkeit. In diesem Abschnitt werden 2 Muster vorgestellt, welche typische Beschränkungen und ihre Konsequenzen deutlich machen.

Schleifen gibt es in fast jedem System. Einige Systeme kennen jedoch nur strukturierte Schleifen (engl. *structured cycles*), die nur einen Eintrittspunkt und einen Austrittspunkt haben (vergleichbar mit WHILE-Schleifen bei Programmiersprachen).

### Muster 10: Arbitrary (beliebige) Cycles

**Beschreibung** Ein Punkt, an dem ein oder mehr Vorgänge wiederholt durchgeführt werden können.

**Synonyme** Loop, Iteration, Cycle

**Problem** In einigen Systemen nicht erlaubt.

**Implementierung**

Arbitrary (beliebige) Schleifen können in *strukturierte Schleifen* umgewandelt werden, wenn sie komplexere Muster wie z.B. Muster 14 enthalten (siehe auch [Kiepuszewski et al. (2000)]). Ein Beispiel wird in Abb. 6 dargestellt. Solche strukturierten Schleifen können meist direkt angewandt werden.

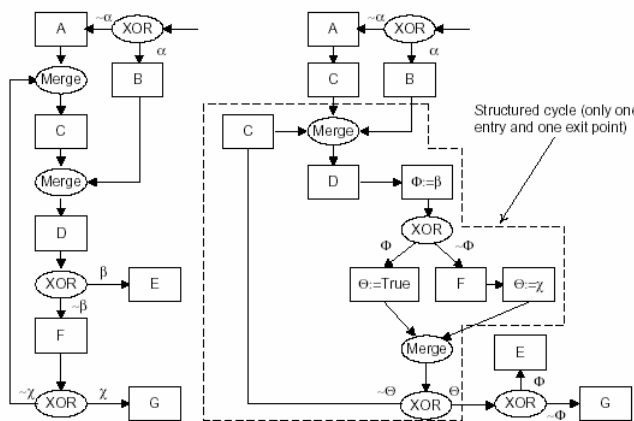


Abbildung 5: Anwendung von arbitrary cycles.

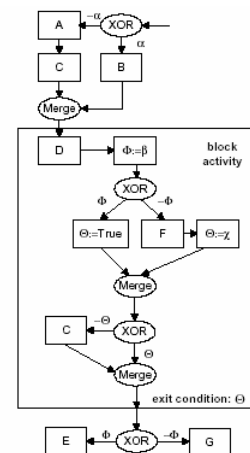


Abbildung 6: Transf. zur Blockstruktur.

### Muster 11: Implicit Termination

**Beschreibung** Ein Subprozess sollte abgeschlossen werden, wenn alle Aktionen aktiviert wurden.

**Problem** Die meisten Systeme brechen ab, wenn ein expliziter Finalpunkt erreicht ist.

**Implementierung**

Einige Systeme unterstützen dieses Muster direkt. Bei den übrigen wird das Modell in ein äquivalentes umgewandelt, das bloß einen Endpunkt hat. Bei einfacheren Modellen geschieht das durch Kombinationen von Joins und Schleifen.

## **2.4 Muster mit multiplen Instanzen**

Alle Muster in diesem Abschnitt beinhalten *multiple Instanzen*. Theoretisch sind sie vergleichbar mit mehrfachen Pfaden. Praktisch jedoch bedeutet dies, dass ein Vorgang zur selben Zeit mehrfach aktiv sein kann. Wir werden sehen, dass das durchaus von Nutzen sein kann! Das Haupt-Anwendungsproblem dabei ist, dass die meisten Systeme diese Muster nicht unterstützen können.

Es gibt 2 Anforderungen an die Muster mit multiplen Instanzen:

1. mehrere Zustände eines Vorgangs zu koppeln, und
2. diese Zustände zu synchronisieren und danach normal fortzufahren.

Die erste Anforderung wird von allen Systemen erfüllt.

### **Muster 12: Multiple Instanzen ohne Synchronisation**

**Beschreibung** Im Einzelfall können multiple Instanzen erzeugt werden. Sie müssen nicht unbedingt synchronisiert werden.

**Synonyme** *Spawn off facility*

**Beispiel:** Ein Kunde bestellt im Internet mehrere Bücher gleichzeitig. Viele Vorgänge (Verrechnung, updating, etc.) erfolgen direkt mit der Bestellung. Um die Vorgänge jedes einzelnen Buches erfassen zu können, brauchen wir multiple Instanzen.

#### ***Implementierung***

Die häufigste Anwendung geschieht durch die Kombinationen aus Schleifen und parallelen Splits (sofern die Systeme sie unterstützen). Siehe hierzu Workflow A in Abbildung 8.

Das Muster 12 wird von den meisten Systemen unterstützt. Problematisch ist nicht etwa, multiple Instanzen zu *generieren*, sondern sie zu *koordinieren*. Die folgenden 3 Muster zeigen, wie gleichzeitig laufende Pfade synchronisiert werden.

Im einfachsten Fall wissen wir, wie viele Vorgänge während des Prozesses aktiv sein werden. Dann brauchen wir nur eine Kombination aus parallelen Splits und Synchronisationen.

### **Muster 13: Multiple Instanzen mit a Priori Design Time Knowledge**

**Beschreibung** Für eine Prozessinstanz kann ein Vorgang mehrfach aktiv sein. Beim Entwurf des Modells ist die Zahl der Instanzen bekannt. Sind alle Instanzen durchgelaufen, müssen neue Vorgänge gestartet werden.

**Beispiel:** Die Bestellung von Gefahrgut benötigt drei verschiedene Berechtigungen.

#### ***Implementierung***

Ist die Zahl der Instanzen von Anfang an bekannt, kann einfach der Vorgang aus dem vorangehenden Modell wiederholt und ein paralleler Split eingebaut werden. Mit der Standard-Synchronisation werden alle Vorgänge nach Durchlauf wieder synchronisiert.

Bei bekannter Anzahl der Instanzen ist die Modellierung einfach. Ist jedoch die Zahl nicht zu bestimmen bevor der Prozess startet, kann das Muster 13 nicht angewendet werden. Dafür stellen wir die nächsten beiden Muster vor.

#### **Muster 14: Multiple Instanzen mit a Priori Runtime Knowledge**

**Beschreibung** Die Zahl der Instanzen eines gegebenen Vorgangs variiert von Fall zu Fall [Casati et al. (1995)], [Jablonski / Busser (1996)]. Sie wird erst bekannt, wenn der Prozess schon läuft, aber noch bevor die Instanzen der Aktivität erzeugt werden.

**Beispiel:** Im Prüfprozess einer wissenschaftlichen Arbeit wird die Aktivität *prüfe\_Arbeit* mehrfach durchgeführt, je nach Inhalt, Gutachtern, und Zeugnissen des Autors. Wenn alle Prüfungen durch sind, geht der Prozess weiter.

**Problem** Solange wir die Zahl der Instanzen nicht kennen, können wir nicht so einfach modellieren. Zur Zeit gibt es nur wenige Systeme, mit denen solche Konstruktionen möglich sind.

#### **Implementierung**

- Einige Systeme bieten zur Realisierung einer gegebenen Anzahl von Instanzen eine spezielle Konstruktion an: Ist die gewünschte Anzahl erreicht, werden sie in einer Bündelkonstruktion realisiert. Sind alle Instanzen eines Bündels durchgelaufen, wird der nächste Vorgang gestartet. Dies ist eine sehr übersichtliche Lösung (siehe Abb. 8, Workflow D).
- Ist die Anzahl der möglichen Instanzen begrenzt, so kann man die gewünschte Route aus AND- und XOR-Splits kombinieren. Mit den XOR-Splits wählt man die Anzahl der Instanzen und löst bestimmte AND-Splits aus. Für jede Anzahl an möglichen Instanzen gibt es einen AND-Split. Der Nachteil dieses Musters ist seine Größe und Komplexität und dass nicht immer die maximal zulässige Anzahl von vornherein bekannt ist (Abb.8, Workflow C).
- Oft kann der gewünschte Prozessablauf durch die festgelegte Reihenfolge der Instanzen einer Aktivität erreicht werden. Angenommen, nach A folgen n Instanzen von B, gefolgt von E. Dann wird erst A ausgeführt und dann die erste Realisierung von B. Jeder Realisierung von B folgt ein XOR-split, um zu entscheiden, ob noch eine weitere Realisierung von B gebraucht wird oder ob E folgt. Die n Realisierungen von B laufen dann nicht parallel, sondern in einer festen Reihenfolge (siehe Abb.8, Workflow B). Jedoch auch diese Lösung ist nicht immer erlaubt.

## **2.5 Zustands-basierte Muster**

Die bisherigen Muster gewährleisten keine dynamische Abbildung von Prozessvorgängen. Um Geschäftsabläufe vollständig und sinnvoll abbilden zu können, benötigt man häufig eine Modellierungsweise, die eine ausdrückliche Beschreibung des Arbeitsablaufs zu einem bestimmten Zeitpunkt darstellen kann. Um solche Muster zu formen, benötigen wir die uns bekannte Petri-Netz-Schreibweise, die uns ermöglicht, den aktuellen Zustand eines Geschäftsprozesses darzustellen.

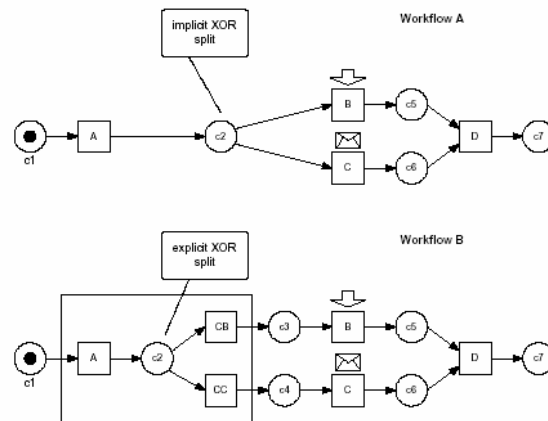


Abbildung 7: Unterschied zwischen einem impliziten und expliziten XOR-Split.

In Abb. 7 erfolgt die Aktivierung von A vor der Aktivierung von B bzw. C. C wird in diesem Modell ausgeführt, sobald z.B. eine externe Nachricht eingegangen ist, wohingegen B von einem internen Mitarbeiter ausgelöst wird. In Workflow A findet die Wahl für eine der folgenden Aktivitäten B oder C erst so spät wie möglich statt. Wenn die externe Nachricht, die C auslöst, vor der Ausführung von B durch einen internen Mitarbeiter ankommt, dann wird C ausgeführt und B gelöscht. In Workflow B wird die Wahl für B oder C bereits nach der Ausführung von Aktivität A festgelegt. Es ist wichtig zu erkennen, dass sich in Workflow A beide Aktivitäten im selben Arbeitsgang befinden; sobald die Wahl für die eine Alternative getroffen wird, wird die andere sofort gelöscht. In Workflow B sind die Aktivitäten in keinem gemeinsamen Arbeitsgang festgelegt, denn die Wahl der einen Aktivität bedeutet nicht, dass die andere Aktivität nicht trotzdem ausgeführt werden kann. Im Folgenden wird die in Workflow A dargestellte verzögerte Wahl formalisiert.

### **Muster 16: (Deferred Choice)**

**Beschreibung:** Betrachtet wird einen Knoten innerhalb des Geschäftsprozesses, an dem einer von mehreren Zweigen gewählt wird. Sobald eine Wahl getroffen wird, fallen die anderen Alternativen automatisch aus. Die Wahl findet so spät wie möglich statt, und zwar erst dann, wenn eine alternative Aktivität tatsächlich beginnt. Die Wahl wird hier also verzögert getroffen.

**Synonyme:** verzögerter XOR-Split, implicit choice, external choice.

**Beispiel:** Die Art des Transportes der bestellten Ware verzögert sich so lange, bis eine der Transportmöglichkeiten realisierbar ist.

**Problem:** Viele Workflow-Systeme können zwar den XOR-Split darstellen, aber haben Schwierigkeiten, ein Muster darzustellen, bei dem die Wahl einer Aktivität verzögert erfolgt.

### **Implementierung:**

- COSA ist eines der wenigen Produkte, das das Aufschieben einer Wahl direkt ermöglicht, zumal COSA als Modellierungssprache Petri-Netze verwendet. Eine verzögerte Wahl („deferred choice“) kann dann wie in Abb. 7 dargestellt werden. Einige andere Systeme können den

„deferred choice“ indirekt unterstützen mithilfe eines bestimmten Konstruktes. Dazu gibt es zwei Lösungsmöglichkeiten, die in Abb. 8 genauer dargestellt werden.

- Muster 16 wird durch einen eindeutigen XOR- Split ersetzt, wenn z.B. eine zusätzliche Aktivität vor die Wahl der Aktivitäten B oder C gesetzt wird. So kann z. B. in Abb. 7 der implizite XOR- Split durch Hinzunahme einer weiteren Aktivität in ein expliziten XOR- Split überführt werden.
- Eine weitere Konstruktion zur Erstellung des „deferred choice“ wird durch eine Verbindung der einzelnen Aktivitäten mithilfe eines AND-Split dargestellt, von denen alle anderen Aktivitäten ausfallen, sobald eine startet. Schwierig wird diese Konstruktion nur, wenn beide, sowohl B als auch C, gleichzeitig starten.

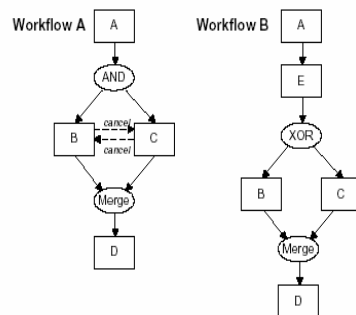


Abbildung 8: Möglichkeiten der „deferred choice“ Implementierung.

Mit Muster 2 und 3 wurden bereits Möglichkeiten dargestellt, Arbeitsabläufe parallel laufen zu lassen. Wenn diese Arbeitsabläufe dieselbe Datenbasis haben, kann es zu Schwierigkeiten kommen, indem die Prozesse zum Stillstand kommen oder neu Gespeichertes verloren geht. Wir stellen in dem Zusammenhang das nachfolgende Muster vor.

### **Muster 17: (Interleaved Parallel Routing)**

**Beschreibung:** Verschiedene Aktivitäten werden hintereinander ausgeführt, wobei nicht vorbestimmt ist, in welcher Reihenfolge sie schalten.

**Synonyme:** Unordered sequence.

**Beispiel:** Die Marine führt bei jeder Testperson zwei verschiedene Tests durch: einen physischen Test und einen mentalen Test. Allerdings ist es egal, in welcher Reihenfolge das passiert.

**Probleme:** Obwohl die meisten Workflow-Systeme gleichzeitig laufende Arbeitsprozesse darstellen können, ist es oftmals nicht möglich, eine Verschachtelung von Arbeitsabläufen darzustellen.

**Implementierung:**

- Eine Möglichkeit besteht darin, die Reihenfolge der Aktivitäten festzulegen, was allerdings dazu führt, dass dadurch die Flexibilität des Prozesses eingeschränkt wird. Diese Konstruktion wird in Workflow A in Abb. 9 dargestellt.

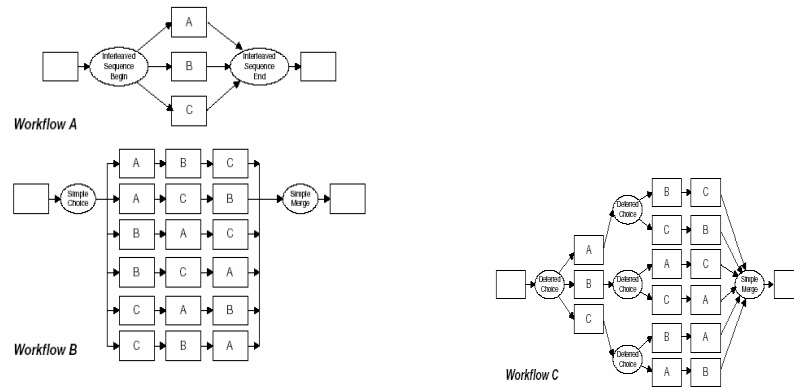


Abbildung 9: Verschachtelte Ausführung von A, B und C.

- Man definiert verschiedene alternative Folgen von Aktivitäten, und vor der Ausführung wird eine Folge mithilfe eines XOR- Split gewählt, so dass die Reihenfolge allerdings vor der Wahl bereits klar ist. Es kann außerdem sehr aufwendig werden, alle möglichen Folgen von Aktivitäten darzustellen, wie Workflow B zeigt.
- Ohne die Reihenfolge vorher festzulegen, kann z.B. mithilfe eines XOR- Split („deferred choice“) die Wahl der nächsten Aktivität von einer Aktivität zur nächsten festgestellt werden, was allerdings zu einer Spaghetti-förmigen Struktur wie in Workflow C führt, da alle möglichen Folgen dargestellt werden müssen.

Wenn Workflow-Systeme als Modellierungssprache Petri-Netze verwenden, dann kann man eine neue Stelle ins Modell einbauen, die für jede Transition direkt erreichbar ist und außerdem zu jeder Transition direkt wieder führt. Mit solch einer Konstruktion wird die Struktur des Modells nicht gefährdet.

Im nächsten Muster wird eine Möglichkeit dargestellt, mit der man innerhalb eines Prozesses feststellen kann, in welcher Phase des Prozesses man sich befindet. Solche Modelle sind notwendig, wenn sich ein Unternehmen mit Klagen oder Stornierungen befassen muss. Dann wird innerhalb des Modells ein Merkmal integriert, das angibt, zu welchem Zeitpunkt des Prozesses die Klage stattfindet.

### **Muster 18 (Milestone)**

**Beschreibung:** Es gibt drei Aktivitäten A, B und C, von denen B erst ausgeführt werden kann, nachdem A ausgeführt wurde, und nicht mehr ausgeführt werden kann, nachdem C ausgeführt wurde.

**Synonyme:** Test arc, deadline, state condition, withdraw message.

**Beispiel:** Ein Kunde kann seine Einkaufs-Bestellung spätestens zwei Tage vor der geplanten Auslieferung zurückziehen.

**Problem:** Innerhalb eines Klage-Prozesses findet ein Wettlauf zwischen Aktivitäten statt, in dem ein „milestone“ integriert werden muss, der angibt, ob eine Aktivität bis zu einem bestimmten Zeitpunkt stattgefunden hat ( z.B. Bestellung rechtzeitig storniert, Stornierung rechtzeitig gemeldet?). Viele

Workflow-Systeme haben allerdings das Problem, dass eine Aktivität, sobald sie aktiviert wurde, nicht vom Muster selber wieder deaktiviert werden kann. Sobald das Merkmal abgeschaltet wird, bedeutet das somit, dass die Meldung zurückgezogen wurde.

Mithilfe einfacher Synchronisation kann solch eine Modell nicht dargestellt werden, weil die Häufigkeit einer hintereinander ausgeführten Aktivität wie z.B. „Bestellung stornieren“ oder „Stornierung gemeldet“ schwankt.

### **Implementierung:**

- Wenn ein Workflow-System als Modellierungssprache Petri-Netze verwendet, dann kann wie in Workflow A, Abb. 10, die Aktivität A beliebig oft nach der Ausführung von Aktivität B und vor der Ausführung von Aktivität C geschaltet werden, mithilfe eines impliziten XOR-Splits.
- In Abb. 10, Workflow B, werden Aktivitäten A, B und C betrachten. Mithilfe eines impliziten XOR-Split kann entweder A oder C ausgeführt werden. Nach der Ausführung von A findet dann wieder eine neue Entscheidung statt, ob C oder ein weiteres Mal A ausgeführt wird. Wenn der Prozess bei C angelangt ist kann er allerdings nicht wieder zu A oder ein erneutes Mal zu C zurückkehren. Diese Modellkonstruktion wird mithilfe eines „deferred choice“ ermöglicht (siehe Muster 16).

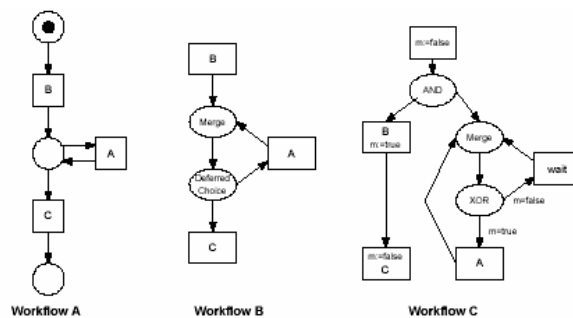


Abbildung 10: Mögliche Anwendung vom „milestone“-Muster.

- In Workflow C wird ein „milestone“-Muster mithilfe einer Variable *m* dargestellt. Zu Beginn des Prozesses wird *m* als „unwahr“ definiert. Eine AND-Verbindung führt zu Aktivität B und zu einer Schleife, durch die A ausgeführt werden soll. Sobald B aktiviert wurde, wird *m* als „wahr“ definiert. Die folgende Aktivität C definiert *m* als „unwahr“. Die parallel laufende Schleife untersucht periodisch, in welchem Zustand sich die Variable *m* gerade befindet. Im Zustand „wahr“ wird A aktiviert, andernfalls läuft die Schleife ein weiteres mal durch und stellt dann wieder den Zustand der Variablen *m* fest, usw. Mit solch einer Darstellung wird A allerdings nicht von der Ausführung von C beeinflusst, denn die Schleife kann hier nicht gestoppt werden, selbst wenn C bereits aktiviert wurde.

## **2.6 Stornierungsmuster**

Nachdem in Muster 16 bereits eine erste Lösung zum Löschen von Aktivitäten dargestellt wurde, wird in den nachfolgenden Mustern diese Eigenschaft noch weiter ausgeführt.

### **Muster19: (Cancel Activity)**

**Beschreibung:** Eine laufende Aktivität wird deaktiviert, z.B. wird ein Zweig im Prozess, der weiter ausgeführt werden soll, wieder gelöscht.

**Synonyme:** Withdraw activity.

**Beispiel:** Um einen Termin einzuhalten, wird ein neues Design, das normalerweise von zwei Ingenieur-Teams kontrolliert wird, nur von einem Team kontrolliert.

**Problem:** Nur wenige der Workflow-Systeme unterstützen die „cancel activity“.

#### ***Implementierung:***

- Wenn das Workflow-Muster 16 unterstützt wird, dann kann man zu einer echten Aktivität eine sog. „Schatten Aktivität“ hinzufügen und beide parallelisiert man mithilfe einer verzögerten Wahl („deferred choice“). Damit wird das Stornieren einer Aktivität ermöglicht. Nachteil ist, dass damit künstlich Aktivitäten eingefügt werden, die nicht zum tatsächlichen Prozess gehören.
- Es ist zwar nicht möglich, eine Aktivität direkt zu stornieren; stattdessen kann aber eine bestimmte Funktion innerhalb von Aktivitäten eingeführt werden, die eine andere Aktivität untauglich macht.

Im obigen Muster haben wir das Streichen einzelner Aktivitäten beschrieben. Es kann aber auch dazu kommen, dass ein ganzer Prozess gelöscht werden muss. Dies wird im Folgenden beschrieben:

### **Muster 20: (Cancel Case)**

**Beschreibung:** Hier werden nun ganze Prozesse entfernt, z.B. werden ganze Arbeits- und Dienstwege gelöscht.

**Synonyme:** Withdraw case.

**Beispiel:** In einem Einstellungsprozess für neue Angestellte zieht ein Bewerber vorzeitig seine Bewerbung zurück.

**Problem:** Nur wenige der Workflow-Systeme unterstützen die direkte Annullierung von Prozesswegen.

#### ***Implementierung:***

- Eine Möglichkeit besteht darin, Muster 19 auf jede der Aktivitäten anzuwenden; es gibt einen Auslöser, der eine Annullierung jeder anderen Aktivität im Arbeitsfluss verursacht.
- Ähnlich wie bei Muster 19 kann hier auch eine bestimmte Funktion eingesetzt werden, die die entsprechenden Eingaben von der Datenbank entfernt.

## 3 Workflow-Management-Systeme

### 3.1 Einführung

In diesem Kapitel werden verschiedene Workflow-Management-Systeme vorgestellt. Es wird dabei untersucht, inwiefern die im vorherigen Kapitel eingeführten Workflow-Muster von den Management-Systemen unterstützt werden.

In Tabelle 1 werden 8 Management-Systeme verglichen und bewertet, in Hinblick auf die Anwendbarkeit der im vorherigen Kapitel eingeführten Workflow-Muster. Die Seminararbeit zeigt große Unterschiede zwischen den einzelnen Workflow-Systemen auf, die mithilfe der Workflow-Muster deutlich gemacht werden.

Bei der Bewertung geht es im Wesentlichen nicht um die Funktionsweise des Produktes, sondern vielmehr darum, inwiefern diese Systeme die aufeinander folgenden, parallelen und wesentlichen Arbeitsabläufe, die in einem Unternehmen vorkommen können, darstellen können.

Im Zusammenhang mit Workflow-Systemen muss zusätzlich erwähnt werden, dass Unternehmen, bevor sie sich für ein wirklich geeignetes System entscheiden, oftmals keine vollständige Analyse durchführen über die für ihr Unternehmen benötigten Arbeitsmuster und abzubildenden Geschäftsprozesse. Daher kann ein für den Geschäftsablauf des Unternehmens wichtiges Muster häufig nicht genügend von dem gewählten Workflow-System unterstützt werden.

Der vorliegende Vergleich basiert auf Informationen, die bereits 2 Jahre alt sind und somit nicht unbedingt garantiert ist, dass die Informationen über die Systeme vollständig sind.

### 3.2 Ergebnisse

In Tabelle 1 werden die Ergebnisse des Vergleichs dargestellt, der sich aus einer früheren Untersuchung, basierend auf [van der Aalst et al.], ergab, in dem weitere Informationen über die tabellierten Workflow-Systeme enthalten sind. Weitere Informationen können in der angegebenen Literatur über die entsprechende Produkte ( Staffware [Staffware (2000)], COSA [Software-Ley (1999)], Eastman Software [Eastman Software (1998)], FLOWer [Pallas Athena (2001)], Domino Workflow [Nielsen et al. (2000)], MQSeries [IBM (1999)], Visual Workflow [FileNet (1997)], SAP/R3 [SAP (1997)]) nachgelesen werden.

Für jede Produkt-Muster-Kombination wurde untersucht, ob sich das Muster direkt im System anwenden lässt. Diese Eigenschaft wird mit einem + notiert. Falls das Muster vom jeweiligen System nur indirekt unterstützt wird, was dann also zu Spaghetti-förmigen Strukturen führt oder zu einer Codierung, dann notieren wir dies mit +/-, wenn wirklich keine Möglichkeit besteht, eine bestimmtes Muster mit einem ausgewählten System darstellen zu können, dann setzen wir ein -.

Es ist zu beachten, dass „direkte Anwendung“ hier nur bedeutet, dass das vorliegende Muster bereits von der grafischen Schnittstelle der Software unterstützt wird, ohne dass eine Lösung gefunden werden muss, wie sie in den Implementierungsabschnitten der Muster in Kapitel 2 beschrieben wird.

Wie man der Tabelle 1 bereits entnehmen kann, gibt es kein Workflow-System, das alle 20 Muster direkt anwenden kann. Einige Muster werden sogar nur von sehr wenigen Produkten unterstützt. Im Abschnitt über Zustands-basierte Muster wird bereits erwähnt, dass diese nur von COSA direkt unterstützt werden können. Muster 14 und 15 können direkt nur bei FLOWer angewendet werden. Die Basis-Muster funktionieren bei allen 8 Systemen, es gibt allerdings keine Möglichkeit für Staffware und FLOWer, Muster 6 weder direkt noch indirekt anzuwenden. Eastman Software ist ein sehr nützliches Tool, um Muster 7-13 darzustellen, die damit alle ohne Änderung von Strukturen etc. dargestellt werden können. SAP ist das einzige Workflow-System, bei dem beide Cancel-Muster direkt dargestellt werden können, ansonsten kann man die „Cancel Activity“ noch durch Staffware oder COSA darstellen. „Cancel Case“ funktioniert hingegen nur bei SAP, alle anderen Workflow-Systeme (außer FLOWer) bieten nicht einmal die Möglichkeit der Kodierung.

Diese lückenhaften Funktionsweisen und unvollständigen Modellierungsmöglichkeiten der angegebenen Workflow-Systeme zeigen deutlich, wie wichtig es für das Unternehmen ist, genaue Analysen durchzuführen über die Muster, die die wesentlichen Geschäftsprozesse des Unternehmens darstellen können, um sich für das beste Produkt entscheiden zu können.

#### **4 Schlußbetrachtung**

Die Seminararbeit gibt uns eine Übersicht über die verschiedenen Workflow-Muster und stellt dar, inwiefern Workflow-Systeme diese Muster umsetzen können.

Die vorliegende Betrachtung, die auf [van der Aalst et al. (2002)] basiert, gibt uns eine kurze Einsicht in die Fehler, Eigenschaften und Grenzen aktueller Workflow-Systeme.

Kürzliche Untersuchungsprojekte über die Häufigkeit von verwendeten Mustern, die in [de Vries/Ommert (2001)] und [de Vries/Ommert (2002)] genauer nachgelesen werden können, ergaben, dass die moderneren Muster, die wir hier beschrieben haben, heutzutage nützlicher sind und häufiger Anwendung finden. Oftmals müssen Workflow-Designer mit einer ungeeigneten Software für bestimmte Prozesse komplexere, Spaghetti- förmige Prozesse oder sogar Kodierungen bilden, um den eigentlichen Geschäftsprozess im Unternehmen vollständig darstellen zu können. Ggf. muss der Geschäftsprozess sogar angepasst werden. Um noch genauer die Einflüsse zu untersuchen, die Workflow-Systeme auf Prozesse bzw. Geschäftsprozessmodellierung haben und um vollständig entscheiden zu können, welches System für ein Unternehmen tatsächlich das sinnvollste ist, sind noch dringend weitere Untersuchungen notwendig.

## Anhang

Muster	Produkte							
	Staffware	COSA	Eastman	FLOWer	Domino	MQSeries	Visual WF	SAP/R3
1	+	+	+	+	+	+	+	+
2	+	+	+	+	+	+	+	+
3	+	+	+	+	+	+	+	+
4	+	+	+	+	+	+	+	+
5	+	+	+	+	+	+	+	+
6	-	+	+/-	-	+	+	+	+
7	-	+/-	+	-	+	+	-	-
8	-	-	+	+/-	+/-	-	-	-
9	-	-	+	+/-	-	-	-	+
10	+	+	+	-	+	-	+/-	-
11	+	-	+	-	+	+	-	-
12	-	+/-	+	+	+/-	-	+	-
13	+	+	+	+	+	+	+	+
14	-	-	-	+	-	-	-	+/-
15	-	-	-	+	-	-	-	-
16	-	+	-	+/-	-	-	-	-
17	-	+	-	+/-	-	-	-	-
18	-	+	-	+/-	-	-	-	-
19	+	+	-	+/-	-	-	-	+
20	-	-	-	+/-		-	-	+

Tabelle 1: Vergleich zwischen den Management-Systemen Staffware, COSA, Eastman, FLOWer, Domino, MQSeries, Visual Workflow und SAP/R3 zur Anwendbarkeit der Workflow-Muster 1-20.



## Literaturverzeichnis

- [Casati et al. (1995)] F. Casati, S. Ceri, B. Pernici, G.Pozzi.  
Conceptual Modelling of Workflows. In: *Lecture Notes in Computer Science*, 1021 (1995), S. 341-354
- [ Pallas Athena (2001)] Pallas Athena.  
*Flower User Manual*. Apeldoorn, Niederlande : Pallas Athena BV, 2001
- [Casati et al. (1998)] F. Casati, S. Ceri, B. Pernici, G.Pozzi.  
Workflow Evolution. In: *Data and Knowledge Engineering*, 24 (1998), S. 211–238
- [Eastman Software (1998)] Eastman Software.  
*RouteBuilder Tool User's Guide*. Billerica USA : Eastman Software Inc., 1998
- [Ellis / Nutt] C.A. Ellis, C.J. Nutt.  
Modelling and Enactment of Workflow Systems. In: *Lecture Notes in Computer Science*, 691 (1993), S. 1–16
- [FileNet (1997)] FileNet.  
*Visual WorkFlo Design Guide*. FileNet Corporation; 1997
- [FileNet (1999)] FileNet.  
*Panagon Visual WorkFlo Architecture*. FileNet Corporation; 1999
- [Hirnschall (2001)] A. Hirnschall.  
*Patterns Discussion in the Context of Co-flow, CONDIS Workflow Management System* Universität Linz, 2001
- [IBM (1999)] IBM.  
*IBM MQ-Series - Getting Started With Buildtime*. Arbeitspapier IBM Deutschland Entwicklung GmbH, Böblingen, 1999
- [Jablonski / Bussler (1996)] S. Jablonski, C. Bussler.  
*Workflow Management: Modelling Concepts, Architecture and Implementation*. . : International Thomson Computer Press, 1996
- [Kiepuszewski et al. (2000)] B. Kiepuszewski, A.H.M. ter Hofstede, c: Bussler.  
On Structured Workflow Modelling. In: *Lecture Notes in Computer Science*, 1789 (2000), S. 431–445
- [Lavana (2000)] H. Lavana.  
*A Universally Configurable Architecture for Taskflow-Oriented Design of a Distributed Collaborative Computing Environment* North Carolina State Universität, 2000

- [Nielsen et al. (2000)]** S.P. Nielsen, C. Easthope, P. Gosselink, K. Gutsze, J. Röle.  
*Using Lotus Domino 2.0*. IBM; 2000
- [SAP (1997)]** SAP.  
*WF SAP Business Workflow*. Walldorf : SAP AG, 1997
- [Software-Ley (1999)]** Software-Ley.  
*COSA 3.0 User Manual*. Pullheim : Software-Ley GmbH, 1999
- [Staffware (2000)]** Staffware (Hrsg.).  
*Staffware 2000 7 GWD User Manual*. Berkshire, UK : Staffware plc, 2000
- [WFMC (1999)]** WFMC.  
*Workflow Management Coalition Terminology and Glossary, Document Status - Issue 3.0*. Workflow Management Coalition; 1999
- [de Vries / Ommert (2001)]** K. de Vries, O. Ommert (2001).  
Advanced Workflow Patterns in Practise: Experiences Based on Pension Processing. In: *Business Process Magazine*, 7 (2001), S. 5–18
- [de Vries / Ommert (2002)]** K. de Vries, O. Ommert.  
Advanced Workflow Patterns in Practise: Experiences Based on Judicial Processes. In: *Business Process Magazine*, 8 (2002), S. 20–23
- [van der Aalst (1998)]** W.M.P. van der Aalst.  
Three Good Reasons for Using a Petri-net-based Workflow Management System. In: *The Kluwer International Series in Engineering and Computer Science*, 1 (1998), S. 161–182
- [van der Aalst / van Hee (2002)]** W.M.P. van der Aalst, K.M. van Hee.  
*Workflow Management: Models, Methods and Systems*. Cambridge : Mit Press, 2002
- [van der Aalst et al.]** W.M.P. van der Aalst, A.H.M. Hofstede, B. Kiepuszewski, A.P. Barros.  
*Workflow Patterns Home Page*.  
<http://www.tm.tue.nl/it/research/patterns> – Letzter Abruf: 01.11.2003
- [van der Aalst et al. (2000)]** W.M.P. van der Aalst, A.H.M. Hofstede, B. Kiepuszewski, A.P. Barros.  
*Workflow Patterns*. Technische Universität Eindhoven, 2000