Universität Kaiserslautern
Fachbereich Informatik
AG Datenbanken und Informationssysteme
Prof. Dr. Theo Härder

ARKTIS/Makros
Eine Makrokomponente als Basis für flexible, erweiterbare WfMS

Diplomarbeit

von

Markus Bon

Betreuer:

Dipl.-Inform. Jürgen Zimmermann

Oktober 1999

Hiermit erkläre ich an Eides statt, daß ich die vorliegende Arbeit selbständig und unter ausschließlicher Verwendung der angegebenen Literatur angefertigt habe.

Kaiserslautern, den 20.10.1999

Markus Bon

Inhaltsverzeichnis

Kapitel 1 Einleitung

Abläufe in Unternehmen sind oft sehr komplex, was die Koordination der einzelnen Tätigkeiten zu einer schwierigen Aufgabe macht. Meist sind viele verschiedene Personen betroffen, die informiert werden, ihren Teil zur Ausführung beitragen und schließlich ihre erfolgreiche (oder auch nicht erfolgreiche) Bearbeitung weitermelden müssen. Diese Personen können aus ganz verschiedenen Bereichen stammen, etwa der Verwaltung, der Produktion oder der Haustechnik, die Kommunikation mittels Belegen über die Hauspost oder sogar der normalen Post ist äußerst zeitraubend. Arbeiten, die gleichzeitig durchgeführt werden könnten, werden oft mangels Koordination nacheinander ausgeführt. Mit einem Satz: Dinge dauern länger, als sie müßten.

In diesem Zusammenhang ist in den letzten Jahren der Begriff des Workflow-Management sehr populär geworden, doch leider gibt es hierzu noch keine oder nur wenig anerkannte, fundierte theoretische Grundlagen. Zahlreiche Firmen haben Workflow-Management-Systeme (WfMS) entwickelt, Software-Pakete, die Workflow-Management unterstützen sollen. In der *Workflow Mangement Coalition* (WfMC) haben sich verschiedene Firmen zusammengefunden, die gemeinsam Standards für Workflow-Management-Produkte entwickeln wollen. Die WfMC, hat untersucht, welche Gemeinsamkeiten die bestehenden WfMS besitzen und wie diese zusammengeführt werden können [WfMC98]. In MOBILE wird der Versuch gestartet, den Begriff des „Workflow-Mangements“ zu fassen. Dabei beschreibt Jablonski ein WfMS als einen intelligenten Terminkalender, der dem menschlichen Bearbeiter mitteilt, was er zu tun hat, ihm aber bei der Lösung seiner Aufgabe freie Hand läßt [Ja95]. Als Anforderungen an das WfMS stellt er die Integrierbarkeit bestehender Softwarekomponenten (Anwendungsprogramme, Datenbanken), Skalierbarkeit des Systems (da im voraus kaum Aussagen zur Anzahl der realisierten Abläufe gemacht werden können) und schließlich Anpaßbarkeit/Erweiterbarkeit, um auf sich ändernde Problemstellungen reagieren zu können.

Weiterhin wird bei der Modellierung als wichtigster funktionaler Qualitätsfaktor die Modularität genannt, d. h., Elemente des Workflow-Modells (die Abbildung des Geschäftsprozesses in die Welt des Workflow-Management-Systems) können separat beschrieben und frei zusammengesetzt werden, kleine Änderungen lassen sich durch überschaubare Anpassung der zugehörigen Beschreibung umsetzen.

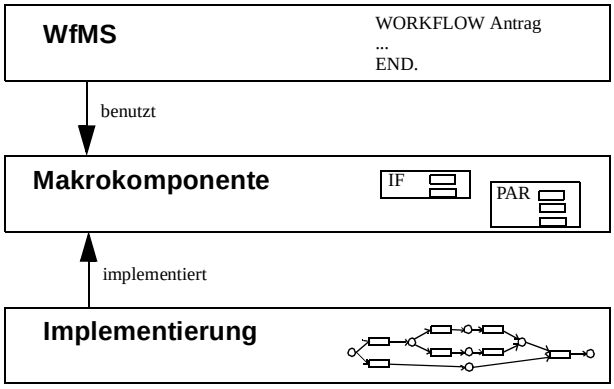
Jablonski unterscheidet in [Ja95] beim Workflow-Management verschiedene Aspekte: Der *funktionale* Aspekt beschreibt, *was* ausgeführt wird. Es werden logische Verarbeitungseinheiten (Workflows) definiert. Der *operationale* Aspekt spezifiziert, *wie* elementare Workflows ausgeführt werden sollen (und damit die eigentliche Funktionalität erbringen). Der *verhaltensbezogene* Aspekt beschreibt einen Kontrollfluß und somit *wann* ein Workflow oder eine Anwendung aus-

geführt werden soll. Im *informationsbezogenen* Aspekt wird der Datenfluß zwischen Workflows behandelt.

Bei der Modellierung dieser Aspekte, insbesondere des verhaltens- und des informationsbezogenen Aspekts, lassen sich immer wiederkehrende Strukturen erkennen, die sich nur in ihren Details unterscheiden. Es liegt nahe, die immer gleichen Teile zu extrahieren und als Gerüst anzubieten, das über Parameter an das jeweilige Problem angepaßt werden kann. Wenige dieser Bausteine, die wir im folgenden als „Makros“ bezeichnen, genügen schon, um die meisten gebräuchlichen Kontrollflußkonstrukte nachzubilden, aber auch Datenfluß läßt sich damit beschreiben. Makros lassen sich miteinander zu komplexeren Gebilden kombinieren, so daß eine hierarchische Struktur entsteht. Werden Makros mit entsprechenden Parametern instanziiert, so entstehen ablauffähige Workflows. Geschieht diese Instanziierung erst zur Laufzeit, erhalten wir sich dynamisch an das Problem anpassende, polymorphe Makros.

Neu definierte Makros können in eine Bibliothek aufgenommen werden, um sie bei Bedarf wieder einsetzen zu können. So entsteht eine umfangreiche Sammlung von „Bausteinen“, mit denen neue Workflows einfach zusammengesetzt werden können. Dies fördert auch die Übersichtlichkeit des Entwurfs. In Abbildung 1 ist das Zusammenwirken von Workflow-Management-System, Makrokomponente und Implementierungsschicht zu sehen. Im WfMS wird ein Prozeß in Form eines Workflows beschrieben, hierzu werden Makros benutzt, die die Makrokomponente zur Verfügung stellt. Die Makros werden wiederum in eine geeignete, ausführbare Implementierung umgesetzt. Die Umwandlung (z. B. in Prozeßkontrollnetze) findet in einer eigenen Schicht statt und ist somit leicht austauschbar.

Abbildung 1



Aufbau der Arbeit

Zu Beginn untersuchen wir näher, was Workflows sind. Es werden zwei Modelle zur Beschreibung von Geschäftsprozessen vorgestellt: Das Modell der WfMC und zum Vergleich dazu das aspektbasierte Modell von MOBILE, das auch unserem weiteren Vorgehen zugrunde liegt. Danach führen wir den Begriff des Makros ein. Wir identifizieren dabei einen Grundstock an Makros, die zur Modellierung von Kontroll- und Datenflüssen vorhanden sein müssen, und zeigen, wie mit Hilfe dieser *Grundmakros* weitere Makros abgeleitet werden können.

Anschließend erweitern wir die Makroidee um komplexere Gebilde. Wir konstruieren *Hybridmakros*, die ein Zusammenwirken von Daten- und Kontrollfluß ermöglichen. Außerdem betrachten wir die Möglichkeit des „Late Binding“ [Ni99] bei *polymorphen Makros*, d. h., Teile des Ablaufs werden dynamisch zur Laufzeit ausgewählt.

Um Makros beschreiben zu können, führen wir eine geeignete Definitionssprache ein. Es werden Mechanismen zur Parameterübergabe vorgestellt und insbesondere die Problematik einer variablen Parameteranzahl behandelt.

Definierte Makros sollen natürlich auch ausgeführt werden können. Zur Implementierung setzen wir *Prozeßkontrollnetze (PCN)* ein, eine Variante der Petri-Netze. Diese werden in Kapitel 6 ausführlich vorgestellt.

Nach diesem Ausflug in die Welt der Netze beschreiben wir den Aufbau der eigentlichen Makrokomponente, die die Verwaltung bestehender Makros, aber auch das Ändern und Neuerzeugen von Makros unterstützt. Eine Ablaufkomponente ermöglicht die Ausführung vollständig definierter Makros.

Wir zeigen ferner, wie ein Compiler die in unserer Definitionssprache erstellten Makrodefinitionen in Prozeßkontrollnetze übersetzt. Die Umsetzung wird an zahlreichen Beispielen verdeutlicht.

Zum Abschluß folgt eine Zusammenfassung der Arbeit, in der die wichtigsten Punkte nocheinmal in aller Kürze rekapituliert werden.

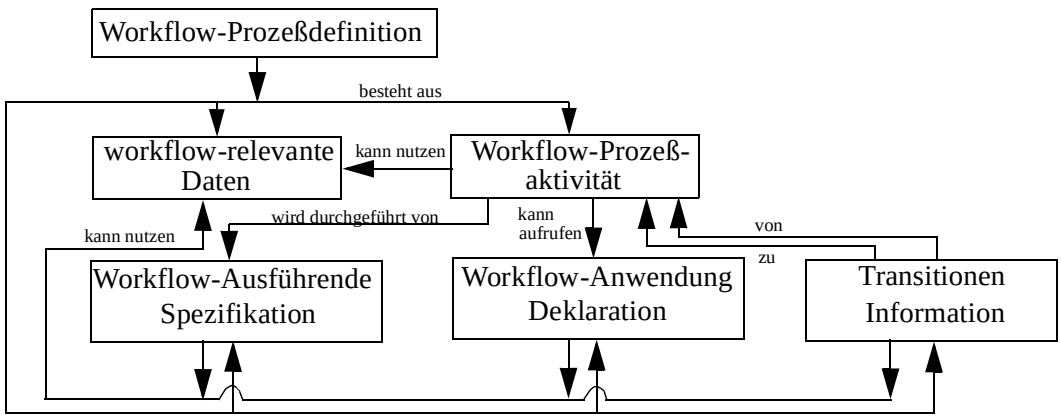
Kapitel 2 Workflows

Workflow-Management-Systeme dienen dazu, Geschäftsprozesse als Ablauf (*Workflow*) zu beschreiben und ihre Ausführung zu überwachen. Dazu müssen die Geschäftsprozesse in einer dem Workflow-Management-System verständlichen Art und Weise beschrieben werden. Die Workflow Management Coalition (WfMC) beschreibt eine *Prozeßdefinition* als *die Repräsentation eines Geschäftsprozesses in einer Form, die automatisierte Bearbeitung, wie z. B. Modellierung oder Ausführung durch ein Workflow-Management-System, zuläßt. Die Prozeßdefinition besteht aus einem Netzwerk von Aktivitäten und ihren Beziehungen zueinander, Kriterien, um den Anfang und das Ende des Prozesses zu erkennen, und Informationen über die einzelnen Aktivitäten, z. B. ausführende Personen, benutzte Anwendungen und Daten u. v. m.* [WfMC98].
Nachfolgend wird das von der WfMC eingeführte Metamodell für Workflow-Geschäftsprozeßdefinitionen erläutert. Anschließend besprechen wir zum Vergleich das von MOBILE vorgestellte Workflow-Modell, das in dieser Arbeit als Grundlage dient.

2.1 Metamodell der WfMC

Die WfMC identifiziert in [WfMC98] fünf grundlegende Einheiten zur Modellierung eines Workflows: Workflow-relevante Daten, ausführende Personen, eingesetzte Applikationen, Prozeßaktivitäten und Transitionen (Abbildung 2).

Abbildung 2 Metamodell zur Definition von Geschäftsprozessen (WfMC)



Die fertige Prozeßdefinition bildet eine Schablone, die, um einen realen Geschäftsprozeß durch das Workflow-Management-System ausführen zu lassen, zu einer **Prozeß-Instanz** erweitert werden kann.

Workflow-relevante Daten sind dabei *Daten, die von einem Workflow-Management-System benutzt werden, um Zustandsübergänge einer Workflow-Instanz, ..., Transitionsbedingungen oder die Auswahl eines Workflow-Ausführenden zu entscheiden* [WfMC99].

Aktivitäten sind definiert als *Beschreibung eines Arbeitsabschnitts, der einen logischen Schritt innerhalb eines Prozesses bildet. Eine Aktivität kann eine manuelle Aktivität sein, die keine Automatisierung durch den Computer zuläßt, oder eine (automatisierte)* ***Workflow-Aktivität****. Eine Workflow-Aktivität braucht menschliche und / oder maschinelle Ressourcen, um die Prozeßausführung zu unterstützen. Sind menschliche Ausführende notwendig, wird der Aktivität ein Workflow-Ausführender zugeordnet* [WfMC99].
Eine Workflow-Aktivität kann atomar sein. In diesem Fall stellt sie die kleinste Arbeitseinheit dar, die in einem Prozeß spezifiziert wird. Sie kann aber auch selbst wiederum ein Workflow sein (Sub-Workflow). Die Workflow-Aktivität bildet dann nur den Container für eine (separat definierte) Prozeßdefinition. Diese Definition enthält eigene Aktivitäten, Transitionen, Applikations- und Ressourcenzuweisungen. Der Austausch von Workflow-relevanten Daten mit dem Sub-Workflow ist über Ein- und Ausgabeparameter möglich.

Ein **Workflow-Ausführender** ist *eine Ressource, die die durch eine Instanz einer Workflow-Aktivität beschriebene Arbeit ausführt. Zugewiesen wird die Arbeit dem Ausführenden über einen oder mehrere Einträge in seiner Arbeitsliste* [WfMC99]. Arbeitslisten sind dem Ausführenden direkt zugeordnet und enthalten alle Arbeitsaufträge, die dieser durchführen soll.

Workflow-Anwendung ist eine *allgemeine Bezeichnung für ein Software-Programm, das mit der Ausführungskomponente interagiert und dabei einen Teil der Bearbeitung übernimmt, der zur Unterstützung einer bestimmten Aktivität notwendig ist* [WfMC99].

Transitionen sind *Punkte während der Ausführung einer Prozeß-Instanz, an denen eine Aktivität endet und der Kontrollfaden an eine startende Aktivität übergeben wird. Transitionen können* ***Transitionsbedingungen*** *besitzen, logische Ausdrücke, die die Reihenfolge der Ausführung von Aktivitäten innerhalb eines Prozesses festlegen können* [WfMC99]. Mit Hilfe von *SPLIT*- und *JOIN*-Bedingungen lassen sich Verzweigungen des Kontrollfadens modellieren, Iteration ist ebenfalls möglich.

2.2 Modell von MOBILE

Bei der Modellierung eines Prozesses kann man diesen aus verschiedenen Perspektiven betrachten [Ja95]:

- was wird ausgeführt *(funktional)*
- wann wird es ausgeführt *(verhaltensbezogen)*
- was wird benötigt *(informationsbezogen)*
- wer führt aus (*organisatorisch)*

Abläufe werden bei der Modellierung in Teilabläufe zerlegt. In [Ja95] definiert Jablonski Workflows als logische Verarbeitungseinheiten, die beschreiben, *was* ausgeführt werden soll. Sie können somit dem *funktionalen Aspekt* zugeordnet werden. Im folgenden werden der strukturelle Aufbau von Workflows näher betrachtet und einige für den späteren Teil der Arbeit wichtige Begriffe eingeführt, daran anschließend der *verhaltensbezogene*, der *informationsbezogene* und der *organisatorische* Aspekt kurz beschrieben.

2.2.1 Struktur von Workflows

Eine grundlegende Eigenschaft von Workflows ist ihr modularer, hierarchischer Aufbau. So kann ein Workflow selbst wieder aus vielen Einzel-Workflows bestehen, die ebenfalls in weitere „kleinere“ Workflows zerlegt werden können. Um die Rolle eines Workflows im Gesamtmodell zu beschreiben, werden folgende Begriffe zur verwendet:

Sub-Workflow: Workflows, die von anderen Workflows benutzt werden.

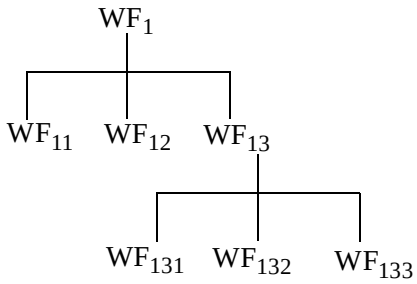
Super-Workflow: Workflows, die Sub-Workflows übergeordnet sind.

Toplevel-Workflow: Workflow, der selbst keinen Super-Workflow übergeordnet hat. Der Toplevel-Workflow ist somit die Spitze der „Zerlegungspyramide“.

Anhand von Beispiel 1 lassen sich diese Begriffe leicht verdeutlichen. WF_1 ist der Toplevel-Workflow. Er benutzt WF_{11}, WF_{12} und WF_{13} als Sub-Workflows und ist für diese Super-Workflow. WF_{13} wiederum hat WF_{131}, WF_{132} und WF_{133} als Sub-Workflows. Die Rolle eines Workflows ist somit flexibel, WF_{13} ist sowohl Super- als auch Sub-Workflow.

Neben der Unterscheidung in Super- und Sub-Workflows kann man Workflows nach ihrem Typ in *elementare* und *komposite* Workflows unterteilen. Elementare Workflows sind Workflows, die nicht weiter in andere Workflows zerlegt werden können, komposite Workflows hingegen benutzen Sub-Workflows. Diese können elementar oder selbst wieder komposit sein. In Beispiel 1 sind WF_{11}, WF_{12}, WF_{131}, WF_{132} und WF_{133} elementare, WF_1 und WF_{13} komposite Workflows.

Beispiel 1 Zerlegung eines Workflows in Sub-Workflows



2.2.2 Kontrollfluß

Der *verhaltensbezogene Aspekt* beschreibt den zeitlichen Ablauf der Ausführung, den Kontrollfluß. Eine Sprache zur Definition eines Workflows muß folglich Elemente zur Beeinflussung des Ablaufs besitzen. Dabei sind klassische Konstrukte wie Sequenz, bedingte Verzweigung und Ite-

ration vertreten, aber auch Erweiterungen wie Parallelausführung, Serialisierung oder 1-aus-n-Auswahl. Vor allem die Parallelausführung von Teilaufgaben verspricht eine deutliche Verbesserung des Zeitbedarfs. Allerdings müssen bei ihr auch Maßnahmen getroffen werden, die eine korrekte Ausführung garantieren, denn nicht alle Teilaufgaben sind auch wirklich parallel ausführbar. Vielmehr kann es zu Konflikten kommen, z. B. bei gemeinsamer Nutzung von Ressourcen.

2.2.3 Datenfluß

Im *informationsbezogenen Aspekt* wird der Weg der Daten durch den Workflow betrachtet. Unter Daten versteht man hier alle für den Ablauf relevanten Datenquellen, z. B. Akten, Dokumente oder Pläne, die auch in nichtelektronischer Form vorliegen können. In MOBILE werden diese als *Produktionsdaten* bezeichnet, um sie von den beim Kontrollfluß eingesetzen *Kontrolldaten* abzugrenzen. Im Datenfluß wird festgelegt, wie Dokumente zwischen verschiedenen Workflows ausgetauscht oder innerhalb eines Workflows weitergeleitet werden. Dabei kann es sich um ein reines Weiterreichen des Gesamtdokuments handeln, aber auch Kopieren, Aufspalten in Teildokumente oder Löschen des Dokuments (z. B. bei Notizen) ist möglich. Die Produktionsdaten sind dem Workflow-Management-System nur durch Repräsentanten bekannt, die Daten an sich stehen nur in den zu ihrer Verarbeitung zuständigen Anwendungen zur Verfügung. Die Repräsentanten können als *„workflow-relevante Produktionsdaten“* im Kontrollfluß integriert werden.

Wir haben uns in unserem Entwurf jedoch für eine möglichst strikte Trennung von Kontroll- und Produktionsdaten entschieden, Kontroll- und Datenfluß laufen getrennt und berühren sich nur an den dafür vorgesehenen Kreuzungspunkten.

2.2.4 Aufbauorganisation

Die Aufbauorganisation beschreibt den *organisatorischen Aspekt* eines Workflows. Es können dabei zwei Punkte unterschieden werden: die *organisatorische Struktur* und die *organisatorische Population*. Eine für Unternehmen typische Struktur ist dabei z. B. eine streng hierarchische Gliederung, die Population besteht aus den Mitarbeitern des Unternehmens. Eine beliebige Aufbauorganisation läßt sich entsprechend aus *organisatorischen Objekten* und *organisatorischen Beziehungen* modellieren, die frei miteinander kombiniert werden können.

Organisatorische Objekte werden weiter in Agenten und Nicht-Agenten unterteilt. Agenten sind dabei alle „faßbaren“ Elemente der Organisation, z. B. die Mitarbeiter und Maschinen des Unternehmens. Nicht-Agenten sind abstrakte Elemente wie Gruppen, Abteilungen oder Rollen.

Organisatorische Beziehungen definieren die Zusammenhänge zwischen organisatorischen Objekten. Typische Beziehungen sind z. B. „ist_Manager_von“, „ist_Mitarbeiter_von“, „nimmt_Rolle_ein“ u. v. m.

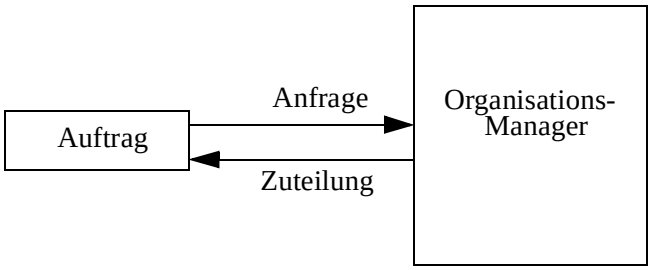
Organisatorische Objekte und Beziehungen werden durch charakteristische Eigenschaften beschrieben, die Zuständigkeiten oder Rechte näher beschreiben. So kann z. B. für einen Bankkassierer festgelegt werden, bis zu welchem Betrag er ohne Nachfrage bei seinem Vorgesetzten Auszahlungen vornehmen darf.

Für die Ausführung von Aufgaben sind die Agenten zuständig. Die Benachrichtigung der Agenten geschieht (wie bei WfMC) über Arbeitslisten. Der Agent selektiert einen Eintrag aus seiner Liste, entscheidet über die Ausführung und löst seine Aufgabe.

Nach der Definition der Organisation und ihrer Agenten bleibt noch zu lösen, wie eine Aufgabe einen geeigneten Agenten findet. Um z. B. ein Tischbein zu schnitzen, braucht man einen Schreiner, der Leiter des Verkaufs wäre sicherlich eine unglückliche Wahl. Daher werden *organisatorische Zuordnungsstrategien (policies)* spezifiziert, die Teile einer Organisation einem Workflow zuordnen. Die Zuordnungsregeln sind mit entsprechenden Prädikaten versehen, die eine situations- und kontextspezifische Zuordnung von Agenten zu Workflows ermöglichen.

In dieser Arbeit werden diese Zuordnungen von einem *Organisationsmanager* vorgenommen, der die Zuordnungsstrategie kapselt. Benötigt ein Auftrag Agenten, schickt er eine Anfrage an den Organisationsmanager, der ihm dann einen oder mehrere geeignete Agenten zuteilt (Abbildung 3)

Abbildung 3 Zuteilung einer organisatorischen Einheit durch den Organisationsmanager



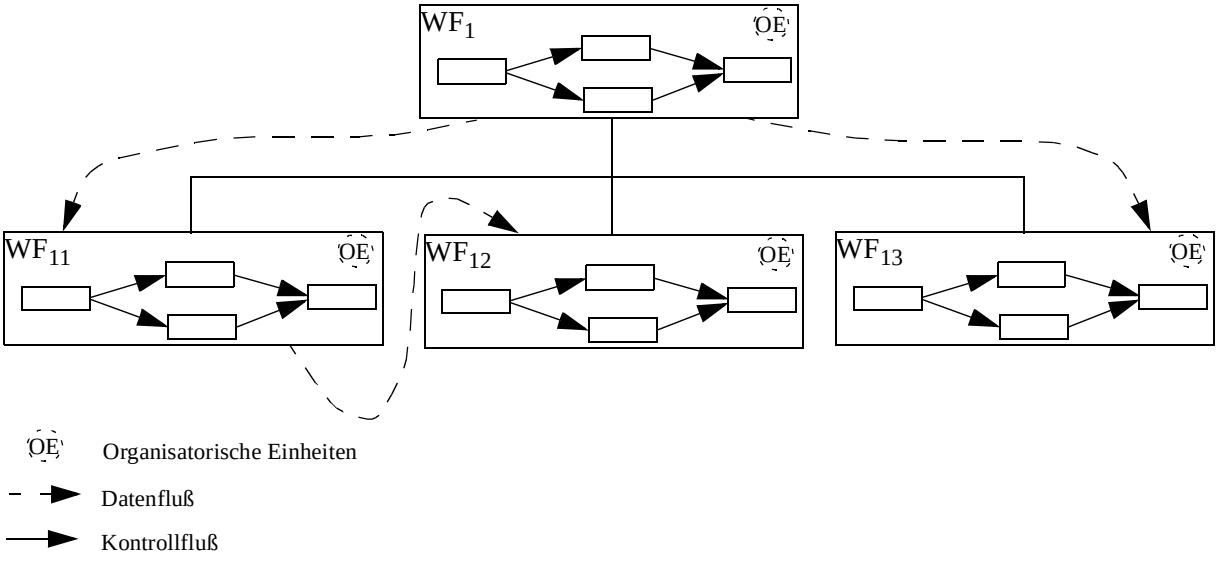
2.2.5 Abhängigkeiten zwischen Aspekten

In diesem Abschnitt wird untersucht, in welcher Beziehung die verschiedenen Aspekte zueinander stehen, und inwiefern sie unabhängig voneinander modelliert werden können.

Der *funktionale Aspekt*, die Zerlegung des Gesamtablaufs in Teilabläufe (Workflows), gibt die grundlegende Struktur vor. Alle anderen Aspekte werden davon direkt beeinflußt. So beschreibt der Kontrollfluß den zeitlichen Ablauf innerhalb der Einzel-Workflows, der Datenfluß den Informationsaustausch zwischen den Workflows. Jedem Workflow sind organisatorische Einheiten zugeordnet, die für die Ausführung der Einzelaufträge zuständig sind.

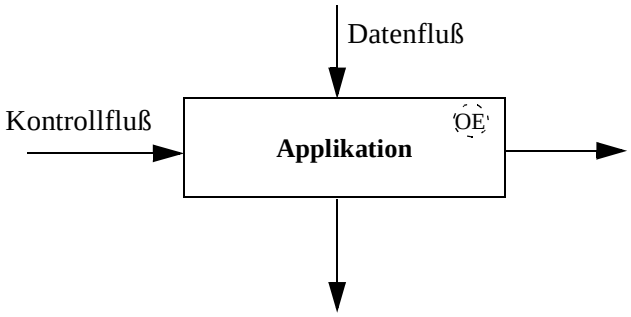
Diese drei Aspekte (verhaltensbezogen, informationsbezogen, organisatorisch) lassen sich weitestgehend unabhängig voneinander modellieren (Abbildung 4).

Abbildung 4 Beziehung verschiedener Aspekte zueinander



Die Zerlegung in Sub-Workflows endet in elementaren Workflows, in denen die Ausführung der Einzelaufträge stattfindet. Hier endet die Unabhängigkeit der Aspekte insofern, daß ein Zusammenführen von organisatorischen Einheiten, Kontroll- und Datenfluß notwendig wird. Nur wenn alle Informationen zusammen vorliegen, kann der Auftrag durchgeführt werden. Dies geschieht in „Applikationen“ (Abbildung 5). Applikationen sind somit natürlicherweise die oben beschriebenen Kreuzungspunkte zwischen den Aspekten.

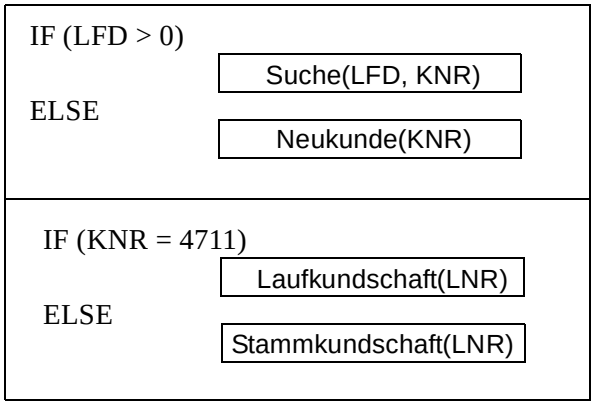
Abbildung 5 Applikation als Kreuzungspunkt verschiedener Aspekte



Kapitel 3 Makros

In vorigen Kapitel wurde der strukturelle Aufbau von Workflows aus Sub-Workflows beschrieben, die wiederum selbst aus Sub-Workflows bestehen können, bis diese Kette durch elementare Workflows abgeschlossen wird. Vergleicht man vor allem die einfachen Sub-Workflows miteinander, fallen schnell Ähnlichkeiten ins Auge. Bestimmte Konstrukte tauchen immer wieder auf, sie unterscheiden sich nur in Details. In Beispiel 2 ist ein Ausschnitt aus einem Workflow zu sehen, in dem zwei bedingte Verzweigungen hintereinander ausgeführt werden; dabei arbeitet der zweite Block mit dem Ergebnis des ersten Blocks, der KNR.

Beispiel 2 Ausschnitt aus einem Workflow mit bedingter Verzweigung



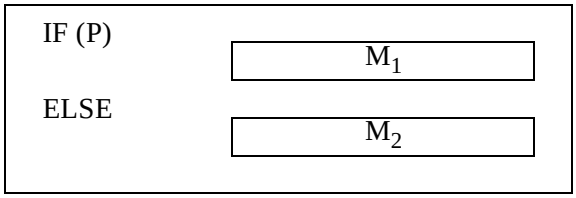
Die immer wiederkehrenden Teile der Deklaration kann man leicht extrahieren und als festes „Gerüst“ anbieten, das über Parameter an das jeweilige Problem angepaßt wird. Dieses Gerüst nennen wir *Makrorumpf*, die davon abgeleiteten Instanziierungen *Makros*. Makrorümpfe sind also flexibel nutzbare Workflow-Bausteine, in ihrer ausführbaren Form nichts anderes als Workflows selbst. Im Gegensatz zu einem Sub-Workflow repräsentiert ein Makro jedoch immer nur einen Aspekt, d. h. es beschreibt einen Kontrollfluß *oder* einen Datenfluß, während ein Sub-Workflow alle Aspekte abdeckt. Makros werden bei der Definition von Workflows benutzt, um die verschiedenen Aspekte sauber getrennt voneinander zu modellieren, Sub-Workflows dagegen dienen zur Strukturierung des Gesamt-Workflows.

In Abbildung 6 ist der Makrorumpf „bedingte Verzweigung“ zu sehen. Dabei sind auch schon die variablen Teile (das Prädikat P und die benutzten Makros M_1 bzw. M_2) als Parameter vorgesehen. Durch Binden dieser Parameter an konkrete Werte entsteht schließlich ein ausführbares Makro. In Beispiel 3 ist der Einsatz der beiden entsprechend instanziierten IF-Makros zu sehen. Eine zum Beschreiben von Makros und Makrorümpfen geeignete Definitionssprache wird in Kapitel 5 vorgestellt

Beispiel 3 Verknüpfung von Makros

Makros können in einer Bibliothek gesammelt werden, die alle bisher definierten Makros zur Verfügung stellt. Makros können so leicht wiederverwendet, bearbeitet oder zur Definition neuer Makros benutzt werden. Durch Einführung von hierarchischen Namen läßt sich auch eine Strukturierung dieser Bibliotheken und damit eine vereinfachte Wiederverwendung von Makros erreichen. Nach Abbildung in eine konkrete Implementierung können Makros mit Hilfe einer Laufzeitumgebung ausgeführt werden. In einem späteren Teil dieser Arbeit wird diese Abbildung mit Prozeßkontrollnetzen (Kapitel 6) durchgeführt.

Abbildung 6 Makrorumpf „bedingte Verzweigung“



Wiederverwendbarkeit ist eine zentrale Eigenschaft. Makros können hierarchisch aus anderen Makros aufgebaut werden. Wir unterscheiden dabei zwischen Grundmakros und abgeleiteten Makros. Grundmakros sind Makros, zu denen eine direkte Abbildung auf die Implementierungsebene existiert, abgeleitete Makros lassen sich mit Hilfe von Grundmakros aufbauen.

In Definition 1 sind die wichtigsten Makroarten aufgeführt. Grundmakros sind dabei nicht zu verwechseln mit elementaren Makros. Grundmakros können durchaus andere Makros benutzen und somit komposite Makros sein. (siehe Abschnitt 3.1). Bei diesen benutzten Makros kann es sich auch um abgeleitete Makros handeln. Das Makro „IF“ beispielsweise ist ein komposites Makro; da aber ein implementiertes Gegenstück existiert, ist es auch Grundmakro.

Definition 1 Makros

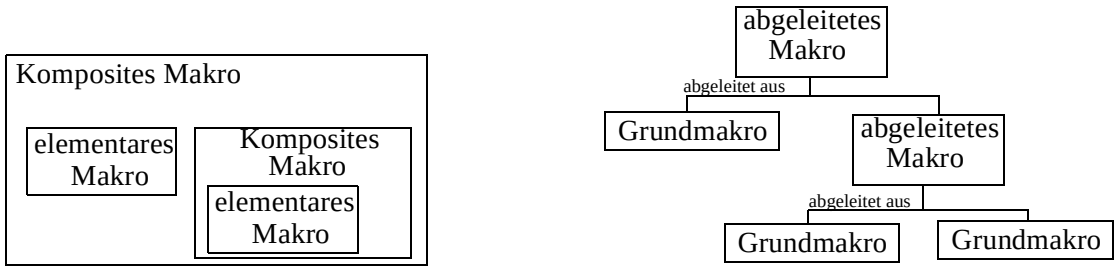
Makrorümpfe sind Workflow-Bausteine, die durch Angabe geeigneter Parameter zu Makros instanziiert werden können.
Makros sind instanziierte Makrorümpfe. Sie entsprechen einfachen, ablauffähigen Workflows.

Elementare Makros sind Makros, die keine anderen Makros als Submakros benutzen.
Komposite Makros sind Makros, die andere Makros als Submakros benutzen.

Grundmakros sind Makros, aus denen Nicht-Grundmakros abgeleitet werden können. Zu ihnen existiert eine direkte Implementierung oder Implementierungsvorschrift.
Abgeleitete Makros sind Makros, die aus Grundmakros und / oder abgeleiteten Makros aufgebaut werden können.

In Abbildung 7 ist dieser Sachverhalt noch einmal dargestellt. Grundmakros bilden einen Baustein-Grundstock, mit dessen Hilfe sich alle anderen Makros ableiten lassen. Bei der späteren Implementierung werden wir sehen, daß es genügt, nur die Grundmakros umzusetzen, deren Anzahl in unserer Betrachtung erfreulicherweise nicht sonderlich groß ist. Um gebräuchliche Kontrollflußkonstrukte zu definieren, genügen schon wenige Elemente wie bedingte Verzweigung, Iteration, Sequenz, Parallelität und Serialität.

Abbildung 7 Beziehung der Makrotypen zueinander



a) komposit / elementar b) Grundmakro / abgeleitetes Makro

Zusätzlich gewinnt man an Flexibilität: Makros lassen sich leicht austauschen. Bei einem Binden zum Ausführungszeitpunkt („Late Binding“ [Ni99]) kommt immer die neueste Version zum Zuge; Teilmakros müssen in diesem Fall erst zum Ausführungszeitpunkt vorhanden sein.

Im folgenden wird die Parameterübergabe an Makrorümpfe näher betrachtet, danach werden einige Grundmakros zur Modellierung von Kontroll- und Datenfluß vorgestellt, die anschließend zum Aufbau komplexerer Makros benutzt werden können.

3.1 Parameterübergabe

Um ein Makro zu instanziieren und deklarieren, benötigt man eine Reihe von Informationen: Wie sehen die Schnittstellen aus, gibt es mehrere Eingangs- oder Ausgangsschnittstellen (die wir

in Anlehnung an das WfMC als Transitionen bezeichnen), handelt es sich dabei um Datenfluß oder Kontrollfluß? Auch die einsetzbaren Parameter müssen näher beschrieben werden, z. B. der Datentyp des Parameters, oder bei Funktionen der erwartete Rückgabewert. Sollen andere Makros als Parameter benutzt werden, muß die Schnittstelle passen. Alle diese Angaben werden in einer besonderen Schnittstellendeklaration beschrieben, die wir in Anlehnung an die Programmiersprache JAVA™ [GJS96] als *Interface* bezeichnen. Bei der Definition eines Makrorumpfes, also dem Festlegen des Verhaltens eines Makros, werden diesem die benötigten *Interfaces* zugewiesen. Bei der Definition des Makros kann dann auf alle dort deklarierten Elemente zugegriffen werden. Diese Elemente sind im wesentlichen:

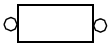
- Eingangstransitionen
- Ausgangstransitionen
- Typ der einsetzbaren Makros
- Rückgabetyp der einsetzbaren Funktionen
- Deklaration von Parametern einfachen Datentyps
- Deklaration von Kollektionen

Kollektionen dienen zur Verarbeitung einer dynamischen Anzahl von Parametern. Sollen z. B. mehrere (instanziierte) Makros parallel von einem Makro „PAR“ ausgeführt werden, so können diese als Kollektion übergeben werden; die Anzahl der Makros muß „PAR“ nicht vorher bekannt gemacht werden.

Verschiedene Makros können natürlich von ein und demselben Interface abgeleitet werden. So haben z. B. das schon angesprochene Makro „PAR“ und das Makro „SER“ zur serialisierten Ausführung von Makros die gleiche Schnittstellendeklaration. Beide Makros sind auch als Parameter geeignet, wenn dieser von eben diesem Interface-Typ sein soll.

3.2 Grundmakros

Wir stellen nun eine Sammlung einfacher Makros vor, mit deren Hilfe es möglich ist, die meisten gängigen Daten- und Kontrollflußkonstrukte nachzubilden. Makros können dabei auf einen bestimmten Typ festgelegt sein, sie sind z. B. nur als Kontrollflußmakros nutzbar. Makros können aber auch universell anwendbar sein. Das nachfolgende „leere Makro“ ist ein Vertreter dieser Sorte.

3.2.1 Das leere Makro 

Das leere Makro ist der einfachste Baustein. Es dient nur zum Verbinden von Elementen oder zum Überbrücken leerer Teile (z. B. einem leeren Zweig einer IF-Anweisung) und hat sonst keine weitere Funktion. Das leere Makro kann sowohl im Kontroll-, als auch im Datenfluß Verwendung finden.

3.2.2 Kontrollfluß

Der Kontrollfluß beschreibt, wann ein Teil des Gesamtablaufs gestartet werden soll. Zur Steuerung des Ablaufs sollen die bekannten Kontrollstrukturen wie Sequenz, bedingte Verzweigung (IF) oder Schleifen (REPEAT...UNTIL) zur Verfügung stehen, aber auch Erweiterungen wie die

Parallelausführung. Im nun folgenden Abschnitt werden eine Reihe von Makrorümpfen vorgestellt, die diese Funktionalität anbieten oder zum Aufbau eines entsprechenden komplexen Makros benutzt werden können.

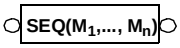
Kontrollflußvariablen

Um den Ablauf zu steuern, ist es natürlich notwendig, Kontrolldaten mitzuführen, die zum Auswerten von Prädikaten benutzt werden können. Für jedes Makro ist eine entsprechende Menge von Eingangs- bzw. Ausgangsvariablen durch die dem Makro zugeordnete Interface-Deklaration festgelegt.

Das Funktionsmakro

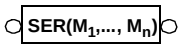
Funktionsmakros realisieren sehr einfache Funktionen wie z. B. das Inkrementieren eines Zählers. Dabei greifen sie nur auf Kontrolldaten zurück. Die entsprechende Erweiterung, um auch Produktionsdaten benutzen zu können, wird im nächsten Kapitel vorgestellt. Funktionsmakros sind elementare Makros.

Die Sequenz



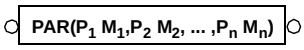
Die Sequenz (oder serielle Ausführung) ist die einfachste „echte“ Kontrollstruktur. Die Makros M_1 bis M_n werden in der Reihenfolge M_1, M_2, ..., M_n ausgeführt. Parameter für dieses Makro ist eine Kollektion mit den Makros M_1 bis M_n als Elementen.

Die Serialisierung



Die Serialisierung erlaubt die Ausführung von Makros in beliebiger Reihenfolge, jedoch immer noch sequentiell. Dies ist sinnvoll, wenn M_i und M_j aufgrund bestimmter Einschränkungen (Sicherheitsbestimmungen, Ressourcenkonflikt) nicht parallel ausgeführt werden können, obwohl sie im Ablauf voneinander unabhängig sind. Die Makros werden daher in der Reihenfolge M_i, M_j oder M_j, M_i nacheinander ausgeführt. Parameter ist eine Kollektion mit den Makros M_1 bis M_n.

Die Parallelität



Oft kommt es vor, daß Teile des Ablaufs voneinander unabhängig sind und parallel ausgeführt werden können; Parallelausführung trägt wesentlich zur Verkürzung der Gesamtausführungszeit bei und sollte daher möglichst oft eingesetzt werden. Die Ausführung eines Teilmakros kann zusätzlich von der Erfüllung eines Prädikats abhängig gemacht werden. Weiterhin wird garantiert, daß jedes Teilmakro genau einmal ausgeführt wird. Die Parallelität ist blockierend, d. h., die Makros M_i werden erst dann ausgeführt, wenn *alle* P_i erfüllt sind; die Ausführung wird dadurch synchronisiert.

Parameter für das Makro Parallelität ist eine Kollektion mit Elementtupeln (P_i, M_i), (i=1, ..., n). P_i sind dabei die Prädikate, M_i die zugehörigen Makros.

Die bedingte Verzweigung

Die Fallunterscheidung ist wie die Sequenz eine fundamentale Kontrollstruktur. Die bedingte Verzweigung (IF) erlaubt eine Steuerung des Ablaufs in Abhängigkeit des Prädikats P, d. h., es

besteht die Möglichkeit, den Ablauf kontextspezifisch zu beeinflussen. Ist Prädikat P erfüllt, wird Makro M_1 ausgeführt, ansonsten M_2.

Der variable Teil dieses Makros besteht aus dem Prädikat P und den Makros M_1 und M_2. Hier kann auch das leere Makro eingesetzt werden, um ein einfaches IF ohne (echten) ELSE-Zweig zu erhalten.

Die Iteration

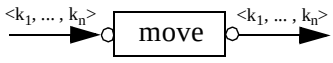
Schleifen sind in allen Programmiersprachen bekannt und bilden auch bei der Modellierung von Abläufen ein mächtiges Werkzeug. Ein Makro kann wiederholt ausgeführt werden, bis ein bestimmtes Abbruchkriterium erfüllt wird. Schleifen können kopf- oder fußgesteuert sein: Im ersten Fall wird die Bedingung *vor* der ersten Ausführung des Makros geprüft, im zweiten Fall danach. Als Grundmakro wird das fußgesteuerte REPEAT-UNTIL-Makro angeboten: Das Makro M wird ausgeführt, danach die Abbruchbedingung P geprüft. Dies wiederholt sich, bis P erfüllt ist. In Abschnitt 3.3 wird mit Hilfe dieses Makros (und der bedingten Verzweigung IF) auch die kopfgesteuerte Schleife erzeugt.

Parameter des Makros sind das Prädikat P und das Teilmakro M.

3.2.3 Datenfluß

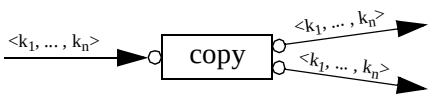
Der Datenfluß beschreibt den Weg der Daten bzw. Dokumente durch den Ablauf. Dabei handelt es sich um eine Erzeuger-Verbraucher-Beziehung. Die hier gezeigten Makros dienen dazu, Dokumente von den Erzeugern zu ihren Verbrauchern zu transportieren bzw. für die weitere Verarbeitung vorzubereiten (insbesondere Daten zu extrahieren oder zu neuen Dokumenten zu verschmelzen). Sie dienen nicht der anwendungsbezogenen Verarbeitung der Dokumente und Daten. Nachfolgend werden einige grundlegende Makros vorgestellt, $<k_1, ..., k_n>$ steht dabei für ein Dokument, das aus n Einzelkomponenten k_i besteht.

Verschieben



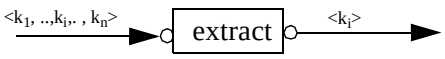
Das Makro **move** realisiert ein reines Weiterreichen des Dokuments. Das Dokument an sich bzw. seine Komponenten werden dabei nicht verändert.

Kopieren



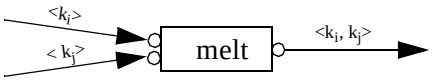
Es kann notwendig werden, mehrere Kopien eines Dokuments zu erzeugen und weiterzureichen. Dies wird mit Hilfe des Makros **copy** möglich. Denkbar ist z. B. die Kopie eines Auftragsdokuments, das einmal zum Archiv, aber gleichzeitig auch zum zuständigen Sachbearbeiter oder ausführenden Techniker muß.

Extrahieren



Ein Dokument enthält oft viele Daten, die der Verbraucher nicht benötigt oder die ihm (z. B. aus Datenschutzgründen) nicht zugänglich sein sollen. Daher wird das Makro **extract** angeboten, um einzelne Komponenten des Gesamtdokuments zur weiteren Verarbeitung auszuwählen. Beispielsweise kann die Kundennummer eines Kunden ausgewählt werden, während die Gesamtsumme aller an diesen Kunden gelieferter Waren verborgen bleibt.

Verschmelzen



Mit dem Makro **melt** kann aus verschiedenen Dokumenten ein neues Dokument zusammengesetzt werden. Wenn mehrere Erzeuger jeweils nur einen Teil des Gesamtdokuments produzieren, so kann dieses vor der weiteren Verarbeitung aus den Einzelkomponenten zusammengefügt und dem Verbraucher zugestellt werden.

3.3 Abgeleitete Makros

Mit den im vorigen Abschnitt eingeführten Makros lassen sich nun noch weitere nützliche Konstrukte bilden.

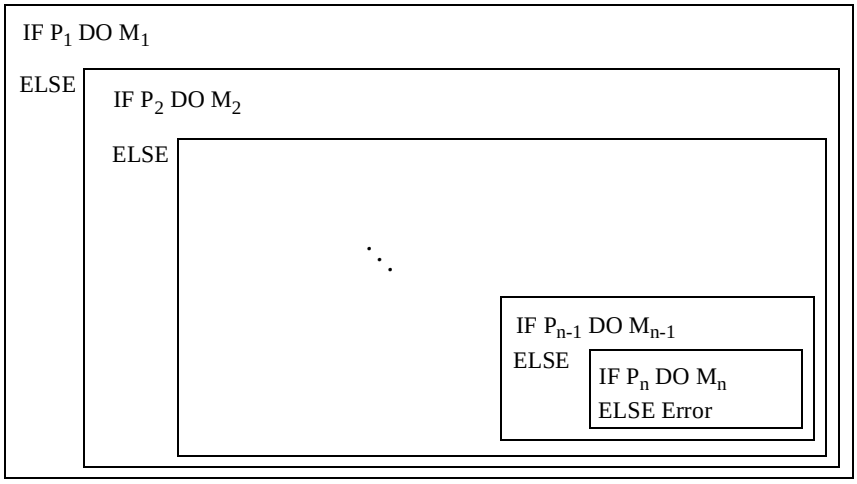
3.3.1 SWITCH (P_1 M_1, ... , P_n M_n)

SWITCH ist die Erweiterung des IF-Konstruktes auf n Zweige und bietet eine „1 aus n“ Auswahl an. Jeder Zweig wird dabei von einem Prädikat bewacht. Diese Prädikate P_i (i=1,...,n) werden der Reihe nach ausgewertet, sobald das erste Pj erfüllt ist, wird das zugehörige Makro M_j ausgeführt und die weitere Auswertung abgebrochen.

Das Makro SWITCH läßt sich über eine Schachtelung von n IF-Makros aufbauen, wie es in Abbildung 8 gezeigt wird. Parameter sind die Makros M_i mit den zugehörigen Prädikaten P_i (i=1,..., n), die als Kollektion übergeben werden.

Die Entscheidung, IF als Grundmakro zu wählen und SWITCH daraus abzuleiten, ist rein willkürlich. Genausogut funktioniert es, SWITCH als Grundmakro mit zugehöriger Implementierung anzubieten und IF als Spezialfall mit n=2 zu betrachten. Die Ausdrucksmächtigkeit bleibt in beiden Fällen gleich.

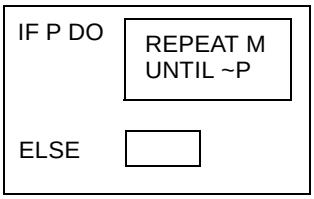
Abbildung 8 Aufbau des Makros SWITCH mit Hilfe des Makros IF



3.3.2 WHILE P DO M

Im Gegensatz zum REPEAT ist WHILE eine abweisende Schleife, das bedeutet, das Abbruchkriterium wird geprüft, bevor der Schleifenkörper zur Ausführung kommt. Die WHILE-Schleife läßt sich leicht mit IF und REPEAT nachbilden. Die abweisende Prüfung geschieht in einem IF, bevor die Schleife zur Ausführung kommt. Abbildung 9 zeigt das entstehende Konstrukt.

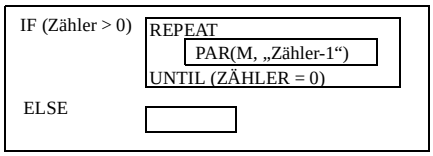
Abbildung 9 Eine WHILE-Schleife aus IF und REPEAT



3.3.3 DO n TIMES

Bei dieser Schleife handelt es sich um eine Zählschleife, d. h., der Schleifenrumpf soll n-mal ausgeführt werden. Bei jeder Ausführung des Rumpfes wird der Zähler dekrementiert, das Abbruchprädikat besteht folglich aus einer Abfrage, ob er noch positiv ist. Modellieren läßt sich die Zählschleife mit Hilfe der Makros IF, PAR und natürlich REPEAT. Parameter sind das zu wiederholende Makro M und der Zähler, initialisiert mit der Anzahl der gewünschten Wiederholungen.

Abbildung 10 Eine Zählschleife als komplexes Makro



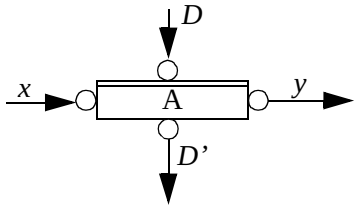
Kapitel 4 Erweiterung der Makroidee

In diesem Kapitel werden neue Makrotypen eingeführt, die eine Verknüpfung der verschiedenen Aspekte ermöglichen (Hybridmakros), bzw. eine dynamische Anpassung eines Ablaufs an sich ändernde Bedingungen erlauben (polymorphe Makros).

4.1 Hybridmakros

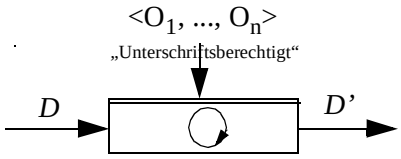
Bisher wurde beschrieben, wie mit Hilfe von Makros Kontroll- und Datenfluß modelliert werden können. Die verschiedenen Aspekte können jedoch nicht immer voneinander getrennt werden, es kommt zu Kreuzungspunkten. Um diesen Zusammenfluß von organisatorischem Aspekt, Daten- und Kontrollfluß zu ermöglichen, führen wir Hybridmakros ein.

4.1.1 Applikation



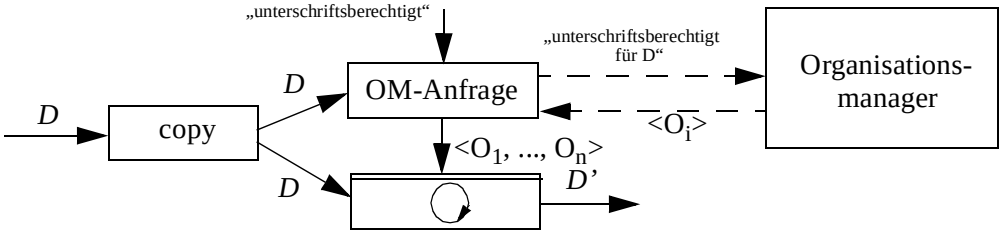
Applikation ist das wichtigste Hybridmakro. Mit den bisher vorgestellten Konstrukten ist noch keine wirkliche Verarbeitung möglich, die eigentliche Funktionalität erbringen jedoch (externe) Applikationen. Das können z. B. Computerprogramme sein, aber auch sogenannte „freie“ Applikationen, die der ausführenden Person die freie Wahl der Mittel lassen. Applikationen sind einerseits in den Kontrollfluß eingebettet, um festzulegen, wann sie ihre Tätigkeit aufnehmen sollen, andererseits benötigen sie zum Erfüllen ihrer Aufgabe das entsprechende Datenmaterial. Das Makro **Applikation** bildet die Schnittstelle, die diese Daten zur Verfügung stellt und die Ausführung anstößt. Die eigentliche Ausführung geschieht jedoch außerhalb der Kontrolle des Ablaufs. Das Workflow-Management-System hat somit auch keine Möglichkeit zum Eingriff, erst nach erfolgreicher Beendigung der externen Applikation erhält es die Kontrolle zurück. Denkbar ist auch, transaktionale Anforderungen innerhalb des Makros zu kapseln und so einen atomaren, konsistenten, isolierten und dauerhaft gültigen Verarbeitungsschritt zu erhalten.

4.1.2 Umlauf



Dokumente müssen oftmals von mehreren Personen bearbeitet werden, wobei die Reihenfolge der Bearbeitung keine Rolle spielt. Ein Vertrag, der mehrfach abgezeichnet werden muß, oder eine Informationsschrift, die jeder als gelesen kennzeichnen muß, sind Beispiele dafür. Das Hybridmakro **Umlauf** setzt genau dieses Verhalten um: Das Dokument D muß von den organisatorischen Einheiten O_i (i=1,...,n) bearbeitet werden und liegt danach zur Weiterbearbeitung bereit. Die organisatorischen Einheiten müssen zuvor durch eine Anfrage beim Organisationsmanager bestimmt werden (Beispiel 4). Dieser ermittelt die Menge der unterschriftsberechtigten Personen, die dann dem Umlaufmakro übergeben werden kann.

Beispiel 4 Anfrage bei Organisationsmanager



4.2 Polymorphe Makros

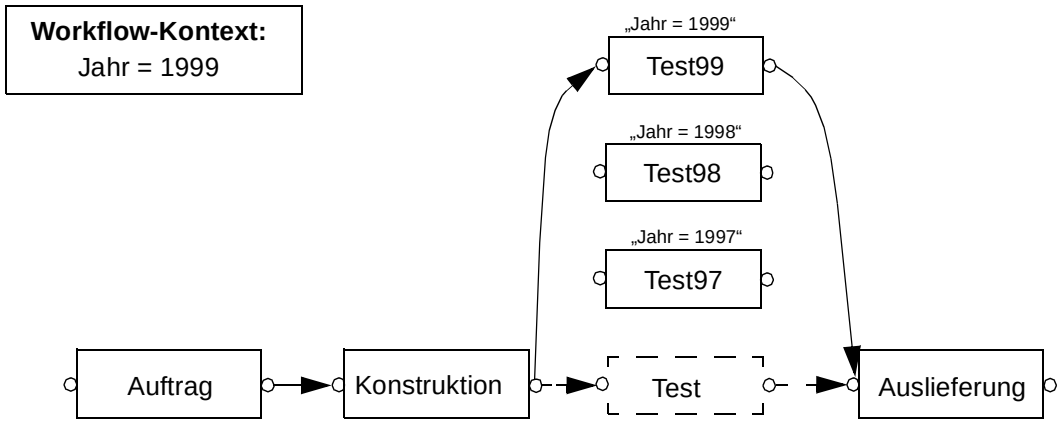
Beim Begriff Geschäftsprozeß denkt man im allgemeinen an Vorgänge, die schnell ausgeführt und schnell wieder beendet werden können. Es gibt jedoch auch Workflows, die über sehr große Zeiträume hinweg laufen. Die Entwicklung eines Produkts kann sich über mehrere Jahre erstrekken. An der Entwicklung eines Autositzes soll dies kurz demonstriert werden: Die Entwicklung kann in einem Workflow „Sitzentwicklung“ durchgeführt werden, der sich in die Sub-Workflows „Auftrag“, „Konstruktion“, „Test“ und „Auslieferung“ unterteilen läßt. Die Konstruktion eines Sitzes ist ein recht zeitaufwendiger Vorgang, bis zum Erreichen der Testphase können Jahre vergehen. Leider können sich in dieser Zeit auch die gesetzlichen Bestimmungen grundlegend verändern, im Test muß aber der aktuelle Stand berücksichtigt werden. Dies ist jedoch nicht möglich, wenn der Ablauf schon zum Startzeitpunkt fest definiert werden muß.

Um dieses Problem zu lösen, benötigen wir ein „Late Binding“, eine Zuordnung zum Ausführungszeitpunkt. Polymorphe Makros lösen dieses Problem. Zum Definitionszeitpunkt wird nur die Schnittstelle des Makros festgelegt, das Makro selbst wird als *dynamic* gekennzeichnet.

Zusätzlich werden ausführbare Makros definiert, die die Rolle des Makros übernehmen können, und Bedingungen formuliert, die die Auswahl eines passenden Makros ermöglichen. Zu diesem Zweck muß noch ein entsprechender Workflow-Kontext definiert werden. Existieren mehrere Makros, deren Prädikat erfüllt ist und die somit potentielle Ausführungskandidaten sind, so muß der Benutzer eingreifen und sich für ein Makro entscheiden. Gibt es kein Makro mit passender Auswahlbedingung, gibt es zwei Möglichkeiten: Entweder wird mit einer entsprechenden Fehlermeldung abgebrochen, es kann aber auch auf ein Defaultmakro zurückgegriffen werden, das extra für diesen Fall definiert wird.

Beim Autositztest ist es nun möglich, ein Makro mit den aktuell gültigen Anforderungen zu formulieren. Als Auswahlkriterium bietet sich in diesem Fall das Datum an. Kommt der Sitz in die Testphase, wird das aktuell gültige Testmakro ausgewählt und anstelle des dynamischen Makros „Test“ ausgeführt. In Beispiel 5 wird dieser Sachverhalt illustriert.

Beispiel 5 Ausführen eines polymorphen Makros



4.2.1 Workflow-Kontext

Um bei der Abarbeitung polymorpher Workflows (Makros) eine Zuordnung zur Laufzeit zu ermöglichen, wird eine Entscheidungsgrundlage benötigt, die mit Workflow-Kontexten eingeführt wird. Jeder Workflow ist mit einem Kontext assoziiert, der alle nach außen sichtbaren Informationen enthält. Von diesen sind für den Auswahlvorgang vor allem die dynamischen Informationen interessant: Aktuelle Werte von Workflow-Parametern, der Startzeitpunkt des Workflows, der ausführende Agent, u. a. Weiterhin kann über den Workflow-Kontext auch auf die aktuelle Systemumgebung (Betriebssystem, Hardwareplattform, vorhandene Applikationen) und den Kontext des Super-Workflows zugegriffen werden.

Einzelinformationen werden als atomare Werte abgelegt und mit Bezeichnern versehen, die Namensräumen zugeordnet sind, die wiederum Bezeichner besitzen. Es können somit mehrere

Namensraumstufen entstehen. Ein Bezeichner kann lokalisiert werden, indem die Bezeichner durch Punkte getrennt aneinandergehängt werden.

Vordefinierte Namensräume eines Workflows Wk sind Wk selbst, Wk.System (hier befinden sich die Informationen zur Systemumgebung) und Wk.Super (der den Kontext des Super-Workflows enthält). Einzelinformationen sind z. B. Wk.startTime oder Wk.System.OperatingSystem.Name.

Prädikate

Mit Hilfe des Workflow-Kontextes können nun Prädikate formuliert werden, die die Auswahl eines ausführbaren Makros bei „Late Binding“ ermöglichen. Prädikate bestehen aus einfachen Prädikaten, die miteinander logisch verknüpft werden. Einfache Prädikate sind in runde Klammern eingeschlossen. Sie bestehen aus binären Vergleichsoperationen von Einzelinformationen untereinander oder mit Konstanten. Ebenfalls einfache Prädikate sind Existenzabfragen auf bestimmte Bezeichner. Als Ergebnis wird immer einer der Boolschen Werte TRUE oder FALSE geliefert. Wird beim Auswerten eines einfachen Prädikats auf einen nichtexistenten Bezeichner zugegriffen, so wird es zu FALSE ausgewertet. Das Prädikat in Beispiel 6 dient z. B. zur Auswahl eines Makros, das einen Test mit den von 1999 bis 2000 gültigen Normen durchführt.

Beispiel 6 Ein Prädikat auf dem Workflowkontext „Test“

```
((Test.Role = „SitzTest“)
AND (Test.startTime >= „01.01.1999“)
AND (Test.startTime <= „31.12.2000“))
```

Die Zuordnung des Prädikats zum auszuwählenden Makro findet mit Hilfe der Makrodeklaration aus Abschnitt 5.1.4 statt.

Probleme

Die dynamische Bindung eines Makros beschert einen gravierenden Nachteil: es kann nicht garantiert werden, daß zum Ausführungszeitpunkt auch wirklich ein ausführbares Makro existiert, dessen Prädikat erfüllt wird und das somit zugeordnet werden kann. Dies kann aber mit einer unbedingten Bindung umgangen werden, einem Makro, dessen Prädikat auf jeden Fall zu TRUE ausgewertet wird.

Es kann auch vorkommen, das mehrere Makros die aktuellen Bedingungen erfüllen. In diesem Fall kann der Benutzer entscheiden, welcher Ausführungskandidat am geeignetsten erscheint und ausgeführt werden soll.

Weiterhin macht die Wiederholbarkeit des Ablaufs Probleme. Da Prädikate auch Zeitinformationen tragen und stets neue Bindungen neu aufgenommen werden können, kann bei der Wiederholung die Auswahl der in Frage kommenden Bindungen vollkommen verschieden sein. Die Ausführung muß folglich komplett mitprotokolliert werden, eine Wiederholung ohne dynamische Bindung stattfinden.

Kapitel 5 Die Definitionssprache

Die in Kapitel 3 und 4 eingeführten Elemente sollen in einfacher Weise beschrieben werden können. Dazu führen wir nun eine geeignete Definitionssprache ein.

5.1 Interface

Um ein Makro benutzen zu können, sind verschiedene Informationen zur Instanziierung nötig. Dazu gehören Eingangs- und Ausgangstransitionen, Typen der einsetzbaren Makros, Funktionen oder auch Daten. Sollen verschiedene Bausteine trotz unterschiedlicher Instanziierung gemeinsame Eigenschaften haben, so ist die Definition eines „abstrakten Makros“ notwendig, das diese Eigenschaften festlegt. Mit Hilfe von Interfaces[1] läßt sich eine solche gemeinsame Schnittstelle für Makros definieren.

5.1.1 Deklaration

Die Deklaration eines Interface geschieht mit Hilfe des Schlüsselwortes „INTERFACE“, gefolgt von „NEW“ (um auszudrücken, daß es sich um eine Neudeklaration handelt) und dem Namen des Interface. Anschließend folgt die Deklaration der zugehörigen Transitionen und Parameter, wie in den Abschnitten 5.1.3 und 5.1.4 beschrieben. Beispiel 7 zeigt die Deklaration eines Interface für Makros mit variabler Parameterzahl.

Beispiel 7 Ein Interface für Makros mit variabler Parameterzahl

```
INTERFACE NEW     DynamicParmInterface;
IN                Eingang [CF, DOMAIN Token1];
OUT               Ausgang[CF, DOMAIN Token2];
COLLECTION        Zweige (FUNC p, MACRO m IS MacroInterface);

END;
```

1. Inspiriert wurde diese Idee durch das Java-Konstrukt „Interface“

5.1.2 Vererbung

Ein Interface kann von einem oder mehreren schon deklarierten Interfaces abgeleitet werden. Es erbt dabei alle Elemente des „Super-Interface“ und übernimmt die dort angegebenen Bezeichner. Diese Elemente stehen direkt zur Verfügung und müssen nicht explizit in der abgeleiteten Interface-Deklaration aufgeführt werden. Umbenennung ist optional mit Hilfe des Schlüsselworts „CORRESPONDS“ möglich. In Beispiel 8 wird die von „DynamicParmInterface“ geerbte Eingangstransition von „Eingang“ in „incoming“ umbenannt, Typ und Domänendefinition bleiben unberührt.

Beispiel 8 Umbenennung von Parametern mit CORRESPONDS

```
INTERFACE NEW par_interface IS DynamicParmInterface dpi;
IN      dpi.incoming CORRESPONDS Eingang;
OUT     ...
```

Erbt ein Interface von mehreren anderen Interfaces, so kann es zu Namenskonflikten kommen. Diese müssen explizit durch Umbenennung eines oder mehrerer Elemente mittels „CORRESPONDS“ aufgelöst werden. Diese Umbenennung gilt jedoch nur für das neue Interface. Ein von diesem abgeleitetes Makro hat aber auch den Makrotyp von allen Super-Interfaces geerbt und kann natürlich auch als entsprechender Parameter eingesetzt werden. Für diesen Fall muß der ursprüngliche Transitionsname weiterhin zur Verfügung stehen, die Umbenennung muß zurückverfolgbar bleiben!

5.1.3 Transitionen

Makros werden mittels Transitionen verbunden. Im Interface werden entsprechend Eingangs- und Ausgangstransitionen festgelegt, und ob es sich dabei um eine Datenfluß- oder Kontrollflußverbindung handelt. Transitionen sind nicht an bestimmte Typen gebunden. Daher muß zusätzlich sichergestellt werden, daß nur Makros miteinander verbunden werden, die auch zueinander passen. Dazu wird ein *Tag* mit einem Typeintrag benutzt, um die Typkonsistenz zu gewährleisten. Zwei Makros können nur miteinander verbunden werden, wenn sie den gleichen Eintrag besitzen. Eine Ausnahme bilden Makros, die problemlos mit allen anderen Makros kombiniert werden können (wie z. B. das leere Makro). Solche Makros werden durch einen leeren Typeintrag gekennzeichnet.

Beispielsweise können Tags benutzt werden, um Makros bestimmten Bereichen zuzuordnen. Ein Makro mit Tag „Buchhaltung“ kann nicht mit einem Makro mit Tag „Produktion“ zusammengefügt werden. So wird verhindert, daß Makros aus verschiedenen Bereichen versehentlich kombiniert werden, deren Zusammenfügen keinen Sinn haben kann.

Zusätzlich wird für jede Transition eine Domäne festgeschrieben und somit der Definitionsbereich, der Daten, die an dieser Stelle dem Makro übergeben werden können. Domänen beschreiben Mengen, deren Elemente einem bestimmten Typ entsprechen. Die Anzahl der an eine Transition übergebenen Daten ist nicht beschränkt.

Sollen zwei Makros verbunden werden, muß natürlich die Domänendefinition der Verbindungstransition in beiden Makros verträglich sein. Eine besondere Rolle spielen „don’t care“-Domä-

nen (symbolisiert durch einen „*“). Spielt für eine Transition die Domäne der Verbindungstransition keine besondere Rolle, so kann diese als „don’t care“ deklariert werden. Solche Transitionen sind mit allen in Frage kommenden Verbindungstransitionen verträglich.

In Beispiel 7 werden zwei Kontrollflußtransitionen deklariert: Eingang und Ausgang. Das Schlüsselwort „IN“ legt dabei Eingang als Eingangstransition, „OUT“ Ausgang als Ausgangstransition fest. „CF“ identifiziert sie als Kontrollflußtransitionen; „Token1“ und „Token2“ sind an anderer Stelle definierte Domänen.

Es können selbstverständlich auch mehrere Eingangs- und Ausgangstransitionen von gleichem oder verschiedenem Typ deklariert werden. Ein Interface für ein Hybridmakro besitzt z.B. zusätzlich Transitionen für den Datenfluß. Transitionen können auch mit Status versehen werden. Die Deklaration

IN: Eingabe [DF, DOMAIN Doc1] WITH EMPTYSTATUS empty;

erzeugt eine lokale Variable „empty“, die genau dann erfüllt ist, wenn noch Eingabewerte an der Eingangstransition „Eingabe“ anliegen. Die Verwendung lokaler Variablen wird in Abschnitt 5.2.2 näher erläutert.

5.1.4 Parameter

In der Interface-Deklaration wird ebenfalls festgelegt, wie die Parameter zur Instanziierung eines Makros aussehen müssen. Parametertypen sind dabei Makros, Funktionen und Daten. Um eine variable Anzahl von Parametern zuzulassen wird zusätzlich das Konzept der „COLLECTION“ eingeführt.

Makros

In Kapitel 3 wird beschrieben, wie Makros hierarchisch aufgebaut werden können. Folglich benötigen wir Makros als Parameter, um komplexe Makros zu konstruieren. Im Interface werden nun die benötigten Makros mit dem Schlüsselwort „MACRO“ aufgeführt, erhalten einen Namen, mit dem sie bei der eigentlichen Makrodefinition angesprochen werden können, und werden mit Hilfe von IS einem Makrotyp (also wiederum einem Interface) zugeordnet. Falls für einen Makroparameter kein spezieller Typ gefordert wird (also jedes schon definierte Makro benutzt werden kann), kann man hierzu ein allgemeinstes Interface benutzen („MacroInterface“), das keinerlei Vorgaben besitzt. Die Deklaration eines Makroparameters kann also wie folgt aussehen:

MACRO m IS MacroInterface;

Zu beachten gilt, daß jedes als Parameter eingesetzte Makro wiederum selbst eine Parameterliste besitzt. Das „rufende“ Makro hat für die Bereitstellung dieser Parameter zu sorgen.

Polymorphe Makros

Die Deklaration eines polymorphen Makros erfolgt analog zur Definition eines normalen Makros, jedoch kennzeichnet das vorangestellte Schlüsselwort *dynamic* den Parameter als polymorph. Es ergibt sich eine Deklaration der Form:

DYNAMIC MACRO m_p IS MakroInterface;

Zu jedem polymorphen Makro müssen die „realen“ Gegenstücke existieren. Diese werden wie gewöhnliche Makros deklariert, jedoch mit dem Schlüsselwort *implements* dem jeweiligen Platz-

halter zugeordnet. Zusätzlich muß noch eine Auswahlbedingung $P_{Auswahl}$ formuliert werden, die festlegt, wann dieses Makro ausgewählt werden soll:

MACRO m_r IS MakroInterface IMPLEMENTS m_p FOR $P_{Auswahl}$;

Alle zur Auswahl nötigen Informationen werden (zur Laufzeit) im *Workflow-Kontext* abgelegt. Dabei handelt es sich um eine Sammlung von Laufzeitinformationen, die ein Bild davon vermitteln, unter welchen Bedingungen das Makro aktuell ausgeführt wird. Dabei kann es sich um völlig verschiedene Informationen handeln, wie z. B. das aktuelle Datum, das Betriebssystem oder der ausführende Agent. $P_{Auswahl}$ ist ein Prädikat, das mit Hilfe des Workflow-Kontextes ausgewertet werden kann. Soll nun ein polymorphes Makro m_p ausgeführt werden, werden für alle Makros, die dieses implementieren, die zugehörigen Auswahlprädikate geprüft. Das Makro, dessen Prädikat erfüllt ist, übernimmt dann die Rolle von m_p und wird ausgeführt. Eine nähere Beschreibung folgt in Abschnitt 7.5, der sich mit der Laufzeitumgebung beschäftigt.

Funktionen

Funktionen werden hauptsächlich zur Auswertung von Prädikaten eingesetzt, können aber auch beliebige andere Aufgaben übernehmen. In der Interface-Deklaration wird für jede benutzte Funktion ein Name und eine zugehörige Signatur festgelegt, etwa in der Form:

FUNCTION p OF BOOLEAN IS (x: REAL, y: INT);

p ist hier ein Prädikat mit zwei Parametern, das je nach Ergebnis der Auswertung TRUE oder FALSE liefert. Da die vollständige Signatur bei Deklaration des Interfaces nicht unbedingt bekannt ist, kann auch auf den IS-Zusatz verzichtet werden, nicht jedoch auf den Ergebnistyp. Die Deklaration vereinfacht sich dann entsprechend:

FUNCTION p OF BOOLEAN;

Die Funktion wird in diesem Fall erst später weiter verfeinert, wenn das tatsächlich ausführbare Makro instanziiert wird und die zugehörige Parameterliste bekannt ist.

Daten

Daten spielen als Parameter eher eine untergeordnete Rolle, trotzdem existieren Fälle, in denen sie sinnvoll sind. Daten sind in diesem Zusammenhang statische Daten zur Definition eines Makros, nicht zu verwechseln mit den Daten, die später das Netz durchfließen,

Eine Zählschleife erwartet z. B. als Parameter die Anzahl der Schleifendurchläufe in Form eines Int-Parameters. Denkbar ist auch das Festlegen von Obergrenzen, mit deren Hilfe z. B. Endlosschleifen verhindert oder Fehler erkannt werden können. Datenparameter können von jedem beliebigen Typ sein. Angeführt wird die Deklaration von dem Schlüsselwort DATA, danach folgt eine Liste von Namen und anschließend der Datentyp:

DATA i, j, k: INT;

Kollektionen

Kollektionen ermöglichen die Übergabe einer variablen Anzahl von Parametern. Sie verhalten sich wie Listen mit Parametertupeln als Elemente. Durch diese Tupelbildung wird auch die Bindung zueinandergehöriger Parameter beschrieben. Bei der Parallelausführung mit Hilfe des Makros „PAR“ soll z. B. ein Teilmakro nur ausgeführt werden, wenn ein zugehöriges Prädikat

erfüllt ist, Makro und zugehöriges Prädikat werden zu einem Tupel zusammengefaßt. In Beispiel 7 ist eine hierzu passende Deklaration zu sehen.

Um auf die einzelnen Elemente der Kollektion zugreifen zu können, sind zusätzliche Funktionen notwendig: „FIRST“ liefert das erste Elementtupel, „REST“ die Liste ohne das erste Element. „FOREACH“ iteriert über alle Elemente. Mit „COUNT“ läßt sich die Anzahl der Elemente bestimmen, das Prädikat „IS_EMPTY“ dient zum Erkennen der leeren Liste. Zusätzlich definieren wir einen Operator „.“, der den Zugriff auf Elemente eines Tupels ermöglicht.

5.2 Makrorümpfe

Interfaces beschreiben, mit welchen Parametern ein Makro instanziiert werden kann. Zusätzlich muß noch das Verhalten des Makros spezifiziert werden. Dies geschieht bei der Deklaration eines Makrorumpfes: In einer speziellen Programmiersprache wird festgelegt, wie die Parameter verarbeitet werden sollen. Die Bezeichnung Rumpf soll dabei ausdrücken, daß die Parameter noch keine festen Werte haben. Makrorümpfe sind vielmehr die instanziierbaren Bausteine, die unsere Makroidee inspiriert haben.

5.2.1 Deklaration

Ein Makrorumpf wird von einem oder mehreren Interfaces abgeleitet, die die Beschreibung der gewünschten Parameter bereitstellen. Die Zuordnung geschieht mit Hilfe des Schlüsselwortes „IS“. In Beispiel 9 wird ein Makrorumpf „Switch“ deklariert, der auf das Interface aus Beispiel 7 zurückgreift. „Switch“ besitzt folglich eine Eingangstransition, eine Ausgangstransition und eine Kollektion mit Prädikaten und ausführbaren Makros.

Beispiel 9 Deklaration des Makrorumpfes für SWITCH

```
MACRO   Switch IS DynamicParmInterface;

BODY    #IF (#EMPTY(Zweige) = #FALSE)
            #MACRO_USE IF(#FIRST(Zweige).p,
                          #FIRST(Zweige).m,
                          #MACRO_USE Switch (#REST(Zweige))
                              GIVING Eingang
                              TAKING Ausgang);
        #ELSE #EXEC Fehler;
        #END;
END.
```

5.2.2 Lokale Variablen

Bei der Definition des Rumpfs kann auf lokale Variablen zurückgegriffen werden. Diese besitzen wie Transitionen einen eindeutigen Namen, Typ (z. B. „CF“ oder „DF“) und eine Domäne. Angesprochen werden sie analog zu Transitionen mit Hilfe von „GIVING“ und „TAKING“.

Eine besondere Rolle spielen Statusvariablen. Eine Transition kann mit Status deklariert werden. „EMPTYSTATUS“ dient z. B. dazu, festzustellen, ob noch Eingabewerte anliegen. Statusvariablen sind ungetypt und domänenlos. Allein das Vorhandensein einer Marke entscheidet, ob der Status gesetzt ist oder nicht, er verhält sich somit wie ein Boolscher Wahrheitswert. Entsprechend kann der Status auch negiert abgefragt werden.

5.2.3 Definieren des Verhaltens

Das gewünschte Verhalten eines Makros wird im BODY-Abschnitt festgelegt. Im wesentlichen besteht dieser aus dem Aufruf ausführbarer Makros („#EXEC“) oder der „Instanziierung“ von Makrorümpfen („#MACRO_USE“). Zusätzlich stehen einige einfache Compileranweisungen zur Verfügung, die durch ein vorangestelltes „#“ gekennzeichnet werden. Dabei handelt es sich im wesentlichen um eine bedingte Verzweigung (#IF), Funktionen und Prädikate zum Abarbeiten von Kollektionen („#FIRST“, „#REST“, „#IS_EMPTY“, „#FOREACH“). In Beispiel 9 ist die Definition des Switch-Makros zu sehen, eine Besprechung folgt in Abschnitt 5.2.5.

Folgende Compileranweisungen stehen zur Verfügung:

1. #TRUE, #FALSE

   Diese beiden Konstanten repräsentieren die Boolschen Wahrheitswerte.

2. #IF (Prädikat) ... #ELSE ...# END

   Eine bedingte Verzweigung. Mit ihr kann, abhängig von einem zu #TRUE oder #FALSE auswertbaren Prädikat, die weitere Expansion des Makros gesteuert werden. Sie wird dazu benötigt, um rekursive Ausdrücke wie in Beispiel 9 abbrechen zu können.

3. #FOREACH

   Diese Anweisung dient dazu, jedem Element der Kollektion ein bestimmtes (strukturell gleiches) Verhalten zuzuweisen, einen eigenen Ausführungszweig. Um den richtigen Zweig auswählen zu können, wird ein Auswahlprädikat benutzt, das Teil der Kollektion sein muß. Mit „#ON“ wird das entsprechende Prädikat ausgewählt. Die Schlüsselworte „#BLOCKED“, „#EXCLUSIVE“ und „#ONCE_ONLY“ beschreiben, ob ein Zweig nur ausgeführt werden darf, wenn alle anderen Zweige ebenfalls ausgeführt werden können („#BLOCKED“), ob immer nur ein Zweig auf einmal aktiv sein darf („#EXCLUSIVE“), und ob ein Zweig erst dann wieder ausgeführt werden darf, wenn alle anderen Zweige ebenfalls abgearbeitet wurden („#ONCE_ONLY“). Eine ausführliche Besprechung folgt im nächsten Abschnitt..

4. #FIRST, #REST

   Mit „#FIRST“ wird das erste Element der Kollektion extrahiert, „#REST“ liefert die Kollektion ohne das erste Element. Diese beiden Funktionen ermöglichen das Abarbeiten von Kollektionen. Sie entsprechen den LISP-Konstrukten „CAR“ und „CDR“.

5. #IS_EMPTY

   Hierbei handelt es sich um ein Prädikat, mit dem festgestellt werden kann, ob eine Kollektion Elemente enthält. Die leere Kollektion liefert „#TRUE“, ansonsten wird „#FALSE“ zurückgegeben.

6. #EXEC

   Hiermit wird ein *ausführbares* Makro angesprochen. Falls das eingesetzte Makro mehrere Eingangs- oder Ausgangstransitionen besitzt, kann mit Hilfe von „GIVING“ bzw. „TAKING“ die richtige Transition ausgewählt werden. Es kann auch ein polymorphes Makro eingesetzt werden.

7. #MACRO_USE

   An dieser Stelle wird ein Makrorumpf mit den entsprechenden Parametern integriert.

8. GIVING, TAKING

   Ein Interface kann mehrere Eingangs- und Ausgangstransitionen besitzen. Wird ein davon abgeleitetes Makro als Parameter eingesetzt, so ist die Zuordnung der Verbindungstransitionen nicht eindeutig bestimmt. Sie kann aber mit Hilfe von „GIVING“ und „TAKING“ festgelegt werden „GIVING“ wählt dabei. eine Eingangstransition des Parametermakros aus, „TAKING“ eine Ausgangstransition.

   Zur Nutzung von lokalen Variablen wird die Syntax erweitert. Soll ein in einer Variablen abgelegter Wert an eine Transition weitergegeben werden, kann dies mit „GIVING Variable TO Eingangstransition“ erreicht werden; „TAKING Variable FROM Ausgangstransition“ legt entsprechend einen Ergebniswert in der Variablen ab.

5.2.4 Expansion von Makros mit variabler Parameteranzahl

In diesem Abschnitt wird näher betrachtet, wie die eingeführten Sprachkonstrukte aus dem vorhergehenden Abschnitt weiterverarbeitet werden und wie mit variabler Parameteranzahl verfahren wird. Dies geschieht anhand der Analyse der Makros „PAR“ zur Parallelausführung von Makros und „SER“ zu deren Serialisierung.

Die Parallelausführung von Teilmakros ist eines der wichtigsten Kontrollkonstrukte. Da Parallelität nicht nur mit zwei, sondern beliebig vielen Teilmakros genutzt werden kann, stehen wir vor dem Problem, eine im voraus unbekannte Anzahl Parameter verarbeiten zu müssen. Mit Hilfe der in 5.2.3 eingeführten Beschreibungselemente läßt sich dies lösen. „#FOREACH“ wird aufgelöst, in dem für jedes Element der Kollektion „Zweige“ ein eigener Ausführungsast erzeugt wird, der p_i als Auswahlprädikat benutzt und m_i (i = 1 ... n) ausführt.

Beispiel 10 Deklaration des Makros PAR (erste Version)

```
MACRO PAR IS DynamicParmInterface;
BODY    #FOREACH (Zweige)
            #ON p #EXEC m;
        #END
END.
```

Diese erste Version der Parallelität ist zwar für beliebig viele Teilmakros geeignet, erfüllt aber noch nicht ganz die Ansprüche, die an dieses Sprachkonstrukt gestellt werden: Jedes Teilmakro m_j kann sofort ausgeführt werden, sobald p_j erfüllt ist, ohne sich mit den anderen Zweigen syn-

chronisieren zu müssen. Sollen aber alle Teilmakros gleichzeitig ausgeführt werden, so muß die Ausführung entsprechend verzögert werden, bis alle Zweig-Eingangsbedingungen gleichzeitig erfüllt sind. Dies wird mit Hilfe des Schlüsselworts „#BLOCKED“ erreicht, das die Ausführung verhindert, bis alle Eingangsbedingungen erfüllt sind.

Nun taucht noch ein weiteres Problem auf: Sind alle Bedingungen erfüllt, kann jeder Zweig sofort ausgeführt werden. Bei einer Konkurrenzsituation kann nun ein Zweig mehrfach ausgeführt werden, ein anderer dafür gar nicht! Es muß also gewährleistet werden, daß jeder Zweig höchstens einmal durchlaufen wird, solange nicht alle Zweige abgearbeitet wurden. Hierzu dient das Schlüsselwort „#ONCE_ONLY“. Nach der Ausführung wird der Ausführungszweig gesperrt und somit eine wiederholte Ausführung verhindert. Diese Sperre wird erst wieder gelöst, wenn alle Teilmakros ausgeführt wurden und ein gültiges Ergebnis vorliegt.

Aus diesen Elementen läßt sich nun ein Makro zur Parallelausführung einer beliebigen Anzahl von Teilmakros zusammensetzen, „#BLOCKED“ und „ONCE_ONLY“ lassen sich konfliktfrei miteinander kombinieren. Die vollständige Deklaration ist in Beispiel 11 zu sehen.

Ein weiterer wichtiger Baustein ist die Serialisierung: Dabei können zwei oder mehr Makros in beliebiger Reihenfolge sequentiell ausgeführt werden. Wie bei der Parallelität ist die Anzahl der Teilmakros beliebig, jedes Teilmakro soll mit gleicher Wahrscheinlichkeit ausgeführt werden können, höchstens einmal und exklusiv. Das Schlüsselwort „#EXCLUSIVE“ ermöglicht genau dies.

Beispiel 11 Deklaration des Makros PAR

```
MACRO PAR IS DynamicParmInterface;

BODY    #FOREACH (Zweige)
            ON p #EXEC m #BLOCKED #ONCE_ONLY;
        #END

END.
```

Mit Hilfe von „#EXCLUSIVE“ und „#ONCE_ONLY“ kann nun das gewünschte Makro „SER“ wie in Beispiel 12 deklariert werden. Auch „#EXCLUSIVE“ und „#ONCE_ONLY“ lassen sich ohne Konflikte miteinander kombinieren.

Beispiel 12 Deklaration des Makros SER

```
MACRO PAR IS DynamicParmInterface;

BODY    #FOREACH (Zweige)
            ON p #EXEC m #EXCLUSIVE #ONCE_ONLY;
        #END

END.
```

5.2.5 Rekursive Deklarationen

Im letzten Abschnitt wurden Makros betrachtet, die eine variable Anzahl voneinander unabhängiger Parameter verarbeiten können. Es können aber auch Abhängigkeiten existieren. Ein Beispiel hierfür ist das „SWITCH“ Makro. Aus n Zweigen soll genau einer ausgewählt werden, die zugehörigen Prädikate sollen jedoch in einer bestimmten Reihenfolge überprüft werden. Die Deklaration aus Beispiel 9 leistet das Gewünschte. Die Kollektion „Zweige“ wird in diesem Fall wie eine lineare Liste behandelt. Das Prädikat „#IS_EMPTY“ dient zum Feststellen der leeren Liste und kann somit als Abbruchkriterium der Rekursion genutzt werden. „#FIRST“ liefert das erste Tupel in der Liste. Mit Hilfe des Punktoperators kann auf die einzelnen Elemente des Tupels zugegriffen werden. „#REST“ liefert die Liste nach dem ersten Tupel. Falls „Zweige“ nicht leer ist, die letzte Ersetzung also noch nicht durchgeführt wurde, wird ein Makro vom Typ „IF“ instanziiert: Das Prädikat des aktuellen Tupels in „Zweige“ wird dabei als Bedingung eingesetzt, das Makro m als auszuführendes Teilmakro, falls p erfüllt ist. Interessant ist der Aufbau der Alternative: Das Makro „SWITCH“ wird bei seiner eigenen Deklaration benutzt! Dadurch entsteht nacheinander eine Kette von verschachtelten „IF“-Makros, die die Prädikatauswertung in der richtigen Reihenfolge gewährleistet.

5.2.6 Verfeinerung

Datenflußmakros dienen zum Transport oder Kopieren ganzer Dokumente, aber auch zum Extrahieren von Einzelelementen oder dem Verschmelzen zu einem neuen Gesamtdokument. Während die ersten zwei Aufgaben keine großen Probleme bereiten und statisch abgebildet werden können, ist dies beim Extrahieren und Verschmelzen nicht ganz so einfach. Hier werden zusätzliche Informationen über die Beschaffenheit der Eingangs- und Ausgangsdokumente notwendig.

Gelöst wird dieses Problem mittels Verfeinerung der Transitionen. Ein Makro, zu dem eine Implementierung existiert, kann mit „REFINES“ angepaßt werden. Dabei können „don´t care“-Domänen umdefiniert werden. In Beispiel 13 wird ein Makro „protoExtract“ eingeführt, das eine „don´t care“- Eingangs- und eine „don´t care“-Ausgangstransition besitzt. Zu diesem Makro existiert auch eine entsprechende Implementierung. Das Makro Document1Extract überschreibt nun die „don´t care“-Transitionen mit den gewünschten neuen Domänen.

Beispiel 13 Verfeinerung von Transitionen

```
MACRO   protoExtract IS NEW protoExtractInterface;
IN:     Eingabe [CF, DOMAIN *];
OUT:    Ausgabe[CF, DOMAIN *];
END.


MACRO Document1Extract REFINES protoExtract;
IN:     Eingabe [CF, DOMAIN Document1];
OUT:    Ausgabe[CF, DOMAIN partOfDocument1];
END.
```

Zur Implementierung des neuen Makros wird eine Kopie des Vorlagemakros angefertigt und die Domänendefinition entsprechend angepaßt.

Kapitel 6 Petri-Netze

Um Makros ausführen zu können, müssen sie erst in eine ausführbare Form übersetzt werden. Wir haben uns in dieser Arbeit dazu entschieden, Makros in Prozeßkontrollnetze (eine Variante von Petri-Netzen) abzubilden, die hierfür recht geeignet scheinen. Zunächst folgt daher eine Einführung in Petri-Netze, anschließend wird deren Erweiterung zu Produktnetzen bzw. Prozeßkontrollnetzen (PCN) näher beleuchtet.

6.1 Petri-Netze

Petri-Netze dienen als Modell zur Beschreibung und Analyse von Abläufen und nebenläufigen, nicht deterministischen Vorgängen. Sie eignen sich hervorragend zur Beschreibung dynamischer Systeme mit fester Grundstruktur, aber auch dynamische Änderungen, wie sie z. B. polymorphe Makros erfordern, lassen sich damit modellieren.

6.1.1 Aufbau

Ein Petri-Netz ist ein gerichteter Graph mit zwei Sorten von Knoten, „Stellen“ und „Transitionen“. Eine Stelle dient dabei als Zwischenablage für Daten, in Transitionen findet die Verarbeitung statt. Dabei werden Eingangsdaten konsumiert und Ausgangsdaten neu erzeugt.

Definition 2 Petri-Netz

Ein *unbeschriftetes Petri-Netz* ist ein Tupel N = (S, T, F) mit

S endliche Menge von Stellen

T endliche Menge von Transitionen, $S \cap T = \emptyset$

$F \subseteq (S \times T) \cup (T \times S)$ eine Flußrelation; die Elemente von F heißen „Kanten“.

$f_i \in (S \times T)$ heißt „Eingangskante“,

$f_o \in (T \times S)$ heißt „Ausgangskante“ einer Transition.

Stellen, von denen Eingangskanten zu einer Transition $t \in T$ führen, heißen „Eingangsstellen“ von t, „Ausgangsstellen“ sind analog dazu Stellen, zu denen eine Ausgangskante von t führt.

Kanten können immer nur von einer Sorte zur anderen führen, eine Stelle ist also nie direkt mit einer anderen Stelle verbunden (dasselbe gilt natürlich auch für Transitionen). Definition 2 zeigt die formale Beschreibung eines solchen Netzes.

Stellen sind Objekte eines bestimmten Datentyps, z. B. BOOLEAN oder NAT. Die Objekte in den Stellen, auch als Marken bezeichnet, tragen selbst keine Informationen (daher können sie auch als einfache schwarze Punkte dargestellt werden). Kann eine Stelle nur zwischen den Zuständen „belegt“ und „nicht belegt“ unterscheiden, so hat sie den Typ BOOLEAN; wird auch noch die Anzahl der vorhandenen Marken berücksichtigt, hat sie entsprechend den Typ NAT.

Die „Markierung“ von Petri-Netzen beschreibt, welche Stelle aktuell mit wievielen Marken besetzt ist (Definition 3).

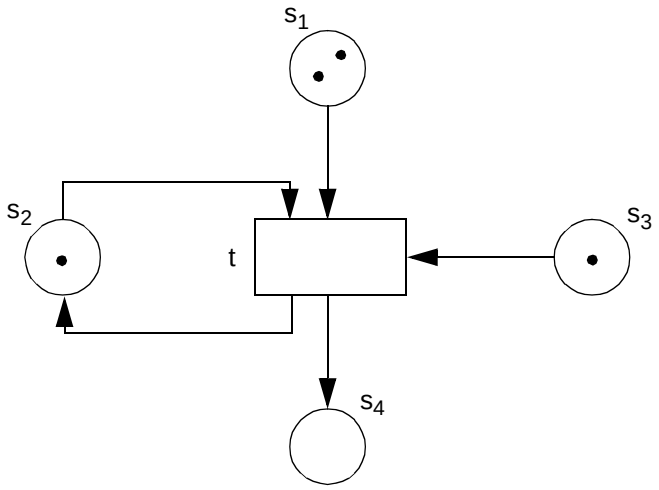
Definition 3 Markierung von Petri-Netzen

Die „Markierung“ von Petri-Netzen ist eine Abbildung M: $S \rightarrow NAT_0$. Sie gibt die Anzahl der Marken an, die jede Stelle enthält.

Diese Vorbetrachtungen und Definitionen sind in Beispiel 14 an einem konkreten Petri-Netz leicht nachzuvollziehen. Das Netz besitzt vier Stellen (s_1 bis s_4) und eine Transition t.

Die Kanten (s_1, t), (s_2, t) und (s_3, t) sind Eingangskanten von t, (t, s_2) und (t, s_4) Ausgangskanten. s_1, s_2 und s_3 sind folglich Eingangsstellen, s_2 und s_4 Ausgangsstellen von t. Die Markierung des Netzes sieht folgendermaßen aus: $M(s_1)=2$; $M(s_2) = 1$; $M(s_3)=1$; $M(s_4)=0$.

Beispiel 14 Markiertes Petri-Netz



$S = \{s_1, s_2, s_3, s_4\}$
$T = \{t\}$
$F = \{(s_1,t), (s_2, t), (s_3,t), (t,s_2), (t,s_4)\}$

$F_i = \{(s_1,t), (s_2, t), (s_3,t)\}$
$F_o = \{(t,s_2), (t,s_4)\}$

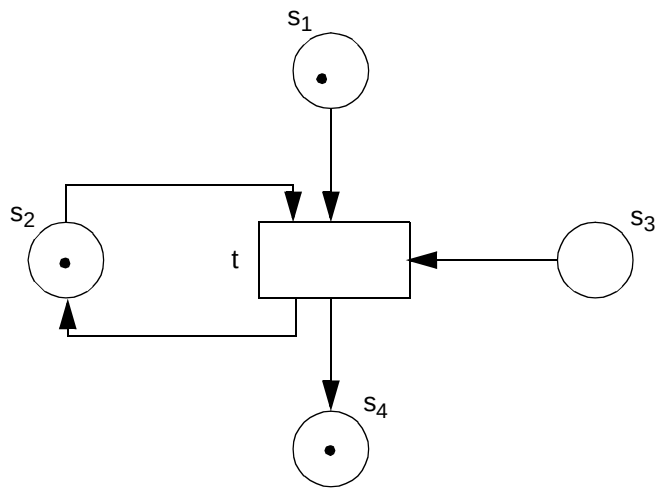
Petri-Netze sollen dynamische Systeme beschreiben. Die Dynamik entsteht im Netz durch „feuern“ von Transitionen. Eine Transition heißt „aktiviert“, wenn in jeder ihrer Eingangsstellen mindestens eine Marke liegt. Beim Feuern der Transition wird aus jeder Eingangsstelle eine Marke entfernt, in jede Ausgangsstelle eine Marke hinzugefügt. Die so entstandene neue Markierung heißt „Nachfolgemarkierung“.

Definition 4 Aktivierungsbedingung für Petri-Netze

Die Transition t ist *aktiviert*, wenn alle Eingangsstellen von t mindestens eine Marke enthalten.

Die Transition t aus Beispiel 14 ist aktiviert, da alle Eingangsstellen besetzt sind. Nach dem Feuern von t entsteht die in Beispiel 15 gezeigte Situation.

Beispiel 15 Petri-Netz nach Feuern der Transition t



6.1.2 Erweiterungen

In diesem Abschnitt werden zwei neue Kantentypen eingeführt, „Verbotskanten“ und „Abräumkanten“. Verbotskanten dienen dazu, eine Transition zu blockieren, sobald sich eine Marke in der zugehörigen Eingangsstelle befindet, Abräumkanten entleeren die Eingangsstelle komplett.

Verbotskanten

Verbotskanten sind spezielle Eingangskanten, die eine Transition am Feuern hindern, wenn sich in ihrer zugehörigen Eingangsstelle eine Marke befindet. Diese Stelle wird als „Verbotsstelle“ bezeichnet. Durch Einführen der Verbotskanten muß natürlich auch die Aktivierungsbedingung für Transitionen, wie in Definition 5, beschrieben erweitert werden.

Durch Einführen von Verbotskanten kann ein logisches NOT modelliert werden. Durch diese echte Erweiterung der Petri-Netze lassen sich alle TURING-berechenbaren Probleme modellieren (ohne Beweis).

Definition 5 Erweiterte Aktivierungsbedingung für Petri-Netze mit Verbotskanten

Die Transition t ist aktiviert, wenn

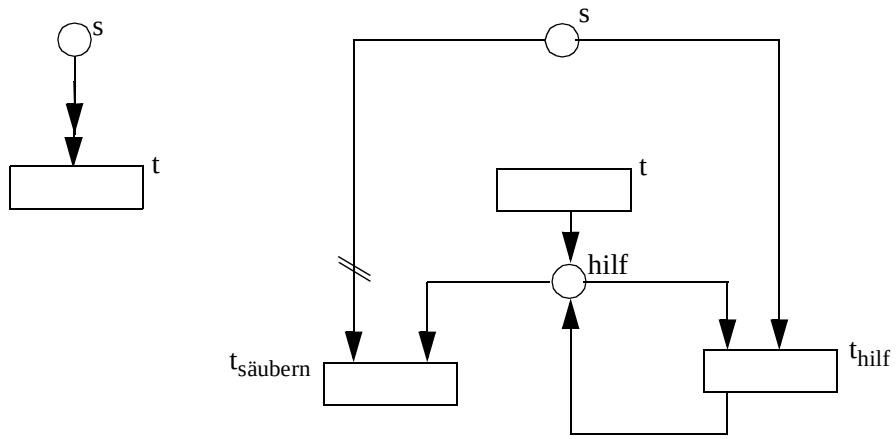
1. alle Eingangsstellen (nicht Verbotsstellen) von t mindestens eine Marke enthalten
2. in keiner Verbotsstelle von t eine Marke liegt.

Abräumkanten ⟶►

Abräumkanten sind spezielle Eingangskanten, die bei Feuern der zugehörigen Transition alle Marken aus der Eingangsstelle entfernen. Diese Eingangsstelle heißt „Abräumstelle“. Abräumkanten sind keine echte Erweiterung und dienen nur zur Vereinfachung der Modellierung.

In Abbildung 11 ist eine äquivalente Darstellung zu sehen. Feuert die Transition t, soll die Stelle s von allen Marken befreit werden. Im linken Bild geschieht dies mittels Abräumkante, im rechten Bild über eine Hilfsstelle mit zugehöriger Abräumtransition t_{hilf}. Feuert t, so wird eine Marke auf hilf abgelegt. Solange sich noch Marken auf s befinden, ist t_{hilf} aktiviert, die Marken werden nacheinander entfernt.

Abbildung 11 Simulation von Abräumkanten



Beispiel 16 zeigt ein Netz mit Verbotskante (s_3, t) und Abräumkante (s_2, t). Die Transition t ist aktiviert, da sich in der Eingangsstelle s_1 eine Marke befindet, nicht jedoch in der Verbotsstelle

s_3. Nach dem Feuern ist die Abräumstelle s_2 leer, die Ausgangsstelle s_4 hat eine zusätzliche Marke erhalten.

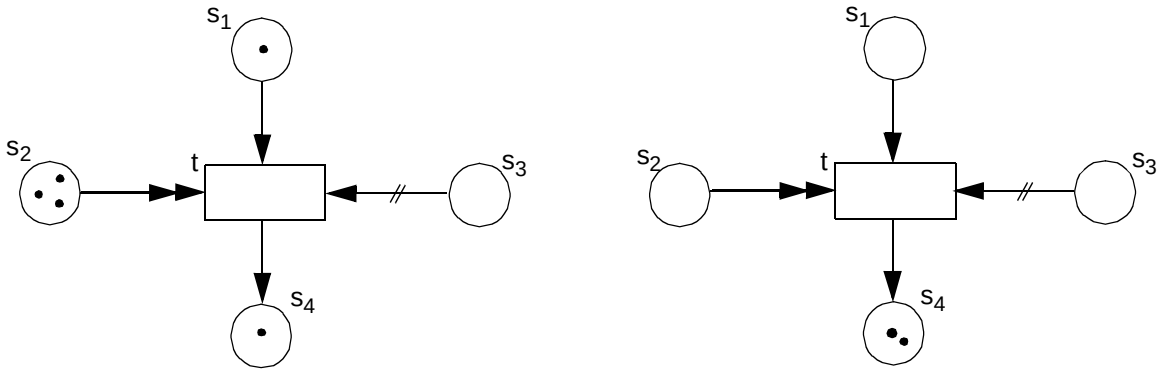
6.2 Produktnetze

Zu Petri-Netzen sind noch viele Erweiterungen denkbar, wie z. B. Marken mit Attributen, Kanten mit Gewichtsfunktionen, um mehrere Marken gleichzeitig fließen zu lassen, oder auch zusätzliche Transitionsbedingungen. Diese Erweiterungen finden sich z. B. in Produktnetzen, die in diesem Abschnitt näher beleuchtet werden.

Ein Produktnetz besteht aus drei Teilen:

1. einem Vorspann
2. einem unbeschrifteten Netz PN = (S, T, $F \cup V \cup A$)
   (V ist die Menge der Verbotskanten, A die Menge der Abräumkanten);
3. einer Beschriftung mit Kantenbeschriftungen für alle Kanten, Transitionsinschriften für Transitionen mit zusätzlicher Transitionsbedingung und der Festlegung von Definitionsbereichen für alle Stellen.

Beispiel 16 Ein Petri-Netz mit Verbots- und Abräumkante



6.2.1 Vorspann

Im Vorspann werden diejenigen Mengen definiert, die später den Definitionsbereich der Stellen bilden sollen. Mengen können Standardmengen sein (NAT_0), endliche Mengen (A = {a_1, ..., a_n}) und alle mit $A \cap B$, $A \cup B$ und $A \backslash B$ gebildetete Mengen. Ebenfalls im Vorspann werden auf diesen Mengen Funktionen definiert, die für Kantenbeschriftungen und Transitionsinschriften benötigt werden. Die Signatur wird wie gewohnt angegeben: $f: M_1 \times \ldots \times M_n \rightarrow M$

6.2.2 Markierung

Für jede Stelle s des Netzes wird ein Definitionsbereich festgelegt:

$$D_s = A_1 \times \ldots \times A_k \text{ bzw. } D_s = \{(x_1, \ldots, x_k) \in A_1 \times \ldots \times A_k | P\},$$

wobei für $k \geq 1$ A_1,..., A_k im Vorspann definierte Mengen sind, P ein Prädikat über (x_1, ..., x_k).

Von diesen Definitionsbereichen leitet sich auch der Name „Produktnetz“ ab. $A_1 \times \ldots \times A_k$ ist ein Produkt von Mengen, D_s eine Teilmenge dieses Produkts.

Eine Marke, die auf der Stelle s liegen darf, hat die Form $(a_1, \ldots, a_k)$ mit $(a_1, \ldots, a_k) \in D_s$, ist folglich ein k-Tupel. k nennt man „die Dimension der Stelle s“. In einer Stelle können auch mehrere gleiche Marken liegen.

Die Markierung einer Stelle des Produktnetzes ist analog zu der Markierung im Petri-Netz eine Abbildung, die angibt, wieviele (gleiche) Marken eine Stelle enthält. Definition 6 beschreibt diesen Sachverhalt: jedem Element des Definitionsbereichs wird eine natürliche Zahl zugeordnet, die angibt, wie oft dieses Element als Marke auf der Stelle s liegt.

Definition 6 Markierung einer Stelle

Die Markierung M_s einer Stelle s ist eine Abbildung $M_s: D_S \rightarrow NAT_0$

Alle Stellen des Produktnetzes haben eine solche Markierung. Zusammen bilden diese Markierungen die Markierung des Produktnetzes.

Definition 7 Markierung eines Produktnetzes

Die Markierung M eines Produktnetzes besteht aus einer Familie von Abbildungen M_s:

$M = (M_s)_{s \in S}$ mit $M_s: D_S \rightarrow NAT_0$

6.2.3 Beschriftung

Die Beschriftung eines Produktnetzes besteht aus Kantenanschriften an allen Kanten, sowie aus Transitionsinschriften, für die zusätzliche Schaltbedingungen existieren. Kantenanschriften dienen zur Zuordnung von Variablen zu den Definitionsbereichen der Stellen.

Kantenanschriften bestehen aus einer formalen Summe von n-Tupeln (+), die mit einer Vielfachheit v_i versehen werden können. An Verbots- und Abräumkanten dürfen n-Tupel nur die Vielfachheit 1 besitzen. Eine Kantenanschrift K der Kante f hat folglich die Form:

$$K(f) = v_1 \langle t_{11}, \ldots, t_{1n} \rangle + \ldots + v_r \langle t_{r1}, \ldots, t_{rn} \rangle$$

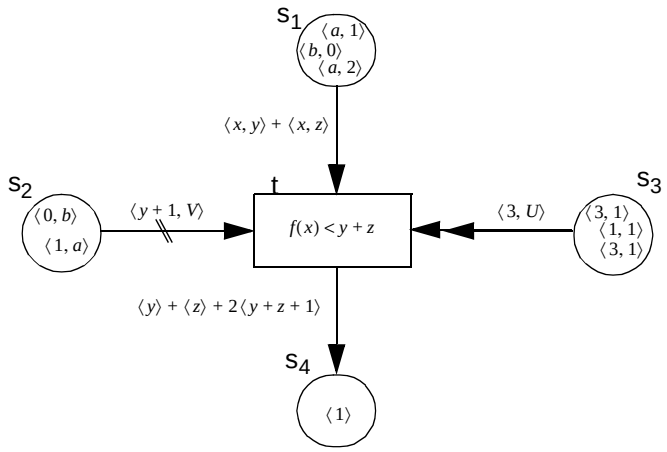
n ist dabei die Dimension der benachbarten Stelle, r eine beliebige natürliche Zahl; die t_{ij} sind Terme, die aus Konstanten, Variablen und Funktionen aufgebaut werden. Alle hierzu benutzten Funktionen müssen im Vorspann definiert werden, die Konstanten müssen aus im Vorspann definierten Mengen stammen.

Variablen gibt es in zwei Arten: gebunden und frei. Alle Variablen an den Eingangskanten der Transition bilden die Menge der bezüglich dieser Transition gebundenen Variablen. Alle Variablen in der Kantenanschrift einer Ausgangskante müssen gebunden sein, jede gebundene Variable muß mindestens einmal als einfacher Term auftauchen.
Freie Variablen treten nur an Verbots- und Abräumkanten auf. Freie Variablen in verschiedenen n-Tupeln müssen ebenfalls verschieden sein.

Transitionsinschriften sind Prädikate in den gebundenen Variablen einer Transition.

Beispiel 17 zeigt ein markiertes, beschriftetes Produktnetz. Im Vorspann sind zwei Mengen A und B definiert, die wir zum Aufbau der Definitionsbereiche und Funktionen nutzen können. Weiterhin wird im Vorspann die Funktion f definiert. Für jede der Stellen s_1 bis s_4 wird ein Definitionsbereich festgelegt. Jede Stelle enthält Marken aus ihrem Definitionsbereich.
Die gebundenen Variablen sind x, y und z, die freien Variablen U und V. Die Transition t hat eine Transitionsinschrift, ein Prädikat über die gebundenen Variablen. Die Transition darf nur feuern, wenn die Variablen x,y und z so belegt werden, daß f(x) < y + z erfüllt wird.

Beispiel 17 Ein markiertes, beschriftetes Produktnetz



Vorspann:

A = {a, b, c}
B = {0, 1, 2}

f: $A \rightarrow NAT_0$ mit f(a) = 1; f(b) = f(c) =0

Definitionsbereiche:

D_{s1} = A x B
D_{s2} = B x A
D_{s3} = NAT_0 x NAT_0
D_{s4} = NAT_0

6.2.4 Belegung

Eine Belegung ist die Abbildung der gebundenen Variablen in die entsprechende Wertemenge. In Beispiel 17 kann die gebundene Variable x alle Werte aus A (a, b oder c) annehmen, y und z ent-

sprechend die Werte der Elemente der Menge B (0, 1 oder 2). Für jede Belegung wird die Aktivierungsbedingung der Transition, auf die gleich noch näher eingegangen wird, geprüft. Beim Feuern der Transition bestimmt die Belegung (in Verbindung mit den entsprechenden Kantenanschriften), welche Marken in welcher Anzahl von einer Stelle entfernt bzw. auf einer Stelle abgelegt werden.

6.2.5 Aktivierungsbedingung

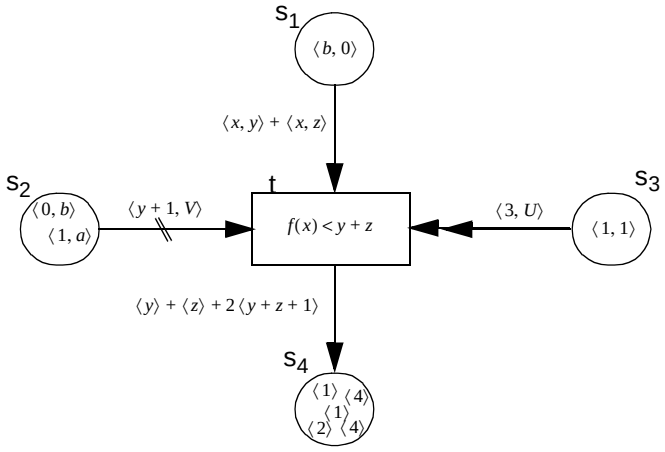
Eine Transition kann nur aktiviert werden, wenn die Belegung B drei Bedingungen erfüllt:

1. Eingangskanten: Für alle Eingangskanten muß gelten, daß den Tupeln aus B(KA) (die Kantenanschrift mit entsprechend B belegten Variablen) mindestens eine gleich große Anzahl entsprechender Marken in der Eingangsstelle gegenübersteht.
2. Verbotskanten: Durch die freien Variablen beschreibt die Kantenanschrift KA der Verbotskante eine Untermenge des Definitionsbereichs D_s der Verbotsstelle s. Die Transition darf nicht schalten, falls s eine Marke aus der Menge B(KA) enthält.
3. Transitionsprädikat: Die Belegung muß das Transitionsprädikat (das durch Anwenden von B auf die Transitionsinschrift erzeugt wird) erfüllen.

6.2.6 Feuern einer Transition

Erfüllt die Belegung B die Aktivierungsbedingung, so kann die Transition feuern. Für alle Eingangs- und Abräumkanten werden Marken entsprechend der Kantenanschriften aus den benachbarten Stellen entfernt. Für alle Ausgangskanten werden entsprechend Marken in die Ausgangsstellen eingefügt.

Beispiel 18 Produktnetz aus Beispiel 17 nach dem Feuern



Beispiel 18 zeigt das Produktnetz aus Beispiel 17 nach dem Feuern der Transition t. Dabei wurde B wie folgt gewählt: x = a; y = 1; z = 2. s_1 enthält genug Marken, um alle Gegenstücke in $B(KA_1)$

zu bedienen. Das Transitionsprädikat ist erfüllt, da f(a) = 1 < (1 + 2). Keine der Marken auf der Verbotsstelle s_2 hat die Form $\langle 3, V \rangle$, die Transition t kann also feuern. Dabei werden aus s_1 die Marken $\langle a, 1 \rangle$ und $\langle a, 2 \rangle$ entfernt, aus s_3 die beiden $\langle 3, 1 \rangle$ Marken. Auf s_4 kommen die Marken $\langle 1 \rangle$, $\langle 2 \rangle$ und zweimal $\langle 4 \rangle$ hinzu.

6.3 Prozeßkontrollnetze (PCN)

Prozeßkontrollnetze sind eine Variante der Produktnetze. Sie unterscheiden sich von letzteren durch den Umgang mit gebundenen und freien Variablen, einer veränderten Domänendefinition und dem Einführen „magischer Stellen“.

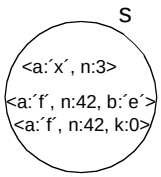
6.3.1 Vorspann

Um die Implementierung zu vereinfachen wird die Definition neuer Mengen im Vorspann aufgegeben. Zur Verfügung stehen nur noch die internen Datentypen des zugrundeliegenden Datenbanksystems. Zum Festlegen der Definitionsbereiche einer Stelle werden ausschließlich diese „flachen“ Mengen benutzt. Weiterhin müssen nicht alle benutzten Funktionen deklariert werden, einfache Ausdrücke können als *anonyme Funktionen* direkt in den Transitionsinschriften verwendet werden. Die Signatur einer Funktion muß nicht mehr vollständig sein, der Typ des Rückgabewerts muß jedoch weiterhin deklariert werden.

6.3.2 Definitionsbereich

Wie bei den Produktnetzen wird auch bei den Prozeßkontrollnetzen jeder Stelle S ein Definitionsbereich zugeordnet. Jedes Tupelelement erhält dabei einen Bezeichner. Im Gegensatz zu den Produktnetzen können Marken jedoch nicht nur k-Tupel (a_1, ..., a_k) sein, die dem Definitionsbereich der Stelle S entsprechen, sondern l-Tupel (a_1, ..., a_k, a_{k+1}, ..., a_l) mit l >=k. Dies bedeutet, auf einer Stelle können alle Marken liegen, die alle Elemente des Definitionsbereichs enthalten, egal, wieviel zusätzliche Information sie noch tragen. Bei Dokumenten interessiert beispielsweise nur der Teil des Tokens, der die Aufgabe eines „Header“ übernimmt. Somit können auf einer Stelle alle Dokumente liegen, die einen entsprechenden Header besitzen, die restlichen Daten können völlig unterschiedlich sein. In Beispiel 19 ist eine Stelle mit gültigen Marken zu sehen.

Beispiel 19 Stelle eines Prozeßkontrollnetzes mit Marken



D_S = a:CHAR x n:NAT_0

Zusätzlich gibt es eine weitere Art von Stellen ohne Definitionsbereich, die wir als *generische Stellen* bezeichnen. Bei diesen Stellen interessiert nicht, welche Marken auf ihnen liegen, sondern nur, ob Marken vorhanden sind. Benötigt wird diese Art Stelle, um einfache Statusabfragen durchführen zu können.

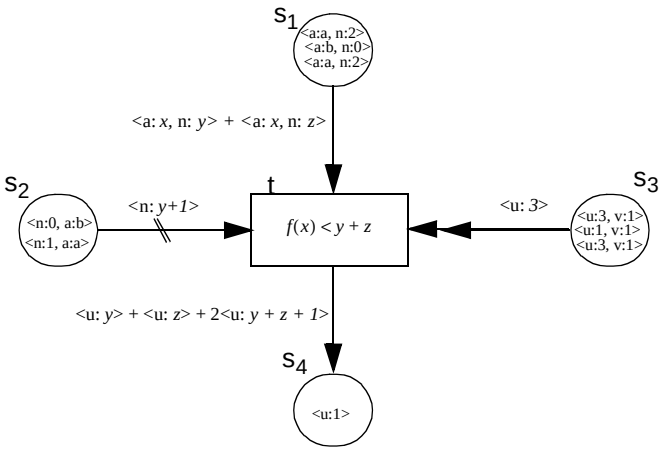
6.3.3 Markierung

Die Definition der Markierung einer Stelle bleibt wie in Definition 6 erhalten. Zu beachten gilt, daß nur die Tupelelemente des Definitionsbereichs betrachtet werden, d. h., die beiden Tupel <a:´f´, n:42, b:´e´> und <a:´f´, n:42, k:0> aus Beispiel 19 sind „gleich“, somit gilt für die Markierung der Stelle S: M_s(<a:´f´, n:42>) = 2.

6.3.4 Beschriftung

Bisher geschah die Bindung der Variablen einer Kantenanschrift an die Elemente der Marke über die Reihenfolge. Bei großem k trägt dies nicht unbedingt zur Übersichtlichkeit bei. Bei Abräum- und Verbotskanten brauchen wir freie Variablen, um nicht interessierende Positionen auszublenden.

Beispiel 20 Ein markiertes, beschriftetes Prozeßkontrollnetz



Vorspann:

f: $CHAR \rightarrow NAT_0$ mit f(a) = 1; f(b) = f(c) = ... = f(z) = 0

Definitionsbereiche:

D_{s1} = a: CHAR x n: NAT_0
D_{s2} = n: NAT_0 x a: CHAR
D_{s3} = u: NAT_0 x v: NAT_0
D_{s4} = u: NAT_0

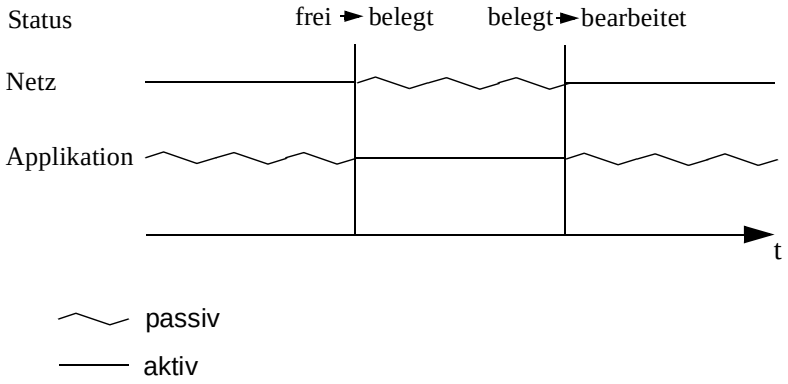
Prozeßkontrollnetze gehen einen anderen Weg: Über den Namen, den jedes Tupelelement beim Festlegen des Definitionsbereichs erhalten hat, ist die Zuordnung der Kantenanschrift möglich. In der Kantenanschrift können gebundene Variablen explizit über den Elementnamen zugeordnet werden. Dadurch ist es nun auch möglich, die Reihenfolge der Elemente in der Kantenanschrift zu ändern. Bei Verbots- und Abräumkanten entfallen die freien Variablen völlig.

In Beispiel 20 ist das Produktnetz aus Beispiel 17 nach Umwandlung in ein Prozeßkontrollnetz zu sehen. Im Vorspann werden nun bei Festlegung des Definitionsbereichs der Stelle die Namen der Tupelelmente festgelegt. Statt <a: *x*, n: *y*> kann nun auch völlig gleichbedeutend <n: *y*, a: *x*> geschrieben werden. An der Abräumkante taucht in der Bedingung <u: 3> keine freie Variable mehr auf, die Position von u innerhalb des k-Tupels und die Anzahl sonstiger Elemente ist für die Kantenanschrift nicht mehr von Bedeutung.

6.3.5 Magische Stellen

Magische Stellen dienen dazu, eine Verbindung zur Außenwelt herzustellen, d. h. in diesen Stellen können Daten außerhalb der Kontrolle des PCN verändert werden. Die Änderungen geschehen unsichtbar für das Netz und somit auf „magische“ Weise.

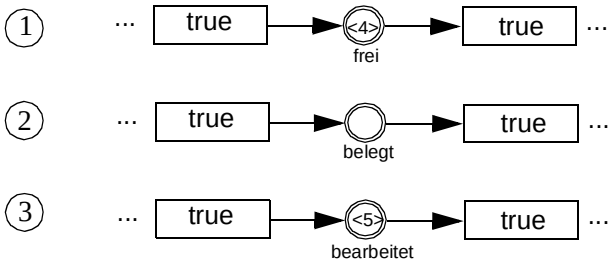
Abbildung 12 Wechsel der Kontrolle an einer magischen Stelle



Sobald ein neuer Auftrag (zur externen Bearbeitung vorgesehene Daten) vorliegt, kann mit der Bearbeitung begonnen werden. In dieser Zeit darf das Netz natürlich nicht weiterschalten, daher verfügt die magische Stelle über ein Zustandsflag. Dieses Flag kann die Werte „frei“, „belegt“ und „bearbeitet“ annehmen. Dabei bedeutet „frei“, daß die externe Anwendung ihre Arbeit beginnen kann. Bei “belegt“ ist die der magischen Stelle zugeordnete Applikation mit der Bearbeitung der Daten beschäftigt; „bearbeitet“ zeigt den Abschluß des Auftrags an, das Netz kann weiterschalten. In Abbildung 12 ist der zeitliche Verlauf zu sehen.

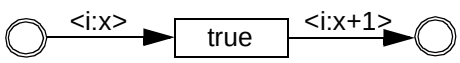
Beispiel 21 Aufruf eines ausführbaren Makros

.... #EXEC INC();



Auch der Aufruf ausführbarer Makros soll nun mit Hilfe von magischen Stellen durchgeführt werden. Beispiel 21 zeigt die Stelle eines PCN, an der ein Zähler inkrementiert werden soll. Dies soll mit Hilfe eines Makros INC(), das in ausführbarer Form vorliegt (Abbildung 13), durchgeführt werden.

Abbildung 13 Das Makro INC()



Ein zu bearbeitendes Datum (in diesem Fall das Token <4>) liegt auf der magischen Stelle, das Flag sitzt auf „frei“. Das zugeordnete externe „INC“ kann seine Arbeit aufnehmen und das Datum bearbeiten (den Wert um eins erhöhen). Das Flag wird auf „bearbeitet“ gesetzt, das Netz kann weiterschalten.

Die Bearbeitung geschieht also in drei Schritten:

1. Ein Datum erreicht die magische Stelle, deren Flag auf „frei“ steht.
2. Das mit der magischen Stelle verbundene Makro wird aufgerufen und bearbeitet das Datum, das Flag wechselt auf „belegt“.
3. Das Ergebnis der Ausführung (das vom gleichen Typ sein muß wie der Ausgangswert) wird auf die magische Stelle gelegt, das Flag auf „bearbeitet“ gesetzt.

Danach kann das Netz mit seiner Bearbeitung fortfahren. Nach entfernen des Tokens wird das Flag auf „frei“ zurückgesetzt.

Kapitel 7 Die Makrokomponente

Eine zentrale Eigenschaft von Makros ist die Möglichkeit ihrer Wiederverwendung. Dem Benutzer müssen daher Möglichkeiten zum Auffinden, Bearbeiten und Einsetzen von Makros zur Verfügung gestellt werden.

Weiterhin muß eine interne Schnittstelle existieren, die die Makrodefinition auf eine konkrete Implementierung (z. B. auf Prozeßkontrollnetze aus Kapitel 6, kurz PCN) abbildet.

7.1 Makroverwaltung

Die Makroverwaltung muß im wesentlichen folgende Funktionalitäten anbieten:

1. Suchen eines Makros

   Dem Benutzer muß ermöglicht werden, ein schon definiertes Makro wieder auffinden zu können. Makros können auch sinnvoll zu Unterbibliotheken zusammengefaßt werden, um die Suchräume zu verkleinern. Dadurch können auch entsprechende Namensräume mit hierarchischen Namen eingeführt werden (z. B. „Basis.Leermakro“)

2. Neudefinition eines Makros:

   Der Benutzer muß die Möglichkeit haben, neue Makros (auch unter Zuhilfenahme schon bestehender Makros) definieren und in die Makrobibliothek aufnehmen zu können. Dazu steht die Makrodefinitionssprache aus Kapitel 5 zur Verfügung.

3. Ändern eines Makros

   Die Bearbeitung eines schon definierten Makros muß möglich sein. Hierbei gilt es jedoch zu beachten, daß unter Umständen auch das unveränderte Makro erhalten bleiben muß (wenn es z. B. als Parameter eines anderen Makros verwendet wurde, das sich gerade in der Ausführung befindet). Mit Versionierung der Makros kann dieses Problem gelöst werden. Jedes „Parametermakro“ liegt in der benötigten Fassung vor. Wird eine ältere Version nicht mehr benutzt, kann sie gelöscht werden („Garbage Collection“). Das Auffinden nicht mehr benötigter Versionen ist Aufgabe des übergeordneten WfMS, in dem die vollständigen Workflow-Definitionen vorliegen und mit der Makrobibliothek abgeglichen werden können.

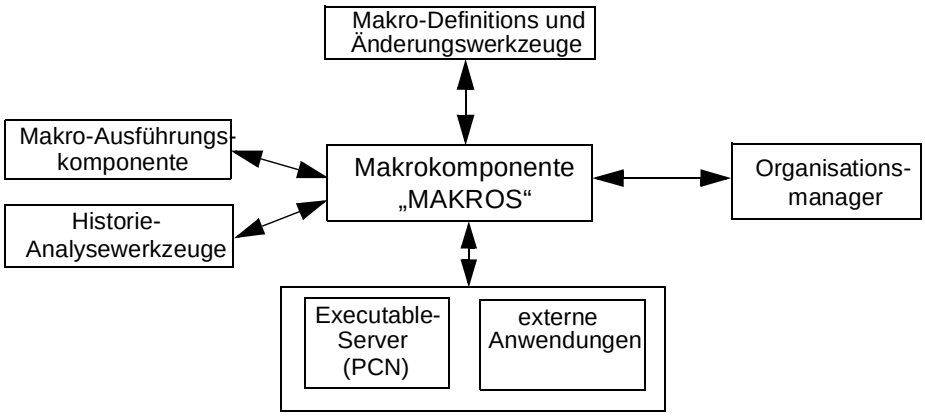
4. Instanziieren eines Makros

   Der Benutzer soll einen Makrorumpf instanziieren können, d. h. die Parameter so füllen, daß ein ablauffähiges Makro entsteht.

5. Definition von Workflows

   Mit Hilfe der vorhandenen Makros können ablauffähige Workflows zuammengesetzt werden, die durch die Ablaufkomponente ausgeführt werden.

Abbildung 14 Die Elemente der Makrokomponente



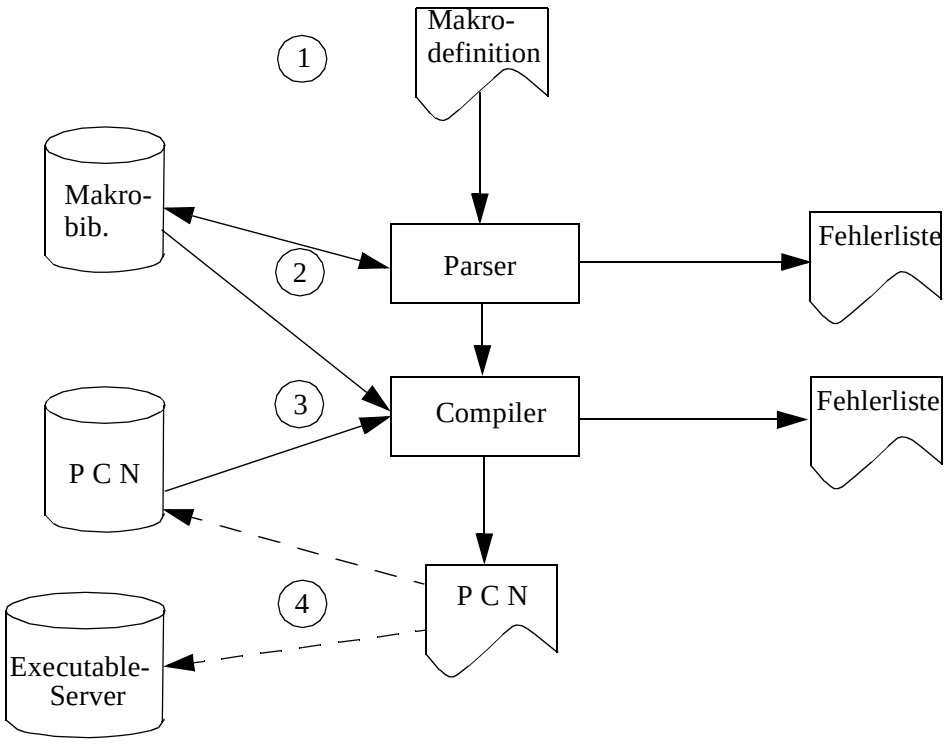
7.2 Erstellen von Makros

Bei der Erstellung neuer Makros muß zunächst unterschieden werden, ob wir ein neues Interface, einen neuen Makrorumpf oder ein ausführbares Makro erzeugen wollen, da sich die Verarbeitung der Definition leicht unterscheidet. In Abbildung 15 ist der Weg eines ausführbaren Makros von der Definitionsdatei bis zum Netz dargestellt: Die Datei mit der Makrodefinition (1) wird vom Parser auf syntaktische Korrektheit und auf Konsistenz mit den aus der Makrobibliothek benutzten Makros, Makrorümpfen und Interfaces geprüft. Komplexe Makros werden solange vereinfacht, bis nur noch direkt auf PCN abbildbare Grundmakros oder bereits in der Bibliothek vorhandene Makros benutzt werden. Treten Fehler auf, so werden diese dem Benutzer als Fehlerliste ausgegeben (2).

Die so bearbeitete Makrodefinition wird nun dem Compiler übergeben, der die eigentliche Abbildung auf ein PCN vornimmt (3). Alle verwendeten Makrorümpfe werden durch ihre PCN-„Gerüste“ ersetzt, Makros durch ihr zugehöriges PCN. Das Ergebnisnetz wird unter eindeutigem Namen ebenfalls wieder in die PCN-Bibliothek eingebracht und steht zur Weiterverarbeitung (und natürlich auch zur Ausführung) zur Verfügung (4). Alle ausführbaren Makros werden im *Executable Server* gesammelt, der diese zur Ausführung als „Untermakroaufruf“ zur Verfügung stellt, vergleichbar mit dem Aufruf eines Unterprogramms bei Programmiersprachen.

Bei Interfaces endet die Verarbeitung bereits beim Parser. Da Interfaces keinen Implementierungsteil besitzen, sondern nur eine Schnittstelle beschreiben, wird (bei syntaktischer Korrektheit) das Interface in die Makrobibliothek aufgenommen. Es kann dann zur Definition neuer Makros genutzt werden.

Abbildung 15 Ablauf der Umwandlung einer Makro-Definition in ein PCN



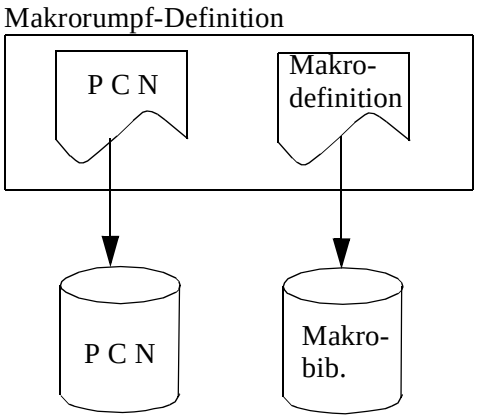
Bei Makrorümpfen ist eine andere Vorgehensweise notwendig. Für jeden Makrorumpf muß ein entsprechendes PCN existieren. Dieses PCN muß „in Handarbeit“ erstellt werden. Es muß fehlerfrei arbeiten, denn es wird ohne weitere Prüfung direkt in die PCN-Bibliothek aufgenommen, der Deklarationsteil entsprechend in die Makrobibliothek.

Die Zweiteilung in Parser und Compiler hat auch zur Folge, daß zwei deutlich trennbare Schichten entstehen: Der Parser arbeitet völlig unabhängig von der gewählten Implementierung (in unserem Falle PCN). Die implementierungsabhängigen Schritte werden vom Compiler übernommen. Dies bedeutet, daß bei einem Wechsel von PCN auf ein anderes Konzept nur der Compiler und die PCN-Bibliothek ausgetauscht werden müssen, die Makrodefinitionen hingegen können ohne Änderung weiterverwendet werden!

7.3 Ändern eines Makros

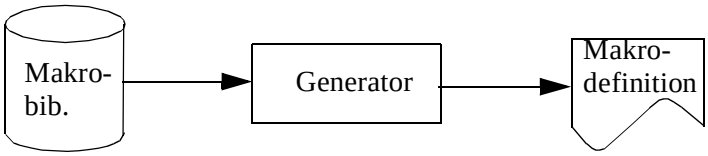
Ein einmal definiertes Makro kann selbstverständlich auch wieder geändert werden. Hierzu werden die in der Makrobibliothek abgelegten Beschreibungsdaten ausgelesen und mit Hilfe eines Generators in eine editierbare Datei umgewandelt. Diese kann bearbeitet werden und anschließend über den in 7.2 beschriebenen Weg neu umgewandelt und eingebracht werden. Die ursprünglich zur Definition des Makros benutzte Datei hat keine Funktion mehr. Insbesondere gehen auch Kommentare und Anmerkungen verloren, jedoch wird durch diese Vorgehensweise garantiert, daß immer die aktuellste Version zur Bearbeitung bereitsteht.

Abbildung 16 Definition eines neuen Makrorumpfes



Beim statischen Binden handelt man sich jedoch Probleme ein. Wird ein Makro als Parameter benutzt und statisch eingebunden, so muß dieses Makro natürlich weiterhin zur Verfügung stehen, da sonst (je nach Art der Änderung) unvorhersehbare Fehler und Effekte auftreten können. Beim dynamischen Binden ist die Wiederholbarkeit eines Ablaufs nicht mehr gewährleistet. Aus diesem Grund werden Makros in verschiedenen Versionen gespeichert, so daß immer auf die tatsächlich benötigte Version des Makros zugegriffen werden kann. Beim Binden muß in diesem Fall mitprotokolliert werden, welche Version verwendet wurde. Bei der Wiederholung eines dynamisch gebundenen Ablaufs erfolgt die Auswahl der Makros mittels des Bindeprotokolls und nicht mehr über die Auswahlprädikate!

Abbildung 17 Generieren einer Makrodefinition aus Daten der Makrobibliothek



7.4 Ein Beispiel

An einem kleinen Beispiel sollen nun die Schritte von der Definition eines Makros bis hin zum ablauffähigen Netz demonstriert werden. Um den Umfang in erträglichem Maß zu halten, verlassen wir hierzu kurz die Welt der Arbeitsabläufe und benutzen ein kleines mathematisches Problem: die Berechnung des größten gemeinsamen Teilers zweier Zahlen. Zu diesem Zweck definieren wir ein Makro „ggT“, das vom Interface „CalcInterface“ aus Beispiel 22 abgeleitet

wird. Alle Eingabe-Token enthalten zwei Elemente k1 und k2, von denen der ggT gebildet werden soll, als Ergebnis wird ein Token mit dem gefundenen Wert generiert.

Beispiel 22 Interfacedeklaration

```
INTERFACE NEW CalcInterface;
IN:    Eingabe   [CF, (k1: INT, k2: INT)];
OUT:   Ausgabe   [CF, (k1: INT)];
END.
```

Nach Festlegen der Schnittstellen müssen wir nun den Körper des Makros ggT formulieren. Die Vorgehensweise ist dabei einfach: Solange die beiden Elemente k1 und k2 verschieden sind, soll der größere Wert um den kleineren vermindert werden. Dabei kommen die beiden elementaren Makros „SUBx-y“ und SUBy-x“ zum Einsatz, deren zugehörige Netze in Anhang A zu sehen sind. Diese sind zur Durchführung der Subtraktion zuständig. Es ergibt sich die in Beispiel 23 gezeigte Definition.

Beispiel 23 Makrodefinition

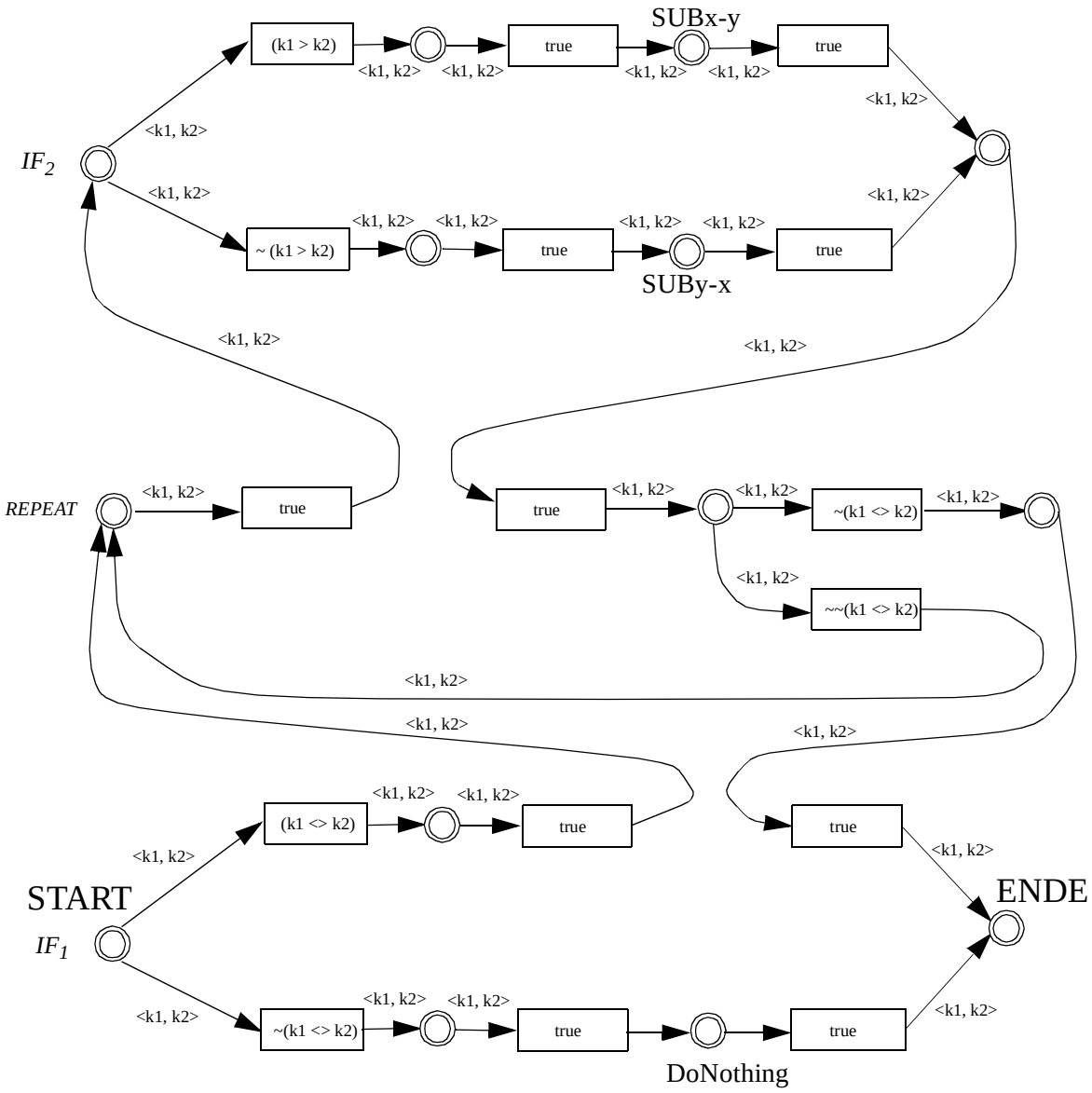
```
MACRO ggT IS CalcInterface;
BODY:   #MACRO_USE WHILE(„(k1 <> k2)“,
                              #MACRO_USE IF(„(k1 > k2)“,
                                                  #EXEC SUBx-y,
                                                  #EXEC SUBy-x))
END.
```

Die Definition wird dem PARSER übergeben. Dieser führt neben der syntaktischen Prüfung auch die Expansion der abgeleiteten Makros zu Grundmakros durch. Jedes abgeleitete Makro wird durch seine BODY-Deklaration ersetzt. Der Vorgang wiederholt sich, bis nur noch Grund- und elementare Makros benutzt werden. In unserem Beispielmakro muß das abgeleitete Makro „WHILE“ expandiert werden. Das Ergebnis zeigt Beispiel 24.

Beispiel 24 Expansion der Makrodefinition

```
MACRO ggT IS CalcInterface;
BODY:   #MACRO_USE IF(„(k1 <> k2)“,
                  #MACRO_USE REPEAT(„~(k1 <> k2)“,
                                    #MACRO_USE IF(„k1 > k2“,
                                                  #EXEC SUBx-y,
                                                  #EXEC SUBy-x)),
                  #EXEC DoNothing)

END.
```

Beispiel 25 Netz des Makros ggT



Zu jedem Grundmakro existiert eine Vorschrift, wie dieses in ein Prozeßkontrollnetz umzusetzen ist. Elementare Makros, die in ausführbarer Form auf dem Executable-Server bereitliegen, werden mit Hilfe magischer Stellen benutzt. Der Compiler nimmt nun die Abbildung der expandierten Makrodefinition aus Beispiel 24 vor. Jede #EXEC-Anweisung wird durch eine magische Stelle ersetzt, die mit dem benutzten elementaren Makro verknüpft wird, jedes #MACRO_USE führt zur Umsetzung in das zugehörige Netz. Die Kantenanschriften werden aus den Domänen der Eingangs- und Ausgangstransitionen gebildet.

Das Ergebnisnetz ist in Beispiel 25 dargestellt. IF_1 ist die Umsetzung des IF-Makros, daß der REPEAT-Iteration zum Simulieren eines WHILE-Makros vorangestellt wurde. Die Teilnetze sind zum einen die REPEAT-Schleife, zum andern das leere Makro „DoNothing“.

Die Iteration führt das Teilnetz IF_2 solange aus, bis beide Komponenten k1 und k2 gleich sind. IF_2 schließlich prüft, welche der beiden Subtraktionen k1 - k2 oder k2 - k1 durchgeführt werden muß und legt das Token auf der entsprechenden magischen Stelle ab.

7.5 Laufzeitumgebung

Mit Hilfe aller definierten Makros und Makrorümpfe lassen sich nun Workflows modellieren, zu PCN umwandeln und ausführen. Hierzu ist eine geeignete Laufzeitumgebung vorgesehen, die diese Aufgabe übernimmt. Nach Festlegen evtl. benötigter Umgebungsvariablen (z. B. für Late Binding) muß das Netz mit einer Startmarkierung belegt, d. h., alle Stellen in einen initialen Zustand versetzt werden. Danach kann das Netz sich selbst überlassen werden, da es nun (bei korrekter Definition) seine Aufgabe ohne weiteren Eingriff erfüllen sollte. Der Ablauf kann mit Hilfe eines *Monitors* verfolgt werden, eine detaillierte Analyse kann nach Abschluß des Laufs vorgenommen werden.

7.5.1 Workflows

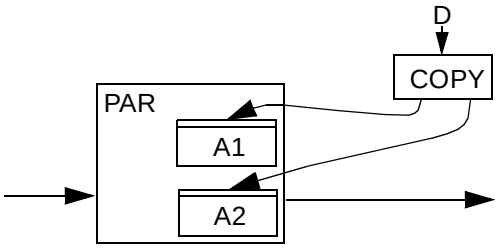
Mit Hilfe der Makrobibliothek haben wir nun die Möglichkeit, auf einfache Art und Weise Workflows zu modellieren. Die bisher erzeugten ausführbaren Makros sind zwar auch Workflows, gehören aber immer nur einem Flußtyp (Datenfluß oder Kontrollfluß) an. Hybridmakros können auf dieser Ebene noch nicht vollständig mit Parametern versorgt werden. Ein kleines Beispiel macht deutlich, warum: Ein Applikationsmakro muß sowohl einem Kontrollfluß- als auch einem Datenflußmakro übergeben werden. Es gibt aber keine Möglichkeit, beide Parameter auf **ein** identisches Netz abzubilden und die Transitionen entsprechend anzubinden.

Daher müssen wir uns zur Definition eines Workflows auf die Ebene des WfMS begeben, um Prozesse unter Zuhilfenahme von Makros vollständig zu beschreiben. Die dazu verwendete Definitionssprache kann sich an der schon vorgestellten Makrodefinitionssprache orientieren, notwendige Erweiterungen sind z. B. Makroinstanzen und Paragraphen zur Abgrenzung der Teilaspekte. Makroinstanzen werden mit eindeutigen Bezeichnern versehen. Auf diese Art und Weise kann an verschiedenen Stellen der Workflow-Definition die gleiche Instanz angesprochen werden.

In Beispiel 26 ist ein kleiner Ausschnitt aus einem Workflow zu sehen, der die Ausführung zweier Applikationen „A1“ und „A2“ veranlassen soll. Beide Applikationen benötigen eine Kopie des Dokuments „D“. Mit Hilfe des COPY-Makros wird „D“ dupliziert und an die zuständige Eingangstransition weitergeleitet. Im Kontrollfluß wird festgelegt, daß „A1“ und „A2“ parallel ausgeführt werden sollen. Die beiden Applikationen müssen folglich den Makros COPY und PAR als Parameter übergeben werden. Hierzu werden zwei Instanzen „m1“ und „m2“ erzeugt, die einerseits bei der Deklaration des Kontrollflusses als Parameter an PAR übergeben werden können, aber auch beim Festlegen des Datenflusses angesprochen werden können!

Die Workflow-Definitionskomponente ist Teil des der Makrokomponente übergeordneten Workflow-Management-Systems und wird hier nicht näher besprochen.

Beispiel 26 Workflow-Ausschnitt mit Hybridmakros



WORKFLOW WF1;

m1 = A1();// A1 und A2 sind Makros vom Typ Applikation
m2 = A2();

CF: ...
#MACRO_USE PAR((„k=1“, m1), („k=2“, m2));
...

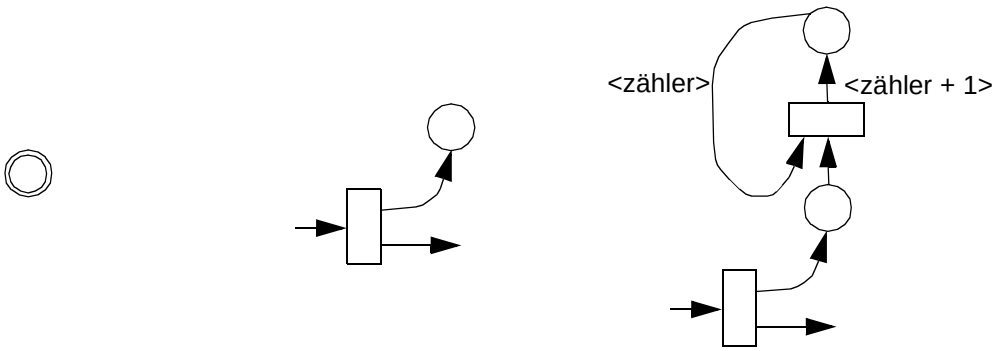
gleiche Instanz!

DF:...
#EXEC COPY()
TAKING Eingabe FROM D
GIVINGAusgabe1TO m1.Eingabe,
Ausgabe2TO m2.Eingabe;
...

7.5.2 Ablaufkontrolle

Um sich über den aktuellen Zustand des Netzes informieren zu können, gibt es einen Netzmonitor, mit dessen Hilfe die Abläufe im Netz beobachtet werden können. Da der Netzmonitor kein

Abbildung 18 Statusstellen



a) Magische Stelle (aktiv) b) passive Statusstelle c) Statusstelle mit Zähler

Teil des Workflows ist, sondern vielmehr der „Außenwelt“ angehört, wird die Rolle der aktiven Statusstelle von den magischen Stellen mit übernommen (Abbildung 18).

Sobald eine magische Stelle erreicht wird, setzt sie eine entsprechende Meldung an den Monitor ab. Eine ausgezeichnete Rolle spielen die magischen Stellen an den Eingangs- und Ausgangstransitionen. Sie melden explizit das Betreten bzw. Verlassen eines Makros (z. B. „IF“... „End of IF“).

7.5.3 Ablaufanalyse

Neben der doch sehr groben Ablaufkontrolle gibt es noch die Möglichkeit, den Zustand eines Netzes nach Ende des Ablaufs detailliert zu untersuchen. Dazu werden passive Stellen benutzt, die nur anzeigen, ob eine Transition passiert wurde (Abbildung 18 b). Alternativ können auch Zähler verwendet werden, die die genaue Anzahl der Transitionsdurchläufe liefern (Abbildung 18 c).

Beispiel 27 Ausgabe eines Analyselaufs mit passiven Statusstellen.

```
IF 1
    1   (k1 <> k2)          PASSED
    2   (true)              PASSED
    REPEAT
        1 (true)            PASSED
        IF 1
            1 (k1 > k2)     PASSED
            2 (true)        PASSED
            SUBx-y
            3 (true)        PASSED
        END OF IF
        IF 2
            1 ~(k1 > k2)    PASSED
            2 (true)        PASSED
            SUBy-x
            3 (true)        PASSED
        END OF IF
        2 (true)            PASSED
        DECIDE 1
            3 ~(k1 <> k2)   PASSED
        DECIDE 2
            4 ~~(k1 <> k2)  PASSED
    END OF REPEAT
    3 (true)                PASSED
END OF IF
IF 2
    1 ~(k1<>k2)             NEVER PASSED
    2 (true)                NEVER PASSED
    DO_NOTHING
        1 (true)            NEVER PASSED
    END OF DO_NOTHING
    3 (true)                NEVER PASSED
END OF IF
```

Um eine Ablaufanalyse durchzuführen, muß zuerst das „Debuglevel“ angegeben (Status oder Zähler) und ein entsprechendes Netz erzeugt werden. Nach dem Lauf wird das gesamte Netz durchlaufen, alle Transitionen geprüft und das Ergebnis ausgegeben. Transitionen werden dabei jeweils pro Zweig von der Eingangsstelle zur Ausgangstelle der Reihenfolge nach numeriert. Stellen, die für mehrere Transitionen als Eingangsstellen dienen, sind ebenfalls als magische Stellen auszubilden, die Einzelzweige werden ebenfalls numeriert und nacheinander ausgewertet. Zur besseren Zuordnung kann auch zusätzlich die Transitionsinschrift mit ausgegeben werden. Ein typischer Analyselauf ist in Beispiel 27 zu sehen. Dabei wurde das ggT- Beispielnetz aus Beispiel 25 mit dem Start-Token <18, 27> ausgeführt und anschließend analysiert.

Mit solchen Analysen ist es möglich, Fehler in Netzen aufzuspüren, es läßt sich schnell überprüfen, ob ein Teilnetz (bei entsprechender Startmarkierung) ausgeführt wurde oder nicht. Weiterhin lassen sich mit Hilfe der Zähler auch leicht Statistiken erzeugen, die besonders stark bzw. schwach genutzte Teile des Netzes identifizieren können.

7.5.4 Der Organisationsmanager

Der Organisationsmanager kapselt die Auswahl geeigneter organisatorischer Einheiten. Ihm ist die Organisation bekannt, die Agenten, Nicht-Agenten, ihre Rollen und charakteristischen Eigenschaften. In Beispiel 28 ist ein Ausschnitt einer typischen Firmenorganisation zu sehen. Mit Hilfe dieser Informationen können organisatorische Zuordnungsstrategien spezifiziert werden. Es können dabei vier Stufen unterschieden werden [Buß98]:

1. *Manuelle Zuweisung*: Das WfMS stellt keine Regeln zur Verfügung sondern erwartet, daß alle Zuordnungen manuell durchgeführt werden (der Organisationsmanager entfällt).
2. *Aufgabenträgerbasierte Zuweisung*: Es können nur Aufgabenträger direkt benannt werden, um einen Workflow zuzuweisen. Andere Kriterien sind nicht möglich.
3. *Rollenbasierte Zuweisung*: Rollen werden als Kriterien angegeben, deren zugeordnete Aufgabenträgern den entsprechenden Workflow zugewiesen bekommen.
4. *Regelbasierte Zuweisung*: In dieser Gruppe werden Zuweisungsregeln verwendet, die aufgrund unterschiedlicher Kriterien Aufgabenträger ermitteln. Entsprechend dem Ergebnis werden Workflows zugewiesen.

Zuordnungsstrategien können dabei beliebig komplex sein, eine ausführlichere Darstellung würde den Rahmen dieser Arbeit sprengen. Zur weiteren Lektüre sei [Buß98] empfohlen.

7.5.5 Workflow-Kontext

In Abschnitt 5.1.4 wird ein Auswahlprädikat zur dynamischen Zuordnung eines Makros beschrieben, das die Rolle eines polymorphen Makros übernehmen soll. Hierzu wurde auf einen Workflow-Kontext verwiesen, mit dessen Hilfe dieses Auswahlprädikat ausgewertet werden soll. Dabei handelt es sich um eine Ansammlung von Laufzeitinformationen, die ein Bild davon vermitteln, unter welchen Bedingungen das Makro aktuell ausgeführt wird.

Einzelinformationen werden als atomare Werte abgelegt und mit Bezeichnern versehen, die Namensräumen zugeordnet sind, die wiederum Bezeichner besitzen. Es können somit mehrere

Namensraumstufen entstehen. Ein Bezeichner kann lokalisiert werden, indem die Bezeichner durch Punkte getrennt aneinandergehängt werden.

Vordefinierte Namensräume eines Workflows Wk sind Wk selbst, Wk.System (hier befinden sich die Informationen zur Systemumgebung) und Wk.Super (der den Kontext des Super-Workflows enthält). Einzelinformationen sind z. B. Wk.startTime oder Wk.System.OperatingSystem.Name.

Beispiel 28 Firmenorganisation und Rollen

Position	Name	Rollen
Abteilungsleiter	Albert Amann	genehmigt Zahlungen über 100 000.- genehmigt Urlaub führt Abteilung A
Gruppenleiter	Boris Bdorf	genehmigt Auszahlungen bis 100 000.- führt Gruppe A3
Kassierer	Christian Cfeld	führt Auszahlungen durch nimmt Einzahlungen entgegen arbeitet in Gruppe A3
Sekretärin	Doris Dfrau	vereinbart Termine verwaltet Urlaubsscheine

Kapitel 8 Die Implementierung

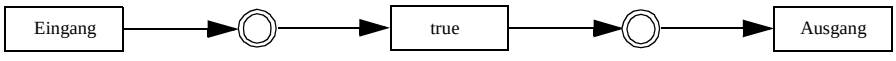
In diesem Kapitel wird die Umsetzung der Makrodefinitionssprache in ausführbare Prozeßkontrollnetze beschrieben. Zuerst werden Grundmakros betrachtet, die direkt in PCN abgebildet werden können, danach folgen Grundmakros mit variablen Parametern. Zum Abschluß werden noch Beispiele für abgeleitete Makros vorgestellt, die aus diesen zusammengesetzt werden können.

8.1 Grundmakros

Grundmakros bilden den Grundstock, mit dem alle anderen Makros aufgebaut werden können. Dabei können Grundmakros mit fester und variabler Parameterzahl unterschieden werden. Bei festen Parametern besitzen die Makros ein direktes Gegenstück in der Netzbibliothek, bei variabler Anzahl werden sie aus einfachen Einzelteilen, die in der Netzbibliothek vorhanden sind, zusammengefügt.

8.1.1 Das leere Makro (DoNothing)

```
INTERFACE NEW DoNothingInterface;
IN:    Eingang [*, DOMAIN *];
OUT: Ausgang[*, DOMAIN *];
END;
```

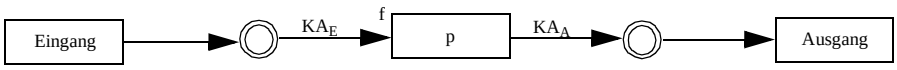


Das leere Makro ist der einfachste Baustein. Er besteht nur aus einer Transition, die jederzeit schalten kann. Eingangs- und Ausgangstransition haben „don´t care“ Domänen und sind jederzeit verknüpfbar. Das dargestellte Netz liegt ausführbar im Executable-Server bereit und kann mit Hilfe der „#EXEC“-Anweisung angesprochen werden. Es steht aber auch zur Expansion mittels „#MACRO_USE“ in der PCN-Bibliothek zur Verfügung. Die Stellen hinter der Eingangs- und vor der Ausgangstransition sind als magische Stellen ausgebildet, um eine Überwachung des Ablaufs durch die Laufzeitumgebung zu ermöglichen.

8.1.2 Funktionsmakros

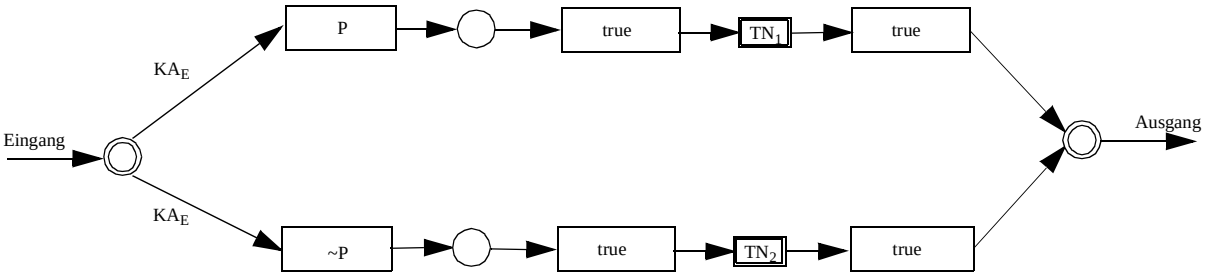
Einfache Funktionen lassen sich oft sehr leicht direkt als Netz abbilden, etwa das Inkrementieren eines Zählers oder das Bilden einer Differenz. Diese einfachen Netze werden als Funktionsmakros bezeichnet. Sie liegen in ausführbarer Form auf dem Executable-Server bereit.

INTERFACE NEW FunctionInterface;
IN: Eingang [CF, DOMAIN EingabeDomain];
OUT: Ausgang[CF, DOMAIN AusgabeDomain];
END;



8.1.3 Die bedingte Verzweigung (IF)

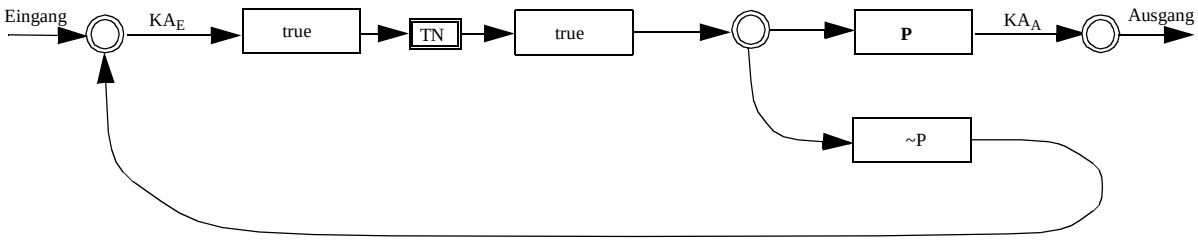
INTERFACE NEW CondInterface;
IN Eingang [CF, DOMAIN *];
OUT Ausgang [CF, DOMAIN *];
FUNCTION p OF BOOLEAN;
MACRO M_1 IS MacroInterface;
MACRO M_2 IS MacroInterface;
END;



Die bedingte Verzweigung ist eine elementare Kontrollstruktur. Abhängig von einem Prädikat p soll entweder das durch Makro M_1 definierte Netz TN_1 ausgeführt werden (wenn p erfüllt ist), oder alternativ das zu Makro M_2 gehörende Netz TN_2. Um die Darstellung nicht zu unübersichtlich werden zu lassen, wurde in der obigen Darstellung auf die Eingangs- und die Ausgangstransition verzichtet. Die Eingangstransition besitzt keine explizite Domäne. Die Anforderungen an die hier ankommenden Token werden durch die Wahl der Parameter p und M_i bestimmt. Die Kantenanschrift KA_E läßt sich aus den über die Parameter bekannten Informationen leicht ableiten. Für das Prädikat p müssen die zur Auswertung notwendigen Komponenten vorhanden sein, ebenso die Elemente der Kantenanschrift des benutzten Netzes TN_i.

8.1.4 Die Iteration (REPEAT)

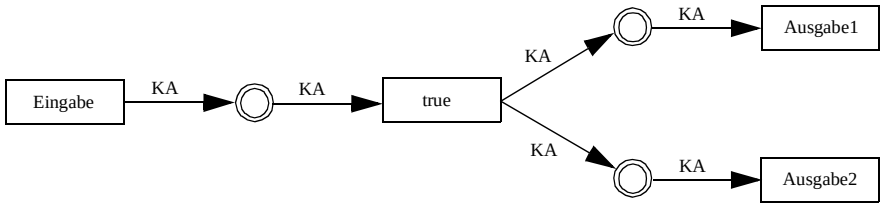
```
INTERFACE NEW IterationInterface;
IN          Eingang  [CF, DOMAIN *];
OUT         Ausgang  [CF, DOMAIN *];
FUNCTION p OF BOOLEAN;
MACRO       M IS MacroInterface;
END;
```



Wie bei den Programmiersprachen sind Schleifen auch bei der Modellierung von Abläufen ein mächtiges Werkzeug. Sie bieten die Möglichkeit, einen Teilablauf mehrfach zu wiederholen, bis eine bestimmte Bedingung erfüllt ist. Das obige Netz zeigt die Implementierung einer fußgesteuerten Schleife, d. h., das Netz TN wird mindestens einmal ausgeführt. Kopfgesteuerte Schleifen und auch Zählschleifen lassen sich leicht aus der fußgesteuerten Schleife ableiten (Abschnitt 8.2). Die Prädikate p und M werden analog zum IF-Makro behandelt.

8.1.5 Dokument kopieren (COPY)

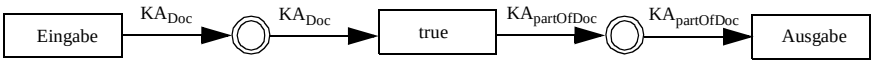
```
MACRO copy IS NEW CopyInterface;
IN:   Eingabe   [DF, DOMAIN *];
OUT:  Ausgabe1[DF, DOMAIN *],
      Ausgabe2[DF, DOMAIN *];
```



Häufig reicht ein Exemplar eines Dokuments nicht aus, es müssen Kopien erstellt werden. Mit dem *copy*-Makro wird dies ermöglicht. Jedes Dokument auf „Eingabe“ wird dupliziert und liegt auf „Ausgabe1“ und „Ausgabe2“ bereit.

8.1.6 Datum extrahieren (EXTRACT)

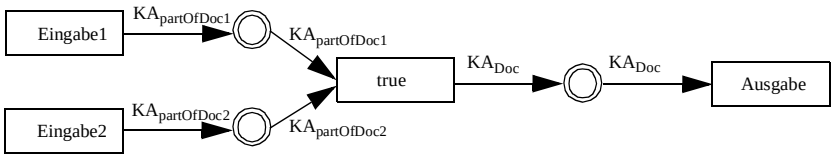
MACRO extract IS NEW ExtractInterface;
IN: Eingabe[DF, DOMAIN Doc];
OUT: Ausgabe[DF, DOMAIN partOfDoc];



Dokumente können aus einer Anzahl verschiedener Teildokumente zusammengesetzt sein. Wird im weiteren Verlauf jedoch nur eine Auswahl dieser Teildokumente benötigt, können diese Teildokumente mit Hilfe des Makros *extract* zu einem neuen, entsprechend verkleinertem Dokument zusammengesetzt werden.

8.1.7 Datum verschmelzen (MELT)

MACRO melt IS NEW meltInterface;
IN: Eingabe1[DF, DOMAIN partOfDoc1],
Eingabe2[DF, DOMAIN partOfDoc2]
OUT: Ausgabe [DF, DOMAIN Doc];



Das Makro *melt* ermöglicht das Verschmelzen von Dokumenten zu einem neuen Gesamtdokument. Dies wird notwendig, wenn Ergebnisse aus unabhängigen Schritten zusammen für einen weiteren Arbeitsschritt zur Verfügung stehen müssen. So können beispielsweise verschiedene Anträge voneinander unabhängig gestellt werden, aber erst wenn alle Genehmigungen gesammelt vorliegen, kann die Verarbeitung fortgesetzt werden.

8.1.8 Expansion von Makros mit variabler Parameteranzahl

Die Abbildung von Makros mit festen Parametern ist recht einfach, da sie durch ein festes Netz beschrieben werden können. Makros wie PAR oder SER können leider nicht ganz so einfach umgesetzt werden, allerdings haben die einzelnen Zweige eine feste Grundstruktur, die Expansion entspricht einer Vervielfältigung dieser Struktur.

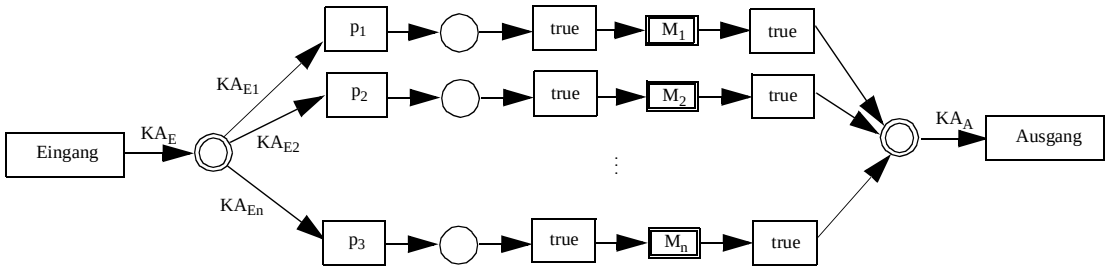
#FOREACH

Mit #FOREACH werden Netze mit einer variablen Zweiganzahl expandiert. Beispiel 29 zeigt die Anwendung: für jedes Tupel (p,m) der Kollektion „Zweige“ wird ein eigener Ast erzeugt. Es ergibt sich das Netz aus Abbildung 19. Das Prädikat p aus der #ON-Klausel übernimmt dabei die Aufgabe, für die richtige Zuordnung der Token zu den Zweigen zu sorgen.

Beispiel 29 Ein Makro mit dynamischer Zweiganzahl.

```
MACRO PAR IS DynamicParmInterface;
BODY:   #FOREACH (Zweige)
            #ON p #EXEC m;
        #END
END.
```

Abbildung 19 Expandieren von #FOREACH



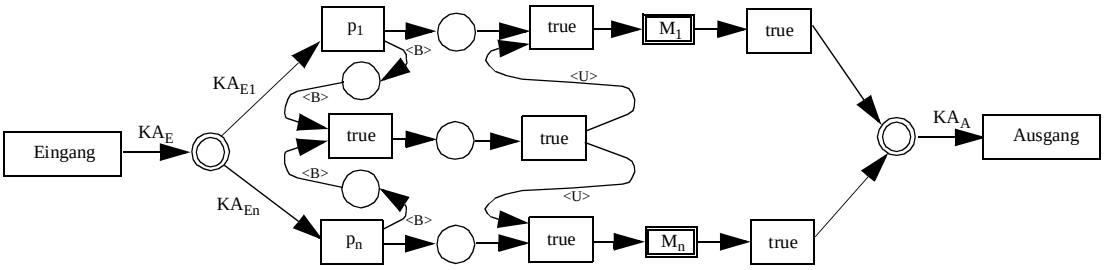
#BLOCKED

MACRO BlockedPar IS DynamicParmInterface;

```
BODY    #FOREACH (Zweige)
            #ON p #EXEC m #BLOCKED;
        #END
END.
```

Mit Hilfe des Schlüsselworts #BLOCKED soll die Ausführung der einzelnen Zweige solange verzögert werden, bis alle Zweige gleichzeitig ausgeführt werden können. Dies wird durch eine zusätzliche Stelle erreicht, deren Eingangstransition erst schaltet, wenn alle Auswahlprädikate mindestens einmal erfüllt wurden. Danach wird mit Hilfe der Ausgangstransition für jeden Zweig ein Freigabetoken erzeugt, so daß die Blockade des Zweigs aufgehoben wird und mit der Abarbeitung des benutzten Makros fortgefahren werden kann. In Abbildung 20 sind die durch #BLOCKED erzeugten zusätzlichen Elemente zu sehen. Bemerkenswert ist, daß der durch #FOREACH erzeugte Teil des Netzes unverändert geblieben ist, die Modifikation kann modular durchgeführt werden!

Abbildung 20 Synchronisation der Ausführung durch Schlüsselwort #BLOCKED



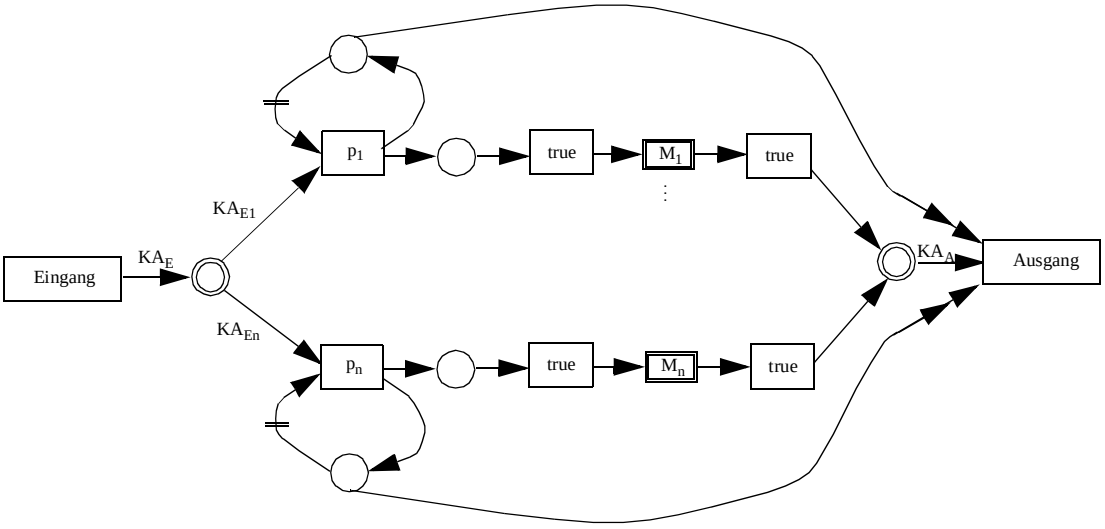
#ONCE_ONLY

MACRO OnceOnlyPar IS DynamicParmInterface;

```
BODY:    #FOREACH (Zweige)
               #ON p #EXEC(m) #ONCE_ONLY;
         #END

END.
```

Die Definition von #FOREACH läßt immer noch die sofortige wiederholte Ausführung von Zweigen zu, unabhängig vom Bearbeitungszustand der restlichen Zweige. Mit #ONCE_ONLY wird ein Zweig nach der Ausführung gesperrt, indem eine Sperrmarke erzeugt wird, die mit Hilfe einer Verbotskante die Zweigeingangstransition sperrt, bis alle Zweige abgearbeitet wurden. Danach wird die Sperrmarke wieder entfernt, das Netz ist für den nächsten Durchgang bereit. Auch die Modifikationen für #ONCE_ONLY können modular durchgeführt werden.

Abbildung 21 Umsetzung von #ONCE_ONLY in ein PCN



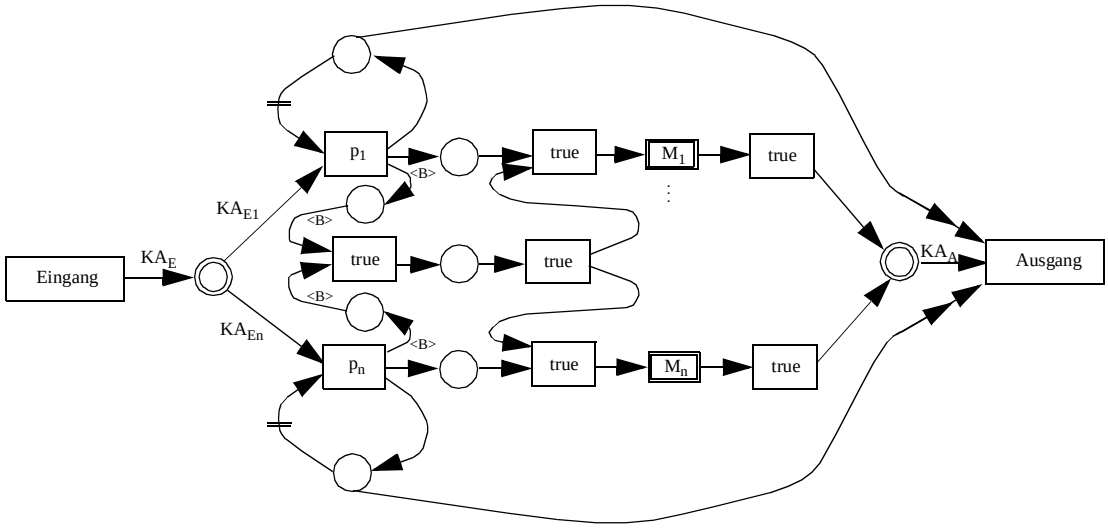
Mit Hilfe dieser Elemente läßt sich nun das Makro PAR sinnvoll beschreiben. In Beispiel 30 ist die zugehörige Deklaration zu sehen.

Beispiel 30 Deklaration des Makros PAR

```
MACRO PAR IS DynamicParmInterface;

BODY    #FOREACH (Zweige)
            ON p #EXEC m #BLOCKED #ONCE_ONLY;
        #END

END.
```

Für jedes Tupel (p,m) aus der Kollektion Zweige wird ein Ast mit Auswahlprädikat p und benutztem Makro m erzeugt. #BLOCKED synchronisiert die Ausgabe, #ONCE_ONLY sorgt für ein faires Konsumieren der Eingangstoken. Das zugehörige Netz zeigt Beispiel 31.

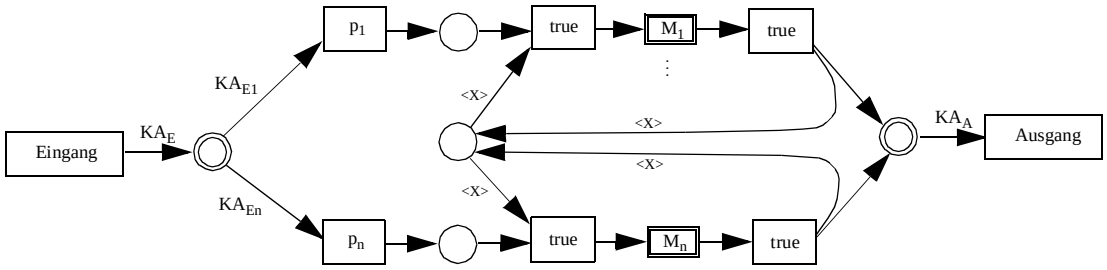
Beispiel 31 Das Makro PAR mit verzögerter, einmaliger Ausführung der Teilmakros



#EXCLUSIVE

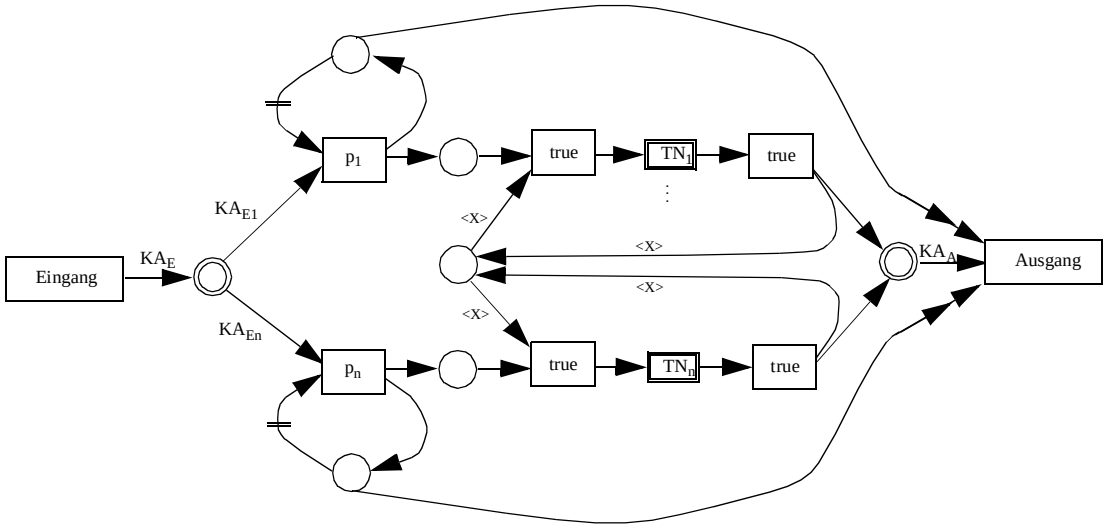
Bei den bisherigen Deklarationen konnten alle benutzten Makros parallel ausgeführt werden. Bei der Serialisierung (SER) darf aber die Ausführung nur exklusiv erfolgen. Mit #EXCLUSIVE kann genau das erreicht werden: Auf einer ausgezeichneten Stelle liegt genau ein Token, dessen Besitz das Ausführungsrecht gewährt. Eine feuerbereite Transition konsumiert dieses Token und blockiert somit alle übrigen Zweige. Nach erfolgreicher Abarbeitung wird ein neues Token erzeugt und auf der Ausgangsstelle abgelegt, die nächste feuerbereite Transition kann das Ausführungsrecht erwerben.

Abbildung 22 PCN zu Schlüsselwort #EXCLUSIVE



In Abbildung 23 sind die zusätzlichen Netzteile abgebildet. Auch diese sind modular hinzugefügt, ohne die restliche Struktur zu verändern.

Abbildung 23 PCN zu Makro SER mit einmaliger und exklusiver Ausführung der Teilmakros



In Beispiel 32 wird das Makro SER deklariert. Wie schon bei PAR wird für jedes Tupel der Kollektion ein Zweig erzeugt, p dient als Auswahlprädikat, #ONCE_ONLY sorgt für einmalige Durchführung. Zusätzlich braucht ein ausführbereites Teilnetz aber noch das durch #EXCLUSIVE hinzugefügte <X>-Token, um das Recht zur Ausführung zu erhalten. In Abbildung 23 ist das gesamte entstehende Netz aufgezeichnet.

Beispiel 32 Deklaration des Makros SER

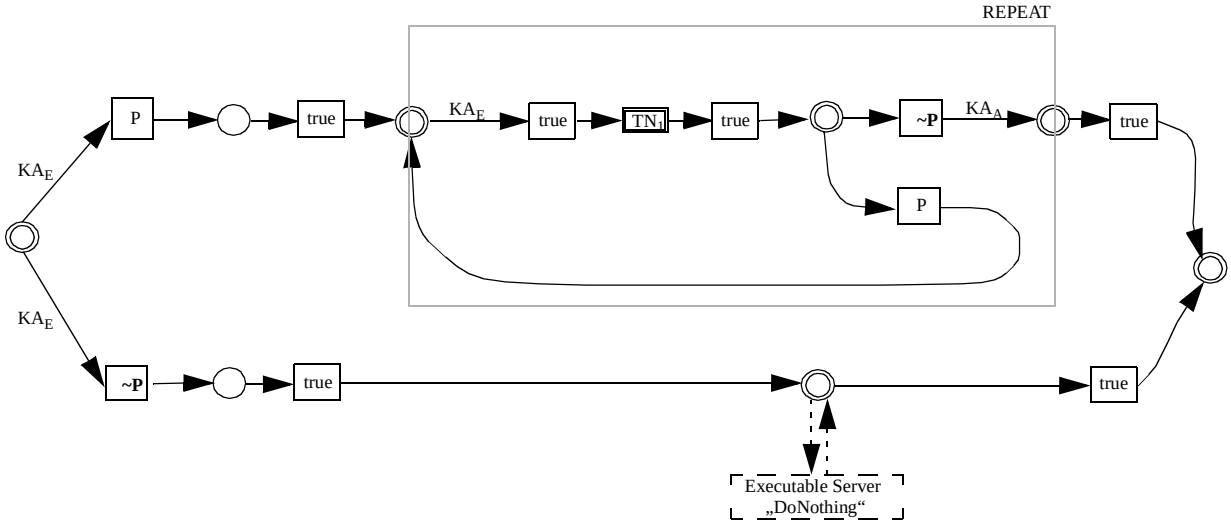
```
MACRO SER IS DynamicParmInterface;

BODY    #FOREACH (Zweige)
            ON p #EXEC m #EXCLUSIVE #ONCE_ONLY;
        #END

END.
```

8.2 Abgeleitete Makros

Nachfolgend werden mit Hilfe der im letzten Abschnitt vorgestellten Makros neue Makros definiert. Die kopfgesteuerte Schleife *WHILE* dient als Beispiel eines reinen Kontrollflußmakros, bei *Umlauf* handelt es sich dagegen um ein Hybridmakro, bei dem Datenfluß und Organisation zusammentreffen.

8.2.1 Kopfgesteuerte Schleife (WHILE)

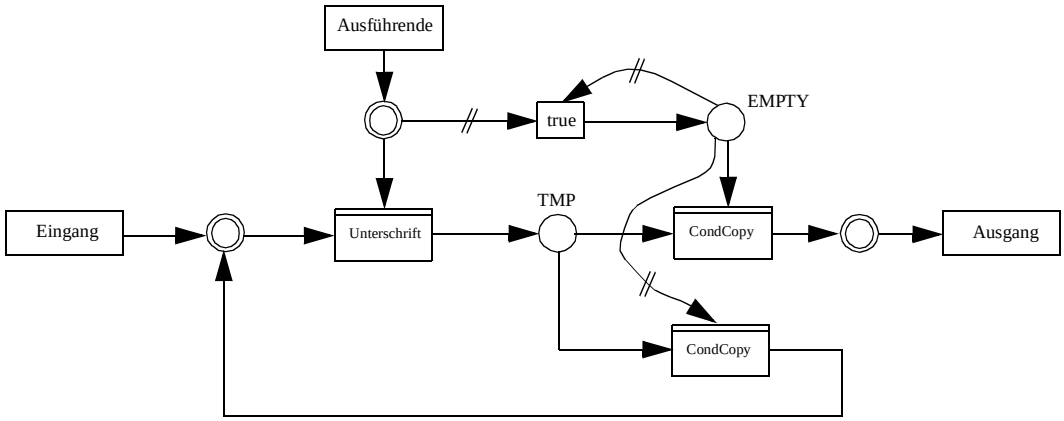
```
MACRO   While IS IterationInterface;
BODY    #MACRO_USE IF(p, #MACRO_USE Repeat(p, m), #EXEC DO_NOTHING);
END.
```



Die kopfgesteuerte Schleife prüft im Gegensatz zur fußgesteuerten Schleife die Abbruchbedingung vor Ausführung des Makros. Sie kann jedoch leicht aus der REPEAT Schleife abgeleitet werden, indem durch ein vorangestelltes IF die Bedingung geprüft wird. Danach kann mit Hilfe von REPEAT die eigentliche Iteration durchgeführt werden. Am entstehenden Netz ist auch sehr schön der Unterschied zwischen Expansion (#MACRO_USE) und Executable-Server-Aufruf (#EXEC) zu sehen. Das REPEAT-Makro wurde durch seine Netzdefinition ersetzt, das (ausführbare) *DoNothing* durch eine magische Stelle, die die Kommunikation mit dem Executable-Server übernimmt.

8.2.2 Umlauf

```
MACRO Umlauf IS NEW UmlaufInterface;
IN:   Eingabe      [DF,   DOMAIN Doc1],
      Ausführende  [ORG,  DOMAIN human]
OUT:  Ausgabe      [DF,   DOMAIN Doc1];

MACRO Unterschrift IS UnterschriftInterface;
END.
BODY:  LOCAL TMP[DF, DOMAIN Doc1];
       #EXEC Unterschrift
            GIVING Eingabe TO Eingabe, Ausführende TO Ausführender
            TAKING Ausgabe TO TMP;
       END.
       #MACRO_USE CondCopy()
            GIVING TMP TO Eingabe, EMPTY TO COND
            TAKING Ausgabe TO Ausgabe;
       END.


       #MACRO_USE CondCopy()
            GIVING TMP TO Eingabe, NOT EMPTY TO COND
            TAKING Ausgabe TO Ausgabe;
       END.
END.
```



Eine weitverbreitete Form, ein Dokument in einer Gruppe jedem zugänglich zu machen, ist der Umlauf. Ein Dokument wird mit einem Laufzettel versehen, auf dem alle vermerkt sind, die dieses Dokument zur Kenntnis nehmen sollen. Wer das Dokument gelesen hat, unterschreibt den Laufzettel und gibt es weiter. Die Reihenfolge der Weitergabe spielt dabei keine Rolle.

Dieses Verhalten wird vom Makro *Umlauf* nachgebildet. Als Eingabe erhält es eine Menge Ausführender, die zuvor vom Organisationsmanager erfragt werden kann, und ein Dokument. Das Makro *Unterschrift* dient zum Verarbeiten des Dokuments. Solange noch nicht alle Ausführende von *Unterschrift* verbraucht worden sind, wird das bearbeitete Dokument wieder zur weiteren Bearbeitung bereitgestellt. Sind alle mit der Bearbeitung fertig, liegt das Endergebnis an der Ausgangstransition bereit.

Kapitel 9 Zusammenfassung und Ausblick

Wir haben uns in dieser Arbeit zunächst mit der Klärung des Begriffs „Workflow“ beschäftigt und zwei Ansätze näher betrachtet. Das Modell der *Workflow Management Coalition* (WfMC) identifiziert verschiedene Einheiten, die zur Definition eines Prozesses notwendig sind: Prozeßaktivitäten, Daten, Ausführende, Anwendungen und Transitionen als Verbindung zwischen Aktivitäten. Als zweiten Ansatz haben wir das aspektorientierte Modell von MOBILE vorgestellt. Dieses beschreibt einen Prozeß aus verschiedenen Blickwinkeln: der *funktionale Aspekt* beschreibt, was ausgeführt wird, der *verhaltensbezogene Aspekt*, wann es ausgeführt werden soll. Datenfluß wird im *informationsbezogenen Aspekt* behandelt, wer ausführt im *organisatorischen*.

Wir haben festgestellt, daß sich bestimmte Strukturen in Workflows wiederholen. Daher haben wir den Begriff des *Makros* eingeführt. Makros sind vordefinierte Workflow-Bausteine, die über Parameter an das jeweilige Problem angepaßt werden können. Zur Verfügung gestellt werden dem Workflow-Management-System die Makros von einer Makrokomponente, die die Verwaltung und die Abbildung auf eine ausführbare Implementierung übernimmt. Wir haben verschiedene einfache Makros vorgestellt, die zum Aufbau komplexerer Strukturen benutzt werden können. Wir haben festgestellt, das schon wenige dieser einfachen Makros, die wir als *Grundmakros* bezeichnen, ausreichen, um die gängigen Kontrollstrukturen, wie z. B. bedingte Verzweigung, Iteration oder Parallelausführung, aber auch Datenfluß zu verwirklichen. Makros erlauben eine bei der Modellierung eine saubere Trennung der Aspekte.

Zur Modellierung des Datenflusses wurden ebenfalls einfache Makros eingeführt, die beispielsweise das Kopieren eines Dokuments oder Extrahieren eines Teildokuments ermöglichen.

Anschließend haben wir unsere Makroidee um zwei wichtige Punkte erweitert. Durch Einführen von *Hybridmakros* haben wir die Möglichkeit geschaffen, verschiedene Aspekte zusammenführen zu können. Zum Ausführen einer Applikation benötigt man den Zeitpunkt (Kontrollfluß), die zu verarbeitenden Daten (Datenfluß) und einen oder auch mehrere Ausführende (Organisation). Hybridmakros formen solche Kreuzungspunkte zwischen den Aspekten.

Die zweite Erweiterung betrifft *polymorphe Makros*. Workflows können sehr lange Laufzeiten haben, sie müssen folglich auf Änderungen der Umgebung, etwa eine Gesetzesänderung, reagieren können. Daher wird bei polymorphen Makros bei der Workflow-Definition nur ein Stellvertreter eingesetzt. Erst zum Ausführungszeitpunkt wird ein real existierendes Makro ausgewählt und ausgeführt („Late Binding“). Zur Auswahl eines geeigneten Makros steht ein *Workflow-Kontext* zur Verfügung, der ein Bild der aktuellen Situation vermittelt. Über *Auswahlattribute* kann somit ein passendes Makro bestimmt werden.

Wir haben eine Definitionssprache vorgestellt, mit deren Hilfe Makros definiert werden können. Es wurde der Begriff des *Interface* eingeführt, der eine Typisierung von Makros erlaubt und eine zuverlässige Parameterübergabe ermöglicht. Außerdem wurden Mechanismen zur Nutzung von Makros mit variabler Parameteranzahl vorgestellt (Kollektionen, parallele und rekursive Expansion).

Um Makros in einer ausführbaren Form zu implementieren, haben wir uns dazu entschieden, diese in Prozeßkontrollnetze abzubilden. Nach einer kurzen Beschreibung der grundlegenden Eigenschaften von Petri-Netzen haben wir erläutert, wie diese zu Produktnetzen erweitert werden. Anschließend haben wir die Veränderungen beschrieben, die schließlich zu Prozeßkontrollnetzen führen. Dabei wurden insbesondere die Kantenanschriften überarbeitet, Domänen neu definiert und das Konzept der *magischen Stelle* eingeführt.

Wir haben uns mit dem Aufbau der Makrokomponente beschäftigt, die Verwaltung und Bearbeitung von Makros beschrieben und an einem Beispiel ausführlich den Weg von der Makrodefinition zum fertigen Netz betrachtet. Daran anschließend haben wir erläutert, wie ein Prozeß mit Hilfe der Makrokomponente definiert werden kann. Danach haben wir die Ablaufkomponente mit ihren Elemente zur Überwachung eines Ablaufs, bzw. seiner Analyse nach Ende der Ausführung, vorgestellt. Wir sind weiterhin näher auf Workflow-Kontexte und die Möglichkeit der Verwendung polymorpher Makros eingegangen.

Zum Abschluß haben wir die Abbildung von Makros auf Prozeßkontrollnetze an zahlreichen Beispielen gezeigt. Die Grundmakros wurden mit ihren zugehörigen Netzen vorgestellt, dazu die Umsetzung verschiedener abgeleiteter Makros, um verschiedenen Konzepte der Expansion zu verdeutlichen.

Wir haben bei dieser Arbeit gesehen, daß eine Zwischenschicht zwischen Prozeßdefinitionskomponente und Implementierungsschicht wertvolle Dienste leisten kann. Neben der Kapselung der eigentlichen Implementierung ergeben sich auch neue Möglichkeiten, wie der Einsatz von polymorphen Teil-Workflows. Die Definition von Prozessen im Workflow-Management-System wird vereinfacht, die verschiedenen Aspekte lassen sich voneinander getrennt sauber formulieren. Die Prozeßdefinition wird durch die Modularisierung übersichtlicher. Einmal definierte Teile eines Workflows können wiederverwendet werden und ersparen so Zeit beim Erstellen neuer Workflows.

Da uns noch keine geeignete Ablaufkomponente für Prozeßkontrollnetze zur Verfügung stand, konnten wir leider noch keine praktischen Erfahrungen mit der Makrokomponente sammeln. Dies wird aber in nächster Zukunft nachgeholt. Weiterhin werden wir untersuchen, inwieweit sich dieser Ansatz zur Abbildung von Workflows in heterogene Ablaufumgebungen eignet.

Kapitel 10 Literaturverzeichnis

[Ja95] Stefan Jablonski
Workflow-Managment-Systeme – Modellierung und Architektur
Thomsons Aktuelle Tutorien, Int. Thomson Publ.
Bonn, 1995

[JBS97] Stefan Jablonski, Markus Böhm, Wolfgang Schulze(Hrsg.)
Workflow-Management – Entwicklung von Anwendungen und Systemen
dpunkt-Verlag
Heidelberg, 1997

[Po96] Stefani Poßner
Entwicklung eines Prozeßmodells für Workflow-Managment-Systeme am Beispiel der Warenwirtschaft in Handelsunternehmen
Diplomarbeit, Universität Kaiserslautern, Fachbereich Informatik
Kaiserslautern, 1996

[Strä99] Markus Sträter
Integration einer Ablaufsteuerungskomponente in ein objekt-relationales DBMS – Konzeption und Implementierungsaspekte
Diplomarbeit, Universität Kaiserslautern, Fachbereich Informatik
Kaiserslautern, 1999

[GJS96] James Gosling, Bill Joy, Guy Steele
The Java™ Language Specification
Addison-Wesley, 1996

[WfMC98] Workflow Management Coalition
Work Group 1
Interface 1: Process Definition Interchange — Process Model
November 1998
Document Number WfMC TC-1016-P
www.aiim.org/wfmc/mainframe.htm

[WfMC99] Workflow Management Coalition
Terminology & Glossary
Februar 1999
Document Number WfMC TC-1011
www.aiim.org/wfmc/mainframe.htm

[ISO97] *High-level Petri Nets – Concepts, Definitions and Graphical Notation*
Committee Draft ISO/IEC 15909
Oktober 1997
www.daimi.aau.dk/PetriNets

[Ni99] Udo Nink
Anbindung von Entwurfsdatenbanken an objektorientierte Programmiersprachen
Shaker Verlag
Aachen, 1999
Zugl.:Kaiserslautern, Univ., Diss., 1999

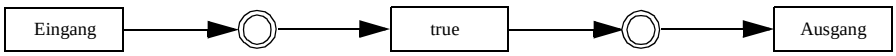
[Bußß98] Christoph Bußler
Organisationsverwaltung in Workflow-Management-Systemen
Deutscher Universitäts Verlag
Wiesbaden 1998

Anhang A Abbildung von Makros auf PCN

A.1 Grundmakros

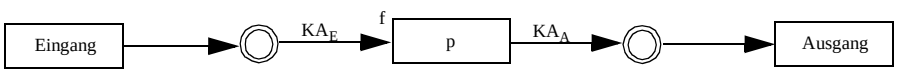
A.1.1 Das leere Makro (DoNothing)

INTERFACE NEW DoNothingInterface;
IN: Eingang [*, DOMAIN *];
OUT: Ausgang[*, DOMAIN *];
END;



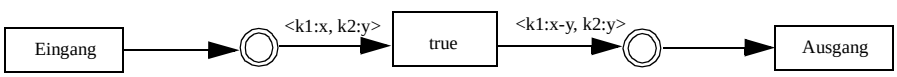
A.1.2 Funktionsmakros

INTERFACE NEW FunctionInterface;
IN: Eingang [CF, DOMAIN EingabeDomain];
OUT: Ausgang[CF, DOMAIN AusgabeDomain];
END;



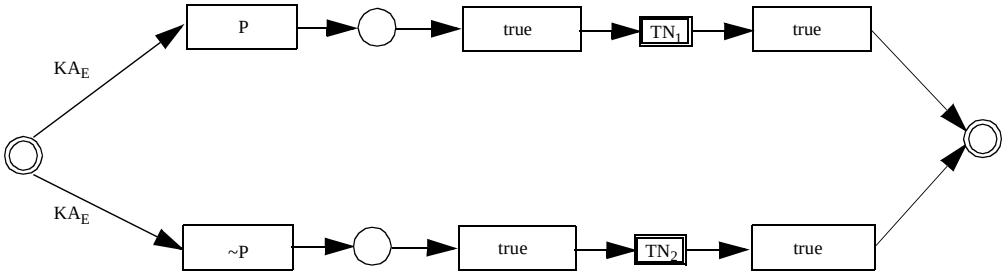
SUBx-y

INTERFACE NEW DualOpInterface;
IN: Eingang [CF, (k1: INT, k2: INT)];
OUT: Ausgang[CF, (k1: INT, k2: INT)];
END;



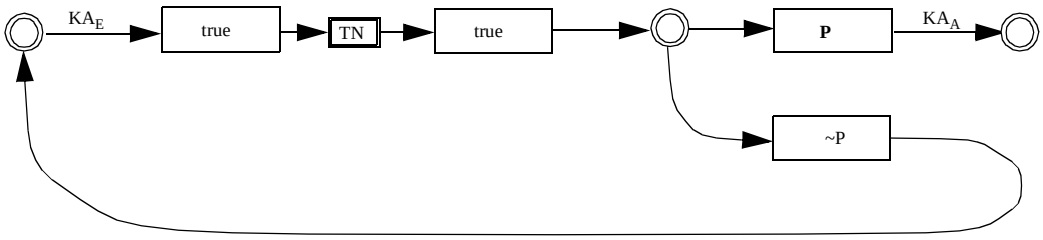
A.1.3 Die bedingte Verzweigung (IF)

INTERFACE NEW CondInterface;
IN Eingang [CF, DOMAIN *];
OUT Ausgang [CF, DOMAIN *];
FUNCTION p OF BOOLEAN;
MACRO M_1 IS MacroInterface;
MACRO M_2 IS MacroInterface;
END;



A.1.4 Die Iteraration (REPEAT)

INTERFACE NEW IterationInterface;
IN Eingang [CF, DOMAIN *];
OUT Ausgang [CF, DOMAIN *];
FUNCTION p OF BOOLEAN;
MACRO M IS MacroInterface;
END;

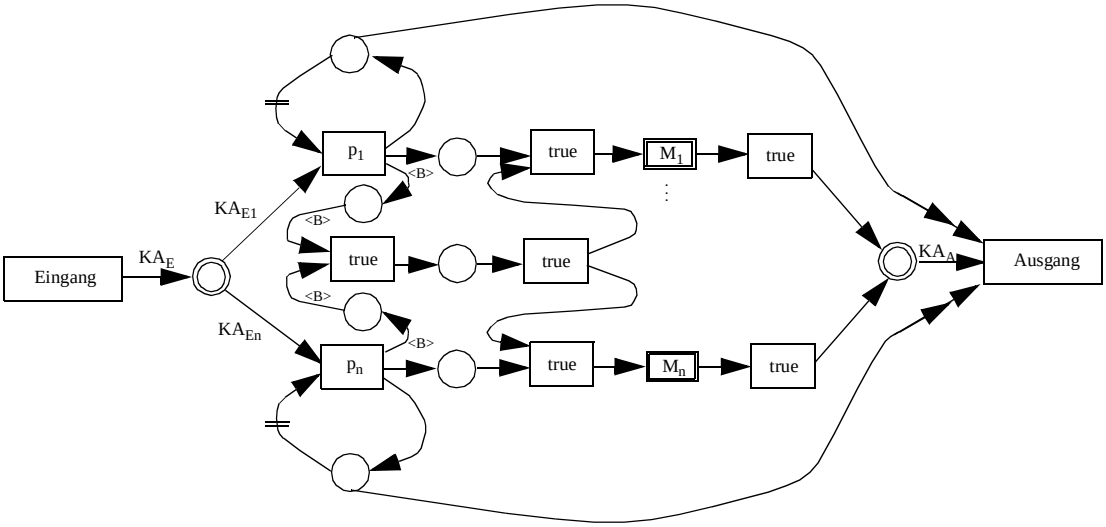


A.1.5 Die Parallelität

MACRO PAR IS DynamicParmInterface;

BODY#FOREACH (Zweige)
ON p #EXEC m #BLOCKED #ONCE_ONLY;
#END

END.

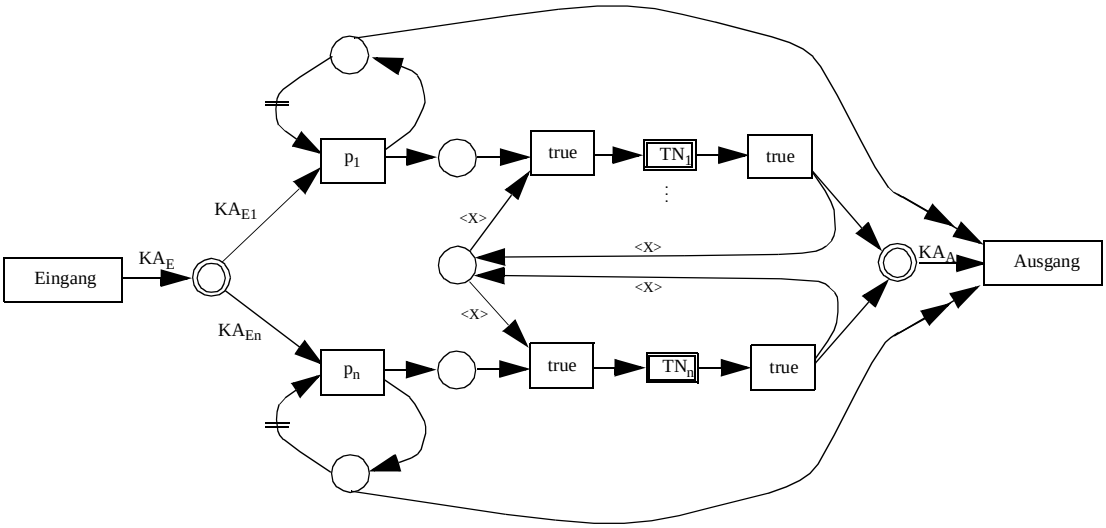


A.1.6 Die Serialisierung

MACRO SER IS DynamicParmInterface;

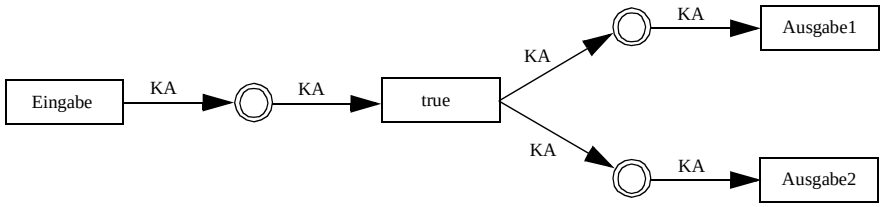
BODY#FOREACH (Zweige)
ON p #EXEC m #EXCLUSIVE #ONCE_ONLY;
#END

END.



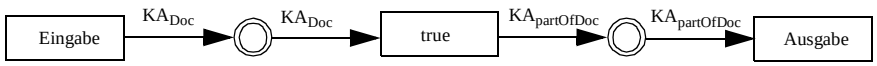
A.1.7 Dokument kopieren (copy)

MACRO copy IS NEW CopyInterface;
IN: Eingabe [DF, DOMAIN *];
OUT: Ausgabe1[DF, DOMAIN *],
Ausgabe2[DF, DOMAIN *];



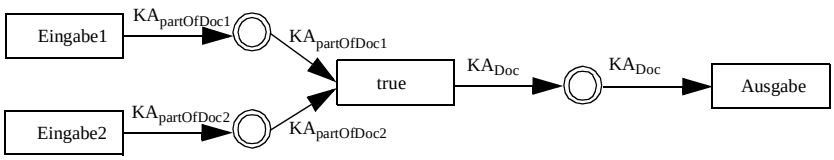
A.1.8 Datum extrahieren

MACRO extract IS NEW ExtractInterface;
IN: Eingabe[DF, DOMAIN Doc];
OUT: Ausgabe[DF, DOMAIN partOfDoc];



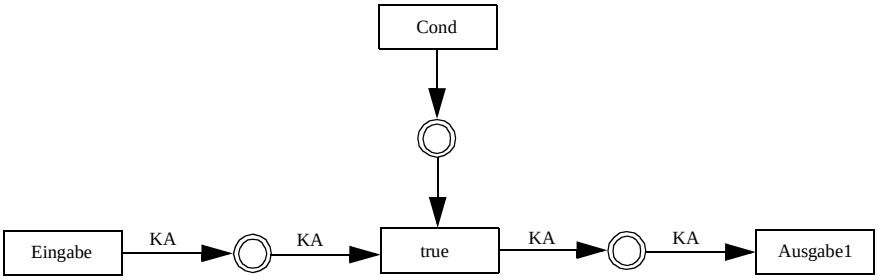
A.1.9 Datum verschmelzen

MACRO melt IS NEW MeltInterface;
IN: Eingabe1[DF, DOMAIN partOfDoc1],
Eingabe2[DF, DOMAIN partOfDoc2]
OUT: Ausgabe [DF, DOMAIN Doc];



A.1.10 Dokument bedingt kopieren (condCopy)

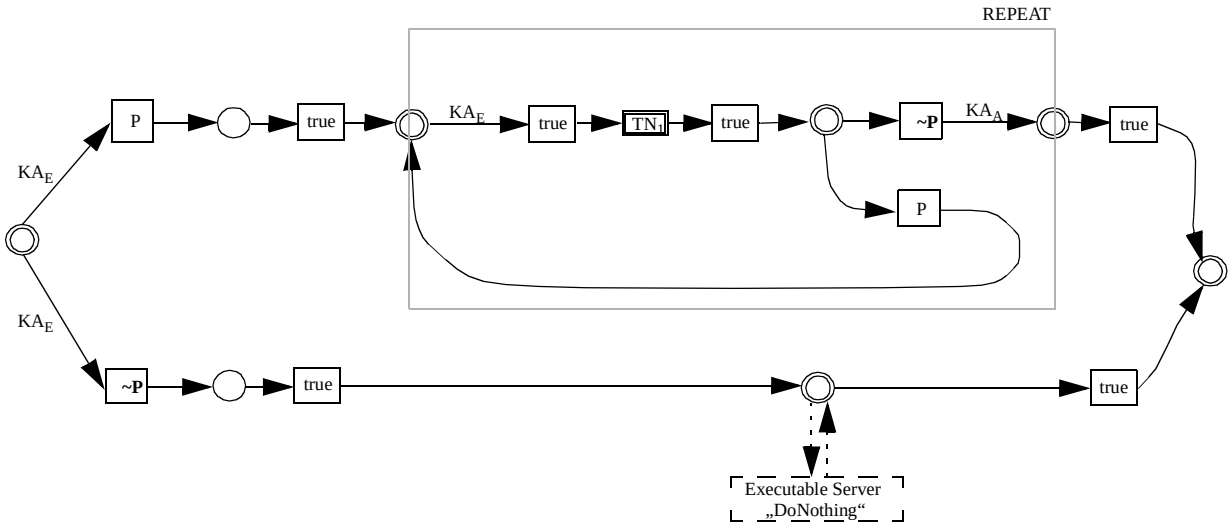
MACRO copy IS NEW condCopyInterface;
IN: Eingabe [DF, DOMAIN *],
Cond [CF, GENERIC];
OUT: Ausgabe [DF, DOMAIN *];



A.2 Abgeleitete Makros

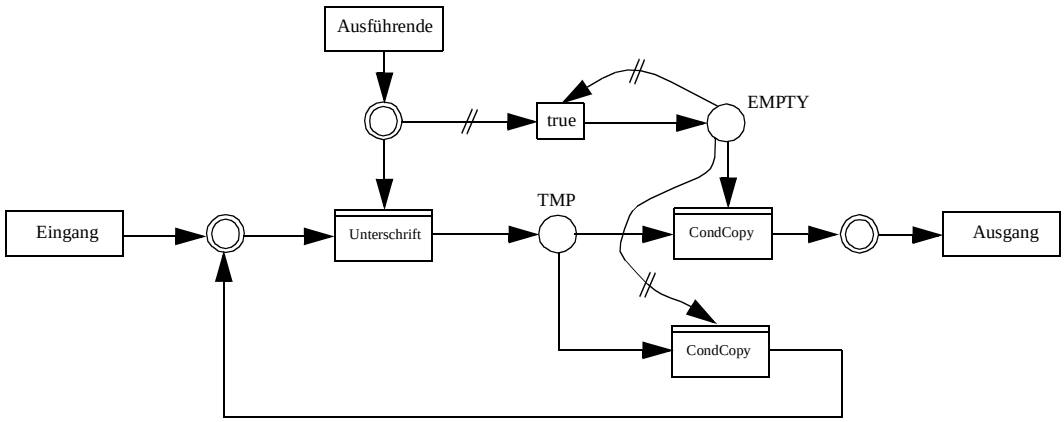
A.2.1 Kopfgesteuerte Schleife (WHILE)

MACRO While IS IterationInterface;
BODY #MACRO_USE IF(p, #MACRO_USE Repeat(p, m), #EXEC DO_NOTHING);
END.



A.2.2 Umlauf

```
MACRO Umlauf IS NEW UmlaufInterface;
IN:   Eingabe      [DF,   DOMAIN Doc1],
      Ausführende  [ORG,  DOMAIN human]
OUT:  Ausgabe      [DF,   DOMAIN Doc1];

MACRO Unterschrift IS UnterschriftInterface;
END.
BODY:   LOCAL TMP[DF, DOMAIN Doc1];
        #EXEC Unterschrift
              GIVING Eingabe TO Eingabe, Ausführende TO Ausführender
              TAKING Ausgabe TO TMP;
        END.
        #MACRO_USE CondCopy()
              GIVING TMP TO Eingabe, EMPTY TO COND
              TAKING Ausgabe TO Ausgabe;
        END.
        #MACRO_USE CondCopy()
              GIVING TMP TO Eingabe, NOT EMPTY TO COND
              TAKING Ausgabe TO Ausgabe;
        END.
END.
```



Anhang B Sprachdefinition

B.1 Interface

interfaceDef = „INTERFACE“ newInterface.

newInterface = „NEW“ interfaceName [IS interfaceName [ident]{„ ,“ interfaceName [ident]} {correspondDef}]
[transition]
{function}
{macro}
{data}
„END“.

interfaceName = ident.

transitionName = ident.

correspondDef = interfaceName “.“ transitionName „CORRESPONDS“ transitionName.

transition = [„IN“ „:“ transitionName „[“ aspectType „ ,“ domainDef „]“
WITH EMPTYSTATUS ident
{„ ,“ transitionName „[“ aspectType „ ,“ domainDef „]“
WITH EMPTYSTATUS ident}]
| [„OUT“ „:“ transitionName „[“ aspectType „ ,“ domainDef „]“
{„ ,“ transitionName „[“ aspectType „ ,“ domainDef „]“ }].

aspectType = „CF“| „DF“| „HUMAN“

domainDef = domainName | ident „:“ BaseType.

function = „FUNCTION“ ident OF funcReturnType [„IS“ „(“ {funcParm}„)“].

funcParm = [datatype „:“ ident {„ ,“ datatype „:“ ident}].

macros = „MACRO“ ident „IS“ interfaceName.

data = simpleData | collectionData.

simpleData = dataType „:“ ident {„ ,“ ident}.

collectionData = „COLLECTION“ collectionName„:“„(“ function|macro|simpleData {„ ,“ function|macro|simpleData} „)“.

collectionName = ident.

B.2 Makros

```
macroDecl    = „MACRO“ macroName „IS“ (interfaceName | newInterface)
               [„LOCAL“ varDecl]
               [„TAG“ string]
               „BODY“ bodyDecl
               „END.“.

macroName    = ident.

varDecl      = ident „[“aspectType „ ,“ domainDef „]“.

bodyDecl     = expandDecl|ifDecl|foreachDecl.

expandDecl   = „#MACRO_USE“ macroName „(“ parmList „)“ connectionDecl
             | „#EXEC“ macroName (connectDecl | varConnectDecl).

connectDecl  = [„#GIVING“ transitionName]
               [„#TAKING“ transitionName]

varConnectDecl = [„#GIVING“ varName „#TO“ transitionName
                 {„ ,“ varName „#TO“ transitionName}]
                 [„#TAKING“ varName „#FROM“ transitionName
                 {„ ,“ varName „#FROM“ transitionName}]

parmList     = {functionName „(“ {ident | dataConst} „)“}
             | collectionName
             | „#FIRST“ „(“ collectionName „)“.ident
             | „#REST“ „(“ collectionName „)“
             | dataConst}.

ifDecl       = „#IF“ „(„ condDecl „=“ boolConst „)“
                  bodyDecl
               „#ELSE“
                  bodyDecl

               „#END“.

foreachDecl  = „#FOREACH“ „(“ collectionName „)“
                  „#ON“ condName
                     bodyDecl
                  [„#BLOCKED“][„#ONCE_ONLY“][„#EXCLUSIVE“]

               „#END“.

condDecl     = „#IS_EMPTY“ „(“ collectionName „)“.

boolConst    = „#TRUE“|„#FALSE“.
```