Arbeitstechniken für die Softwareentwicklung

Hinrich E. G. Bonin[1]

14-Feb-1994 – 30-Jun-2004

[1]Prof. Dr. rer. publ. Dipl.-Ing. Dipl.-Wirtsch.-Ing. Hinrich E. G. Bonin, University of Applied Sciences, Fachhochschule Nordostniedersachsen, Volgershall 1, D-21339 Lüneburg, Germany.

Kurzfassung

„Informatik hat viele Facetten.“([17] S. V). Eine davon ist die Bildung von Modellen. Arbeitstechniken für die Softwareentwicklung, ein Buch zum *Selbststudium* und für *Lehrveranstaltungen*, vermittelt Wissen zum Modellieren von neuen und existierenden Softwaresystemen. Modellieren ist dabei ein (Abstraktions-)Prozeß, um wesentliche Aspekte des Systems und dessen Verhalten durchschaubar darzustellen. Aus der großen Menge mehr oder weniger formaler Darstellungsmittel wurden Entscheidungstabellen, Ablaufpläne, Struktogramme, Pseudocode-Notation (Process Design Language), (Petri-)Netze und Ward/Mellor-Diagramme ausgewählt. Diese Wahl berücksichtigt einerseits den Verbreitungsgrad und ermöglicht andererseits nicht nur das Darstellen von sequentiellen sondern auch von nebenläufigen Vorgängen.

Abstract

“Computer Science has many facets.” ([17] p. V) One is building models. Arbeitstechniken für die Softwareentwicklung, a book for self-study and for lectures, presents practical knowledge to build models for new and existing software systems. To work out such models is a process of abstraction with the target of making transperent the relevant aspects and behavior of the system. Decision tables, flow diagrams, Nassi/Shneiderman-diagrams, process design language (pseudo code), (Petri-)nets and Ward/Mellor-diagrams are selected from the big set of more or less formal means of representation. On the one hand this selection takes into consideration the commonly used means and on the other hand the option to represent sequential and concurrent processes.

Vorwort

Das Zeichnen von Kreisen, Rechtecken, Pfeilen und ihre Beschriftung sind wesentliche Bestandteile der Softwareentwicklung. Bildlich formuliert: Wie das Ballspielen einen Ball bedingt, so erfordert das Entwicklen von Software den Einsatz von graphischen Darstellungen. Aus Kreisen, Rechtecken und Pfeilen entstehen Ablaufpläne, Jackson-Diagramme, Struktogramme, Petri-Netze und vieles andere mehr. Das Verstehen solcher Pläne, Diagramme, Netze etc., – kurz und ein wenig kritisch „*boxo-logie*“ genannt – ist für alle Beteiligten und Betroffenen zwingend notwendig.

Boxologie

Diese Buch behandelt die „*boxo-logie*“ im Zusammenhang mit Arbeitstechniken, das heißt im Kontext von Prinzipien, Methoden und Produktionshilfen („Werkzeugen“). Es skizziert zunächst Spannungsfelder bei der Softwareentwicklung. Vor diesem Hintergrund erläutert es bewährte, weit verbreitete Arbeitstechniken. Ausführlich dargestellt ist das Modellieren mit Entscheidungstabellen, Struktogrammen, Pseudocode-Notation und (Petri-)Netzen. Erläutert sind neben den theoretischen Grundlagen praxisorientierte Handlungsempfehlungen mit vielen Beispielen inklusive Übungsaufgaben mit Lösungen.

Die erörterten Arbeitstechniken ermöglichen einerseits die Präzisierung und Formalisierung von (weitgehend) sequentiellen Abläufen, andererseits auch die von nebenläufigen Abläufen. So entfällt die populäre „Sequentiallisierungsschere“ im Analysestadium. Man ist nicht mehr gezwungen, „Nebenläufiges“ sequentiell abzubilden, nur weil die geeignete Arbeitstechnik fehlt. Mit der Technik von (Petri-)Netzen sind daher zusätzliche Problemarten graphisch modellierbar.

Dieses Buch ist konzipiert als ein *Arbeitsbuch zum Selbststudium und für Lehrveranstaltungen.* Die Leserin und der Leser ohne Vorkenntnisse mögen es Abschnitt für Abschnitt durcharbeiten. „Vorbelastete“ können sich zunächst anhand der Zusammenfassungen der einzelnen Abschnitte orientieren. Damit das Buch auch als Nachschlagewerk hilfreich ist, enthält es sowohl Vorwärts- wie Rückwärts-Verweise.

Arbeitsbuch

Wie jedes Fachbuch so hat auch dieses eine lange Geschichte. Begonnen wurden die ersten Aufzeichnungen im Jahre 1985 als Material für eine Vorlesung „Kontrollstrukturen“ an der Hochschule Bremerhaven, Studiengang Systemanalyse [8]. Erst 1991 wurden die zwischenzeitlich lose und kaum systematisch gesammelten Notizen neu überarbeitet und als FINAL, Heft 1 Mai 1991 (ISSN 03939-8821) *Systemanalyse – Arbeitstechniken* herausgegeben. Ergänzt und modifiziert entstand daraus dieses Buch.

Während der Arbeit am Manuskript lernt man erfreulicherweise stets

dazu. Das hat jedoch auch den Nachteil, daß man laufend neue Unzulänglichkeiten am Manuskript erkennt. Schließlich ist es trotz solcher Schwächen der Öffentlichkeit zu übergeben. Ich bitte Sie daher im voraus um Verständnis für Unzulänglichkeiten. Willkommen sind Ihre konstruktiven Vorschläge, um die Unzulänglichkeiten Schritt für Schritt weiter zu verringern.

Danksagung:

Für das Interesse und die Durchsicht einer vorhergehenden Fassung danke ich meinen Kollegen Prof. Dr. Fevzi Belli (Universität Paderborn), Prof. Dr. Jürgen Burgermeister (FH Nordostniedersachsen) und Prof. Dr. Volker Speidel (Hochschule Bremerhaven). Ohne die kritischen Diskussionen mit Studentinnen und Studenten im Rahmen der Lehrveranstaltungen: Kontrollstrukturen (Hochschule Bremerhaven) und Anwendungsprogrammierung (Fachhochschule Nordostniedersachsen) wäre dieses Buch nicht in dieser Form entstanden.

Lüneburg, September 1992 Hinrich E. G. Bonin

Vorwort zur 2. und 3. Auflage

Wer arbeitet macht Fehler – und findet auch welche! Erfreulicherweise haben viele „Durcharbeiter“ Druckfehler und Layout-Unzulänglichkeiten entdeckt, die jetzt beseitigt werden konnten. Zusätzlich sind weitere Übungsaufgaben und Klausuraufgaben aufgenommen worden. Dazu gekommen sind im Anhang Informationen zum Leistungsnachweis im Fach „Anwendungsprogrammierung“ (APG) an der FH Nordostniedersachsen, Lüneburg. Die Notation für ein ET-Verbundsystem wurde im Hinblick auf eine Anfangssequenz ergänzt.

Lüneburg, Juli 1993 & Februar 1994 Hinrich E. G. Bonin

Vorwort zur 1. Web-Auflage

Ein Manuskript, welches in der Lehre eingesetzt wird, sollte den Studierenden über das *World Wide Web* verfügbar sein. Daher wurde die 3. Auflage von ihrer Ursprungsfassung als LaTeX-Dokument in ein Adobe PDF-Dokument überführt:

```
http://as.fhnon.de/publikation/sysall.pdf
```

Jetzt sind die einzelnen Abschnitte direkt referenzierbar, einerseits über das Inhaltsverzeichnis und andererseits über den umfassenden Index.

Lüneburg, April 2001 Hinrich E. G. Bonin

Vorwort zur 2. Web-Auflage

Im rasanten Wandel der Informationstechnik ist es nicht einfach einen Text, zusammengesetzt aus vielen Dateien, auf der jeweils aktuellen Plattformen verfügbar zu halten. Diese Auflage ist primär bedingt durch eine notwendige Umstellung auf eine TEX- & LATEX-Plattform mit dem Betriebssystem *Microsoft Windows XP Professional*. Benutzt werden zur Zeit:

- Editor: GNU Emacs 21.2.1
- Layoutaufbereitung mit TEX & LATEX: Version 3.14159 (Web2c 7.3.7x); LaTeX2e
- Indexgenerierung: `makeindex`, Version 2.13, 07-Mar-1997
- DVI-Datei → Postscript-Datei: `dvips(k) 5.90a`; Copyright 2002 Radical Eye Software `http://www.radicaleye.com`
- Postscript-Datei → Adobe PDF-Datei: Acrobat Distiller^{TM} 5.0
- Sicherheitsmerkmale: Adobe Acrobat^{TM} 5.0

Mit der klassischen Schreibmaschine wurden wichtige Wörter unterstrichen. In Analogie dazu wurden in der ersten Web-Phase daher klickbare Verweise (*Links*) stets unterstrichen. In einem total vernetzen Dokumenten sind *Links* aber keine Besonderheit mehr, sondern eher der Normalfall. Jetzt sind sie daher nicht mehr besonders hervorgehoben.

Zunehmend sind Arbeitstechniken für die Softwareentwicklung von leistungsfähigen Softwarewerkzeugen geprägt. Das Manuskript verweist jetzt mittels *Links* auf solche *Tools*.

Lüneburg, April 2003 Hinrich E. G. Bonin

Vorwort zur aktuellen Web-Auflage

In den vorhergehenden Auflagen standen die ausgewählten Arbeitstechniken noch weitgehenden nebeneinander. Zunehmend tritt jedoch ein weitgefasstes Paradigma „Modellierung“ in den Fokus. Im Kontext der Informatik dient die Modellierung primär zwei Zielen:

1. die Schaffung des *Domänenmodells*,
   also die Schaffung der Abbildung eines Ausschnittes der realen Welt und die Darstellung der Aufgabe der Informationsverabreitung,

2. die Schaffung des *Systemmodells*,
   also die Schaffung einer Vorlage für die Arbeitsweise eines informatischen Systems.

Lüneburg, 30. Juni 2004 Hinrich E. G. Bonin

Notation

In diesem Buch wird erst gar nicht der Versuch unternommen, die weltweit übliche Informatik-Fachsprache Englisch zu übersetzen. Es ist daher teilweise „mischsprachig“: Deutsch und Englisch. Aus Lesbarkeitsgründen sind nur die männlichen Formulierungen genannt; die Leserinnen seien implizit berücksichtigt. So steht das Wort „Programmierer“ hier für Programmiererin und Programmierer.

Zeichen/Symbol	Erläuterung
$\equiv$	„definiert als“ bzw. „konstruiert aus“
$\leftarrow$	Zuweisung
$\neg$	Negation bzw. Komplement
&	Junktor für Nebenläufigkeit
;	Junktor für Sequenz

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Konfliktfelder der Softwareentwicklung

„Wann ist der erste Termin,
zu dem du nicht beweisen kannst,
daß du nicht fertig sein kannst.“
([40] Seite 151)

Der Begriff *Software* gehört heute zum Alltagsprachgebrauch. Er ist daher unscharf und wird mit unterschiedlichen Bedeutungen benutzt. Allgemein bezeichnet man mit Software eine Menge von Programmen mit und ohne Daten zusammen mit den Dokumenten, die für ihre Anwendung notwendig und nützlich sind. Software ist damit Programm plus Dokumentation. Software im engeren Sinne sind alle Texte, die den potentiell universellen Computer für eine bestimmte Aufgabe spezifizieren, insbesondere diejenigen Texte, die als Instruktionen (Anweisungen, Befehle, Kommandos etc.) der Veränderung von Daten dienen. Unter Software im weiten Sinne verstehen wir Programme einschließlich ihrer Programmdokumentation plus alle vorbereitenden Dokumente (Problemuntersuchung, Anforderungsanalysen, Entwürfe, . . .) plus alle nachbereitenden Dokumente (Testprotokolle, Integrationspläne, Installationsanweisungen, . . .).

Diese Interpretation fußt nicht nur auf dem Produkt„Software“, sondern bezieht auch den Prozeß der Entwicklung und der Fortschreibung (Erweiterung, Wartung / Pflege) des Produktes (zumindest zum Teil) ein.

Wegweiser

Der Abschnitt *Konfliktfelder der Softwareentwicklung* erläutert:

- das Konglomerat von vielfältige Entscheidungsfragen im Rahmen des Entwicklungsprozesses,
  ↪Seite 22 . . .
- den Betrachtungsstandort,
  ↪Seite 25 . . .
- das Problem der Formulierung in der Sprache des Anwendungsgebietes und der notwendigen Übersetzung in die Sprache der Informatik,
  ↪Seite 32 . . .
- das Wechselspiel zwischen Abstraktion und Konkretisierung anhand von allgemeinen Prinzipien und
  ↪Seite 37 . . .
- die Softwareentwicklung als Teilgebiet einer Ingenieurdisziplin.
  ↪Seite 41 . . .

1.1 Konglomerat von Enscheidungsfragen

Mikrosicht

Makrosicht

Der Entwicklungsprozeß von Software wirft im besonderen Maße konflikt- und risikobehaftete Entscheidungsfragen auf. Ihre globale Nennung scheitert an der Universalität des Computers. Die technokratische Mikrosicht betont häufig die Erreichung einer hohen Produktivität und Qualität unter Einhaltung vorgegebener Restriktionen (Kosten, Termine). Bei einer Makrosicht ist es die strategische Bedeutung für die Organisationseinheit (Unternehmung, Verwaltung) auf ihrem Wege in die„Informationsgesellschaft“, die als Orientierungspunkt ausschlaggebend ist. Dies führt dann zu Argumentationen wie:

> „In der Informationsgesellschaft haben wir die Produktion von Wissen systematisiert und dadurch unsere Geisteskraft verstärkt. Um ein Gleichnis aus dem Industriezeitalter zu verwenden – das Wissen wird heutzutage zur Massenproduktion, und dieses Wissen ist die Triebkraft unserer Wirtschaft.“ ([45] S. 30)

Ob die Softwareentwicklung tatsächlich im Zusammenhang mit einer Massenproduktion von Wissen zu verstehen ist, erscheint fraglich.[1] Hier diskutieren wir nicht weiter, ob eine gelungene Softwareentwicklung Ordnung in das heutige Chaos der Informations-Überschwemmung und -Verseuchung bringt und uns letztlich Wissen beschert. Wir wollen uns auch nicht auf eine unkritische Technokratensicht zurückziehen. Vielmehr interessiert uns die Softwareentwicklung aus der Perspektive einer Übertragung von Arbeit auf Computer.[2]

W-Fragen

In diesem Zusammenhang stellen sich die altbekannten Fragen nach dem Warum, Wozu, Weshalb usw. Wichtige dieser sogenannten „*W-Fragen*“ lassen sich wie folgt formulieren:

1. Was soll wozu entwickelt werden?
   Zu definieren sind Ziele–Zwecke, Nutznießer–Verlierer, Interessengegensätze. Im Mittelpunkt einer Zieldiskussion stehen:

   - die Rationalisierung,
   - die Qualitätsverbesserung, zum Beispiel präzisere Karten im Vermessungswesen, und/oder
   - neuartige Leistungen (Innovationen), zum Beispiel Entwicklungsprognosen durch Zusammenführung riesiger Datenmengen.

   Erfordert beispielsweise die Softwareentwicklung für eine programmierte Waschmaschine eine andere Arbeitstechnik als die Softwareentwicklung für ein rechnergestütztes Finanzwesen? Zu klären ist daher, ob trotzdem dieselben Grundsätze und Methoden einen universellen, „tragfähigen“ Rahmen bilden (vgl. Abschnitt 1.4 Seite 37).

2. Wer soll entwickeln?
   Führt der Team-Ansatz zum Erfolg? Man kombiniere Spezialisten (Fachexperten) aus dem Arbeitsbereich, in dem die Software eingesetzt werden soll, mit Spezialisten der Softwareentwicklung (Informatiker), gebe ihnen Mittel (Finanzen & Zeit) und erhält dann nach Ablauf der vorgegebenen Projektzeit die gewünschte Software. Wer soll in diesem Projektteam der maßgebliche Entwickler sein – quasi der Architekt des Gebäudes? Einer der Fachexperte oder ein Informatiker?

Team-Ansatz

[1] Eher mag die provokative These gelten: „Wir ertrinken in Informationen, aber hungern nach Wissen.“ ([45] S. 41).

[2] Näheres vgl. Abschnitt 1.2.5 Seite 27.

3. Wie soll entwickelt werden?
   Wie lautet das Leitmotto?
   Welches Denkmodell (Paradigma) ist geeignet?

Denkmodell

Ist Softwareentwicklung die Erzeugung eines „Produktes“, das maßgeblich vom ersten Impuls – dem Auftrag – geprägt ist und dessen Erstellung ein deduktiver Prozeß ist? Lassen sich ingenieurmäßig aus einem Softwareentwicklungsauftrag:

- die Problemstellung präzise ableiten,
- die Softwarekonstruktion vollziehen,
- das Ergebnis prüfen und freigeben?

deduktiver Prozeß?

Handelt es sich um eine klar bestimmbare Aktivitätenfolge, bei der das Ergebnis der vorhergehenden Aktivität den Lösungsraum der nachfolgenden Aktivität einschränkt (definiert)? Sind die einzelnen Phasen und die damit verbundenen Arbeiten im voraus festlegbar? Oder ist das klassische Denkmodell eines linearen Phasendurchlaufes ungeeignet und durch ein neues zu ersetzen? Vielleicht durch ein Denkmodell der permanenten Wissenszunahme, charakterisiert durch das Begriffspaar *lernendes System* (vgl. Abschnitt 3.1 Seite 58ff)

Bevor neue Lösungsansätze zu diskutieren sind, stellen sich weitere, grundsätzliche Fragen:

4. Wodurch unterscheidet sich eine Softwareentwicklung von einer klassischen, mechanischen Ingenieuraufgabe?

Ingenieuraufgabe?

Was sind zum Beispiel die Unterschiede im Vergleich zum Bau eines Schiffes? Sind signifikante Differenzen feststellbar? Warum nehmen wir nicht einfach die bewährten Rezepte aus dem Schiffbau, dem Städtebau etc.? Softwareentwicklung analog zur Architektur der Bauhausbewegung [48, 6]? Erzwingt die Differenz ein grundsätzlich anderes Handlungskonzept?

konträre Positionen!

Solche bunt zusammengewürfelten Fragen, die im Rahmen der Softwareentwicklung auftreten, erlauben vielfältige, teilweise konträre Antworten. Es gilt daher im voraus den Betrachtungsstandort des Beantworters (Autors) zu skizzieren.

Die Wahl des Betrachtungsstandortes[3] ist von genereller Bedeutung für die Problemwahrnehmung und die vertretenen Lösungswege und Lösungen.

[3] Der auch „Interessenstandort“ sein mag.

> „Wenn man bis zum Hals in einem Fluß steht, sind die Strömungsgeschwindigkeit, die Wassertemperatur, der Verschmutzungsgrad und die Entfernung zum Ufer für einen um einiges wichtiger als die Schönheit der Landschaft, durch die der Fluß fließt.“ ([5] Seite 21)

1.2 Betrachtungsstandort

. . . und aus dem Chaos
sprach eine Stimme zu mir:
„Lächle und sei froh,
es könnte schlimmer kommen!“
. . . und ich lächelte und war froh
und es kam schlimmer . . . !

(verbreitete Ingenieurerfahrung)

Dazu postulieren wir vier Arbeitsthesen:

1. Differenziertes Zukunftsbild
2. Kritik an der klassischen Softwareentwicklung
3. „Human Factor“-Dominanz
4. Skepsis vor „Profis“

Diese Arbeitsthesen, die im Sinne des Begriffs Postulat einsichtig sein sollten, werden kurz skizziert.

1.2.1 Differenziertes Zukunftsbild

Softwareentwicklung ist eine wesentliche Triebfeder des technischen Fortschritts. Diesem Fortschritt sind wir nicht ausgeliefert.

> „Weder treiben wir auf apokalyptische Abgründe zu, noch werden wir in paradiesische Gefilde emporgetragen.“ ([49] Seite 10)

Dieses differenzierte Bild zeigt nicht die vielzitierte „neutrale Technik“. Beim einfachen Gegenstand, dem einfachen Werkzeug – was der Computer und damit auch der PC nicht ist – zum Beispiel einem Messer kann zwischen Instrument und Zweck (Brotschneiden oder Mord) berechtigterweise unterschieden werden. Beim Panzer beispielsweise nicht; er dient dem Zweck der Vernichtung. Er ist kein neutrales „Werkzeug“, sondern ein zweckorientiertes Gerät. Analog dazu ist die Softwareentwicklung ebenfalls zweckorientiert zu sehen, das heißt es gibt Grenzen für die verantwortbare Softwareentwicklung (vgl. Abschnitt 1.2.6 Seite 30).

zweckorientierte Technik

1.2.2 Kritik an der klassischen Softwareentwicklung

Die herkömmliche Softwareentwicklung orientiert sich am Leitmotto „Denke wie ein Computer!“. Gerade dieses Motto ist zu kritisieren.[4]

Obwohl manche der heutigen Handlungsempfehlungen diese „historische“ Vorgehensweise propagieren, ist sie überholt. Es ist nicht zunächst darüber nachzudenken, was der Computer als ersten Schritt vollziehen müßte, dieses dann zu dokumentieren und dann zu überlegen was der Computer als nächsten Schritt zu tun hätte, und so weiter ... Diese „Gedanken“ in Computerschritten lassen sich heute sicherlich in größeren Schritten beschreiben, um sie dann später zu einer Reihe kleinerer Schritte zu verfeinern. Trotz alledem ist dieses eher historisch erklärbare und einstmals bewährte Motto zu verwerfen. Heute sind Denkwelten (Paradigmen) erfolgversprechend, unabhängig von der konkreten Reihenfolge von Vollzugsschritten der Hardware. Zu nennen sind zum Beispiel funktional- und objektgeprägte Denkwelten.[5]

1.2.3 „Human Factor“-Dominanz

Die Softwareentwickler sollten von den Technokraten gelernt haben:

Es klappt nicht!

Durch die feingesponnenen Netze der Netzpläne, Phasenpläne, Entscheidungstabellen und Organigramme entschlüpft immer wieder der Mensch als nicht programmierbares Wesen.[6]

Die Gestaltung der Mensch-Maschine-Kooperation[7] kann nicht Hard-/-Software zentriert erfolgen. Sie muß den Menschen als dominierenden Gestaltungsfaktor, als „*human factor*“, konkret einbeziehen.

1.2.4 Skepsis vor Technikprofis

Niemand kann bezweifeln, viele Softwareentwicklungen sind gescheitert.[8] Häufig wird die Ursache den Anwendern (Auftraggebern & Benutzern) zugewiesen, weil diese sich nicht präzise artikulieren können; ihre Motive, Wünsche, Interessen, Anforderungen etc. bleiben unklar. Der Ruf nach dem „*emanzipierten Anwender*“, der ordnungsgemäße, leicht

Anwender-schuld

[4] Vgl. zur Kritik zum Beispiel [9].

[5] Vgl. hierzu zum Beispiel [10].

[6] Vgl. hierzu zum Beispiel [33].

[7] Vgl. Abschnitt 1.2.5 Seite 27

[8] Zum Beispiel: Grundbuchautomation? Integrierte Transportsteuerung der Bundesbahn (kurz: ITS)? Management Informationssysteme für einen gesamten Konzern, für ein Bundesland, für die Bundesrepublik insgesamt ...?

realisierbare Pflichtenhefte formuliert, wird von den Softwareprofis immer mächtiger.

Vielfältige Projekterfahrungen zeigen die Medaille eher von der anderen Seite (vgl. dazu auch [33] S. 64):

- Die „Profis“ sind verliebt in ihre Babies, so daß sie alles abwehren – bis es zu spät ist.
- Die „Profis“ konzentrieren sich auf rein technische Aspekte und sehen die soziale Seite der Veränderung unzureichend.
- Die „Profis“ stellen ein fiktives technisches Optimum über die Akzeptanz.
- Den „Profis“ fehlt es an Respekt vor Tradition und Improvisation.

1.2.5 Mensch-Maschine-Kooperation

Zur Abwägung des Nutzens und der damit verbundenen negativen Wirkungen (Riskien & Gefahren) ist der mittels Software spezifizierbare Computer primär als universelle Rationalisierungsmaschine zu betrachten. Universell heißt dabei:

- Alle Arbeits- und Lebensbereiche sind berührt.
- Alle Organisationsgrößen und Sektoren werden erfaßt.
- Alle Beschäftigten, unabhängig von Status und Qualifikation sind betroffen.

Diese universelle Betroffenheit wird überall spürbar. Jeder erwartet: **Ambivalenz**

- einerseits positive Computerbeiträge, zum Beispiel wissenschaftlichen Fortschritt in der Medizin, Wettbewerbsfähigkeit der Wirtschaft oder Befreiung von stumpfsinniger Routinearbeit, und
- andererseits negative Folgen, zum Beispiel mehr Möglichkeiten zur Bürgerüberwachung, Arbeitsplatzvernichtung, stumpfsinnige Arbeit als Datenver- und -entsorger oder mehr Konzentrationsnotwendigkeit.

Der Einzelne gehört daher selten in ein Pro- oder Kontra-Lager, sondern ist in sich gespalten. Seine Grundeinstellung zur Softwareentwicklung ist ambivalent.

Das eine Softwareentwicklung auf dem Hintergrund „universelle Rationalisierungsmaschine“ kombiniert mit intrapersoneller ambivalenter

Bewertung nicht pauschal akzeptiert wird, ist offensichtlich. Insbesondere im sensiblen Bereich „gläserner Mensch“ ist massiver Widerstand angebracht.

Unstrittig sind im Rahmen der Übertragung von betrieblichen Aufgaben auf Computer folgende Fakte (vgl. [15]):

> Ein Hard-/Softwaresystem übernimmt von arbeitenden Menschen (Hilfs-, Fach- und/oder Sachbearbeiters) Teile des Arbeitsprozesses, der Gestaltungs- und Kontrollmöglichkeiten. Es bildet einen Teil der Erfahrungen und Qualifikationen ab und tritt ihm zugleich als Ordnungsschema für den Arbeitsprozeß insgesamt wieder gegenüber. Vom Management übernimmt das System einen Teil der Führungs- und Organisationsaufgaben, indem es Prozesse abbildet, nach denen die Arbeit zwingend abzuwickeln ist. Damit übt es Kontrollfunktionen selbst aus.

Aus dieser Transformationsperspektive[9] stellen sich für die Softwareentwicklung folgende Fragen:

- Wie schwierig (komplex & kompliziert) ist die in Betracht gezogene Arbeit?
- In welchen bewältigbaren „Brocken“ ist sie aufteilbar?
- Geht es um eine sogenannte „*eins-zu-eins*“-Automation oder ist ein neues Abwicklungsmodell für die Arbeit erst zu entwerfen?
- Was sind in diesem Zusammenhang Muß- und Wunschkriterien?
- Wo sind die kritischen Bereiche und Schnittstellen? Wann ist „anspruchsvolle“ Software erforderlich? Wann reicht die „08/15“-Software?
- Wie zuverlässig muß gearbeitet werden? Wie lange kann der „Kollege Computer“ krankfeiern?

Aus der Sicht des Informatikers formuliert, drängen sich folgende Aspekte auf:

- Geht es um eine Übertragung monotoner, klar strukturierter und wohl definierter Prozesse oder um komplexe Prozesse der Informationsverarbeitung, die intelligenten Umgang mit diffusem Wissen erfordert?

[9] Innerhalb der zu ziehenden Grenzen, vgl. Abschnitt 1.2.6 Seite 30

- Sind die zu programmierenden Abläufe aus manuellen Prozessen (unmittelbar) bekannt oder sind die Abläufe primär kognitive Prozesse und daher nicht direkt beobachtbar?

- Besteht die Verarbeitung primär aus homogenen, strukturierten Massendaten oder hetorogenen, unstrukturierten Wissenseinheiten?

- Entsteht Komplexheit primär aufgrund des Umfanges der Datenmenge oder durch die Reichhaltigkeit der Wissensstrukturen?

- Geht es um relativ wenige Datentypen mit vielen Ausprägungen (Instanzen) eines Typs oder um viele Strukturtypen mit oft wenigen Instanzen eines Typs?

Im Zusammenhang mit diesem Transferbild „Arbeit $\Longrightarrow$ Computer“ ist zu erörtern, wie eine sinnvolle Zusammenarbeit zwischen Mensch und Computer gestaltet werden kann. Zu diskutieren ist jedoch weniger die Optimierung des Computers als „Werkzeug“. Hilfreicher erscheint eher die aufgeworfenen Fragen aus einem Rollenverständnis einer Kooperation zu betrachten.

Im Sinne einer solchen „Mensch-Maschine-Kooperation“ sind die Arbeiten (Aufgaben & Leistungen) beider Kooperationspartner zu definieren, das heißt, bildlich gesprochen, vertraglich festzulegen. Zu ermitteln ist daher, welche Arbeit jeweils für den Menschen und welche für den Computer prädestiniert ist. Während der Mensch individuell und zeitlich nicht konstant arbeitet, zeichnet sich der Computer durch die sichere Wiederholbarkeit einmal programmierter Vorgänge aus.

Die Software sollte daher nicht so konzipiert werden, daß der Mensch in die Rolle eines stupiden Datenzulieferns („Eintipper“) und Datenentsorgens („Papierabreißers“) gedrängt wird. Die Mensch-Maschine-Kooperation bedarf einer bewußten Gestaltung, insbesondere im Hinblick auf den anzustrebenden Automationsgrad (= Beschäftigungsgrad des Computers). Nicht weil sie softwaretechnisch aufwendig abzubilden ist, sollte eine stupide Aufgabe gleich dem Menschen zugeordnet werden, das heißt in logisch zu Ende gedachter Konsequenz:

> Es gibt auch eine **abzulehnende Halbautomation oder „Zuwenig“-Automation**.

Der provokative Begriff Kooperation verdeutlicht, daß es sich primär um einen Prozeß der sinnvollen Arbeitsteilung zwischen Menschen und Computern handelt, der in jedem Einzelfall bewußter, das heißt, die

Arbeitsteilung

Vor- und Nachteile abwägender Entscheidungen bedarf. Eine Softwareentwicklung, die diesen risiko- und konfliktreichen Entscheidungsprozeß ausklammert, greift zu kurz, weil sie erst ansetzt, wenn die wesentlichen Entscheidungen der Problemlösung schon getroffen sind. Softwareentwicklung konzentriert sich dann nur noch auf einen mehr und mehr automatisierbaren Umsetzungsprozeß. Gerade das Einbeziehen der Entscheidungen über die Arbeitsaufteilung zwischen Mensch und Maschine im Rahmen der anstehenden Automationsaufgabe verweist auf einen Unterschied zu einem klassisch-mechanischen Ingenieurprojekt.

Bei der Erstellung von großen oder sehr großen Softwareprodukten[10] ist es daher notwendig das Vorgehensmodell (lineare Phasenkonzept) nicht auf einmal auf das gesamte zu erstellende System anzuwenden, sondern vorab Entscheidungen über die angestrebte „Mensch-Maschine-Kooperation“ zu treffen. Anders formuliert: Das Gesamtsystem ist in überschaubare und beurteilbare Teilsysteme oder Teilbereiche aufzuteilen. Zum Kreieren und Koordinieren der Teilsysteme empfiehlt es sich, eine Strategiephase, auch „Nullphase“ genannt, vorzuschalten (vgl. Abbildung 1.1 Seite 31).

1.2.6 Grenzen des Computereinsatzes

Ethik

Das skizzierte Bild der „Mensch-Maschine-Kooperation“ (vgl. Abschnitt 1.2.5 Seite 27) zwingt dazu, Grenzen des Computereinsatzes und damit der Softwareentwicklung zu akzeptieren. Die Universalität des Computers ist aus vielerlei Motiven, zum Beispiel aus der Ethik des Ingenieurs heraus, zu begrenzen. Zu ziehen sind folgende Grenzen:

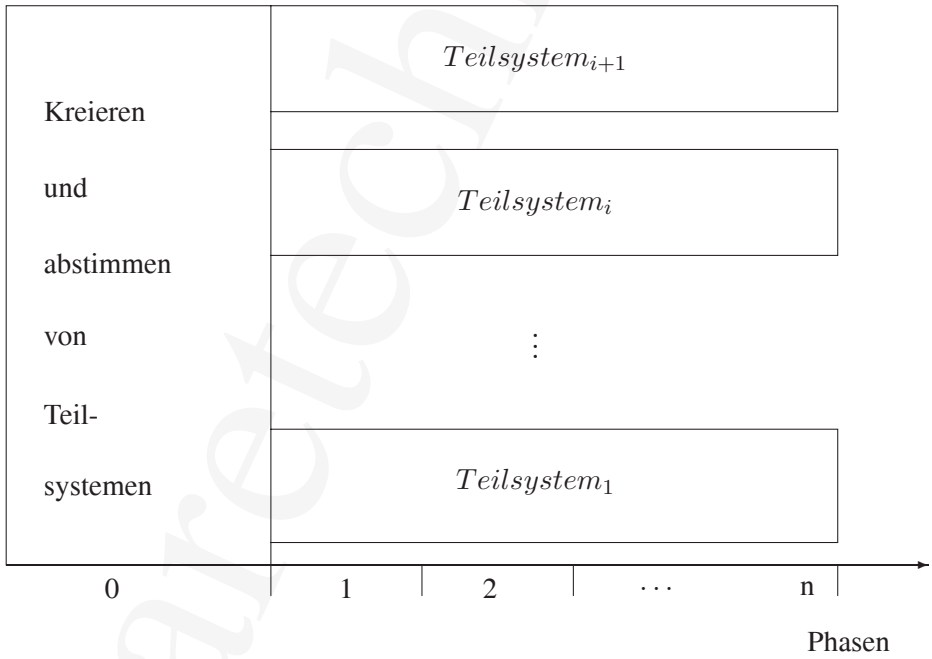
- Grenze des fachlich verantwortbaren Computereinsatzes,

  das heißt keine Softwareentwicklung für Bereiche bei denen aufgrund eines verfehlten Vertrauens in die Leistungsfähigkeit von Software, unverantwortbare Risiken eingegangen werden.

- Grenze des zwischenmenschlich verantwortbaren Computereinsatzes,

  das heißt, dort wo Computer aufgrund einer verfehlten Gleichsetzung von Menschen mit Maschinen eingesetzt werden sollen, zwischenmenschlicher Austausch behindert wird, menschliches Erleben verkümmert, menschliche Zuwendung wegfällt und „soziale Netze“ zerstört werden.

[10] Vgl. Abschnitt 2.4 Seite 54ff.



Legende:

Phase 0	$\equiv$	Festlegung einer Strategie für die Arbeitsteilung zwischen Mensch und Maschine; Schaffung der Rahmenbedingungen für die (Teil-)Systeme.
Phasen $1, \ldots, n$	$\equiv$	Phaseneinteilung für die (Teil-)Systeme entsprechend dem jeweiligen Phasenkonzept (vgl. Abbildung 2.1 Seite 51).

Abbildung 1.1: „Nullphase“ bei großen Softwareprodukten

- Moralisch/politische Grenzen des Computereinsatzes,

  das heißt keine Software, um Computer in die Lage zu versetzen, das zu tun, was ohne Computer nicht gemacht werden darf.

Es gilt Verantwortung für das zu übernehmen, was wir in die Software „gießen“. Softwareentwickler tragen diese Verantwortung mit und dürfen sich nicht hinter denjenigen, der den Auftrag erteilt, verstecken.

1.3 Problem: Fach- $\Leftrightarrow$ Informatiksprache

Häufig sind die ersten Phasen eines Softwareprojektes, zum Beispiel die Problemanalyse, die Definition der Ziele und Konflikte, die Ist-Aufnahme des System(umfeldes), die Analyse des Bedarfs und der Schwachstellen etc. (vgl. Abbildung 2.1 Seite 51) maßgeblich vom Auftraggeber („Bauherrn“) und den späteren Benutzern („Bewohnern“) bestimmt. In den anschließenden Phasen, zum Beispiel des technischen Entwurfs, der Feinplanung, der Programmierung, des Integrationstestes, dominieren die Systemdesigner („Architekten“) und Programmierer („Bauunternehmer“).

Beide Parteien nutzen – und sind geprägt von – unterschiedlichen Terminologien[11]. Einerseits ist es die Fachsprache des Automationsfeldes, andererseits ist es die Terminologie der Informatik („Computerwelt“).

Bei einer Softwareentwicklung sind Problem, die ihre Ursache in den unterschiedlichen Terminologien haben, zu lösen. Es bieten sich dafür zwei Lösungsansätze an:

1. Streben nach einer einheitlichen Terminologie, das heißt, im Softwareprojekt wird ein gemeinsamer, von jedem beherrschter Sprachschatz aus (Anwendungs-)Fach- und Informatikbegriffen erarbeitet.

2. Beibehaltung unterschiedlicher Terminologien und Vereinbarung einer Übersetzungsstelle.

Abbildung 1.2 Seite 33 zeigt vier Abstraktionsebenen: Fachebene, Anwendungsmodellebene, Entwurfsebene und Implementationseben. Sie verdeutlichen, daß bei der Softwareentwicklung ein Übergang von der Fachterminologie („Fachwelt“) zur Informatik-Terminologie („Informatik-Welt“) erforderlich ist (ähnlich [31] S. 40).

[11] Unter einer Terminologie versteht man allgemein die Gesamtheit der in einem Fachgebiet üblichen Fachwörter und Fachausdrücke und die Lehre von ihnen.

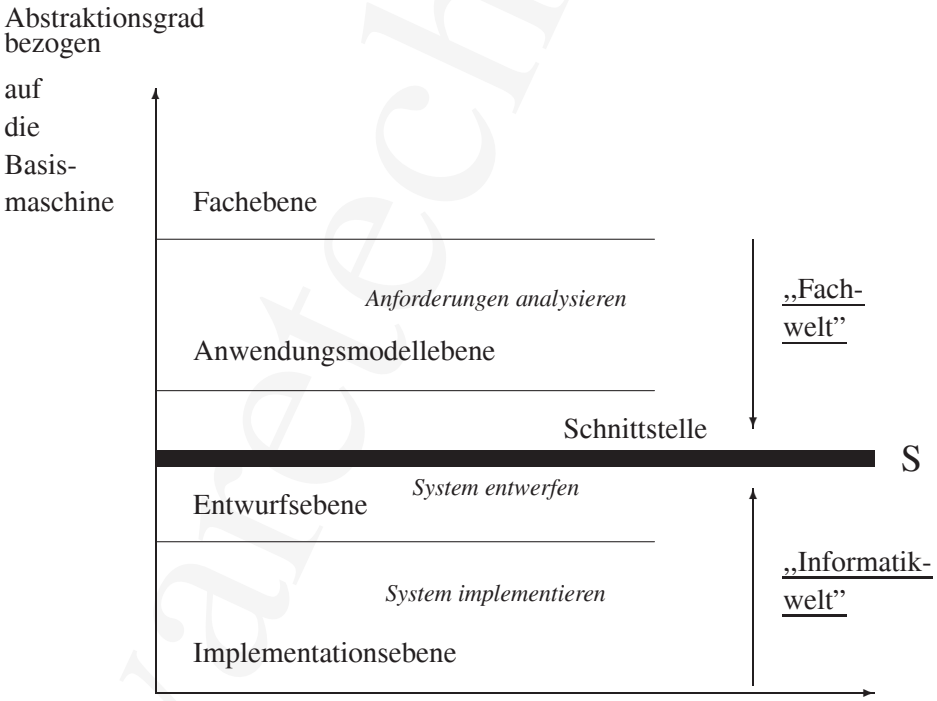


Abbildung 1.2: Verdeutlichung des Terminologie-Problems

Abstraktions-Ebene	Charakteristische Begriffe		
	Objekte $\mathcal{O}$	Tätigkeiten $\mathcal{T}$	Wechselwirkungen $\mathcal{O} \Leftrightarrow \mathcal{T}$
Fachebene	Dokumente, technische Gegenstände	Aktivitäten, Aufgaben, Vorgänge	Beziehungen, Zusammenhänge Zusammenarbeit
Anwendungs-modellebene	Informationen	Informations-flüsse	Modelle
Entwurfs-ebene	Daten, Datenmengen, Datenbereiche	Funktionen, Operationen, Prozesse	Bausteine, abstrakte Datentypen
Implementations-Ebene	Objekte, Variablen, Daten(speicher)	Prozeduren, Unter-programme	Moduln Transformationen Kontrollstrukturen

Tabelle 1.1: Beispiele für charakteristische Begriffe

Zwei Maschinenmodell

Anders formuliert, es handelt sich um die Notwendigkeit eine sogenannte Benutzermaschine (≡ die Anforderungen und Vorstellungen aus der Fachwelt) auf eine sogenannte Basismaschine (≡ eine konkrete Soft- und Hardwarekonfiguration) abzubilden.

Während auf der Fachebene und der Ebene des Anwendungsmodells die gebräuchlichen Fachbegriffe aus dem Bereich der zu automatisierenden Aufgabe („Fachwelt“) Verwendung finden, um die Phänomene, Fakten, Sachverhalte, Aussagen etc. möglichst eindeutig und effektiv zu beschreiben, bedarf die Ebene der Implementation der Präzision der Terminologie der Informatik (vgl. Tabelle 1.1 Seite 34).

Mit dem Zwang auf der Ebene der Implementation in der Terminologie der Informatik zu argumentieren und zu formulieren, zeichnet sich rückwärts betrachtet die Notwendigkeit ab, das Ergebnisdokument der vorhergehenden Phase ebenfalls in der Terminologie der Informatik zu formulieren. Anderernfalls wäre ein Deduktionsprozeß – im Sinne des Phasenkonzeptes – nicht gewährleistet.

Ausgehend von der Fachebene einerseits und der Ebene der Implementation anderseits zeichnet sich eine Grenzlinie ab, bei der die beiden unterschiedlichen Terminologien zusammentreffen. Diese Schnittstelle (S in Abbildung 1.2 Seite 33) kann auf zwei grundlegende Weisen gemeistert werden (vgl. Tabelle 1.2 Seite 36):

1. Vorab wird für die Akteure aus dem Fachbereich und der Informatik eine gemeinsame Sprache definiert.

Hierzu gehören die Ratschläge, die mit einem gemeinsamen Begriffslexikon operieren und dem Fachbereich zum Beispiel zumuten, Entscheidungstabelle oder Pseudocode zu kennen, also Sprachmittel der Informatik zu verstehen.

gemeinsame Sprache

Dieser Ansatz unterstellt: Wenn die Fachebene ein hinreichendes Verständnis der Informatik gewinnt und die Informatiker ein hinreichendes Wissen über die Automationsaufgabe, dann entsteht eine gemeinsame, allseits verstandene Sprache, die die Schnittstelle S leichter überwinden läßt.

2. Die Schnittstelle S wird bewußt herausgearbeitet. Oberhalb (gemäß Abbildung 1.2 Seite 33) wird ausschließlich mit Begriffen des Fachbereichs operiert, unterhalb mit definierten Informatikbegriffen.

   Die Übersetzung wird als bewußter Akt gestaltet und im allgemeinen von einem Übersetzer, dessen „Muttersprache“ die Terminologie der Informatik ist, vollzogen.

Übersetzung

   Hierzu gehören die Empfehlungen, die ein besonderes Dokument betonen, zum Beispiel ein Pflichtenheft, das von der Terminologie der Informatik (Datenverarbeitung) frei sein sollte. Getragen werden diese Ratschläge von der Sorge, daß mit der Terminologie der Informatik einhergehend das WIE, also die Ausprägung der späteren Lösung einfließt und damit das WAS des Fachbereichs frühzeitig beeinflußt (eingeschränkt) wird.

Ob der Ansatz I („gemeinsame Sprache“) oder der Ansatz II („Bereichsabgrenzung mit Übersetzung“) erfolgversprechender ist, hängt von einer Menge von Faktoren ab. Bedeutsam sind sicherlich (vgl. Tabelle2.2 Seite 53):

- die Größe des zu entwicklenden Softwareproduktes und
- die Klarheit der Zielkriterien und der erfolgreichen Arbeitstechnik zu Beginn der Softwareentwicklung.

Fundierte Entscheidungskriterien lassen sich der Literatur nicht entnehmen. Im wesentlichen wird man sich auf eine projektspezifische Gewichtung der Vor- und Nachteile der beiden Ansätze abstützen. Tabelle 1.2 Seite 36 vermittelt einen Einstieg für eine solche kontextabhängige Bewertung.

Aussagen über die zu erfüllenden Leistungen eines Softwareproduktes sowie über dessen qualitative und/oder quantitative Eigenschaften

	Ansatz I	Ansatz II
	Gemeinsame Sprache	Übersetzung
Vorteil	Einfluß der „Fachwelt“ bricht nicht ab – gemeinsames Lernen möglich	Nutzung von Spezialisierungsvorteilen – klare Schnittstellen für die Verantwortungsbereiche
Nachteil	enge Machbarkeitsgrenzen (nur begrenzt kann JEDER BEIDES verstehen)	Probleme der Kommunikation und daher zusätzliche Komplexität – Problem der korrekten Übersetzung

Tabelle 1.2: Lösungsansätze zum Terminologie-Problem

Kriterium	**Lastenheft**	**Pflichtenheft**
Erstellungszeitpunkt:	Definitionsphase	Planungsphase
Intention:	*Stop-or-go*-Frage	Vertragsgrundlage
Detailierungsgrad:	Grobe Übersicht	Umfassende Beschreibung
Hauptakteure:	Auftraggeber Projektleiter (Fachexperte)	Auftraggeber Projektleiter Fachexperte Systemanalytiker

Tabelle 1.3: Anforderungsdokumentation: Lasten- & Pflichtenheft

werden als Anforderungen (engl.: requirements) bezeichnet. Je nachdem welcher Lösungsansatz für die Terminologieproblem gewählt wurde, sind diese als freier Text oder stärker formalisiert notiert. Ihre meist verbindliche Fixierung in Form einer Softwareproduktdefinition wird als Lastenheft oder Pflichtenheft bezeichnet, insbesondere wenn es als Basis für die Vergabe von Arbeiten an Dritte (zum Beispiel an ein Softwarehaus) dient. Die Begriffe Lastenheft und Pflichtenheft werden im Alltag häufig als Synonyme verwendet. Üblicherweise werden sie jedoch aufgrund ihres Entstehungszeitpunktes und ihres Detailierungsgrades unterschieden (↪ Tabelle 1.3 S. 36).

Pflichtenheft

Der Auftragnehmer (das Softwarehaus) übernimmt die Verpflichtung ein Softwareprodukt, mit den spezifizierten Leistungen und Eigenschaften zu liefern. Für den Auftragnehmer ist das Pflichtenheft das Schlüsseldokument. Es bildet die verbindliche Grundlage, um in der Art und Weise eines Deduktionsprozesses die nachfolgenden Phasen zu

durchlaufen und gesichert zum fertigen Produkt zu kommen.

Wegen seiner großen Bedeutung enthalten praxisorientierte Empfehlungen häufig einen detaillierten Gliederungsvorschlag für das Pflichtenheft. Tabelle 1.4 Seite 38 zeigt einen komprimierten Gliederungsvorschlag (ähnlich [1, 54]). Dieser Vorschlag ist hier nicht als direkt umsetzbares Kochrezept zu verstehen. Er soll vorab Ideen zum Pflichtenheft verdeutlichen. Dabei geht es im wesentlichen um:

Produkt- & Prozeßdokument

- die Aufteilung der Ziele/Zwecke des Produktes in einen Abschnitt „*Produktleistungen & Produkteigenschaften*“ und in einen Abschnitt „*Anforderungen an den Realisierungsprozeß*“,
- die Einführung von Abgrenzungsaspekten, das heißt um das Beschreiben von Zielen/Zwecken, die mit dem Produkt bewußt nicht erreicht werden sollen,
- das ausführliche Beschreiben der Einsatzbedingungen und des Produktumfeldes, als Darstellung derjenigen Fakten, Sachverhalte etc., die im Rahmen dieser Softwareentwicklung nicht gestaltbar sind, sondern als gegebene Randbedingungen („*Sachzwänge*“) aufzufassen sind,
- die funktionale Beschreibung als Definition der „Benutzermaschine“,
- das Erwähnen eines Anhanges, in dessen allgemeinen Teil relevante Dokumente (zum Beispiel Herstellerbeschreibungen der Hardwarekomponenten) aufgenommen werden sollen, und
- zum Schluß (leicht abtrennbar!) ein vertraulicher Anhang, der Informationen enthält, die nur einem begrenzten Personenkreis zugänglich gemacht werden sollen (zum Beispiel Betriebsvergleiche, Preisabschläge, Sonderkonditionen etc.).

1.4 Abstraktion ⇔ Konkretisierung

Maintenance

Allerorts wird die sogenannte Software-Krise beschworen. Die Entwicklungsabteilungen von Unternehmen und Verwaltungen sind damit beschäftigt, die existierenden Anwendungen zu sanieren, zu warten und zu pflegen. Häufig entpuppen sich solche „maintenance“-Arbeiten als Beseitigung von Ungereimtheiten in der Anforderungsdokumentation. Die Software-Krise basiert auf der Komplexität der eingesetzen Software. Sie wächst mit steigenden Automationsgrad und mit dem Streben

1. Ziele/Zwecke des Produktes
   (a) Ergebnisorientierte Leistungen und/oder Eigenschaften
      i. Mußaspekte
      ii. Wunschaspekte
      iii. Abgrenzungsaspekte
   (b) Realisierungsprozeßbezogene Anforderungen
      i. Mußaspekte
      ii. Wunschaspekte
      iii. Abgrenzungsaspekte
2. Randbedingungen für das Produkt
   (a) Produkteinsatz
      i. Anwendungsbereich
      ii. Benutzergruppen
      iii. Betriebsbedingungen
   (b) Produktumfeld
      i. Nutzbare Ressourcen (Soft-/Hardware)
      ii. Produktschnittstellen
      iii. Voraussichtliche Veränderungen
3. Funktionale Beschreibung („Benutzermaschine“)
   (a) Bedienungsmaßnahmen der verschiedenen Benutzer
   (b) Normal-, Ausnahme- und Fehlerreaktionen
   (c) Informationsflüsse (Eingaben/Ausgaben)
4. Anhang
   (a) Allgemeiner Anhang
   (b) Vertraulicher Anhang

Tabelle 1.4: Software-Produktdefinition (Gliederungsvorschlag)

nach größer Integration, das heißt mit dem Zusammenfassen von verschiedenen Teilsystemen zu einem Gesamtkomplex.

Der allgemeine Ratschlag zur Meisterung der Komplexität: Denke besser! ist zwar begrüßenswert, stößt aber sehr schnell an seine Grenzen. Diese Grenze auszuweiten, um höhere Komplexität bewältigen zu können, verspricht der Übergang zu einer ingenieurmäßigen Arbeitstechnik. Damit ist Software im Sinne der Ingenieurdisziplin „*Software Engineering*“ zu konstruieren. Die Arbeitstechniken dieser Disziplin befassen sich mit:

- Konstruktionsempfehlungen gemäß bisher offensichtlich bewährter Konstruktionen und Vorgehensweisen,
- Vorschläge für den Einsatz nützlicher Pruktionshilfen („Werkzeuge“; engl.: tools) und
- Ratschlägen oder Anweisungen aufgrund heuritischer Strategien.

Ausgangspunkte für ein rational erklärbares, zielorientiertes und letztlich erfolgreiches Vorgehen bilden Prinzipien. Sie entsprechen im Bild der Mathematik den „Axiomen“, das heißt den allgemein „geglaubten“ und nachweislich zweckmäßigen (Handlungs-)Grundsätze. Solche Prinzipien sind mit folgenden Schlagwörtern charakterisierbar:

- Prinzip der Abstraktion
  Wechselspiel zwischen Abstraktion und Konkretisierung gegenüber Phänomenen, Fakten, Wünschen etc. („reale Welt“)
- Prinzip der Strukturierung
  Erkennen und Ordnen von Phänomenen, Fakten, Wünschen etc. („reale Welt“)
- Prinzip der Hierarchisierung
  Hierarchische Gliederung zum Beispiel in die Relationen: ist-Teil-von und nutzt-Teil
- Prinzip überschaubarer Subeinheiten
  Modularisierung und Konzentration (Lokalität)
- Prinzip der Selbstdokumentation
  Dokumentationsnormen und Dokumentationsstandards
- Baukasten-Konstruktionsprinzip
  Mehrfachnutzung möglichst weniger Bausteintypen

Zur Bewältigung von Komplexität ist das Prinzip der Abstraktion bedeutsam. Obwohl im Anwendungsfall häufig eng miteinander verbunden, unterscheiden wir drei allgemeine Formen der Abstraktion (vgl. zum Beispiel [41]):

Invarianten

- generalisierende Abstraktion
  Sie ist geprägt durch das Auffinden von Invarianten, das heißt von Größen, Eigenschaften, Relationen etc., die in Bezug auf bestimmte, das jeweilige System betreffende Veränderungen (Transformationen, Operationen etc.) unverändert bleiben.

Verselbständigung

- isolierende Abstraktion
  Ihre vorgenommene Verselbständigung einzelner Eigenschaften und Relationen oder von Gruppen von Eigenschaften, Relationen und dergleichen vereinfacht die Übersicht über den betrachteten Bereich und erleichtert die jeweiligen Phänomene in einer Theorie zu erfassen.

ideale Objekte

- idealisierende Abstraktion
  Sie schafft ideale Objekte, die sich von den wirklichen Objekten nicht nur dadurch unterscheiden, daß manches Unwesentliche weggelassen wird, sondern auch dadurch, daß sie mit Eigenschaften ausgestattet werden, die die existierenden Objekte nicht oder nur angenähert besitzen. Ein Beispiel ist der unendlich große Speicher der Turing-Maschine, den ein real existierender Computer nicht haben kann.

Das Überführen einer in bestimmter Hinsicht abstrakten Beschreibung in eine in gleicher Hinsicht weniger abstrakten Beschreibung nennt man konkretisieren, die Umkehrung von abstrahieren ([31] S. 203).

Die generalisierende Abstraktion dient dazu Gegenstände oder Sachverhalte zu Klassen zusammenzufassen, deren Elemente in einer bestimmten Hinsicht als gleich behandelt werden sollen. Abstraktion und ihr Gegenteil die Konkretisierung stehen daher in Bezug zu Etwas. Zum Beispiel kann eine Problembeschreibung abstrakt oder konkret sein:

- bezüglich Phänomenen, Fakten und/oder Sachverhalten der realen Welt oder
- bezüglich der Basismaschine, das heißt bezüglich einer definierten Soft-/Hardware-Konfiguration.

„Die Abstraktionsmethode liefert uns die Möglichkeit, so zu reden, als ob wir über neue Gegenstände (abstrakte Ob-

> jekte) reden – obwohl wir nur in neuer Weise über die bisherigen Gegenstände (konkrete Objekte) reden . . . Die Abstraktion geschieht dadurch, daß wir uns auf solche Aussagen über Objekte beschränken, deren Gültigkeit sich bei der Ersetzung eines Objektes durch ein äquivalentes nicht ändert. Solche Aussagen wollen wir „invariant“ bezüglich einer gegebenen Äquivalenzrelation nennen.“ (Lorenz 1975 zitiert nach [53] S. 43)

Die Abstraktion zeigt sich als ein reales Phänomen; es entsteht durch Wiederholung, durch die Äquivalenz von Ergebnissen von Handlungen oder Vorgängen. Es ist Grundlage für das Entstehen von elementarer Information, einer Verknüpfung von konkretem Träger und abstraktem Inhalt (vgl. [53]).

Im Zusammenhang mit der Softwareentwicklung bezieht sich das Wechselspiel zwischen Abstraktion und Konkretisierung primär auf Funktionen und Daten. Wir unterscheiden daher:

- die funktionale Abstraktion[12] von
- der Datenabstraktion.

1.5 Ingenieurdisziplin: Softwareentwicklung

Die Softwarentwicklung ist ein Teilgebiet der Ingenieurdisziplin Softwaretechnik (engl.: Software Enginieering). Ein Einstiegsverständnis für den Begriff „Software Engineering“ vermittelt die Etymologie, also die Lehre von der „wahren“ Bedeutung des Wortes, das heißt vom Ursprung des Wortes. „Ingenium“ [lateinisch] verweist auf natürliche Begabung, Scharfsinn und Erfindungskraft, auf Logik und Mathematik. „Ingenieur“ [französich: Kriegsbaumeister] verweist auf Management und Kooperation mit dem Ziel ein größeren Produkt zu schaffen. (vgl. [32] S. 49)

Sofware Engineering läßt sich anhand der Tätigkeiten gliedern (vgl. Tabelle 1.5 Seite 42): einerseits in die originären (ursprünglichen) Tätigkeiten der Softwareentwicklung und -anwendung und andererseits in Dienstleistungstätigkeiten dafür (vgl. [31] S. 205). .

Software Engineering

1.5.1 Tätigkeiten

Die Sofwareentwicklung selbst ist aufteilbar in drei grobe Tätigkeitsbereiche:

[12] Manchmal auch als prozedurale oder operationale Abstraktion bezeichnet.

Software Engineering				
Originäre Tätigkeiten der Softwareentwicklung und -anwendung		Dienstleistungstätigkeiten für die Softwareentwicklung und -anwendung		
Softwareentwicklung	Anwendung der Software	Projektmanagement	Qualitätssicherung	Bereitstellung von Arbeitstechniken

Tabelle 1.5: Einordnung der Softwareentwicklung

- Ermittlung und Festlegung der Anforderungen,
- Konzeptionierung der Software und
- Realisierung der Software.

Tabelle 1.6 Seite 43 verfeinert diese Klassifikation durch Nennung wesentlicher Tätigkeiten. Die dort benutzten Bezeichnung für die einzelnen Tätigkeiten, wie zum Beispiel „analysieren“, „definieren“ etc., sind Alltagsbegriffe. Sie werden in verschiedensten Zusammenhängen auch weniger strikt (salopp) verwendet. Für dieses Buch erklärt Tabelle 1.7 Seite 44 ihre Bedeutung (ähnlich [31] S. 203)

1.5.2 Rollen & primäre Interessen

Die Tätigkeiten im Rahmen einer Softwareentwicklung sind (im allgemeinen) verbunden mit einer Menge von Personen. Sie alle vertreten unterschiedliche „Rollen“ und Interessen. Der Auftraggeber hat andere Interessen als die späteren Benutzer. Der Datenschutzbeauftragte, der Betriebsrat, die Wartungsmannschaft etc. zählen zu den „zu beteiligenden Personen“ oder zumindest zu den Betroffenen. Die Tabelle 1.8 Seite 45 skizziert die üblichen „Rollen“ und die damit verbundenen „primären Interesse“. Zur Verdeutlichung ist eine Analogie zur Architektur gezogen.[13]

[13] Vgl. zum Beispiel [9].

Softwareentwicklung					
Festlegung der Anforderungen (Requirements Engineering)		Konzeption des Softwaresystems		Realisierung des Softwaresystems	
Problem analysieren*	Anforderungen definieren*	Bausteine spezifizieren*	Bausteine konstruieren*	Bausteine implementieren (Programmieren)*	Bausteine integrieren (montieren)*

Legende:

* ≡ Inklusive begleitende (simultane) Dokumentation und (schrittweises) Prüfen

Tabelle 1.6: Softwareentwicklung: Einteilung der Aktivtäten

Tätigkeit	Erläuterung
analysieren	Etwas als gegeben hinnehmen und genauere Kenntnisse darüber gewinnen.
definieren	Etwas, zum Beispiel einen unbekannten oder nur teilweise bekannten Begriff, mit Hilfe anderer, bekannter Begriffe beschreiben und festlegen.
präzisieren	Etwas in weniger mißverständlicher Art und Weise darstellen.
formalisieren	Eine formale Beschreibungsform einführen oder Etwas von einer teilweise oder nicht formalen Beschreibungsform in eine im größeren Ausmaße formale Beschreibungsform überführen.
spezifizieren	Etwas durch sprachliche Festlegung seines Gebrauchs oder seines Ergebnisses präzisieren.
konstruieren	Ein spezifiziertes Etwas durch weitere Angaben und Darstellungen konkretisieren.
implementieren	Programmieren und prüfen von (konstruierten) Bausteinen.
montieren	Bausteine zu einem größeren Etwas zusammensetzen.

Legende:
Ähnlich [31] S. 203.

Tabelle 1.7: Softwareentwicklung: Tätigkeiten

Rollen	Primäre Interessen
Auftraggeber (Bauherr)	Optimale Erreichung seiner Ziel- und Wunschvorstellungen, Nutzen- und Kostenfragen
Systemdesigner (Architekt)	Konzeptions- und Entwurfsfragen, um das Konglomerat an gewünschten, erhofften Softwareleistungen ($\equiv$ Benutzermaschine) auf die konkrete (das heißt vorhandene oder zubeschaffene) Computer ($\equiv$ Basismaschine) abbilden zu können.
Programmierer (Bauunternehmer)	Verständlichkeits- und Interpretationsfragen, um durchschaubare und zuverlässige Programme erstellen zu können.
Betreiber & Operateure (Hausmeister)	Leichte Bedienbarkeit und Zuverlässigkeit
Wartungsdienst (Handwerker)	Änderungs- und Anpassungsfreundlichkeit, insbesondere Nachvollziebarkeit der (Kontroll-)Strukturen
Weitläufig Betroffene (Nachbarn)	Fragen der Schadensbegrenzung, das heißt Einflußnahme auf die Ausschaltung von negativen Wirkungen; Schnittstellen, Auflagen
Spätere Benutzer (Bewohner)	Akzeptanz, Softwareergonomie; Erfüllung der fachlichen Anforderungen zuverlässig und in hinreichender Qualität
Zu Beteiligende, zum Beispiel Betriebsrat oder Datenschutzbeauftragte (Baubehörden)	Erfüllung von Gesetzen, Vorschriften, Standards und Normen.

Tabelle 1.8: Softwareentwicklung: Rollen & Interessenschwerpunkte

1.5.3 Erläuterung der Argumentationskette

Heute versuchen wir in der Regel Software zu konstruieren mit den Eigenschaften[14]:

- Industrial Strength (≡ für den harten Alltagseinsatz)
- Resolutely Simple (≡ keine unnötige Komplextität)
- Multi-Platform (≡ für mehrere Betriebssysteme)

Solche Ziele bedingen eine Softwareentwicklung, die geprägt ist vom Ingenieuransatz. Dieser ergibt sich aus der Disziplin „Software Engineering“, die als Dach zur Erörterung ihres Teilgebietes „Softwareentwicklung“ dient. Charakteristisch für einen Ingenieuransatz ist ein methodisch fundiertes, erfolgsicherndes Vorgehen. Dazu werden aus allgemeingültigen Handlungsgrundsätzen (Prinzipien) konkretere Handlungsanweisungen (Methoden) abgeleitet. Um ein ordnungsmäßes Vorgehen im Sinne einer nützlichen Methode zu gewährleisten, dienen Produktionshilfen (Instrumente, „tools“). Im Wechselspiel zwischen Methoden und Produktionshilfen entstehen praxisorientierte „Arbeitstechniken“ (vgl. Abbildung 1.3 Seite 47).

Ein solche praxisorientierte Arbeitstechnik bildet zum Beispiel die Präzisierung und Formalisierung mittels Entscheidungstabellen. Entscheidungstabellen basieren theoretisch auf dem Lokalitätsprinzip, das heißt, Zusammengehörendes wird zusammen auf engem – und damit leicht überschaubarem – Raum dargestellt.

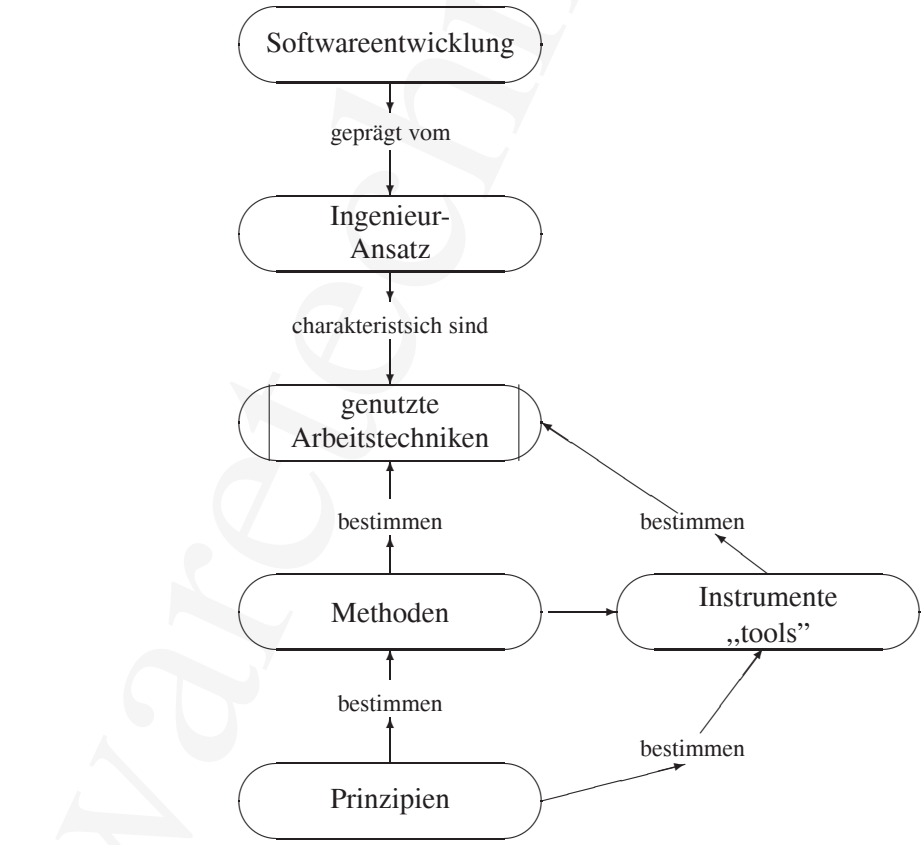
Dieses Buch erläutert *bewährte und weit verbreitete Arbeitstechniken* anhand von Beispielen und Übungsaufgaben. Neben Entscheidungstabellen, Ablaufplänen, Pseudocode und Struktogrammen – für (weitgehend) sequentielle Modelle – werden Petri-Netze eingehend erörtert. Sie sind eine Arbeitstechnik zur Modellierung von *nebenläufigen Vorgängen*.

Die einzelnen Arbeitstechniken sind miteinander verknüpft dargestellt. Zum Beispiel werden Formulierungen in Pseudocode mit Darstellungen von Struktogrammen verglichen. Solche Gegenüberstellungen lassen die Vor- und Nachteile der jeweiligen Arbeitstechnik leicht erkennen.

Welche Arbeitstechnik im einzelnen Anwendungsfall nützlich ist, hängt (zumindest) vom Problemtyp und der (Software-)Produktgröße ab. Eine Skizze einer Problem-/Produkt-Einordnung ist daher den einzelnen Arbeitstechniken vorangestellt.

Das „Denkmodell“ (Paradigma) bei einer Softwareentwicklung kann sehr verschieden sein. Zum Beispiel kann es primär vom Phasenmodell

[14] Vgl. zum Beispiel die Produktankündigung von SoftQuad Software



Legende:

≡ Betrachtungsschwerpunkt

Abbildung 1.3: Argumentationskette

oder vom Prototyping ausgehen. Um solche Unterschiede exemplarisch zu erläutern, ist der Ansatz Prototyping vorab im Kapitel 3.1 Seite 58 besonders ausführlich dargestellt.

Prinzipiell ist eine bestimmte Arbeitstechnik in vielen Bereichen – quasi unabhängig vom Problemtyp, der Produktgröße und dem „Denkmodell“ – anwendbar. Ihre Hauptvorteil wird jedoch (im Regelfall) erst im eigentlichen Bestimmungsbereich zur Geltung kommen. So kann man zum Beispiel unstrittig auch mit Entscheidungstabellen geschachtelte Iterationen modellieren, obwohl der ursprüngliche Einsatzbereich eher „verzwickte“ Bedingungskombinationen sind. Die geschachtelten Iterationen wären wahrscheinlich durchschaubarer mittels Struktogrammen abgebildet.

Kapitel 2

Problem-/Produkt-Einordnung

Arbeitstechniken umfassen einerseits Prinzipien (Handlungsgrundsätze) und Methoden (erfolgversprechende, begründbare Vorgehensweisen) und andererseits Produktionshilfen („Werkzeuge“). Welche Arbeitstechniken im Anwendungsfall nützlich sind, ist (zumindest) abhängig vom Problemtyp und der Produktgröße.

Allzweck-Arbeitstechniken bei der Softwarekonstruktion gibt es nicht! Bezogen auf die (Vor-)Kenntnisse zum Zeitpunkt des Projektstartes unterscheiden wir: Vollzugs-, Definitions-, Zielerreichungs- und Steuerungsprobleme.

Wegweiser

Der Abschnitt *Problem-/Produkt-Einordnung* erläutert:

- den Begriff der Software-Spezifikation,
  ↪Seite 50 . . .
- Typen von Anforderungen,
  ↪Seite 50 . . .
- Problemarten bei den vielfältigen Konstruktionsaufgaben und
  ↪Seite 53 . . .

- den Einfluss der Produktgröße.
  ↪Seite 54 . . .

2.1 Software-Spezifikation

Systemanalyse im Sinne von Spezifizieren heißt, eine zunächst noch weitgehend unklare/unbekannte Aufgabenstellung durch sprachliche Festlegung ihres Ergebnisses oder ein zunächst noch unbekanntes Objekt durch sprachliche Festlegung seines Gebrauchs zu präzisieren.

Lebenszyklus

Publikationen zur Thematik „Softwarespezifikation“ gehen von unterschiedlichen Begriffsdefinitionen aus. Üblicherweise basieren sie auf der Abgrenzung einzelner Phasen des Lebenszyklus und der Beschreibung ihrer Dokumente.

Ziel ist dabei die Spezifikation mit der Präzision der Mathematik [1] und der Verständlichkeit der Umgangssprache.

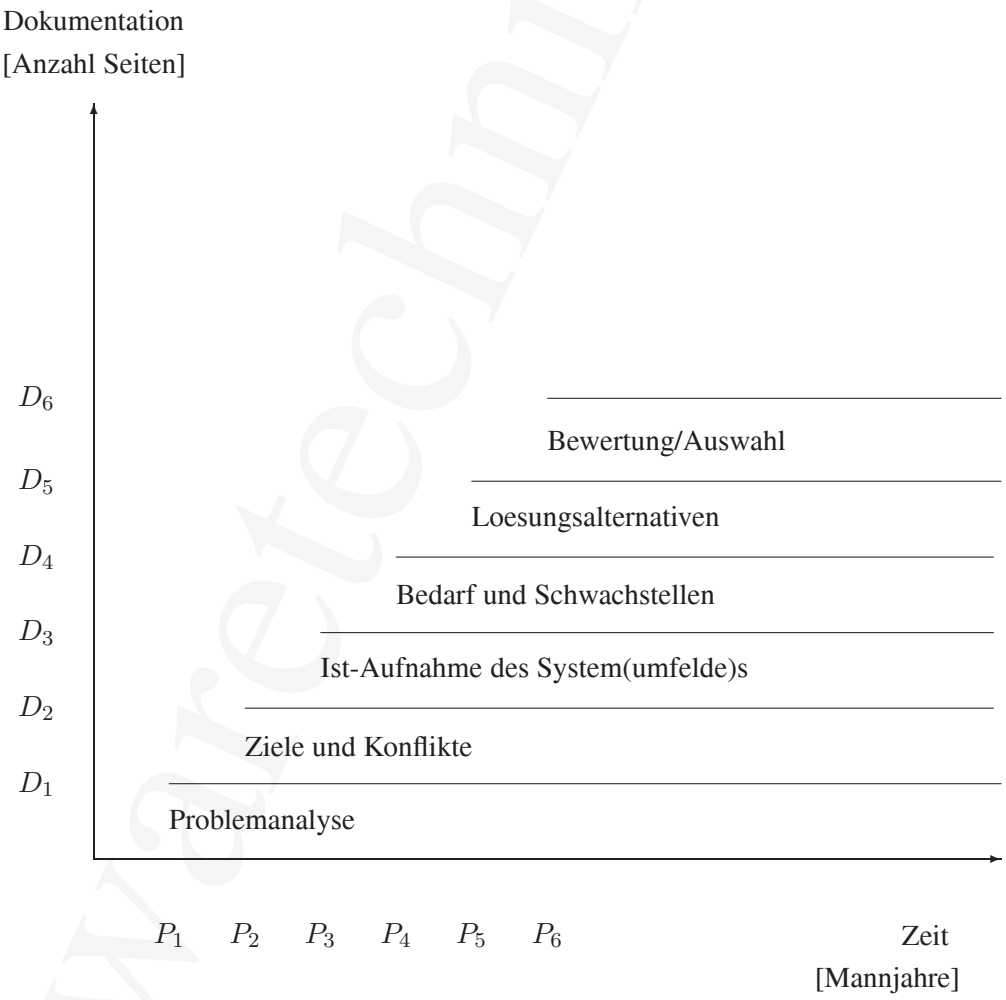
Spezifizieren im Sinne eines Präzisierens umfaßt alle Phasen. In den ersten Phasen bezieht sich die Spezifikation auf die Analyse und Synthese möglicher Konzeptionen. Ausgehend von einer Problemanalyse, werden mit der Aufnahme der Ziele und Konflikte sowie der Ist-Situation des bisherigen System(umfelde)s der Bedarf und die Schwachstellen präzisiert. Alternative Lösungansätze sind dann anhand der präzisierten Ziele und Konflikte zu bewerten (vgl. Abbildung 2.1 Seite 51). Anschließend folgt das Spezifizieren im engeren Sinne, das heißt das Entwerfen und Formulieren von Quellcodetexten.

2.2 Typen von Anforderungen

Anforderungen

Die Anforderungen können sich auf sehr unterschiedliche Produkte beziehen. Beispiele sind numerische Berechnungen (Lösungen von Differentialgleichungen), geometrische Berechnungen (3-dimensionale Modelle), Datenaggregationen (Besoldungsbescheide), Transaktionsprozesse (Platzreservierungssysteme), Wissensakquisition (Diagnosesysteme), Kommunikationsprozesse (Elektronische Post) und vieles mehr.

[1] Exkurs: Algebraische Spezifikation. Konzentriert man sich auf die Phase, in der anwendungsspezifische Datentypen zu präzisieren sind, dann besteht die Spezifikation aus Sorten, Operationen und Gleichungen. Diese Spezifikation notiert die Datentypen im Sinne von Algebren.



Legende:
$D_i \equiv$ Dokument als Ergebnis der Phase P_i

Abbildung 2.1: Beispiel: Strukturierung der ersten Phasen

	Anforderungs-Kategorie	Kernfrage	Beispielformulierungen (Programm DIAGNOSE)
1	funktionale Anforderungen	Was soll das System tun?	Aussagen sind aus Regeln abgeleitet oder werden beim Benutzer nachgefragt.
2	Anforderungen an das fertige Produkt	Was ist das System?	Das Dialogsystem DIAGNOSE ist ein Produkt für den Einsatz bei mehr als 100 Kfz-Betrieben.
3	Anforderungen zur Systemumgebung	Was sind die Einsatzbedingungen?	DIAGNOSE setzt das Betriebssystem MS-DOS Version 6.0 voraus.
4	Anforderungen an die Zielstruktur	Was ist der Bewertungshintergrund?	DIAGNOSE verkürzt die Fehlersuchzeit zumindest bei angelernten Kräften.
5	Anforderungen zur Projektdurchführung	Was sind die Ressourcen für das Projektmanagement?	DIAGNOSE Version 1.0 ist innerhalb von 3 Monaten mit 2 Mann zu entwickeln.

Tabelle 2.1: Kategorien von Anforderungen

Zu Projektbeginn Klarheit über: die Arbeits-Techniken			die Ziele: groß I	die Ziele: gering II
1	erfolgversprechende Prinzipien, Methoden und Instrumente zur Problemlösung	bekannt	Vollzugsprobleme	Definitionsprobleme der Automationsaufgabe
2		unklar	Zielerreichungsprobleme	Steuerungsprobleme eines kontinuierlichen Herantastens

Tabelle 2.2: Problemarten

2.3 Problemarten

Die Vielfalt der Konstruktionsaufgaben macht unterschiedliche Arbeitstechniken notwendig.

Arbeitstechniken umfassen einerseits Prinzipien (Handlungsgrundsätze) und Methoden (erfolgversprechende, begründbare Vorgehensweisen) und andererseits Produktionshilfen („Werkzeuge“). Erste sind nicht, zweite sind (teilweise) selbst Software.

Empfehlungen zur Arbeitstechnik haben zumindest die Aufgabenart und die Aufgabengröße zu berücksichtigen. Allzweck-Arbeitstechniken sind genauso unsinnig wie Allzweck-Worte. Es ist beispielsweise nicht unerheblich, ob ein administratives System oder ein System zur Steuerung eines technischen Prozesses zu spezifizieren ist.

Im ersten Fall ist das Nutzen/Kostenverhältnis einzelner Systemleistungen disponibel oder unklar. Wir haben es mit Definitionsproblemen der Automationsaufgabe zu tun.

Im zweiten Fall mag die geforderte Reaktionsgeschwindigkeit das eigentliche Konstruktionsproblem sein. Das Ziel und die Arbeitstechnik seien klar; das Problem liegt in der Durchführung (Vollzugsproblem). Ausgehend von unserem (Vor)-Wissen über die erfolgversprechende Arbeitstechnik und der Klarheit der Zielkriterien, sind verschiedene Problemfelder zu unterscheiden.

Ist die Konstruktionsaufgabe ein Vollzugsproblem (Feld I.1 in Tabelle 2.2 Seite 53), dann beziehen sich die erfolgversprechenden Arbeitstechniken zur Spezifikation auf:

Vollzugsproblem

- eine schrittweise Verfeinerung (vertikale Ebene) und
- eine modulare Strukturierung (horizontale Ebene)

von Anforderungen. Für beide Fälle bedürfen wir einer Rechnerunterstützung.

2.4 Produktgröße

Die Konstruktionsgröße bestimmt welche (rechnergestützten) Arbeitstechniken anzuwenden sind. Bei einer kleinen Konstruktionsaufgabe ist die Spezifikation der einzelnen Verarbeitungsprozesse und der Datenrepräsentation zu meistern. Bei einer sehr großen Aufgabe sind zusätzlich die Fortschritte der systemnahen Software und der Hardware während der benötigten Planungs- und Realisierungszeit einzukalkulieren. Außerdem ändern sich Anforderungen in dieser relativ langen Zeit. Zu spezifizieren ist daher eine Weiterentwicklung, pointiert formuliert: ein Evolutionskonzept.

Tabelle 2.3 Seite 55 skizziert benötigte Arbeitstechniken in Abhängigkeit zur Konstruktionsgröße. Dort ist der Umfang des geschätzten Quellcodetextes nur ein grober Maßstab. Die Angabe des Aufwandes in„Mannjahren“ (kurz: MJ)[2] ist umstritten und berechtigt kritisierbar (’The Mythical Man-Month’ [16]). Hier dienen die LOC- und MJ-Werte nur zur groben Unterscheidung, ob ein kleines oder großes Team die Aufgabe bewältigen kann. Bei einer größeren Anzahl von Programmierern ist eine größere Regelungsdichte erforderlich. Die Dokumentationsrichtlinien, die Arbeitsteilung und die Vollzugskontrollen sind entsprechend detailliert zu regeln. Es bedarf einer umfassenden Planung und Überwachung des gesamten Lebenszyklus.

Mann-jahr

[2] Im Zeitalter der Gleichberechtigung wäre es angemessen einen andernen Begriff zu verwenden – zum Beispiel „Personenjahr“. Ein solcher Begriff hat jedoch bisher noch keinen Eingang in die Fachliteratur gefunden.

Lfd	Konstruktions-Kategorie	Konstruktionsgröße [LOC]	Aufwand [MJ]	Erforderliche Arbeitstechniken (Prinzipien, Methoden, Instrumente) zur:
	A	B	C	D
1	Kleine Konstruktion	< 1.000	< 0.5	o funktionalen Strukturierung o Datenrepäsentation
2	Mittlere Konstruktion	< 10.000	< 4	o Projektplanung o Projektüberwachung den den gesamten Lebenszyklus o Anforderungsanalyse (Requirements Engineering) o plus (1)
3	Große Konstruktion	< 100.000	< 25	o Durchführbarkeitsstudie o Definition von Datennetzen und Datenbankmanagement o Konfigurationsmanagement o plus (2)
4	Sehr große Konstruktion (noch größer „zerfallen“ in eigenständige Teile	mehrmals 100.000	> 25	o Softwareevolution o Hardwareevolution o Dynamik der Anforderungen o plus (3)

Legende:
LOC ≡ Umfang der Quellcodetexte (Lines of Code)
MJ ≡ Mannjahre

Tabelle 2.3: Konstruktions-Kategorien und Arbeitstechniken

Kapitel 3

Exemplarische Vorgehensweisen

Eine Spezifikation einer koordinierten Vorgehensweise bei der Abwicklung eines Vorhabens wird als *Prozessmodell* bezeichnet. Es definiert sowohl den *Input*, der zur Abwicklung einer Aktivität erforderlich ist, als auch den *Output*, der als Ergebnis der Aktivität erzeugt wird. Dabei spielt die Zuordnung zu einer durchführenden Instanz (*Worker*) eine wichtige Rolle.

Wegweiser

Der Abschnitt *Exemplarische Vorgehensweisen* erläutert:

- den „Ansatz Lernen“ als tragendes Konzept unter dem Begriff *Prototyping*
  ↪Seite 58 . . .

- als Beispiel für den „Ansatz strikt geplanter Phasen“ den *Rational Unified Process*,
  ↪Seite 64 . . .

- als Beispiel für den „Ansatz einer projektübergreifenden Strategie“ das *Capability Maturity Model for Software* (CMM) und
  ↪Seite 67 . . .

- ein Beispiel für den „Ansatz einer Kooperation“ zur Schaffung des Domänenmodells und des Systemmodells.
↪Seite 70 . . .

3.1 Prototyping

Prototyping kann nützlich sein zur Klärung und Festlegung der Systemleistungen (Anforderung, Automationsumfang), zur Überprüfung der Machbarkeit des Designs und als iterative Vorgehensweise, wobei der Prototyp[1] zu einem immer leistungsfähigeren Produktionssystem heranwächst. Prototyping bietet die Chance durch Tatsachen zu überzeugen.

Das Risiko liegt in der Wahl und Modifikation des Prototypen. Es besteht die Gefahr, daß man sich nicht dem gewünschten Ziel nähert (Konvergenz) und die gefordertere Qualität nicht erreicht (Qualitätssicherung).

3.1.1 „Lernen“ als Ansatz

Für alle Akteuer (Beteiligte und Betroffene) versucht der Ansatz „Lernen“ eine bejahende (Grund-)Einstellung zu komplexen Vorhaben aufzubauen und zu verstärken. Vergleichbar zur Softwareentwicklungsmethode des „bootstrapping“[2], bei dem man sich von einem primitiven Programm zu einer wesentlich leistungsfähigeren Software „hochzieht“, ist auf der Basis einer Strategie des Überzeugens durch Tatsachen und Prototypen die Lösung und die Akzeptanz dafür zu entwickeln.

Ähnlich dem „bootstrapping“, bei dem zunächst eine funktionsfähige Urzelle erforderlich ist, setzt der erste Schritt ein minimales Akzeptanzpotential voraus.

Bedingung ist es daher, den ersten Realisierungsschritt für einen Prototyp, ein Pilotprojekt oder einen Probebetrieb in einer Umgebung zu starten, bei der dieses Minimalpotential nachweislich vorhanden ist. Gemäß dem Bild vom „bootstrapping“ bedeutet eine dominierende positive Einstellung zum Vorhaben, also ein hohes Akzeptanzpotential, daß die geplante Lösung schneller erreicht wird.

Bereitschaft zum Prototyping

[1] Im Rahmen von Software ist „Prototyp ein unglückliches Wort“ [43]. Häufig bezeichnet es, z.B. in der Fertigungsindustrie, ein Produktmuster, dessen Stückkosten zwar relativ hoch sind, das aber keine Investitionen für eine Serienfertigung beansprucht. Der Prototyp eines Fahrrades ist natürlich kostenintensiver als ein Fahrrad aus der Serie.

[2] Bootstrapping (abgeleitet von bootstrap: Hilfsschlaufe zum Anziehen von Stiefeln) ist eine Methode, die insbesondere beim Compilerbau und in Zusammenhang mit dem Laden eines Betriebssystems angewendet wird.

Dieser Ansatz betont das Lernen[3] anhand von konkreten Beispielen (Prototypen). Zur Entwicklung passender Beispiele dient das Prototyping.

Unter Prototyping versteht man:

> Die Verwendung vorhandener oder mit geringerem Aufwand erstellbarer Lösungen als Arbeitsmodell und als Muster für die Entwicklung weiterer Vorstellungen, Anforderungen und Lösungen.

Prototyping ist ein Prozeß, ein dynamischer Vorgang der Modellierung eines tatsächlich arbeitsfähigen Systems (vgl. Abbildung 3.1 Seite 60). Der konkrete Betrieb des Prototyps – auch unter der abschwächenden Einordnung als befristeter Probebetrieb oder als Pilotprojekt – verändert die Situation im Automationsfeld. Positive und negative Erfahrungen mit dem Prototyp prägen die Akzeptanz.

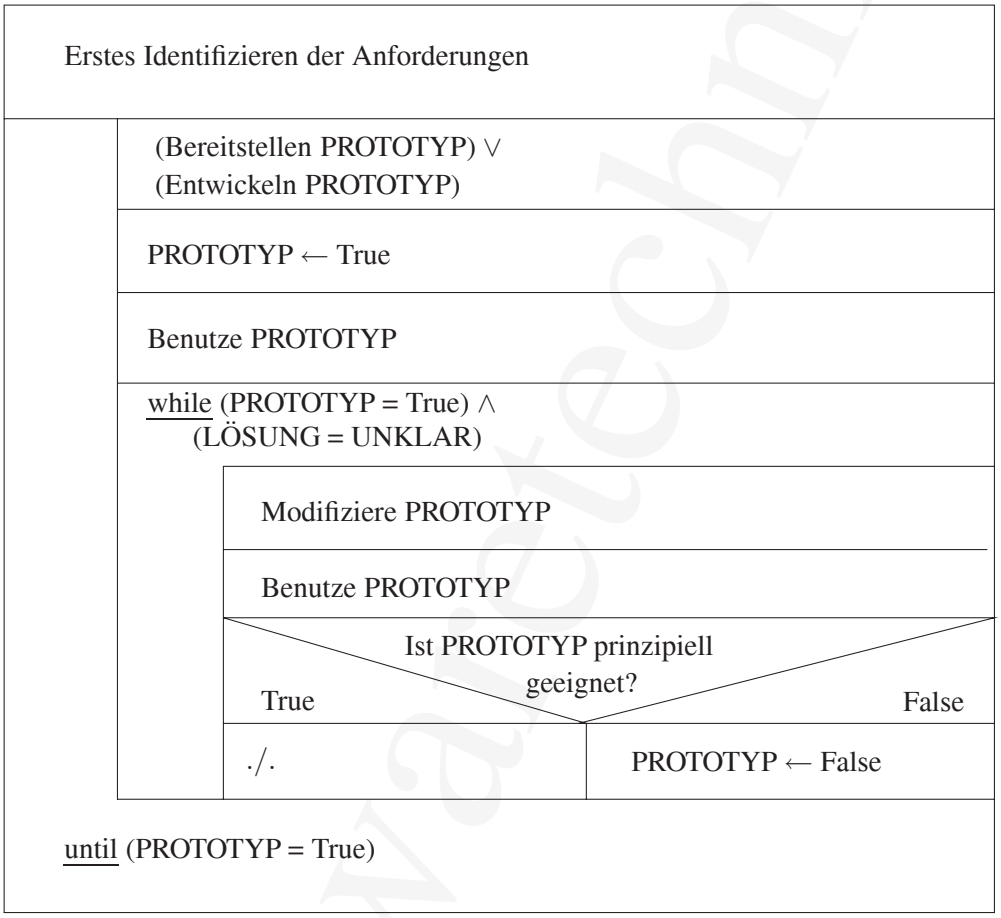
Das Lernen anhand von Prototypen ist an die Bedingung geknüpft, daß einerseits Fehler und andererseits erkannte Fehlentwicklungen und Unzulänglichkeiten kurzfristig ausräumbar sind. Es bedarf daher Vorkehrungen, um die einzelnen Veränderungsschritte des Prototyps so weit wie möglich zielorientiert zu steuern. Dazu gehört, daß Prototyping nicht im „aufsichtsfreien Raum“ stattfinden kann.

Konvergenz

Die Feststellung, daß die Fortentwicklung des Prototyps in die gewünschte Zielrichtung läuft und nicht von dieser divergiert, ist vom Auftraggeber im zeitlichen Zusammenhang mit dem jeweiligen Modifikationsschritt zu treffen. Ausreichende Reversibilität besteht selten über eine relativ lange Zeitspanne vieler Veränderungsschritte, sondern im wesentlichen zum jeweiligen Vorgängerschritt.

Die Erfolgskontrolle im Hinblick auf die Konvergenz mit dem angestrebten Ziel ist deshalb synchron zur Modifikationsschrittfolge zu vollziehen. Grundlage dafür ist die resultatsbezogene Kommunikation zwischen allen Akteuren (Beteiligten und Betroffenen). Eine solche Kommunikation, gestützt auf konkrete eigene Erfahrungen mit dem Leistungspotential des Prototyps, ist weit mehr geeignet, für den Entwicklungsprozeß bedeutsame Fakten aufzudecken als Befragungen, die ohne derartigen Erfahrungsfundus durchgeführt werden.

[3] In diesem Zusammenhang kommen außerdem in Betracht: Lernen anhand von Analogien, Lernen von Heuristiken. Lernen in der trivialsten Form besteht in der Aufnahme von Fakten als Auswendiglernen.



Legende:
Graphische Darstellung nach DIN 66261 (Struktogramm, vgl. [46])

Abbildung 3.1: Ablaufskizze *Prototyping*

3.1.2 Aufgaben und Arten

Eine allgemein anerkannte Begriffsdefinition ist für Prototyping weder in der DV-Praxis noch in der Wissenschaft erkennbar.

Begriffsklärung *Prototyping*:

> In der Regel wird Prototyping definiert im Bereich, der durch die Extremfälle **Wegwerfmuster**[4] und **Zwischenresultat** eines Entwicklungsprozesses[5] abgesteckt ist. Dabei werden arbeitsfähige Muster unterstellt. Teilweise werden in das Prototyping auch deskriptive Modelle und abstrakte Modelle miteinbezogen.

Prototyping ist im Rahmen der Realisierung einer komplexen Softwareentwicklung geeignet:

1. Zur Klärung und Festlegung des Automationsumfanges (der wünschenswerten Eigenschaften, der Anforderungen);
2. zur Überprüfung der Machbarkeit des Systemdesigns (der„Architekten“-Entwürfe) und
3. als iterative Vorgehensweise, wobei der Prototyp — durch Abdeckung zusätzlicher Anforderungen — zu einem immer leistungsfähigeren Produktionssystem ausgebaut wird.

Beim Prototyping wird unterschieden zwischen:

1. *exploratorischem Prototyping*,
2. *experimentellem Prototyping* und
3. *evolutionärem Prototyping*.

Das evolutionäre Prototyping wird auch als Entwicklung in Versionen (engl.: versioning) bezeichnet. Zur Abgrenzung von (1) und (2) wird häufig auch im Fall (3) ein Pilotsystem gesehen, das im Gegensatz zu Prototypen ordnungsgemäß entworfen, ingenieurmäßig konstruiert und auf eine längere Nutzungszeit ausgelegt ist.

Ideen zur Modifikation des Prototyps sollten durch die verwendete Prototyp-DV-Technik rasch umsetzbar (erprobbar) sein und nicht wegen ungenügenden dv-technischem Leistungspotential mit dem Hinweis

[4] *Expandable prototyping* oder *disposable prototyping*
[5] *Evolutionary prototyping* oder *incremental development*

in der Art „ist zu aufwendig“ abgewiesen werden müssen. Dieser Anspruch auf konkrete (Mit-)Gestaltung bedingt daher eine entsprechende Leistung der Basismaschine im Hinblick auf Flexibilität und Anpassungsfähigkeit.

> The prototype systems development strategy is based on expending dollars on hardware in order to get the most out of the people doing the development [29] S. 134.

In diesem Zusammenhang ist unwesentlich, ob der jeweilige Prototyp anschließend weggeworfen wird oder unmittelbar als Produktionssystem dient. Dies ist eine ökonomische Frage und eine prinzipielle Machbarkeitsfrage. Bei knappem Budget stößt die Bewilligung eines Wegwerfmusters erfahrungsgemäß auf besonders große Widerstände, so daß damit ein höherer Überzeugungs- und Durchsetzungsaufwand verbunden ist.

3.1.3 Zielerreichung & Endekriterium

Das Benutzen und Modifizieren eines Prototyps ist ein kreativer Prozeß, der das Finden neuer (Lösungs-)Ideen begünstigt. Ausgehend von den identifizierten groben Vorstellungen über abzudeckende Anforderungen ist ein geeigneter existierender Prototyp auszuwählen oder, falls nicht verfügbar, mit Hilfe eines geeigneten „Werkzeuges“ zu konstruieren.

Beim Benutzen des Prototyps werden Leistungen bewertet, Unzulänglichkeiten erkannt und Ideen zur Modifikation des Prototyps simultan entwickelt. Aufgrund der praktisch unbeschränkten Softwareverbesserungsmöglichkeiten stellt sich die Frage, wann der kreative (Lern-)Prozeß abzubrechen ist.

Im Falle der Überprüfung der Machbarkeit des Systementwurfes ist das Endekriterium „PROTOTYP = True“[6] relativ eindeutig als „Lösung ist lauffähig?“ interpretierbar.

Endekriterium

Zur Klärung und Festlegung der Benutzeranforderungen ist ein unmittelbar allen Akteuren einsichtiges Endekriterium schwierig definierbar. Auf der Basis eines fairen Entscheidungsprozesses zur Definition der Konsens- oder Kompromißlösungen ist unter Beachtung der Leitlinie schrittweises Vorgehen mit Teilnutzenrealisierung ein Endekriterium zu vereinbaren. Anhaltspunkt ist aus der ökonomischen Effizenzperspektive das überproportionale Anwachsen des Prototypingaufwandes bezogen auf die erwartbare Lösungsverbesserung.

Beim „*evolutionary*“ Prototyping ist die Vereinbarung eines geeigneten Endekriteriums mit einem geringen Fehlinvestitionsrisiko verknüpft,

[6] Vgl. Abbildung 3.1 Seite 60

da simultan mit der Prototypmodifikation eine nutzenerhöhende Betriebsverbesserung einhergeht.

3.1.4 Chancen

Der Zyklus aus Benutzen des Prototyps und Revidieren/Modifizieren des Prototyps bis zur Erfüllung des Endekriteriums bietet die Chance, aus den Beteiligten und Betroffenen Partner und Gefährten auf dem Weg zu einer tragfähigen und unter den gegebenen Umständen akzeptablen Lösung zu machen.

Die Partizipation beim Prototyping bezieht sich auf den gesamten Klärungsprozeß, das heißt sowohl auf die Bewertung des Prototyps (Analyse) als auch auf die Verbesserung des Prototyps (Design). Während erfahrungsgemäß beim Ablauf die Einschwingaktivitäten: Erstes Identifizieren der Anforderungen und Bereitstellen/Entwickeln eines arbeitsfähigen Prototyps vorwiegend Auftraggeber- (Bauherrn-) und Systementwickler- (Architekten-) orientiert erfolgen, bietet der Iterationsprozeß die Möglichkeit für die wirkungsvolle Partizipation[7] der tatsächlich Betroffenen (späteren Datenversorger und Datenentsorger).

Ob sich bei diesem Prozeß in der Praxis eine entsprechende partnerschaftliche Teamarbeit entwickelt, ist abhängig:

- einerseits von der positiven (Grund-)Einstellung der Akteure als Voraussetzung für eine konstruktive Teammitarbeit,
- andererseits vom organisatorischen und hard-/softwaretechnischen Umfeld.

3.1.5 Risiken

Im Zusammenhang mit einer wirksamen Partizipation stellt sich die Frage, ob die (Mit-)Gestaltung im Rahmen des Prototyping im Vergleich zu autoritären Architektenlösungen zu besseren Lösungen führt. Anders formuliert: Sind die Prototypbenutzer gute Architekten?

[7] Partizipationsschwerpunkte. In die Partizipation sind die vier folgenden Aspekte einzubeziehen:

1. Systembewertung,
2. Erklärungshilfe für unerwünschte Systemzustände (Bedarf und Schwachstellen),
3. Prognose künftigen Systemverhaltens und
4. Verbesserung des Systemverhaltens.

Diese Frage zeigt die große Abhängigkeit des Erfolges von den jeweiligen Benutzern des Prototyps. Es ist daher entscheidend, möglichst fähige Benutzer auszuwählen. Beim Prototyping werden nicht die am besten verfügbaren, sondern die besten Mitarbeiter benötigt.

Darüber hinaus können Aufwand und Umfang der organisatorischen Veränderungen für den Prototypeinsatz die rasche Korrektur von Fehlentscheidungen erschweren. Die Umkehrbarkeit der im Rahmen des Prototyping getroffenen organisatorischen Maßnahmen kann sich für den Auftraggeber zu einem Problem entwickeln. Auch das Prototyping birgt für den Auftraggeber die Gefahr prägender, im praktischen Sinne irreversibler Vorentscheidungen durch den Systementwickler. Es kommt daher auf eine Übereinstimmung der Intentionen von Auftraggeber und Systementwickler an, um diktatorische Alleingänge des Systementwicklers im Schutze des Prototyping zu vermeiden.

3.2 Rational Unified Process

„Der *Rational*[8] *Unified Process* ist ein Prozessmodell, dass unter anderem den Einsatz der *Unified Modeling Language* (UML) beschreibt. Der *Rational Unified Process* ist aber auch ein Prozessmodell, das Projektleiter in die Lage versetzt, objektorientierte Projekte zu managen.“(↪[58] S. V)

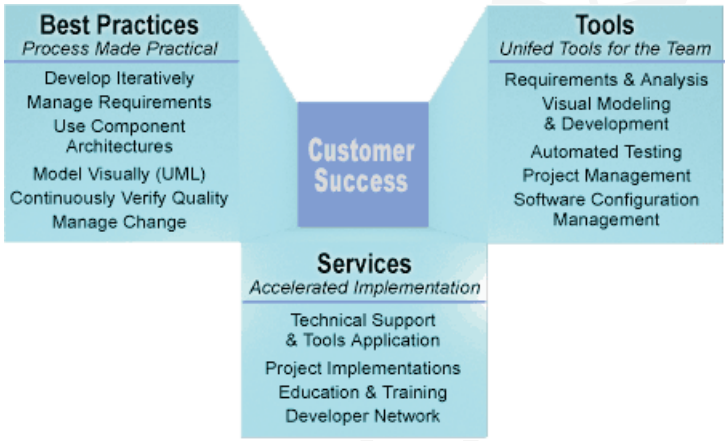
Es handelt sich dabei um ein objekt-orientiertes Prozessmodell auf der Basis von *Workflows*, die parallel über die folgenden vier Phasen ablaufen:

1. Konzeptualisierungs-*Phase*
2. Entwurfs-*Phase*
3. Konstruktions-*Phase*
4. Übergangsphase*Phase*

Charakteristisch für den *Rational Unified Process* ist die umfassende Unterstützung durch diverse Softwarewerkzeuge[9] und der Ansatz den

[8] Das Unternehmen *Rational Software* wurde 1981 gegründet und befasste sich zunächst primär mit der Programmierung in ADA.

[9] Derzeit (2003) zusammengefasst als *Rational Suite*^{TM} *Enterprise*:
„Rational Suite Enterprise is designed for software development project managers and software development teams that are faced with the challenge of completing complex projects in shorter periods of time.“↪`http://www.rational.com/products/entstudio/index.jsp` (online 19-May-2003)



Legende:
Quelle ↪`http://www.rational.com/images/paradox/keystone_450x278.gif` (online 19-May-2003)

Abbildung 3.2: Grundlagen für den *Rational Unified Process*

gesamten Entwicklungsprozess auf der Grundlage von vielfältigen Erfahrungen (*Best Practices*) zu gestalten (↪Abbildung 3.2 S. 65).

3.2.1 Workflows

Der *Rational Unified Process* besteht aus den folgenden fünf *Core Workflows*:

1. Geschäftsprozessmodellierungs-*Workflow*
2. Anforderungsmanagement-*Workflow*
3. Analyse- und Design-*Workflow*
4. Implementierungs-*Workflow*
5. Verteilungs-*Workflow*

Hinzu kommen noch die drei unterstützenden Workflows, die sogenannten *Supporting Core Workflows*:

1. Konfigurations- und Change-Management-*Workflow*
2. Projektmanagement-*Workflow*
3. Umgebungs-*Workflow*

3.2.2 Workflow-Elemente

Die Workflows werden durch die folgenden Elemente beschreiben:

1. Worker
   ≡ Personen, die innerhalb eines Vorhabens eine bestimmte Aktivität durchführen.

2. Artefakte
   ≡ ein Teil an Information, das produziert, modifiziert oder vom Prozess genutzt wird und dem Versionsmanagement unterliegt. Ein Artefakt kann ein Modell, ein Modellelement oder ein Dokument sein.

3. Aktivitäten
   ≡ in sich abgeschlossene Folgen von Tätigkeiten, deren Unterbrechung kein sinnvolles Ergebnis liefern würde.

4. Phasen
   ≡ zeitliche und/oder ergebisbezogene Abschnitte.

5. Konzepte

6. Toolmentoren
   ≡ Software[10], die bei der Verwendung von (neuen) Software-Tools hilft.

7. Richtlinien
   ≡ Regeln die angebe wie Aktivitäten abzuwickeln sind.

8. Templates
   ≡ Dokumentvorlagen, die Orientierung und Beispiele (für Neulinge) geben.

9. Reports

10. Checkpoints

Dabei kann ein *Workflow* selbst wieder aus *Workflows* bestehen, so gesehen kann ein *Workflow* ebenfalls ein Element von einem *Workflow* sein.

Im *Rational Unified Process* bezeichnet man diese Elemente auch als Schlüsselkonzepte. Sie stehen in enger Wechselwirkung, beispielsweise führen die *Worker Aktivitäten* durch, die als Ergebnis ein bestimmtes *Artefakt* produzieren.

[10] Zum Beispiel Toolmentor von Rational Rose.

3.3 Capability Maturity Model

SEI

Das *Capability Maturity Model for Software* (CMM oder auch SW-CMM; ≈ Reifegrad des Entwicklungprozesses) wurde (und wird) vom *Software Engineering Institute* (SEI) der *Carnegie Mellon University*[11] aufbauend auf Arbeiten von Watts Humphrey entwickelt. CMM beschreibt die Prinzipien und Praktiken um den Reifegrad des Softwareprozesses zu steigern. Dabei dient CMM zur Klassifikation des Reifegrades. CMM spezifiziert den Entwicklungsprozeß mit folgenden Mitteln (↪ Abbildung 3.3 S. 71):

- *Maturity Level*
  Er beschreibt den Reifegrad des Entwicklungsprozesses.
- *Key Process Area*
  Es handelt sich um Schlüsselprozesse, die je nach Reifegrad zu verfolgen sind.
- *Common Features*
  Es handelt sich um eine Unterteilung der Schlüsselprozesse in gemeinsame Aufgabenbereiche.
- *Key Practices*
  Es handelt sich um Anweisungen, die die Schlüsselprozesse erfüllen.

CMM ist in die fünf *Maturity Levels* gegliedert:

1. **Initial**
   Level I
   Dieser CMM-Reifegrad beschreibt eine Organisation, bei der keine stabile Umgebung für die Entwicklung und Wartung von Software vorhanden ist. In der Regel wird in kritischen Situationen von geplanten Vorgehensweisen abgewichen. Der Erfolg von Projekten hängt primär von Einzelpersonen ab. Die Leistungen der Organisation sind unkalkulierbar, da die Softwareentwicklung quasi als ein *ad hoc*-Prozeß durchgeführt wird. Weder die Kosten noch der Zeitbedarf sind planbar.(Hinweis: Das SEI betrachtet dies als niedrigste Reifegrad und identifiziert daher keine *Key Process Areas*.)
2. **Repeatable**
   Level II
   Die Projekte werden auf der Basis konkreter Erfahrungen geplant und durchgeführt. Es gibt Managementtechniken. Die Kosten und

[11] Copyright by Carnegie Mellon University, Web-Site ↪ `http://www.sei.cmu.edu/cmm/cmm.sum.html` (online 25-Nov-2003).

Zeitpläne sowie die Produktqualität werden überwacht. Eine Aufgabe ist die Institutionalisierung von Steuerungsprozessen, um ein erfolgreiches Management in Folgeprojekten wiederholen zu können. Es geht hier um:

- das Anforderungsmanagement, also um das Erfassen und Verwalten der Systemanforderungen sowie um die Reaktion auf Anforderungsänderungen.
- die Projektplanung, also um Aufwandsabschätzung, die Planung von Zeiten, Ressourcen und Budget sowie um die Analys von Risiken.
- die Projektüberwachung, also um das Berichtswesen, den Vergleich von IST und SOLL mit einer Kontrolle des Fortschrittes um Korrekturmaßnahmen durchzuführen.

Die *Key Process Areas* auf diesem Level sind:

- das Management von Subkontrakt, also die Auswahl von Partnern, die Vergabe von Aufgaben an Partner. — Die Planung, Durchführung, Kontrolle und das Berichtswesen vollziehen Partnern.
- die Qualitätssicherung, also das Erstellen von Plänen zur Gewährleistung von Qualität, die Prüfung der Prozeß- und Produktqualität sowie das Berichtswesen für die höheren Managementebenen.
- das Konfigurationsmanagement, also die Planung und Durchführung der Verwaltung aller Produkte im Entwicklungsprozeß.

In diesem Reifegrad werden Kundenanforderungen und Arbeitsprodukte kontrolliert. Es existieren grundlegende Managementtechniken. Damit besteht die Möglichkeit der Einsichtnahme und der Reaktion auf Abweichung durch das Management oder den Kunden— allerdings nur zu bestimmten Projektmeilensteinen. Die Abläufe zwischen den einzelnen Meilensteinen bleiben weitgehend ungesteuert.

Meilensteine

3. **Defined**

Level III

Der Prozeß zur Entwicklung und Wartung von Software ist dokumentiert. Er ist in der Organisation als *Standardsoftwareprozeß* definiert. Eine Expertengruppe ist für diesen Standardsoftwareprozeß verantwortlich und paßt ihn gegebenenfalls den aktuellen

Erfordernissen an. Es existieren Trainingsprogramme. Die *Key Process Areas* auf diesem Level sind:

- die Organisationsprozesse, also das Einführen von Verantwortlichkeiten sowie die Koordination der Prozeßverbesserung.
- die Prozeßdefinition, also die Entwicklung des Standardsoftwareprozesses sowie die Vorgaben für die projektspezifischen Anpassung.
- das Training, also die Planung und Durchführung von Trainingsprogrammen.
- das integrierte Softwaremanagement, also die Anpassung des Standardsoftwareprozesses an die Projektgegebenheiten sowie das Planen und Managen des projektbezogenen Softwareprozesses.
- die Produktentwicklung, also das Umsetzung des projektbezogenen Softwareprozesses.
- die Gruppenkommunikation, also die Aufrechterhaltung der Kommunikation zwischen den beteiligten Gruppen sowie die Verständigung über die Anforderungen und Aufgaben.
- die Reviews, also die Planung und Durchführung von Reviews sowie die Identifikation und Korrektur von Fehlern.

Im Gegensatz zum Reifegrad *Repeatable* sind nun die Teilaktivitäten zwischen den Meilensteinen vom Management und anderen Gruppen aus sichtbar. Es gibt einen Konsens über die Verantwortlichkeiten und definierte Rollen aller Prozeßbeteiligten.

4. **Managed**

Level IV

Metriken

Es sind quantitative Vorgaben zur Qualität definiert. In den einzelnen Projekten werden die Produktivität und die Qualität mit vorgeschriebenen Metriken gemessen. Die Ergebnisabweichungen von einzelnen Prozessen werden auf besondere Umstände oder geringe Störungen zurückgeführt. Es werden Abhilfemaßnahmen eingeleitet. Der Entwicklungsprozeß ist kalkulierbar. Es sind präzise Werte über Zeit-und Kostenziele angebbar. Die *Key Process Areas* auf diesem Level sind:

- das quantitative Prozeßmanagement, also die quantitative Prozesskontrolle sowie die Identifikation der Abweichungen.

- das Qualitätsmanagement, also die Entwicklung eines quantitativen Verständnisses für die Qualität des Produktes und des Erzeugungsprozesses.

Vergleichbar zum Reifegrad *Defined* sind nun auch die Zwischenphasen dem Management und/oder dem Kunden gegenüber durchschaubar. Zusätzlich sind aufgrund der quantitativen Ergebnisse auch Korrekturen in den Zwischenphasen möglich.

Level V

5. **Optimizing**
   Der Fokus liegt auf einer Verbesserung des Prozesses. Dazu werden Innovationen bei den Methoden, Techniken und Tools erprobt und gegebenfalls eingeführt. Stets wird angestrebt, den Entwicklungsprozeß selbst zu verbessern. Die *Key Process Areas* auf diesem Level sind:

   - die Fehlervermeidung, also die Identifikation von Fehlerquellen sowie die Änderung des Entwicklungsprozesses und die Übernahme der Änderungen in den Standardsoftwareprozeß.
   - der Technologiewandel, also die Identifikation und Übernahme besserer Techniken.
   - der Prozeßwandel, also die Verbesserung des Entwicklungsprozesses.

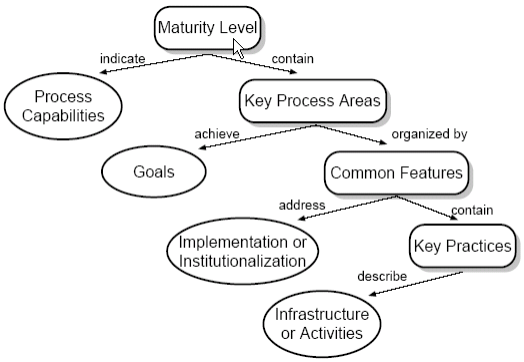
   Aufbauend auf den quantitativen Analyse des Reifegrades *Managed* wird der Entwicklungsprozeß in Richtung auf das Optimum angepasst. Dies betrifft neben den einzelne Zwischenphasen auch den gesamten Entwicklungsprozeß.

3.4 Kooperation — Domänenmodell & Systemmodell

Die Modellierung in der Informatik verfolgt primär zwei Ziele:

1. die Schaffung des *Domänenmodells*,
   also die Schaffung der Abbildung eines Ausschnittes der realen Welt und die Darstellung der Aufgabe der Informationsverabreitung,

2. die Schaffung des *Systemmodells*,
   also die Schaffung einer Vorlage für die Arbeitsweise eines informatischen Systems.



Legende:
Quelle: Universität GH Essen, Wirtschaftsinformatik und Softwaretechnik, ↪http://nestroy.wi-inf.uni-essen.de/Lv/mod1/05-handout.pdf (online 25-Nov-2003)

Abbildung 3.3: Capability Maturity Model

Die Modelle der Informatik haben nicht nur Ausprägungen im Sektor Computerhardware und Software, sondern auch im Sektor Planung, Organisation, Steuerung und Kontrolle. In diesem Kontext sind — zumindest bei größeren Projekten – vielfältige „Laien“ und Experten mit unterschiedlicher Prägung und verschiedenem Wissen sowie mannigfaltigen Erfahrungen zu beteiligen. Es geht daher um eine zielführende Kooperation zwischen allen Zubeteiligenden. Diese Kooperation ruht primär auf vier Säulen ([17] S. vii):

1. Pragmatik
2. Formalismen
3. Methodik
4. Werkzeuge

Dieser Modellierungsansatz mit starker Betonung der Kooperation läßt sich formaler beschreiben. Dazu wird eine Notation in *Extensible Markup Language* (XML) verwendet. Die Struktur der einzelnen XML-Elemente ist in Form einer *Document Type Definition* (DTD) notiert. Die Datei `Modellierung.xml` (S. 72) erläutert in dieser XML-Struktur die obigen vier Punkte. Die Datei `Modellierung.dtd` (S. 72) enthält die zugehörige DTD.

DTD-Datei `Modellierung.dtd`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Grundlagen der Kooperation bei der Modellierung
     im Sinne von Broy/Steinbrüggen
     Bonin 28-Jun-2004                    -->
<!ELEMENT Modellierung
  (Pragmatik, Formalismus, Methodik, Werkzeug)>
  <!ATTLIST Modellierung
    modellart (Domäne | System) "Domäne"
    perspektive CDATA #REQUIRED>
<!ELEMENT Pragmatik (Paragraph)+>
<!ELEMENT Formalismus (Paragraph)+>
<!ELEMENT Methodik (Paragraph)+>
<!ELEMENT Werkzeug (Paragraph)+>
<!ELEMENT Paragraph (#PCDATA)>
  <!ATTLIST Paragraph
    label ID #REQUIRED
    version (Abgenommen | Entwurf) "Entwurf"
    keywords CDATA #IMPLIED>
<!-- End of DTD: Modellierung.dtd          -->
```

XML-Datei `Modellierung.xml`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Grundlagen der Kooperation bei der Modellierung
     im Sinne von Broy/Steinbrüggen
     Bonin 28-Jun-2004                    -->
<!DOCTYPE Modellierung SYSTEM "Modellierung.dtd">
<Modellierung
  modellart="System"
  perspektive="Kooperation">
<Pragmatik>
<Paragraph label="p1"
  keywords="Beschreibungstechniken"
  version="Entwurf">
In der Softwaretechnik haben sich eine Reihe
von anschaulichen Beschreibungstechniken,
insbesondere diagrammatischer Art, herausgebildet.
</Paragraph>
<Paragraph label="p2"
  keywords="Boxologie"
  version="Entwurf">
Die vielfältigen Diagramme werden auch als Boxologie
bezeichnet.
</Paragraph>
</Pragmatik>
<Formalismus>
```

```
<Paragraph label="f1"
  keywords="Mathematik"
  version="Entwurf">
Wie üblich im Ingenieurbereich sind die Aneignung
und die Anwendung der einschlägigen Zweige der
Mathematik zwingend notwendig.
</Paragraph>
<Paragraph label="f2"
  keywords="Präzision, Kosten, Nutzen"
  version="Entwurf">
Die Mathematik gewährleitet Präzision und
Zuverlässigkeit. Sie hilft bei der Kostenbeherrschung
und der Nutzenabschätzung.
</Paragraph>
</Formalismus>
<Methodik>
<Paragraph label="m1"
  keywords="Abstraktion"
  version="Entwurf">
Die Abstraktion ist die Haupttechnik, um die
Komplexität von Software zu meistern.
</Paragraph>
<Paragraph label="m2"
  keywords="Approximation"
  version="Entwurf">
Die Abstraktion spielt eine ähnlich vereinfachende
Rolle wie die Approximation in anderen
Ingenieurbereichen.
</Paragraph>
</Methodik>
<Werkzeug>
<Paragraph label="w1"
  keywords="CASE"
  version="Abgenommen">
Methodisches Vorgehen erfordert in der Regel
den Einsatz von Spezialsoftware. Diese
bezeichnet man als CASE-Werkzeuge
(Computer Aided Software Engineering Tools).
</Paragraph>
</Werkzeug>
</Modellierung>
<!-- End of object Modellierung.xml  -->
```

Kapitel 4

Modellieren mit ET

In der Alltagspraxis müssen Problemlöser (Systemanalytiker, Systemdesigner oder Programmierer) häufig mit einer großen Menge von WENN . . . DANN . . . Beziehungen umgehen. Diese sind mit Entscheidungstabellen (kurz Singular & Plural: ET) präzise und transparent beschreibbar. Dabei bilden Regeln die Verknüpfung von Bedingungen und Aktionen.

Mehrere ET können zu einem Verbundsystem verknüpft werden. Sequenz, Selektion, Iteration (Rekursion) sind mittels ET abbildbar. Die ET-Technik ist daher eine vollständige formale (Programmier-)Sprache.

Wegweiser

Der Abschnitt *Modellieren mit ET* erläutert:

- den Zweck und die Anwendungsfelder,
  ↪Seite 76 . . .
- verschiedene ET-Typen,
  ↪Seite 79 . . .
- die zweckmäßige Schrittfolge zur ET-Erstellung,
  ↪Seite 84 . . .

- die ET-Analyse,
  ↪Seite 86 . . .

- ET-Verbundsysteme,
  ↪Seite 91 . . .

- die Linearisierung von nichtlinearen Kontrollstrukturen um sie dann als ET-Verbundsystem abbilden zu können,
  ↪Seite 100 . . .

- in Form einer Bilanzierung die Vor- und Nachteile einer ET-Modellierung und
  ↪Seite 110 . . .

- anhand von Übungen vielfältige Anwendungsmöglichkeiten.
  ↪Seite 113 . . .

4.1 Zweck & Anwendungsfelder

Im Rahmen einer Systemanalyse sind ET vorteilhaft bei Situationen (Sachverhalten), bei denen eindeutig definierbare Bedingungen zu eindeutig definierbaren Aktionen führen. Eine ET dokumentiert:

- die Voraussetzungen (Bedingungen), mit denen festgestellt wird, welcher Entscheidungsfall vorliegt, und
- die Handlungen (Aktionen), die in diesem Fall zu vollziehen sind (vgl. Tabelle 4.1 Seite 77).

Für die ET gibt es eine normierte Notation (vgl. DIN 66241). Wir ergänzen diese durch Bezeichner für Bedingungen und Aktionen (vgl. Tabelle4.1 Seite 77).

Der Zwang, den Sachverhalt in Bedingungen und Aktionen einzuteilen, unterstützt den Prozeß der Präzisierung einzelner Beschreibungen und Aussagen. Die tabellarische Anordnung und die Analysemöglichkeiten der dokumentierten Fälle im Hinblick auf (syntaktische) Vollständigkeit und (formale) Widerspruchsfreiheit schaffen mehr Transparenz (Durchsichtigkeit, Klarheit) über die darzustellende Situation.

Beim Grundaufbau einer ET gilt:

Tabellenname		R1,R2, . . . , Rn
B1	Bedingungsteil	Bedingungsanzeigerteil
B2		
⋮		
⋮		
⋮		
Bk		
A1	Aktionsteil	Aktionsanzeigerteil
A2		
⋮		
⋮		
⋮		
Am		

Legende:

$B1, \ldots, Bk$	$\equiv$ Bedingungsbezeichner
$A1, \ldots, Am$	$\equiv$ Aktionsbezeichner
$R1, \ldots, Rn$	$\equiv$ Regelbezeichner

Tabelle 4.1: Komponenten einer ET

- Zwischen einzelnen Bedingungen einer Bedingungsfolge $(B_1, \ldots, B_k)$ besteht eine logische UND-Beziehung[1].
- Zwischen einzelnen Aktionen einer Aktionsfolge $(A_1, \ldots, A_m)$ besteht eine logische UND-Beziehung.
- Eine Bedingungsfolge steht zur zugehörigen Aktionsfolge (gleiche Spalte) in einer $WENN \ldots DANN \ldots$ Beziehung, das heißt, trifft eine Bedingungsfolge zu, dann ist die entsprechende Folge von Aktionen auszuführen.
- Bei der ET vom Typ Eintreffer-ET sind die Regeln $(R_1, \ldots, R_n)$ in einem exklusiven ODER-Bezug[2], das heißt trifft eine Regel zu, dann trifft keine andere Regel mehr zu.[3]
- Bei der ET vom Typ Mehrtreffer-ET ist nach dem Zutreffen einer Regel zu prüfen, ob eine weitere Regel zutrifft. Ist dies der Fall, dann ist auch diese auszuführen. Die Richtung der Regelabarbeitung ist bei einer Mehrtreffer-ET vorab festzulegen. Üblicherweise ist eine Folge „von-links-nach-rechts“ definiert. Trifft eine Regel R_j zu, dann ist nach Vollzug die Prüfung bei Regel R_{j+1} fortzusetzen. (Näheres zu Mehrtreffer-ET vgl. zum Beispiel [57] oder [2].)

In der Mitte der fünfziger Jahre entwickelte man quasi gleichzeitig bei General Electric Company decision structure tables und bei Sutherland Company management rules. Die davor bekannten Hilfsmittel zur Analyse und Beschreibung von Prozessen (zum Beispiel Ablaufdiagramme) reichten für die komplexer werdenden Programme nicht mehr aus.

- $1959 \Rightarrow$ Erste Analysen und Dokumentationen mittels ET.
- $1960 \Rightarrow$ Entwicklung der ET-Sprache TABSOL.
- $1962 \Rightarrow$ Spezifikation der ET-Sprache DETAB-X. Sie sollte in COBOL-61 integriert werden. Dies gelang jedoch nicht.
- $1963 \Rightarrow$ ET-Precompiler für COBOL
- $heute \Rightarrow$ ET-Technik ist Bestandteil von leistungsfähigen CASE-Tools[4]

[1] Im folgenden notiert mit dem Symbol: $\wedge$

[2] Im folgenden notiert mit dem Symbol: $\vee$

[3] Vorausgesetzt die ET ist „korrekt“ , das heißt hier, sie ist ohne Redundanz.

[4] Akronym: Computer Aided Software Engineering $\approx$ Produktionshilfen für Softwaretechnik.

4.2 ET-Typen

Die ET lassen sich anhand ihrer Bedingungen und Bedingungsanzeiger einteilen. Damit unterscheidet man zum einen begrenzte, erweiterte und gemischte ET und zum anderen syntaktische vollständige ET. Bei einer bestimmten Notation der Regeln spricht man von der kanonischen Normalform.

Komplexe Bedingung:

- Die Bedingungen – eingetragen in den Feldern (B1, ..., Bk) – sind elementare syntaktische Einheit einer ET. Sie sind Variable mit einer endlichen Anzahl möglicher Ausprägungen.
- Die Bedingungen notieren wir mit den Symbolen $a, b, c, \ldots$.
- Die zu diesen Bedingungen gehörenden Mengen der Ausprägungen notieren wir mit: $[a], [b], [c], \ldots$.
- Die jeweiligen Ausprägungen bilden die Bedingungsanzeiger[5]. Für sie notieren wir die Symbole:

$$\begin{pmatrix} a_1 & a_2 & \ldots & a_m \\ b_1 & b_2 & \ldots & b_n \\ c_1 & c_2 & \ldots & c_k \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Einfache Bedingung:

- Für den Fall, daß eine Bedingung a zwei Ausprägungen hat, kann die Form: $[a] \equiv$ „J“ oder „N“ benutzt werden.

ELSE-Regel:

- Werden nicht alle Ausprägungen einer Bedingung x explizit angegeben, dann kann der ELSE-Bedingungsanzeiger sie abdecken. Er bezeichnet die Ausprägungen, die nicht durch andere Bedingungsanzeiger beschrieben sind.

Komplexe Aktion:

- Die Aktionen – eingetragen in den Feldern (A1, ..., Am) – sind elementare syntaktische Einheit einer ET. Sie sind Variable mit einer endlichen Anzahl möglicher Ausprägungen.

[5] engl.: condition entries

- Die Aktionen notieren wir mit den Symbolen $\alpha, \beta, \gamma, \ldots$.
- Die zu diesen Aktionen gehörenden Mengen der Ausprägungen notieren wir mit: $[\alpha], [\beta], [\gamma], \ldots$.
- Die jeweiligen Ausprägungungen bilden die Aktionsanzeiger. Für sie notieren wir die Symbole:

$$\begin{pmatrix} \alpha_1 & \alpha_2 & \ldots & \alpha_m \\ \beta_1 & \beta_2 & \ldots & \beta_n \\ \gamma_1 & \gamma_2 & \cdots & \gamma_k \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Einfache Aktion:

- Für den Fall, daß eine Aktion α zwei Ausprägungen hat, kann die Form: $[\alpha] \equiv$ „X“ oder „ “ benutzt werden. Dabei gilt:

  „X“ $\equiv$ Die Aktion ist durchzuführen.

  „ “ $\equiv$ Die Aktion ist nicht durchzuführen.

4.2.1 Begrenzte ET

Unter folgenden Voraussetzungen bezeichnen wir eine ET als begrenzt:[6]

- Eine ET, die innerhalb ihrer Regeln ausschließlich Bedingungsanzeigerkombinationen aus den Anzeigern:

  „J“ $\equiv$ Die Bedingung ist erfüllt.

  „N“ $\equiv$ Die Bedingung ist nicht erfüllt.

  „-“ $\equiv$ Die Anwendung der Regel ist nicht davon abhängig, ob die Bedingung erfüllt oder nicht erfüllt ist. Man kann dieses Zeichen als Dokumentation der Irrelevanz betrachten. Die Bedingung kann geprüft werden, nur ist das Prüfungsergebnis unerheblich.

  „♯“ $\equiv$ Ausschlußanzeiger. In diesem Fall ist die Entscheidung für die Bedingung nicht definiert; ihr Wahrheitswert kann nicht ermittelt werden.[7]

Irrelevanz

- und im Aktionsteil die Notationen:

[6] engl.: limited entry table

[7] Ist hier wegen der Vollständigkeit angegeben und wird in den Beispielen nicht verwendet.

ET-Auszahlung-buchen		R1	R2	R3	R4	R5	R6	R7
B1	Kontonummer angegeben?	J	J	J	J	N	N	N
B2	Kontonummer $<= 40000$?	J	J	J	N	-	-	-
B3	Betrag $>= 50.00$ DM ?	-	J	N	-	J	N	N
B4	Zweck genannt?	J	N	N	-	-	J	N
A1	Dem Chef vorlegen		X					X
A2	Zurück an die Fachabteilung schicken					X		
A3	Umkontieren auf Konto „Sonstiges“			X			X	
A4	Vorgang an Abteilung 2 abgeben				X			
A5	Buchung vollziehen	X		X			X	

Tabelle 4.2: Beispiel: Begrenzte Eintreffer-ET

„X“ $\equiv$ Die Aktion ist durchzuführen.

„“ $\equiv$ Die Aktion ist nicht durchzuführen.

aufweist

- ist eine begrenzte ET, weil die Definition der Bedingungen und Aktionen sich nicht in den Regelteil hineinzieht. Tabelle 4.2 Seite 81 zeigt eine begrenzte ET.

4.2.2 Erweiterte ET

Eine erweiterte ET[8], die innerhalb der Regeln andere Werte als die der begrenzten ET aufweist, das heißt deren Bedingungen und/oder Aktionen erst im Zusammenhang mit den Ausprägungen vollständig definiert sind, ist eine erweiterte ET. Eine erweiterte ET kann zweckmäßig sein, um die Zahl der Bedingungen und/oder Aktionen gering zuhalten. Im Vergleich zur begrenzten ET können so voneinander abhängige Bedingungen ($\equiv$ *nicht disjunkte* Bedingungen) vermieden werden.

Tabelle 4.3 Seite 82 zeigt eine erweiterte ET. Tabelle 4.4 Seite 82 bildet diesen Sachverhalt mit einer begrenzten ET ab.

[8] engl.: extended entry table

ET-Zahlungen-melden		R1	R2	R3
B1	Kontonummer?	638	752	891
B2	Einzahler?	Schulze	Krause	Meyer
B3	Währung?	DM	Gulden	DM
A1	Vollzug melden	X	X	X
A2	Einladung zum Abend-essen verschicken		X	

Tabelle 4.3: Beispiel: Erweiterte Eintreffer-ET

ET-Zahlungen-melden		R1	R2	R3
B1	Kontonummer = 638 ?	J	N	N
B2	Kontonummer = 752 ?	N	J	N
B3	Kontonummer = 891 ?	N	N	J
B4	Einzahler = Schulze ?	J	N	N
B5	Einzahler = Krause ?	N	J	N
B6	Einzahler = Meyer ?	N	N	J
B7	Währung = DM ?	J	N	J
B8	Währung = Gulden ?	N	J	N
A1	Vollzug melden	X	X	X
A2	Einladung zum Abend-essen verschicken		X	

Tabelle 4.4: Beispiel: Begrenzte Eintreffer-ET

ET-Wissenserwerb		R1	R2	R3
B1	Enthält $< text >$ das Wort „Software“ ?	J	N	N
B2	Enthält $< text >$ das Wort „Geld“ ?	-	J	N
A1	$< text >$ klassifizieren und archivieren	X		
A2	$< text >$ unverzüglich lesen	X	X	
A3	$< text >$ vergessen			X
A4	$< text >$ vernichten			X

Legende:
Syntaktische Vollständigkeitsprüfung:
2 einwertige Regeln (R2 und R3) $\Longrightarrow$ 2 Fälle
1 zweiwertige Regeln (R1) $\Longrightarrow$ 2 Fälle
$\hookrightarrow$ 4 Fälle $\equiv 2^2$ Regeln

Tabelle 4.5: Beispiel: Syntaktisch vollständige ET

4.2.3 Syntaktisch vollständige ET

Eine ET gilt als sytaktisch vollständig, wenn die in den Regeln aufgeführten Kombinationen der Bedingungsanzeiger alle formal zulässigen elementaren Bedinungsanzeigerkombinationen repräsentieren und jeder zulässige Aktionsanzeiger Bestandteil mindestens einer der aufgeführten Aktionsanzeigerkombinationen ist. Tabelle 4.5 Seite 83 zeigt eine syntaktisch vollständige begrenzte Eintreffer-ET.

4.2.4 ET in kanonischer Normalform

Eine Et in kanonischer Normalform hat folgende Eigenschaften:

- Die ET verfügt über Regeln mit elementaren Bedingungsanzeigerkombinationen. Die Regeln sind nach Maßgabe ihrer Bedingungsanzeigerkombinationen lexikographisch geordnet.
- Die ET befindet sich in syntaktisch vollständiger Form, wenn alle elementaren Bedingungsanzeigerkombinationen in den Regeln aufgeführt sind.

kanonische Normalform

- Die ET in syntaktisch vollständiger Form bildet das Maximum an möglichen Regeln ab.

Tabelle 4.6 Seite 84 zeigt eine syntaktisch vollständige ET in kanonischer Normalform.

ET-FOO		R1	R2	R3	R4	R5	R6	R7	R8
B1	Bedingung-1 ?	J	J	J	J	N	N	N	N
B2	Bedingung-2 ?	J	J	N	N	J	J	N	N
B3	Bedingung-3 ?	J	N	J	N	J	N	J	N
A1	Aktion-A	X							
A2	Aktion-B		X						X
A3	Aktion-C			X				X	
A4	Aktion-D				X		X		
A5	Aktion-E					X			

Tabelle 4.6: Beispiel: Kanonische Normalform

ET-Mitgliedschaft		R1	R2	R3
B1	Alter $<$ 18 Jahre ?	J	N	N
B2	Alter $<$ 65 Jahre ?	-	J	N
A1	Jugendliches Mitglied	X		
A2	Erwachsenes Mitglied		X	
A3	„Senioren“			X

Legende:
Nicht mögliche Kombination mit Hilfe des Zeichens irrelevant („-“) versteckt.

Tabelle 4.7: Beispiel: Abhängige Bedingungen & Irrelevanz

4.2.5 ET mit abhängigen Bedingungen

Bei der kanonischen Normalform unterstellen wir, daß die Bedingungen voneinander unabhängig sind ($\equiv$ disjunkte Bedingungen), damit alle Kombinantionen auch tatsächlich vorkommen können. Sind die Bedingungen jedoch untereinander abhängig, dann können nicht alle Regeln der kanonischen Normalform „sinnvoll“ sein.

In einem solchen Fall haben wir zwei Möglichkeiten. Entweder verstecken wir die Unmöglichkeit der logischen Kombination mit Hilfe des Zeichens „Irrelevant“ (vgl. Tabelle 4.7 Seite 84) oder führen eine zusätzliche „Warnungsaktion“ (vgl. Tabelle 4.8 Seite 85) ein.

4.3 Schrittfolge zur ET-Erstellung

Eine stets gültige Empfehlung zum Vorgehen beim Erstellen einer ET ist nicht angebbar. Im Sinne einer Daumenregel sind die folgenden Schritte

ET-Mitgliedschaft		R1	R2	R3	R4
B1	Alter $<$ 18 Jahre ?	J	J	N	N
B2	Alter $<$ 65 Jahre ?	J	N	J	N
A1	Jugendliches Mitglied	X			
A2	Erwachsenes Mitglied			X	
A3	„Senioren“				X
A4	Unmögliche Kombination		X		

Legende:
Nicht mögliche Kombination mit Hilfe einer zusätzlichen Aktion kenntlich gemacht.

Tabelle 4.8: Beispiel: Abhängige Bedingungen & zusätzliche Aktion

gedacht, wenn eine verbale Situationsbeschreibung vorliegt oder zunächst zu erheben ist.

1. Logisches Aufteilen eines komplexen Sachverhaltes ($\approx$ Strukturieren) in mehrere „einfachere“ Sachverhalte, die jeweils in einer ET abbildbar sind (Näheres dazu vgl. Abschnitt 4.5 Seite 91). **Strukturieren**
2. Feststellen aller Bedingungen und Aktionen, welche die $WENN$ ... $DANN$ ... Beziehungen eines Sachverhalt betreffen. Markieren dieser Bedingungen und Aktionen im Text.
3. Aufstellen einer Liste der markierten Bedingungen.
4. Aufstellen einer Liste der markierten Aktionen.
5. Gleiche oder „ähnliche“ Bedigungen und Aktionen aus den Listen streichen.
6. Bedingungen und Aktionen präzisieren, das heißt im Regelfall neu und zwar mehr mathematisch formulieren. Ziel sind kurze prägnante Aussagen.
7. Tabellenname vergeben und in ET eintragen.
8. Bedingungen und Aktionen (logisch sinnvoll) in ET eintragen.
9. Bedingungs- und Aktionsanzeiger setzen.
10. ET-Analyse.
11. ET-Abnahme

12. Für den nächsten (Teil-)Sachverhalt eine weitere ET aufstellen bis der komplexe (Gesamt-)Sachverhalt ganz abgebildet ist.

4.4 ET-Analyse

Bedingungen, Aktionen und Regeln sollen eine gegebene Situation realitätsnah abbilden. Die ET-Analyse kann die Korrektheit dieser Realitätsabbildung nur bedingt überprüfen. Inhaltlich fehlende oder inhaltlich falsche Bedingungen und/oder Aktionen sind nicht erkennbar. Die analysierbaren Lücken und Widersprüche bei den Regeln können jedoch Anlaß sein, den ET-Erstellungsprozeß nochmals zu überprüfen.

Ausgehend von einer syntaktisch korrekten ET umfaßt die ET-Analyse folgende Verfahren:

- Konsolidierung, das heißt Verringerung des ET-Umfanges durch Zusammenlegung von Regeln.
- Widerspruchstest und Redundanztest, das heißt Erkennen und Beseitigen von Widersprüchen und „überflüssigen“ Regeln.
- syntaktische Vollständigkeitsprüfung, das heißt Erkennen von fehlenden Regeln.

4.4.1 Konsolidierung

Bei der Konsolidierung einer ET geht es um die Beschreibung der Situation mit einer möglichst kleinen Anzahl von Regeln. Die These lautet:

> Je weniger Regeln in die Betrachtung einzubeziehen sind,
> desto leichter ist eine ET zu durchschauen.

Starte mit Aktions-anzeiger!

Eine Zusammenfassung von einzelnen Regeln, also eine Konsolidierung, setzt voraus:

1. Bedingungsanzeigerfolgen, die zu identischen Aktionsanzeigerfolgen führen. Es werden daher zunächst die Aktionsanzeigerfolgen der einzelnen Regeln betrachtet. Enthält die ET keine gleichen Aktionsanzeigerfolgen, dann ist auch keine Konsolidierung möglich.
2. Regeln mit identischen Aktionsanzeigerfolgen unterscheiden sich nur in **einem** Bedingungsanzeiger.
3. Die konsolidierte Regel ist noch nicht in der ET enthalten.

Auf eine konsolidierte Regel kann das Konsolidierungsverfahren erneut angewendet werden, wenn die genannten Konsolidierungsvoraussetzungen mit weiteren Regeln bestehen.

Die obige Voraussetzung 3 ist für eine begrenzte ET wie folgt zu interpretieren:

> Innerhalb einer Gruppe, bei der alle Entscheidungsregeln gleiche Aktionsanzeigerfolgen haben, sind Regelpaare festzustellen, deren Bedingungsanzeigerfolgen sich nur in einem Bedingungsanzeiger („J“ oder „N“ ; „J“ oder „-“; „N“ oder „-“) unterscheiden. Jedes derartige Regelpaar wird durch eine neue Regel ersetzt, bei der anstatt des differierenden Bedingungsanzeigers das Zeichen für Irrelevanz („–“) steht. Sonst weist diese verdichtete Regel die bisherigen Bedingungs- und Aktionsanzeigerfolgen auf.

Die obige Voraussetzung 3 ist für eine erweiterte ET wie folgt zu interpretieren:

> Zur Konsolidierung ist das skizzierte Verfahren entsprechend anzuwenden. Es ist eine Bedingung festzustellen, die mit allen (Anzeiger-)Ausprägungen in der Gruppe vertreten ist, wobei die betroffenen Bedingungsanzeigerfolgen sich nur in diesem Bedingungsanzeiger unterscheiden. Ist dies der Fall, dann ist für diese Bedingung das Zeichen für Irrelevanz („–“) zu setzen.

Tabelle 4.9 Seite 88 zeigt ein Konsolidierungsbeispiel.

Eine Variante der ET-Konsolidierung bietet die sogenannte ELSE-Regel. Bei einer ET mit mehreren unterschiedlichen Bedingungsanzeigerfolgen und gleichen Aktionsanzeigerfolgen kann eine Zusammenfassung zu einer Regel, die im Bedingungsanzeigerteil die Eintragung `ELSE` (oder „Sonst“) aufweist, angebracht sein. Voraussetzung ist, daß die anderen Fälle durch die bleibenden Regeln syntaktisch vollständig abgedeckt werden.

Die `ELSE`-Regel birgt die Gefahr, daß vorher nicht ausgewiesene, aber mögliche Bedingungsanzeigerfolgen, danach automatisch zur Aktionsanzeigerfolge ELSE führen. Dies muß nicht stets dem gegebenen Sachverhalt entsprechen. Tabelle 4.10 Seite 89 zeigt eine Konsolidierung mit der ELSE-Regel.

4.4.2 Widerspruchs- und Redundanztest

Eine Eintreffer-ET enthält einen formalen Widerspruch, falls mindestens zwei Bedingungsanzeigerfolgen inhaltlich gleich sind, aber zu un-

ET-FOO vor der Konsolidierung:

ET-FOO		R1	R2	R3	R4	R5	R6	R7	R8
B1	Bedingung-1 ?	J	J	J	J	N	N	N	N
B2	Bedingung-2 ?	J	J	N	N	J	J	N	N
B3	Bedingung-3 ?	J	N	J	N	J	N	J	N
A1	Aktion-A	X							
A2	Aktion-B		X						X
A3	Aktion-C			X	X			X	
A4	Aktion-D			X	X		X		
A5	Aktion-E					X			

Konsolidierbar sind die beiden Regeln R3 und R4.
Die neue Regel ist R'.

ET-FOO		R1	R2	R'	R5	R6	R7	R8
B1	Bedingung-1 ?	J	J	J	N	N	N	N
B2	Bedingung-2 ?	J	J	N	J	J	N	N
B3	Bedingung-3 ?	J	N	-	J	N	J	N
A1	Aktion-A	X						
A2	Aktion-B		X					X
A3	Aktion-C			X			X	
A4	Aktion-D			X		X		
A5	Aktion-E				X			

Tabelle 4.9: Beispiel: Konsolidierung

ET-BAR vor der Konsolidierung:

ET-BAR		R1	R2	R3	R4	R5	R6	R7	R8
B1	Bedingung-1 ?	J	J	J	J	N	N	N	N
B2	Bedingung-2 ?	J	J	N	N	J	J	N	N
B3	Bedingung-3 ?	J	N	J	N	J	N	J	N
A1	Aktion-A	X					X		
A2	Aktion-B		X	X	X	X		X	X

Konsolidierbar sind die Regeln R2,R3,R4,R5,R7 und R8.
Die neue Regel ist R'.

ET-BAR		R1	R6	R'
B1	Bedingung-1 ?	J	N	E L S E
B2	Bedingung-2 ?	J	J	
B3	Bedingung-3 ?	J	N	
A1	Aktion-A	X	X	
A2	Aktion-B			X

Tabelle 4.10: Beispiel: ELSE-Regel

ET-Widerspruchs- und -Redundanztest		R1	R2	R3
B1	Bedingungsanzeigerfolgen sind gleich ?	J	J	N
B2	Aktionsanzeigerfolgen sind gleich ?	J	N	-
A1	Widerspruchsbeseitigung durchführen		X	
A2	Redundanzbeseitigung durchführen	X		
A3	Nach Korrektur erneut ET-Widerspruchs- und -Redundanztest anwenden	X	X	
A4	Geprüfte ET ohne Widerspruch und Redundanz.			X

Tabelle 4.11: Widerspruchs- und Redundanztest

terschiedlichen Aktionsanzeigerfolgen führen. Zur Auflösung eines solchen Widerspruchs sind ergänzende Informationen (Rückfragen oder Recherchen) erforderlich; es sei denn, es handelt sich um einen behebbaren Erstellungsfehler.

Starte mit Bedingungsanzeiger!

Sind mindestens zwei Bedingungsanzeigerfolgen gleich und führen diese zu gleichen Aktionsanzeigerfolgen, dann genügt eine Regel. Die redundanten Regeln sind zu streichen.

Die Tabelle 4.11 Seite 90 skizziert das Verfahren zum Testen auf Widersprüche und Redundanz bei ET.

4.4.3 Syntaktische Vollständigkeitsprüfung

Begrenzte ET

Bei k Bedingungen ist die syntaktische Vollständigkeit bei 2^k Regeln gegeben. Zum Beispiel sind bei 5 Bedingungen 32 unterschiedliche Regeln erforderlich. Zu beachten ist bei der Bestimmung dieser Vollständigkeit das Irrelevanzzeichen. Jedes Irrelevanzzeichen symbolisiert die beiden Fälle J und N. Zur Bestimmung der Vollständigkeit gilt: Ist die Zahl der Irrelevanzzeichen in einer Regel gleich j, dann steht diese Regel für 2^j Regeln. Tabelle 4.12 Seite 91 zeigt eine syntaktisch unvollständige ET.

Irrelevanzzeichen

Erweiterte ET

Die Bestimmbarkeit der Regelanzahl für die syntaktische Vollständigkeitsprüfung einer ET bedingt, daß alle möglichen Anzeigerausprägungen jeder Bedingung aus der ET (oder ihrer zusätzlichen Dokumentation) entnehmbar sind. Syntaktische Vollständigkeit liegt vor, wenn die Anzahl der unterschiedlichen Regeln gleich dem Produkt aus der Anzahl der Ausprägungen der einzelnen Bedingungen ist.

Beispiel: Verkehrsampel

ET-BAZ		R1	R2	R3
B1	Bedingung-1 erfüllt?	J	N	N
B2	Bedingung-2 erfüllt?	-	-	N
B3	Bedingung-3 erfüllt?	-	J	N
A1	Aktion-A	X		X
A2	Aktion-B		X	X

Legende:
Syntaktische Vollständigkeitsprüfung:
1 einwertige Regeln (R3) $\Longrightarrow$ 1 Fälle
1 zweiwertige Regeln (R2) $\Longrightarrow$ 2 Fälle
1 vierwertige Regeln (R1) $\Longrightarrow$ 4 Fälle
$\hookrightarrow$ 7 Fälle $\not\equiv 2^3$ Regeln

$\mapsto$Es fehlt die Bedingungsanzeigerfolge: N J N

Tabelle 4.12: Beispiel: Unvollständige ET

$B1$ Ampel: grün, gelb, rot, rot und gelb, gelb blinkt
$\hookrightarrow$ 5 Anzeiger
$B2$ Wetter: Sonne, Regen, Schnee
$\hookrightarrow$ 3 Anzeiger
$B3$ Tempo: null, mittel, schnell, sehr schnell
$\hookrightarrow$ 4 Anzeiger
$\Longrightarrow$ syntaktische Vollständigkeit: $5 * 3 * 4 = 60$ Regeln.

4.5 ET-Verbundsystem

Üblicherweise erfordert die Abbildung eines Sachverhaltes eine so große Anzahl von Regeln, daß eine ET unübersichtlich und unhandlich wird. Pragmatisch betrachtet: Die ET paßt nicht mehr auf ein DIN A4 Blatt, wenn sie mehr als ca. 16 Regeln aufweisen müßte. Um die Transparenz zu sichern, ist der Sachverhalt daher in mehr als einer ET darzustellen. Die ET sind miteinander zu verknüpfen. Es entsteht ein ET-Verbundsystem.

Beispiel: Aufteilung von 6 Bedingungen in drei ET

- Wir nehmen an, es sei eine begrenzte einfache ET mit 6 Bedingungen aufzustellen. Diese habe für die syntaktische Vollständigkeit 64 Regeln.

- Eine Aufteilung dieser ET in drei ET aufgrund der logischen Zusammenhänge führt zu $3 * 4 = 12$ Regeln. Die drei ET haben insgesamt nur ca. $1/5$ der Regeln. Ihre Darstellung ist wesentlich leichter zu durchschauen.

Die Aufteilung (Strukturierung) eines komplexen Sachverhaltes in einzelne durchschaubare ET (Module) erfordert eine Dokumentation der Verknüpfung zwischen den einzelnen ET, um den Gesamtzusammenhang abzubilden. Nach der Art dieser Verknüpfung unterscheidet man zwischen offenen Verbundsystemen und geschlossenen Verbundsystemen.

4.5.1 Offenes ET-Verbundsystem

GO TO

Die Verknüpfung der ET mit einer anderen ET erfolgt über eine (zusätzliche) Aktion, die eine GO TO Anweisung (ohne Rücksprunganweisung) enthält.

Beispiel: $A3$ GO TO ET 1.4.1

4.5.2 Geschlossenes ET-Verbundsystem

Um eine ET als Unterprogramm (mehrfach aufrufbare Entscheidungssituation) gestalten zu können, enthält die aufrufende ET eine (zusätzliche) Aktion: „Führe ET-XY aus und fahre anschließend mit der nächsten Aktion (bzw. Bedingung) fort“. Dies entspricht dem `PERFORM` in COBOL oder dem üblichen *run*-Statement in sogenannten Pseudocode-Notationen. Die aufgerufene ET ($\equiv$ das Unterprogramm) kann zur Verdeutlichung eine (zusätzliche) Aktion: „Kehre zurück!“ ($\equiv$ return) aufweisen. Die so verknüpften ET bilden ein geschlossenes ET-Verbundsystem.

Die Verknüpfungsform mittels GO TO Anweisung ist selten zweckmäßig. Im Regelfall ist ein geschlossenes Verbundsystem vorzuziehen, weil dann die Rückkehrstelle im voraus feststeht.

run & return

Mit den Aktionen run und return lassen sich Verbundsysteme realisieren, deren ET (Module) hierarchisch miteinander verknüpft sind. Bei einer Hierarchie enthält zum Beispiel die oberste ET-1 eine Aktion A_i : run ET-1-1 und diese dann eine Aktion A_j : run ET-1-1-1 usw. Mit Hilfe eines solchen hierarchischen Verbundsystems können wir leicht den sogenannten „top down“-Ansatz in der Systementwicklung verwirklichen.

Tabelle 4.13 Seite 93 zeigt eine ET, die durch ein Verbundsystem mit drei ET ersetzt wird.

ET-GESAMT		R1	R2	R3	R4	R5	R6
B1	$Q = TRUE$?	J	J	J	N	N	N
B2	$R = TRUE$?	J	J	N	J	N	N
B3	$S = TRUE$?	J	N	-	-	J	N
A1	Aktion-A	X	X		X	X	X
A2	Aktion-B	X		X	X	X	X
A3	Aktion-C	X	X	X	X	X	
A4	Aktion-D	X	X	X			
A5	Aktion-E	X	X	X		X	X

Verbundsystem als Ersatz für ET-GESAMT:

ET-TOPLEVEL		R1	R2
B1	$Q = TRUE$?	J	N
A1	run ET-Q-TRUE	X	
A2	run ET-Q-FALSE		X

ET-Q-TRUE		R1	R2	R3
B1	$R = TRUE$?	J	J	N
B2	$S = TRUE$?	J	N	-
A1	Aktion-A	X	X	
A2	Aktion-B	X		X
A3	Aktion-C	X	X	X
A4	Aktion-D	X	X	X
A5	Aktion-E	X	X	X
A6	return	X	X	X

ET-Q-FALSE		R1	R2	R3
B1	$R = TRUE$?	J	N	N
B2	$S = TRUE$?	-	J	N
A1	Aktion-A	X	X	X
A2	Aktion-B	X	X	X
A3	Aktion-C	X	X	
A4	Aktion-E		X	X
A5	return	X	X	X

Legende:
In ET-Q-FALSE fällt die Aktion „Aktion-D“ fort, da diese nicht angekreuzt ist.

Tabelle 4.13: Beispiel: Geschlossenes ET-Verbundsystem

4.5.3 *run*-ET-*selbst*

Wir erweitern die Einsatzmöglichkeiten von ET durch Abbildung der Iteration. Eine Iteration[9] ermöglicht das wiederholte Anwenden (Durchlaufen) einer ET. Abbildung 4.1 Seite 95 zeigt die graphische Darstellung einer Iteration in Jackson Notation [36]. Der Modul GESAMT besteht aus einer beliebig langen Folge von Moduln WIEDERHOLUNGSTEIL. Dabei ist zu beachten:

- GESAMT ist die Iteration und
- die Häufigkeit des Auftretens von WIEDERHOLUNGSTEIL ist eine Eigenschaft von GESAMT und nicht von WIEDERHOLUNGSTEIL.

Kurzzeichen *

Der Stern in der rechten oberen Ecke dokumentiert das mehrfache Auftreten von WIEDERHOLUNGSTEIL innerhalb von GESAMT. Eine Iteration wird beendet:

- entweder durch Zutreffen einer Abbruchbedingung, wobei die Ereignisprüfung vor oder nach einem Durchlauf erfolgen kann,
- oder nach Vollzug einer vor Beginn der Iteration festgelegten Anzahl von Durchläufen.

In vielen Programmiersprachen werden zur Kennzeichnung einer Iteration Bezeichner wie `DO, WHILE, REPEAT, UNTIL` oder `FOR` benutzt.

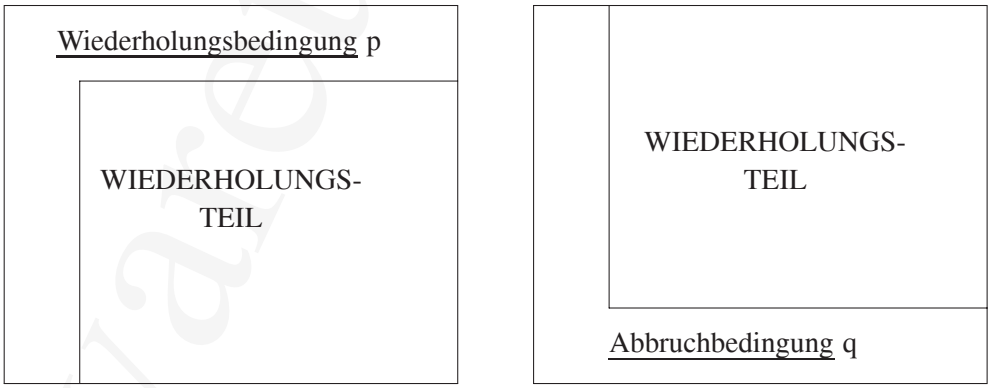
Schleifentyp

Im Rahmen des Entwurfes von linearen Kontrollstrukturen werden Iterationen oft in der von I. Nassi und B. Shneiderman 1973 vorgeschlagenen Form dargestellt (vgl. Abbildung 4.2 Seite 95). Die Iteration mit der Bedingungsprüfung vor jedem Wiederholungsdurchlauf wird auch abweisende Schleife oder salopp: „kopfgesteuerte“ Schleife genannt. Sie ist zu unterscheiden von der Iteration mit der Bedingungsprüfung nach jedem Wiederholungsdurchlauf. Diese wird als nichtabweisende Schleife oder salopp: „fußgesteuerte“ Schleife bezeichnet. Im ersten Fall, dem while-Konstrukt, wird so lange wiederholt wie die Bedingung erfüllt ist. Im zweiten Fall, dem until-Konstrukt, wird der Wiederholungsvorgang beendet, wenn die Bedingung erfüllt ist.

Die Iteration kann bei ET mit einer Aktion „run ET-selbst“ (oder „Wiederhole ET-selbst“) dargestellt werden. Ein Beispiel zeigt Tabelle 4.14 Seite 96.

[9] auch Repetition genannt

Abbildung 4.1: Graphische Darstellung der Iteration



Wiederholung mit vorausgehender Bedingungsprüfung (kurz: while-Konstrukt)	Wiederholung mit nachfolgender Bedingungsprüfung (kurz: until-Konstrukt)
Notiert in einem Pseudocode:	
while p do B od	do B until q od

Abbildung 4.2: Iteration — Struktogramm (DIN 66261)

$AKKUMULATOR \leftarrow 0$

ET-Wortzählung		R1	R2	R3
B1	Enthält $< text >$ kein Wort?	J	N	N
B2	Erste Wort im $< text >=< wort >$?	-	J	N
A1	$AKKUMULATOR \leftarrow 1 + AKKUMULATOR$		X	
A2	$< text > \leftarrow < text\ reduziert\ um\ erstes\ Wort >$		X	X
A3	Wiederhole ET-Wortzählung		X	X
A4	Gib AKKUMULATOR aus	X		

Tabelle 4.14: Beispiel: Iteration

Dabei handelt es sich stets um ein while-Konstrukt. Ist ein vorgegebenes until-Konstrukt in einer ET abzubilden, dann ist es zunächst durch eine Ersatzlösung mit einem while-Konstrukt abzubilden. Abbildung 4.3 Seite 97 zeigt die Transformation.

Auch eine rekursive Lösung läßt sich mit Hilfe von ET notieren. Tabelle 4.15 Seite 98 zeigt die iterative Lösung (Tabelle 4.14 Seite 96) in rekursiver Form. Dabei verwenden wir die Notation entsprechend dem Fakultätsbeispiel in der Originalarbeit von I. Nassi und B. Shneiderman [46].

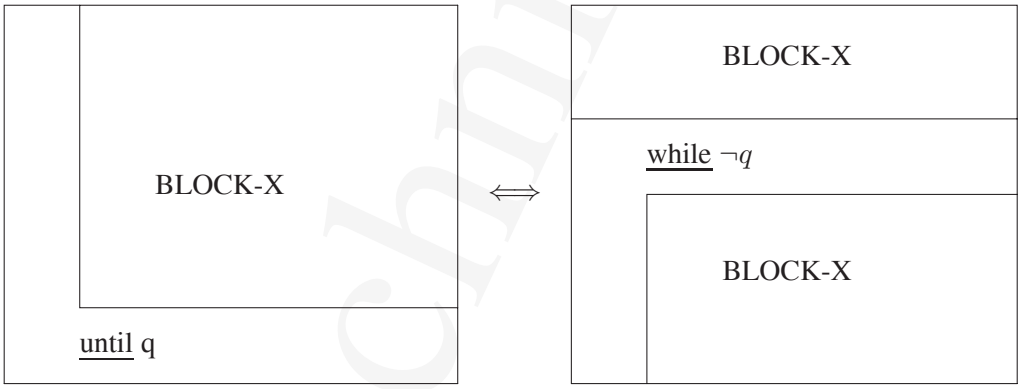
4.5.4 ET ohne Bedingung

Ein Problem ist die Abbildung einer Sequenz; zum Beispiel das Setzen der Iterationsvariablen auf einen Anfangswert. Ist zunächst – ohne eine Bedingung – eine Aktion durchzuführen, bevor bedingte Aktionen folgen, dann bedarf es für einen solchen Vorlauf einer „konsequenten“ Interpretation der ET-Technik.

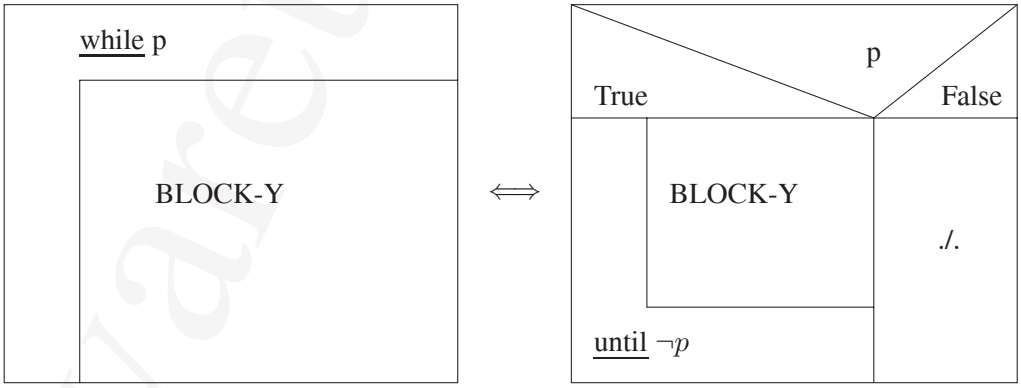
Wir unterstellen, daß auch bei keiner Bedingung sich die Anzahl der Regeln bei einer begrenzten Eintreffer-ET nach 2^n mit $n = 0$ berechnet. Da $2^0 = 1$, kann die ET-VORLAUF eine Regel ausweisen. Tabelle 4.16 Seite 98 zeigt eine ET ohne Bedingung.

In der Praxis bildet man diesen Vorlauf auch mit Hilfe einer mehr oder weniger willkürlichen gewählten „Ersatzbedingung“ab. Pragmatisch läßt sich eine solche Anfangssequenz auch direkt über die erste ET schreiben (vgl. Tabelle 4.14 Seite 96). Tabelle 4.17 Seite 100 verdeutlicht dieses Notationsproblem anhand der Sequenz von Abbildung 4.4 Seite 99.

Fall 1: *Until*-Konstrukt ersetzt durch ein *while*-Konstrukt



Fall 2: *While*-Konstrukt ersetzt durch ein *until*-Konstrukt



Legende:
Vgl. Abbildung 4.2 Seite 95

„./.“ ≡ Identitätsfunktion (zweckmäßig zur Kennzeichnung, daß eine Blockeintragung nicht vergessen wurde).

Abbildung 4.3: *while*⇔*until*-Konstruktumwandlung

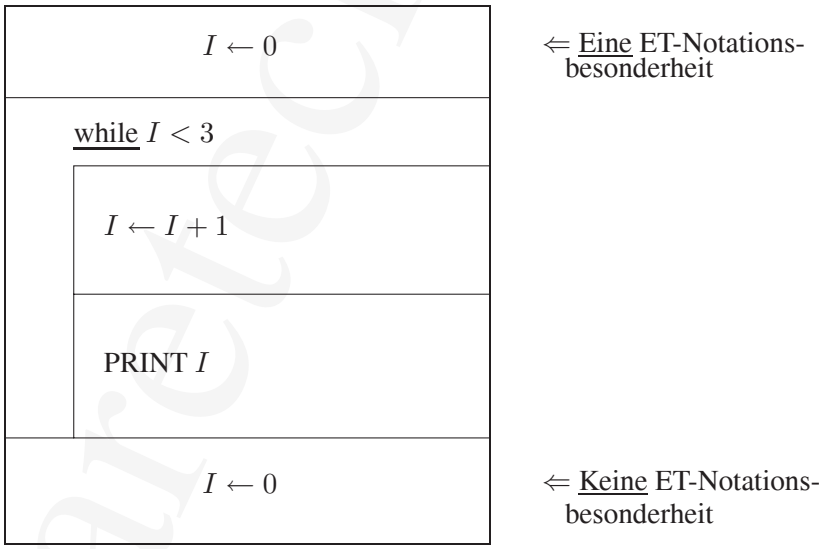
ET-Wortzählung$[< wort >< text >]$		R1	R2	R3
B1	Enthält $< text >$ kein Wort?	J	N	N
B2	Erste Wort im $< text >=< wort >$?	-	J	N
A1	Wortzählung $\leftarrow 0$	X		
A2	Wortzählung $\leftarrow$ 1+ call ET-Wortzählung$[< wort >$ $< text\ reduziert\ um\ erstes\ Wort >]$		X	
A3	call ET-Wortzählung $[< wort >$ $< text\ reduziert\ um\ erstes\ Wort >]$			X
A4	return Wortzählung	X	X	X

Legende:
call $\equiv$ run; verweist hier auf die Funktionsapplikation
vgl. iterative Lösung in Tabelle 4.14 Seite 96

Tabelle 4.15: Beispiel: Rekursion

ET-VORLAUF		R1
A1	$I \leftarrow 0$	X
A2	run ET-HAUPTTEIL	X

Tabelle 4.16: ET ohne Bedingung



Legende:
Vgl. Tabelle 4.17 Seite 100

Abbildung 4.4: ET & Sequenz

ET-VORLAUF		R1	R2
B1	Erster Durchlauf?	J	
A1	$I \leftarrow 0$	X	
A2	run ET-HAUPTTEIL	X	
A3	end	X	X

ET-HAUPTTEIL		R1	R2
B1	$I < 3$	J	N
A1	$I \leftarrow I + 1$	X	
A2	PRINT I	X	
A3	run ET-HAUPTTEIL	X	
A4	$I \leftarrow 0$		X
A5	return		X

Legende:
Vgl. Abbildung 4.4 Seite 99.

Tabelle 4.17: Bedingung: „Erster Durchlauf?“

4.6 ET & PAP-Linearisierung

Die ET-Technik kann sogenannte lineare Kontrollstrukturen, die sich aus den Bausteinen Sequenz, Iteration und Alternative zusammensetzen, direkt abbilden. Nicht-lineare Kontrollstrukturen, wie sie mit einem PAP (Programmablaufplan) notierbar sind (vgl. Abbildung 4.5 Seite 102), müssen vorab in eine lineare Kontrollstruktur überführt werden. Für diese Linearisierung bieten sich zwei Verfahren an:

1. Duplizierung von Konstrukten (Funktionen und Prädikaten)
2. Einführung einer zusätzlichen Steuerungsgröße (Label Structure Program)

4.6.1 Duplizierung von Konstrukten

Duplikate

Bei einer Linearisierung mit Hilfe der Duplizierung von Konstrukten vefolgen wir den Ablauf und zeichnen Konstrukte die mehrmals durchlaufen werden mehrfach. Abbildung 4.6 Seite 103 zeigt den ersten Umkonstruktionsschritt der nicht-lineraren Beispielstruktur (vgl. Abbildung 4.5 Seite 102).

ET-START		R1	R2
B1	p	J	N
A1	A		X
A2	run ET-L		X
A3	B	X	
A4	run ET-R	X	

ET-L		R1	R2
B1	q	J	E L S E
B2	r	J	
A1	B	X	
A2	run ET-A	X	
A3	run ET-L	X	
A4	return		X

ET-R		R1	R2
B1	r	J	E L S E
B2	q	J	
A1	A	X	
A2	run ET-B	X	
A3	run ET-R	X	
A4	return		X

ET-A		R1	R2
B1	r	J	N
A1	A	X	
A2	return	X	X

ET-B		R1	R2
B1	q	J	N
A1	B	X	
A2	return	X	X

Legende:
Vgl. Abbildung 4.5 Seite 102.

Tabelle 4.18: PAP & Duplizierung ⇒ ET-Verbundsystem

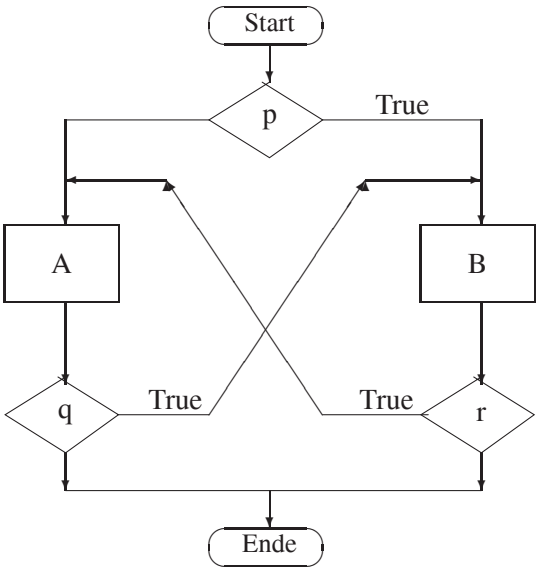


Abbildung 4.5: Beispiel: Nicht-lineare Konstrollstruktur

Aus dem until-Konstrukt:
do A . . . B until $\neg r$ od
wird bei $\neg q$ heraugesprungen. Dieses zweite Verlassen der Schleife verhindern wir, durch eine Zusammenfassung zu einer Abbruchbedingung. Diese Umkonstruktion („Umbiegen von Schleifenaussprüngen“) ist nicht in allen Fällen gleichwertig zur vorgegebenen Struktur.[10] Abbildung 4.6 Seite 103 zeigt diesen zweiten Umkonstruktionsschritt. Entstanden ist eine lineare Struktur, die direkt als geschlossenes ET-Verbundsystem darstellbar ist. Tabelle 4.18 Seite 101 zeigt dieses ET-Verbundsystem.

4.6.2 Label Structure Program

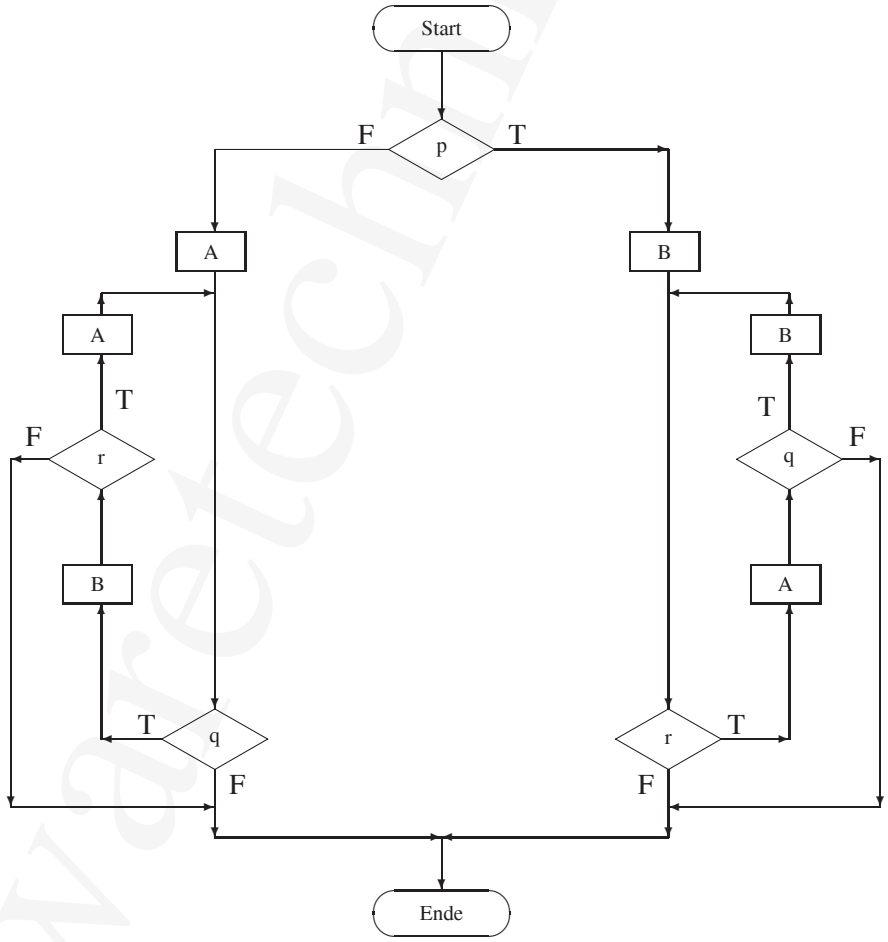
Knotennetz

Wir können einen nicht-linearen PAP als ein Netz von Knoten betrachten. Beim Erreichen eines Knotens steht fest, welcher Knoten danach folgt. Die Identifizierungsnummer des Nachfolgeknotens können wir in einer zusätzlichen Variablen L ($\equiv$ label) festhalten.

Anfangswert
Endewert

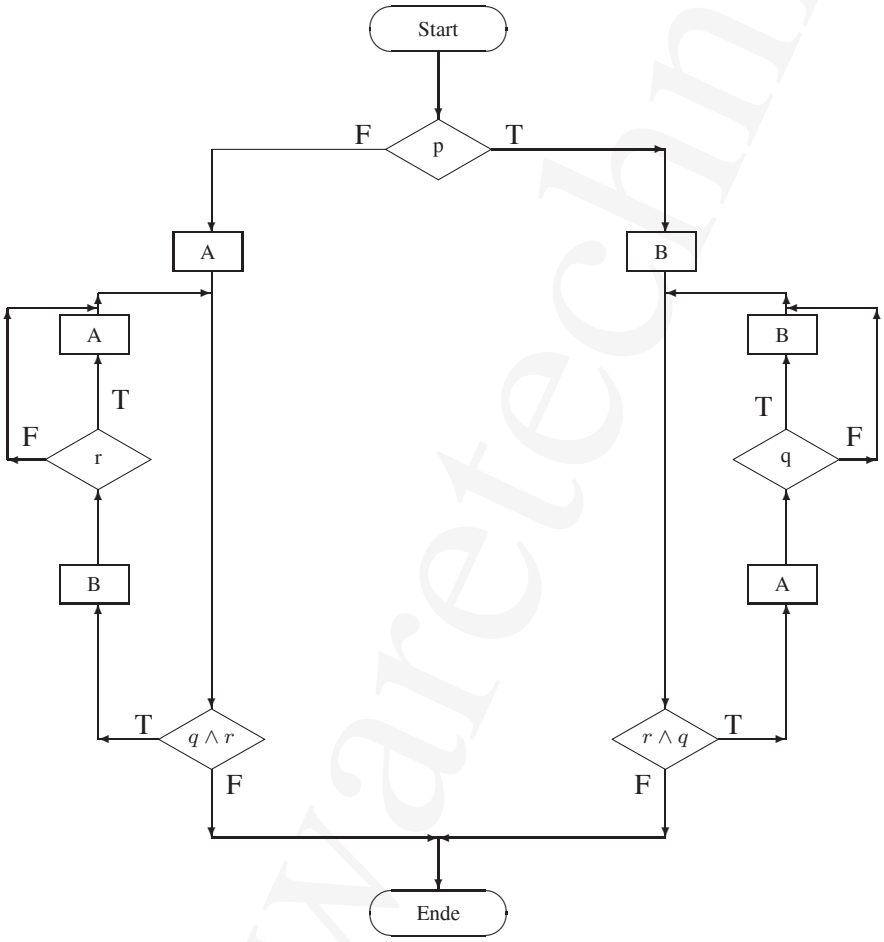
Wir bezeichnen den PAP-Start mit der Knotennummer 1 und weisen der Variablen L den Wert 1 zu. Das PAP-Ende bekommt die Knotennummer 0. Alle anderen Knoten bekommen eine beliebige, aber eindeutige Knotennummer (vgl. Abbildung 4.8 Seite 105). Mit einer solchen Knotenummerierung können wir das Durchlaufen des PAP als ein

[10] Zum Beispiel nicht bei dem Prädikat r, wenn dessen Prüfung mit Seiteneffekten verbunden wäre.



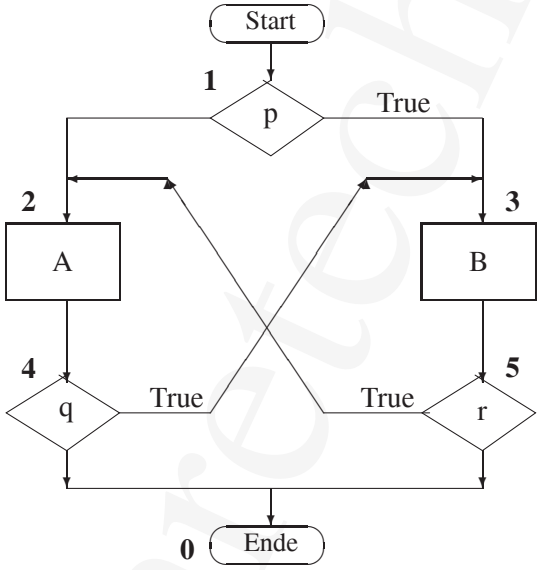
Legende:
Vgl. Abbildung 4.5 Seite 102.

Abbildung 4.6: Ansatz Duplizierung: 1. Schritt



Legende: Vgl. Abbildung 4.5 Seite 102 und Abbildung 4.6 Seite 103.

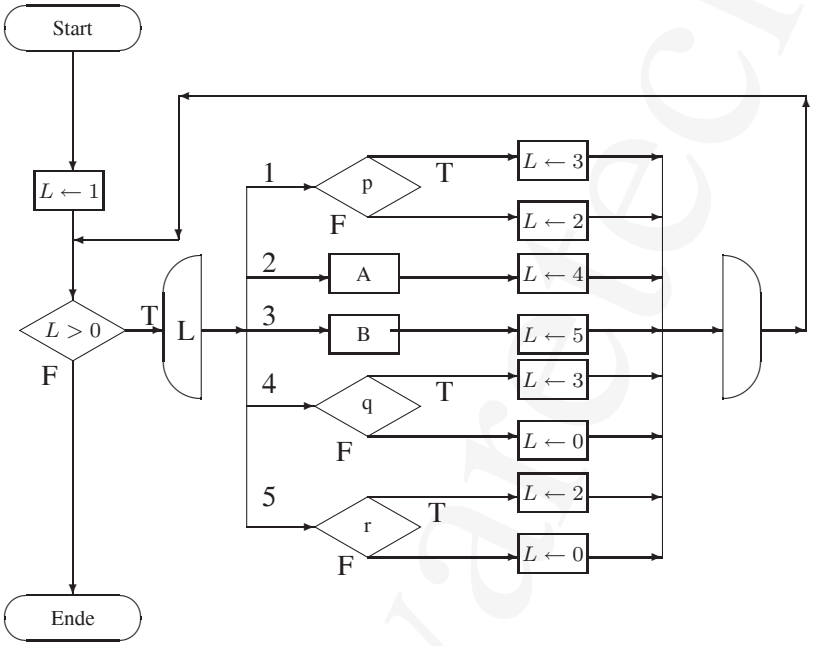
Abbildung 4.7: Ansatz Duplizierung: 2. Schritt



Legende:
Vgl. Abbildung 4.5 Seite 102.

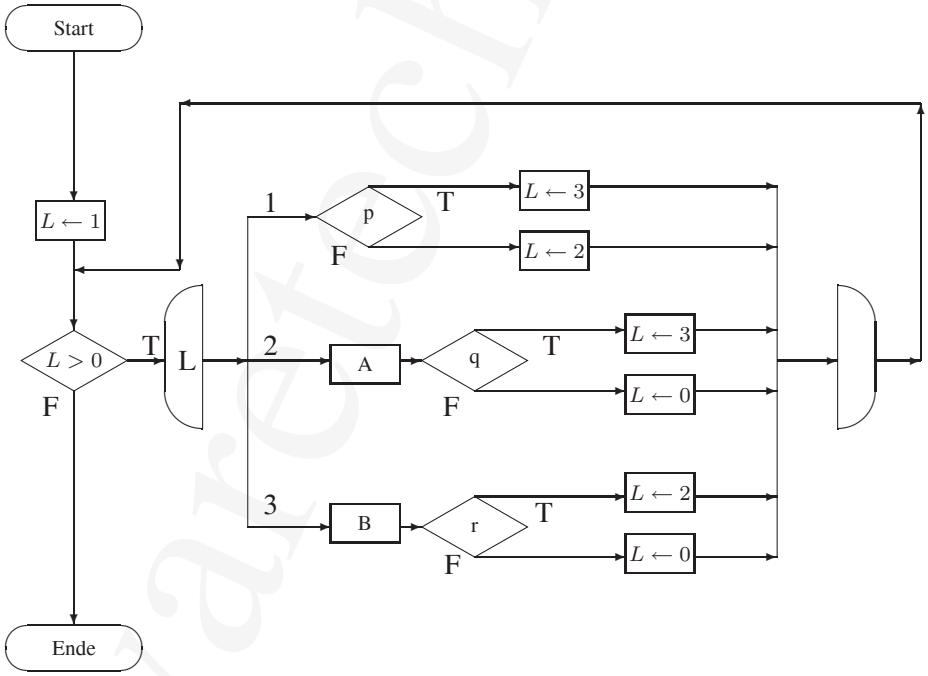
Startknoten	$\equiv 1$
Knoten	$\equiv$ beliebig nummeriert
Endknoten	$\equiv 0$

Abbildung 4.8: PAP $\Rightarrow$ Netz mit Knoten



Legende:
Vgl. Abbildung 4.8 Seite 105.

Abbildung 4.9: Knotennetzdurchlauf mit while-Konstrukt



Legende:
Reduzierbar sind die Knoten: 4 und 5
Vgl. Abbildung 4.9 Seite 106.

Abbildung 4.10: Reduziertes *Structure label program*

ET-VORLAUF		R1
A1	$L \leftarrow 1$	X
A2	*run* ET-PAP	X

ET-PAP		R1	R2	R3	R4
B1	$L = 1$	J	N	N	N
B2	$L = 2$	-	J	N	N
B3	$L = 3$	-	-	J	N
A1	run ET-1	X			
A2	A		X		
A3	run ET-2'		X		
A4	B			X	
A5	run ET-3'			X	
A6	Wiederhole ET-PAP	X	X	X	
A7	Ende				X

ET-1		R1	R2
B1	p	J	N
A1	$L \leftarrow 2$		X
A2	$L \leftarrow 3$	X	
A3	return	X	X

ET-2'		R1	R2
B1	q	J	N
A1	$L \leftarrow 0$		X
A2	$L \leftarrow 3$	X	
A3	return	X	X

ET-3'		R1	R2
B1	r	J	N
A1	$L \leftarrow 0$		X
A2	$L \leftarrow 2$	X	
A3	return	X	X

Legende:
Vgl. Abbildung 4.10 Seite 107.

Tabelle 4.19: PAP & Label $\Rightarrow$ ET-Verbundsystem

proc PAP
$L \leftarrow 1$
while $L > 0$
do
if $L = 1$ then $Knoten_1, L \leftarrow Knoten_1^{Nachfolger}$ fi
if $L = 2$ then $Knoten_2, L \leftarrow Knoten_2^{Nachfolger}$ fi
if $L = 3$ then $Knoten_3, L \leftarrow Knoten_3^{Nachfolger}$ fi
$\vdots$
od
corp PAP

Tabelle 4.20: Schablone: *Label structure program*

while-Konstrukt abbilden. Dieser Durchlauf ist in PDL einfach notierbar, wie Tabelle 4.20 Seite 109 zeigt.

Man spricht bei dieser Art von Linearisierung – unter Beibehaltung des englischen Ausdruckes – häufig von einem label structure program (vgl. zum Beispiel [42]). Im Gegensatz zur Umkonstruktion mit Hilfe von Duplikaten ist dieses Verfahren stets und ohne aufwendige Analyse der Struktur anwendbar.

Bezogen auf den nicht-linearen Beispiel-PAP erhalten wir ein while-Konstrukt mit 5 Konstrukten:
if ... then ... fi
Diese Konstrukte können wir zu einem case-Konstrukt zusammenfassen. Abbildung 4.9 Seite 106 zeigt dieses label structure program.

weniger Zweige

Wird innerhalb des while-Konstruktes der Label L auf einen bestimmten Wert nur einmal gesetzt, dann können wir an die Stelle dieser Zuweisung gleich den entsprechenden case-Zweig notieren. Auf diese Art reduzieren wir die Zweige im case-Konstrukt. Achtung: Diese Reduktion können wir nicht mit dem Startknoten-Wert ($L = 1$) vornehmen, weil wir sonst nicht in das Netz einsteigen. In unserem Beispiel können wir die Zweige 4 und 5 reduzieren. Abbildung 4.10 Seite 107 zeigt das reduzierte label structure program. Dieser jetzt lineare PAP ist direkt als ein geschlossenes ET-Verbundsystem abbildbar. Tabelle 4.19 Seite 108 zeigt dieses ET-Verbundsystem.

Für die manuelle Durchführung (mit Bleistift und Papier) ist die

Lineares zusammenfassen!

Anzahl der beherrschbaren Knoten relativ gering. Es empfiehlt sich daher eine komplexe Struktur zunächst auf die linearen Anteile hin zu analysiseren. Ein linearer Teilbereich wird zu einem Block zusammengefaßt. Dieser Block bekommt dann eine Knotennummer. Nach der Aufstellung des reduzierten label structure program können die Zusammenfassung wieder aufgelöst und an die jeweilige Stelle eingefügt werden.

Für die Nummerierung solcher Zusammenfassung schlagen Richard C. Linger, Harlan D. Mills und Bernard I. Witt folgende Notation vor ([42] S. 128):

Identifizierung / zusammengfaßte Knotenanzahl

Idendifizierung	1. Stelle	$\equiv$ Zusammenfassungsschritt
	2.-3. Stelle	$\equiv$ Laufende Nummer
Zusammengefaßte Knotenanzahl	$\equiv$ Zahl der Knoten im linearen Teilbereich	

Unterstellen wir eine Sequenz von zwei if ... then ... else ... fi Konstrukten, dann ergibt sich nach diesem Vorschlag folgende Nummerierung:
1. Zusammenfassungsschritt: 100/3 und 101/3
2. Zusammenfassungsschritt: 200/6

4.7 Fazit

Die ET sind ein nützliches Darstellungsmittel zum Präzisieren von $WENN$ ... $DANN$ ... Beziehungen. Ihre Syntax hilft beim Analysieren und Notieren aller formal-logischen (möglichen) Fälle.

Tabelle 4.21 Seite 111 faßt die ET-Typen nochmals zusammen.

Das Einbeziehen der Iteration und des ET-Verbundes lassen dieses Darstellungsmittel zu einer vollständigen formalen (Programmier-)-Sprache werden.

Die folgende Gegenüberstellung einer Beschreibung weniger Anforderungen zeigt die Vorteile der Darstellung mittels ET. Dabei wurde eine Mehrtreffer-ET notiert (vgl. Tabelle 4.22 Seite 112).

Beispiel: Plausibilitätsprüfung

1. Verbaler Text: Das Programm soll drei Datenelemente auf Plausibilität überprüfen. Ist das erste Datenelement falsch, wird eine Fehlernachricht ausgegeben. Ist das zweite Datenelement nicht

<table>
<tr><th colspan="4">ET-Typen

– Eintreffer-ET oder Mehrtreffer-ET –</th></tr>
<tr><th colspan="2">begrenzte ET</th><th>erweiterte ET</th><th>gemischte ET</th></tr>
<tr><td colspan="2">linker Tabellenteil beschreibt Bedingungen und/oder Aktionen vollständig</td><td rowspan="2">Bedigungen und Aktionen sind erst zusammen mit dem rechten Tabellenteil vollständig definiert</td><td rowspan="2">zusammengesetzt aus begrenzter und erweiterter Art</td></tr>
<tr><td>einfache ET

Bedingungs-anzeiger:
J oder N
<u>Standard-ET</u></td><td>komplexe ET

Bedingungs-anzeiger:
J, N, $-$ oder $\sharp$
<u>Sonderfall</u>
ELSE-Regel</td></tr>
</table>

Tabelle 4.21: Zusammenfassung: ET-Typen

richtig wird ebenfalls eine Fehlermeldung ausgegeben. Ist das dritte Datenelement falsch, soll das Programm den Wert $Unbekannt$! annehmen.

2. Verbaler Text mit Bezeichnern:

   $A1$ Das Programm $PLAUSIBEL?$ überprüft die Datenelemente $ELEMENT_1$, $ELEMENT_2$ und $ELEMENT_3$.

   $A1.1$ Ist $ELEMENT_1$ falsch, gibt $PLAUSIBEL?$ die Fehlernachricht F_1 aus.

   $A1.2$ Ist $ELEMENT_2$ falsch, gibt $PLAUSIBEL?$ die Fehlernachricht F_2 aus.

   $A1.3$ Ist $ELEMENT_3$ falsch, nimmt $PLAUSIBEL?$ den Wert $W_1 = Unbekannt!$ an.

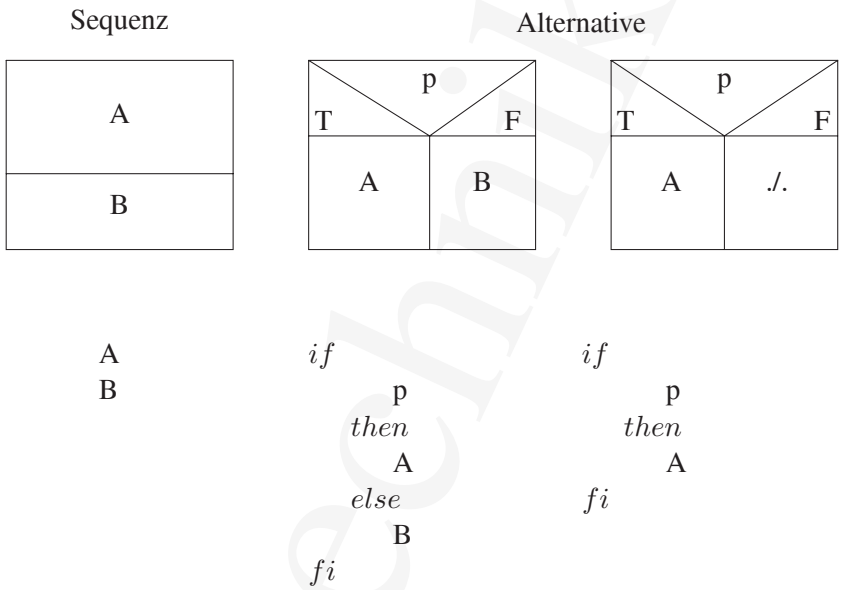
3. Mehrtreffer-ET (vgl. Tabelle 4.22 auf Seite 112)

ET-PLAUSIBEL?		R1	R2	R3	R4
B1	$ELEMENT_1$ falsch?	J	-	-	N
B2	$ELEMENT_2$ falsch?	-	J	-	N
B3	$ELEMENT_3$ falsch?	-	-	J	N
A1	Fehlernachricht F_1	X			
A2	Fehlernachricht F_2		X		
A3	PLAUSIBEL? nimmt Wert $W_1 = Unbekannt!$ an			X	
A4	Fall nicht beschrieben				X

Legende:
Mehrtreffer-ET (Abarbeitungsfolge R1, R2, R3)

Tabelle 4.22: Beispiel: ET-PLAUSIBEL?

Bei den Lösungen 2 und 3 ist jede einzelne Anforderung direkt angebbar. Jedes nachfolgende Dokument, zum Beispiel der Quellcodetext, kann sich auf diese Identifizierung stützen. Bei einer Modifikation sind die Änderungsbereiche problemlos adressierbar, zum Beispiel:
A1.3 Wert $W_1 = UNKLAR$ (Lösung 2)
ET-PLAUSIBEL? A3 Wert $W_1 = UNKLAR$ (Lösung 3).



Legende:

Abbildung 4.11: Erläuterung der PDL-Konstrukte

4.8 Übungen

4.8.1 Aufgabe: ET-Verbundsystem erstellen

Stelle Sie ein geschlossenes ET-Verbundsystem auf, daß das Verfahren zur Konsolidierung einer begrenzten Eintreffer-ET abbildet. Die zu prüfenden ET ($\equiv$ Prüfling P) sei redundanz- und widerspruchsfrei. Die Tabelle 4.23 zeigt eine Lösung. Sie steht auf Seite 114.

4.8.2 Aufgabe: PDL $\Rightarrow$ ET

Abbildung 4.12 Seite 115 zeigt die Kontrollstruktur eines Programms notiert in Process Design Language (PDL) oder auch Pseudocode genannt. Diese Formulierung nutzt die beiden Konstrukte: Sequenz und Alternative. Abbildung 4.11 Seite 113 zeigt ihre Definition in Form von Struktogrammen.

Beginn und Ende eines Programms sind mit den Kennworten proc, Kurzform für Procedure und corp, umgekehrte Kurzform für Procedure, markiert. Das Kennwort run bezeichnet den Aufruf eines Programms. **Pseudo-code**

Lösen Sie folgende Aufgaben:

ET-VORLAUF		R1
A1	$RZ1 \leftarrow 1$	X
A2	$RZ2 \leftarrow 1$	X
A3	*run* ET-MAIN	X

ET-MAIN		R1	R2
B1	$RZ1 = RZ2$	N	EL-
B2	$AktAF(RZ1) = AktAF(RZ2)$	J	SE
A1	run ET-ITERATIONSSCHRITT		X
A2	run ET-KONSOLIDIEREN	X	

ET-ITERATIONSSCHRITT		R1	R2	R3
B1	$RZ2 < R_{Anzahl}$	J	N	N
B2	$RZ1 < R_{Anzahl}$	-	J	N
A1	$RZ1 \leftarrow RZ1 + 1$		X	
A2	$RZ2 \leftarrow RZ1$		X	
A3	$RZ2 \leftarrow RZ2 + 1$	X		
A4	run ET-MAIN	X	X	
A5	return			X

ET-KONSOLIDIEREN		R1	R2
B1	$BedAF(RZ1)$ gleich bis auf eine B_i $BedAF(RZ2)$	J	N
A1	Streichen von Regel($RZ2$)	X	
A2	Ersetzen von B_i in Regel (RZ1) durch „-“	X	
A3	$RZ1 \leftarrow 1$	X	
A4	$RZ2 \leftarrow 1$	X	
A5	run ET-ITERATIONSSCHRITT	X	X
A5	return	X	X

Legende:

$RZ1, RZ2$ ≡ Zeiger auf eine Regel des Prüflings
$AktAF$ ≡ Aktionsanzeigerfolge
$BedAF$ ≡ Bedingungsanzeigerfolge
R_{Anzahl} ≡ Anzahl der Regeln des Prüflings

Tabelle 4.23: Konsolidieren als ET-Verbundsystem

```
proc MAIN
    if
            α = 1
        then
            if
                    β = 1
                then
                    if
                            γ = 1
                        then
                            A
                    fi
                    B
                else
                    C
            fi
        else
            run PART
            D
    fi
    E
corp

proc PART
    if
            (δ = 1) ∧ (ϵ = 1)
        then
            F
        else
            if
                    (ϵ ≠ 1) ∨ (ϑ ≠ 1)
                then
                    G
                else
                    H
            fi
            K
    fi
corp
```

Abbildung 4.12: Beispiel: Kontrollstruktur in PDL-Notation

ET-MAIN		R1	R2	R3	R4	R5	R6	R7	R8
B1	$\alpha = 1$?	J	J	J	J	N	N	N	N
B2	$\beta = 1$?	J	J	N	N	J	J	N	N
B3	$\gamma = 1$?	J	N	J	N	J	N	J	N
A1	A	X							
A2	B	X	X						
A3	C			X	X				
A4	run ET-MODUL-Y					X	X	X	X
A5	D					X	X	X	X
A6	E	X	X	X	X	X	X	X	X

ET-PART		R1	R2	R3	R4	R5	R6	R7	R8
B1	$\delta = 1$?	J	J	J	J	N	N	N	N
B2	$\epsilon = 1$?	J	J	N	N	J	J	N	N
B3	$\vartheta = 1$?	J	N	J	N	J	N	J	N
A1	F	X	X						
A2	G			X	X		X	X	X
A3	H					X			
A4	K			X	X	X	X	X	X
A5	return	X	X	X	X	X	X	X	X

Tabelle 4.24: PDL $\Rightarrow$ ET (Lösung zu 1)

1. Transformieren Sie die Kontrollstruktur von Abbildung 4.12 Seite 115 in ein geschlossenes ET-Verbundsystem mit Eintreffer-ET.

2. Können Ihre ET konsolidiert werden? Wenn ja, führen Sie die Konsolidierung durch.

3. Prüfen Sie, ob Ihre ET syntaktisch vollständig sind.

PDL $\Rightarrow$ ET: Lösung zur Frage 3)
ET-MAIN und ET-PART sind begrenzte Eintreffer-ET. Sie sind syntaktisch vollständig, da sie jeweils 8 Regeln bei 3 Bedingungen aufweisen.

4.8.3 Aufgabe: Abhängige Bedingungen

Die Vereinigung von Managern „LOGE-Ψ“ hat folgende Regelung in der Satzung festgeschrieben:

Konsolidierbar sind: R3/R4 und R5/R6/R7/R8

ET-MAIN		R1	R2	R3	R4
B1	$\alpha = 1$?	J	J	J	N
B2	$\beta = 1$?	J	J	N	-
B3	$\gamma = 1$?	J	N	-	-
A1	A	X			
A2	B	X	X		
A3	C			X	
A4	run ET-PART				X
A5	D				X
A6	E	X	X	X	X

Konsolidierbar sind: R1/R2 und R3/R4/R7/R8

ET-PART		R1	R2	R3	R4
B1	$\delta = 1$?	J	N	N	-
B2	$\epsilon = 1$?	J	J	J	N
B3	$\vartheta = 1$?	-	J	N	-
A1	F	X			
A2	G			X	X
A3	H		X		
A4	K		X	X	X
A5	return	X	X	X	X

Tabelle 4.25: PDL $\Rightarrow$ ET (Lösung zu 2)

Ist das zu versteuernde Einkommen eines Mitglieds kleiner DM 150.000,-, dann beträgt der Mitgliedsbeitrag DM 500,- . Ist es kleiner als DM 300.000,-, dann beträgt der Beitrag DM 1000,-. Ist es größer als DM 300.000,-, dann erwartet die Vereinigung eine angemessene Zahlung.

Stellen Sie eine Entscheidungstabelle (Typ: begrenzte, komplexe Eintreffer-ET) auf und erläutern Sie das Darstellungsproblem. Tabelle 4.26 zeigt Lösungen. Sie stehen auf der Seite 118.

Lösung A: Zeichen für Irrelevanz „versteckt“ die semantisch nicht sinnvollen (unmöglichen) Bedingungsanzeigerfolgen.

ET-MITGLIEDSBEITRAG		R1	R2	R3	R4
B1	Einkommen $<$ DM 150.000,- ?	J	N	N	N
B2	Einkommen $<$ DM 300.000,- ?	-	J	N	N
B3	Einkommen $=$ DM 300.000,- ?	-	-	J	N
A1	DM 500,-	X			
A2	DM 1000,-		X		
A3	Angemessene Zahlung				X
A4	Keine Regelung			X	

Lösung B: Besondere Warnaktion A5 für die semantisch nicht sinnvollen (unmöglichen) Bedingungsanzeigerfolgen.

ET-MITGLIEDSBEITRAG		R1	R2	R3	R4	R5	R6
B1	Einkommen $<$ DM 150.000,- ?	J	J	N	N	N	N
B2	Einkommen $<$ DM 300.000,- ?	-	-	J	J	N	N
B3	Einkommen $=$ DM 300.000,- ?	J	N	J	N	J	N
A1	DM 500,-		X				
A2	DM 1000,-				X		
A3	Angemessene Zahlung						X
A4	Keine Regelung					X	
A5	Warnung (semantisch unsinnig)	X		X			

Tabelle 4.26: ET: Lösung „Mitgliedsbeitrag“

4.8.4 Aufgabe: Präzisieren durch ET

Ein kommunales Theater plant eine Werbeaktion. Es ist daher beabsichtigt, die Eintrittspreise durch Rabattgewährung attraktiver zu gestalten. Der Leiter formuliert als ersten Entwurf folgenden Werbetext:

> Es wird ein Rabatt von 5% gewährt, wenn die Karte 30 Werktage vor der Veranstaltung gekauft wird. Für eine Gruppe von 7 oder mehr Personen, die so frühzeitig ihre Karten kauft, wird sogar 10% Rabatt gewährt. Eine Gruppe erhält auch dann einen Rabatt, wenn sie später kauft. Allerdings müssen die Karten dann 3 Werktage vorher gekauft werden. In diesem Fall beträgt der Rabatt jedoch nur 8%. Da Jugendliche, das heißt Personen jünger als 14 Jahre, schon heute nur 50% des normalen Preises zahlen müssen, gilt die Rabattregelung dieser Werbeaktion für sie nicht.

Stellen Sie ein geschlossenes ET-Verbundsystem (Typ: begrenzte, komplexe Eintreffer-ET) auf, das diese Regelung abbildet. Tabelle 4.27 zeigt die Lösung. Sie steht auf der Seite 120.

4.8.5 Aufgabe: ET-Analyse

Beantworten Sie folgenden Fragen:

1. Ist die ET-PROBE (vgl. Seite 121) konsolidierbar? Die Tabelle 4.29 zeigt die Lösung. Sie steht auf Seite 121.

2. Enthält die ET-PROBE einen Widerspruch? Zeigt die ET-Probe einen syntaktisch vollständigen Regelsatz? Kann die ET-PROBE mit einer zusätzlichen Regel vervollständigt werden? Die Tabelle 4.30 zeigt die Lösung. Sie steht auf Seite 121.

4.8.6 Aufgabe: Iteration

Eine Lieferantendatei (`LIEF-DAT`) ist auszuwerten, so daß alle Datensätze sequentiell gelesen werden. Beim Ende von `LIEF-DAT` ist die Nachricht „Alles verarbeitet!“ auszugeben. Die Datensätze der `LIEF-DAT` haben ein Merkmal M. Datensätze mit $M = A$ sind nicht bedeutsam. Sie werden übersprungen. Bei $M = B$ ist die Nachricht „Dubios!“ auszugeben. Bedeutsam sind Datensätze mit $M = C$. Für sie gilt folgendes: Jeder Datensatz der `LIEF-DAT` enthält für jeden Monat ein Monatsumsatzfeld. Für jeden der 12 Monate ist zu prüfen, ob der Wert

ET-WERBEAKTION		R1	R2	R3
B1	Alter < 14 Jahre ?	J	N	N
B2	Mitglied einer Gruppe?	-	J	N
A1	50% Rabatt	X		
A2	run ET-GRUPPE		X	
A3	run ET-EINZEL			X

ET-GRUPPE		R1	R2	R3	R4
B1	Gruppe ≥ 7 Mitglieder ?	J	J	J	N
B2	Kaufdatum 30 Werktage vorher?	J	N	N	-
B3	Kaufdatum 3 Werktage vorher?	-	J	N	-
A1	10% Rabatt	X			
A2	8% Rabatt		X		
A3	Kein Rabatt			X	
A4	run ET-EINZEL				X
A5	return	X	X	X	X

ET-EINZEL		R1	R2
B1	Kaufdatum 30 Werktage vorher?	J	N
A1	5% Rabatt	X	
A2	Kein Rabatt		X
A3	return	X	X

Tabelle 4.27: ET-Verbundsystem: Lösung „Werbeaktion“

im Monatsumsatzfeld numerisch ist; wenn ja, dann ist der Monatsumsatz in das entsprechende Feld der Jahrestabelle (`JAHR-TAB`) zu addieren. Vor Beginn der Verarbeitung ist `JAHR-TAB` auf den Anfangswert 0 zu setzen.

Bilden Sie diesen Sachverhalt mit Hilfe der ET-Technik ab. Tabelle 4.31 zeigt eine Lösung. Sie steht auf Seite 123.

4.8.7 Aufgabe: Bedingung oder Aktion

Die Sportzeitschrift *condition* (Heft 10, 1992, Seite 49) veröffentlichte folgende Erkenntnis:

ET-PROBE		R1	R2	R3	R4
B1	$Q = FALSE$?	J	J	N	N
B2	$R = FALSE$?	J	-	J	N
B3	$S = FALSE$?	J	N	N	N
A1	Vollzug I	X	X		
A2	Vollzug II			X	X

Tabelle 4.28: Beispiel: ET-PROBE

ET-PROBE		R1	R2	R3'
B1	$Q = FALSE$?	J	J	N
B2	$R = FALSE$?	J	-	-
B3	$S = FALSE$?	J	N	N
A1	Vollzug I	X	X	
A2	Vollzug II			X

Legende:
R1 ist mit R2 nicht konsolidierbar, da diese Regeln sich in zwei Bedingungsanzeigern unterscheiden.

Tabelle 4.29: ET-PROBE (Lösung zu 1)

ET-PROBE		R1	R2	R3	R4
B1	$Q = FALSE$?	J	J	N	E
B2	$R = FALSE$?	J	-	-	L S
B3	$S = FALSE$?	J	N	N	E
A1	Vollzug I	X	X		
A2	Vollzug II			X	
A3	???				X

Legende:
Tabelle 4.28 Seite 121 enthält keinen formalen Widerspruch. Sie ist unvollständig. Es fehlen die drei Regeln mit den Bedingungsanzeigerfolgen:
„J N N“ , „N J J“ und „N N J“

Tabelle 4.30: ET-PROBE (Lösung zu 2)

1	Mann läuft zu viel ⟶	2
2	Frau ist allein, verärgert ⟶	3
3	Frau geht eigene Wege ⟶	4
4	Mann über das Verhalten der Frau verärgert ⟶	5
5	Mann läuft noch mehr ⟶	2

Bilden Sie diese mit Hilfe einer Entscheidungstabelle ab. Diskutieren Sie die Frage der Formulierung von Bedingungen und Aktionen. Tabelle 4.32 zeigt eine Lösung. Sie steht auf Seite 124.

ET-LIEF-DAT-Analyse		R1	R2	R3	R4	R5	R6
B1	Erster ET-Durchlauf ?	J	N	N	N	N	N
B2	`LIEF-DAT`-Ende ?	-	J	N	N	N	N
B3	$M = A$?	-	-	J	N	N	N
B4	$M = B$?	-	-	-	J	N	N
B5	$M = C$?	-	-	-	-	J	N
A1	Löschen `JAHR-TAB`	X					
A2	$I \leftarrow 1$	X					
A3	Ausgabe „Dubios!“				X		
A4	run ET-ADDITION					X	
A5	„Unbekannter M-Wert!“						X
A6	Read `LIEF-DAT`	X		X	X	X	X
A7	„Alles verarbeitet!“		X				
A8	Wiederhole ET-LIEF-DAT-Analyse	X		X	X	X	X

ET-ADDITION		R1	R2	R3
B1	$I \leq 12$?	J	J	N
B2	`LIEF-DAT-MONAT(I)` numerisch ?	J	N	-
A1	`JAHR-TAB(I)` $\leftarrow$ `JAHR-TAB(I)` + `LIEF-DAT-MONAT(I)`	X		
A2	$I \leftarrow I + 1$	X	X	
A3	„Nicht numerische Eintragung!“		X	
A4	Wiederhole ET-ADDITION	X	X	
A5	$I \leftarrow 1$			X
A6	return			X

Tabelle 4.31: Lösung: LIEF-DAT

ET-VORLAUF		R1
A1	Mann läuft zu viel	X
A2	ABBRUCH ← False	X
A3	*run* ET-BEZIEHUNGSKISTE	X

ET-BEZIEHUNGSKISTE		R1	R2
B1	ABBRUCH = True ?	J	N
A1	Frau ist allein, verärgert		X
A2	Frau geht eigene Wege		X
A3	Man über das Verhalten der Frau verärgert		X
A4	Mann läuft noch mehr		X
A5	run ET-BEZIEHUNGSKISTE		X
A6	Stop Running	X	

Legende:
Zyklusende ist unbekannt – daher keine Modifikation von ABBRUCH!

Tabelle 4.32: Lösung: Endlosschleife

Kapitel 5

Modellieren mit (Petri-)Netzen

Netze, entwickelt aus dem von Carl Adam Petri vorgeschlagenen Grundtyp, eignen sich zur Beschreibung dynamischer Systeme, die eine feste Ablaufstruktur besitzen, wie zum Beispiel die Abwicklung von Bürovorgängen. Graphische Netz-Darstellungen machen das Modellverhalten durchschaubar.

Mit Hilfe der mathematisch fundierten Netztheorie können Eigenschaften eines entworfenen Netzes nachgewiesen werden.

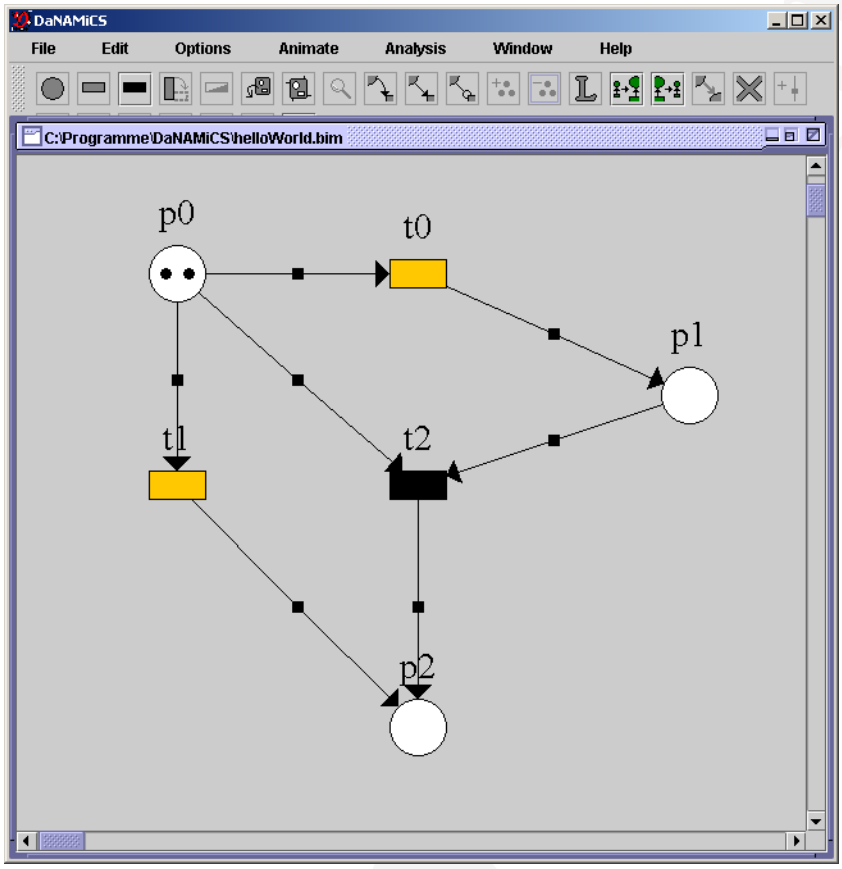
Wegweiser

Der Abschnitt *Modellieren mit (Petri-)Netzen* erläutert:

- die Grundbegriffe anhand von einführenden Beispielen,
  ↪Seite 127 . . .

- den Fall von knappen Ressourcen abgebildet mit Netzen vom Typ „Stelle/Transition‘,
  ↪Seite 152 . . .

- Netze mit unterscheidbaren Marken,
  ↪Seite 164 . . .



Legende:
Darstellung einer Kostprobe mit dem Petri-Netz-Werkzeug `DaNAMiCS` ↪Abschnitt C.2 S. 246

Abbildung 5.1: Kleine Kostprobe dargestellt in `DaNAMiCS`

- das markengerechte Verfeinern und Kombinieren von Netzen im Rahmen des Modellierens,
  ↪Seite 172 . . .

- das Nachweisen von Netzeigenschaften anhand eines umfangreicheren Beispiels,
  ↪Seite 190 . . .

- in Form einer Bilanzierung die Vor- und Nachteile einer Modellierung mit Petri-Netzen und
  ↪Seite 203 . . .

- anhand von Übungen Anwendungsmöglichkeiten.
  ↪Seite 204 . . .

5.1 Grundbegriffe und einführende Beispiele

Netze vom Grundtyp bestehen aus zwei Knotensorten: Stellen s („Datenablagen“ , dargestellt als Kreise) und Transitionen t (beschreiben „Datenverarbeitung“, dargestellt als Rechtecke). Die Kanten (Verbindungen zwischen zwei Knoten) führen stets von einer zur anderen Sorte. Stellen, deren Kanten von s nach t laufen, heißen Eingabestellen von t. Andere Stellen, zu denen von t aus Kanten führen, heißen Ausgabestellen von t. Ein so gerichteter Graph repräsentiert die feste Struktur des Modells. Um Ablaufvorgänge zu modellieren, werden Stellen mit Marken belegt, die beim Schalten von Transitionen weitergegeben werden. Eine einfache Schaltregel bestimmt: t kann schalten (oder „feuern“), wenn jede Eingabestelle von t mindestens eine Marke enthält. Schaltet t, dann wird aus jeder Eingabestelle von t eine Marke entfernt und jeder Ausgabestelle eine Marke hinzugefügt. Zahlreiche Modifikationen des Petri-Netz-Grundtyps (zum Beispiel Kapazität einer Stelle oder Kanten mit Gewichtangabe) erweitern die Anwendungsmöglichkeiten.

Anhand von Beispielen[1] sind Netztypen, ihre Handhabung und der Nachweis von Netzeigenschaften skizziert. Es handelt sich um eine praxisbezogene Einführung, die einen ersten Überblick über Einsatzmöglichkeiten vermitteln will.

[1] Für die Durchsicht einer vorhergehenden Version danke ich Herrn Goetz (FH Nordostniedersachsen) und Herrn J. Knowles (DAK, Hamburg). Die Beispiele „Versionsmanagement“ und „Projektplan“ hat Herr Knowles vorgeschlagen.

Im folgenden ist für uns das Modellieren ein Abstraktionsprozeß, der einige wesentliche Aspekte des (realen oder fiktiven) Systems und dessen Verhalten in einem Modell sichtbar macht. Das Modellieren mit Hilfe von Petri-Netzen dient hier der Analyse des Nachrichtenflusses (Daten- und Kontrollflusses) in Informationssystemen.

nebenläufige Vorgänge

Schwerpunkt unserer Betrachtung sind sogenannte nebenläufige Vorgänge. Solche Vorgänge liegen zum Beispiel vor wenn im Arbeitsalltag zwei Sachbearbeiter miteinander einen Fall bearbeiten. Beide sprechen sich miteinander ab und erarbeiten dann ein Stück unabhängig voneinander ihr jeweiliges (Teil-)Ergebnis um dann erneut miteinander zu kommunizieren. Einzelne Arbeitsschritte laufen nebeneinander, andere erfordern ein Nacheinander.

Bei nebenläufigen Vorgängen treten vielfältige Analysefragen auf, wie z.B:

- Konflikte
- Konfusionen
- Verklemmungen
- Sicherheit
- ...

Zum Modellieren von Systemen mit nebenläufigen Vorgängen hat Carl Adam Petri in seiner Dissertation Netze mit zwei Sorten von Knoten vorgeschlagen [47]. Eine Knotensorte für die passiven Komponenten und eine für die aktiven Komponenten des Systems. Die Verbindungen zwischen den Knoten sind abstrakte, gedankliche Beziehungen zwischen Komponenten, zum Beispiel für logische Zusammenhänge, Zugriffsrechte, räumliche Nähe oder direkte Kopplung. Sein Netz (vom Grundtyp) besteht aus:

- Stellen[2] [hier mit s bezeichnet]
- Transitionen[3] [hier mit t bezeichnet]
- und gerichteten Kanten (Flußrelationen)

Damit ein System auf verschiedenen Abstraktionsebenen modellierbar ist, kann zum Beispiel eine gegebene Transition wiederum als ein Netz verfeinert werden (Näheres Abschnitt 5.4.1). Wir verwenden daher auf allen Abstraktionsebenen dieselbe graphische Notation. Aus einem groben Netz ist Schritt für Schritt ein verfeinertes Netz entwickelbar.

[2] auch Plätze (engl.: places) oder Kanäle genannt
[3] lateinisch: Übergang, auch Instanzen genannt

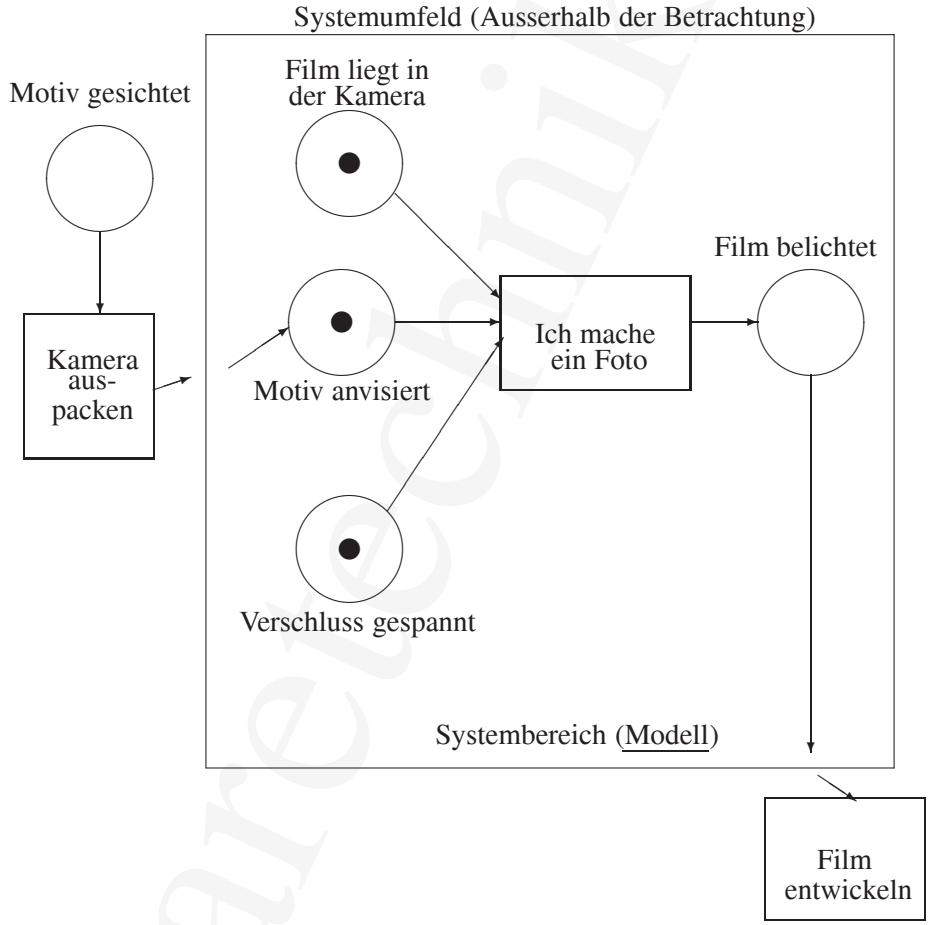


Abbildung 5.2: Beispiel: Fotografieren

5.1.1 Erste Kostproben

Wir zeichnen Transitionen als Rechtecke und bezeichnen sie mit t. Faßt man Transitionen im Sinne von Hürden auf, dann ist eine Abbildung als schwarzer Balken üblich. Stellen zeichnen wir als Kreise und bezeichnen sie mit s (vgl. Abbildung 5.5 auf Seite 133).

In einem Netz dürfen Kanten (Pfeile) nur jeweils von einer Knotensorte zur anderen Knotensorte führen. Stellen, deren Kanten von s nach t laufen, heißen Eingabestellen von t. Andere Stellen, zu denen von t aus Kanten führen, heißen Ausgabestellen von t. Ein so gerichteter Graph repräsentiert die feste Struktur des Modells.

Die Ablaufvorgänge modellieren wir mit sogenannten Marken. Dazu werden Stellen mit Marken belegt, die beim Schalten von Transitio-

nen weitergegeben werden. Eine einfache Schaltregel bestimmt: t kann schalten (oder „feuern“), wenn jede Eingabestelle von t mindestens eine Marke enthält. Schaltet t, dann wird aus jeder Eingabestelle von t eine Marke entfernt und jeder Ausgabestelle eine Marke hinzugefügt.

bool-- Petri-- Netz

Abbildung 5.3 ist die Stelle s_2 mit einer Marke versehen. Diese Marke können wir als ein Objekt vom Datentyp boolean (wahr, nicht wahr) ansehen. Sie zeigt an, daß die Aussage von s_2 ($\equiv$ Person ist arbeitsbereit) wahr ist. Nicht markierte Stellen haben Aussagen, die nicht wahr sind. Ein Netz mit solchen Marken heißt Bedingungs-Ereignis-System oder auch bool-Petri-Netz (vgl. zum Beispiel [21]). Die Transition t_2 hat eine erfüllte Eingangsstelle (Vorbedingung). Sie kann schalten, wenn ihr Ereignis Person geht zur Arbeit eintritt. Wir sprechen von einem aktivierten Ereignis. Nach dem Schalten ist die Stelle s_3 markiert, das heißt die Aussage Person am Arbeitsplatz ist wahr. Die Nachbedingung von t_2 ist erfüllt.

Bei einem Bedingungs-Ereignis-System[4] tragen Stellen nur eine oder keine Marke; mehrere Marken auf einer Stelle wären als mehrfach wahr zu interpretieren und daher hier unsinnig. Daraus folgt: Das Schalten einer Transition bedingt, daß die Nachbedingung noch keine Marke enthält.

Die einfache Schaltregel[5] lautet:

if Vorbedingungen erfüllt
$\wedge$
Nachbedingungen $\neg$erfüllt
then
Transition (Ereignis)
kann schalten (eintreten)
fi

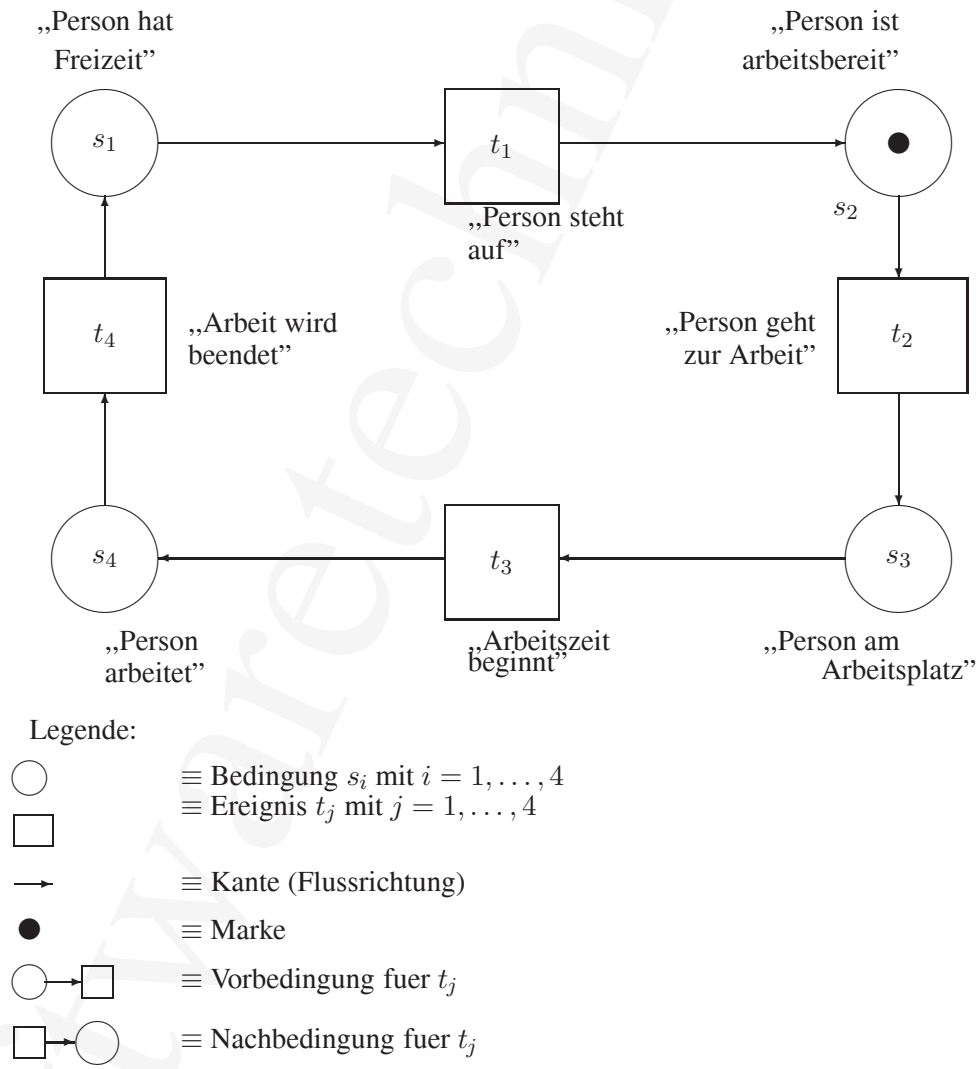
Bei Transitionen unterscheiden wir zwischen

1. der Schaltkonzession (Aktivierung) und

2. dem Schalten (engl.: firing), worunter das tatsächliche Eintreten des Übergangs verstanden wird.

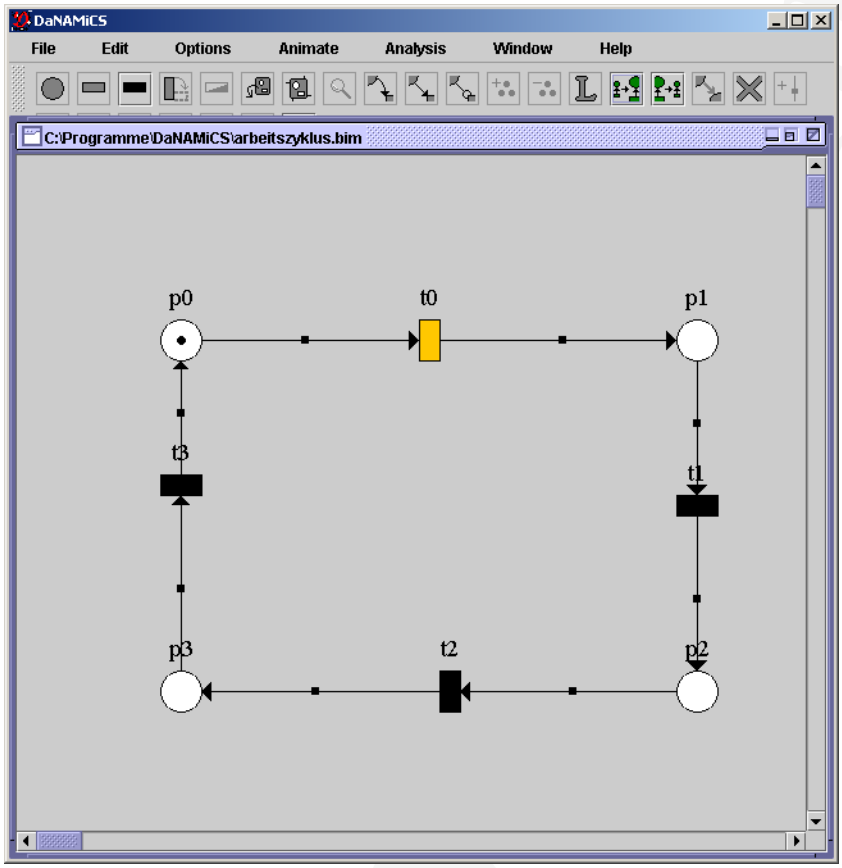
[4] Die Kapazität der Stelle (Näheres vgl. Abschnitt 5.2.1 Seite 152), das heißt die Menge der Marken, die sie aufnehmen kann, ist im gesamten Netz hier 1.

[5] Auch als „strong rule“ bezeichnet (vgl. zum Beispiel [60]). Die „soft rule“ läßt die Belegung einer Stelle mit mehreren Marken zu.



Dargestellt ist als Anfangszustand: Person ist arbeitsbereit

Abbildung 5.3: Beispiel: Arbeitszyklus



Legende:
Darstellung des Beispiels „Arbeitszyklus“ (↪Abbildung 5.3 S. 131) mit dem Petri-Netz-Werkzeug `DaNAMiCS` ↪Abschnitt C.2 S. 246

Abbildung 5.4: Beispiel: Arbeitszyklus dargestellt in `DaNAMiCS`

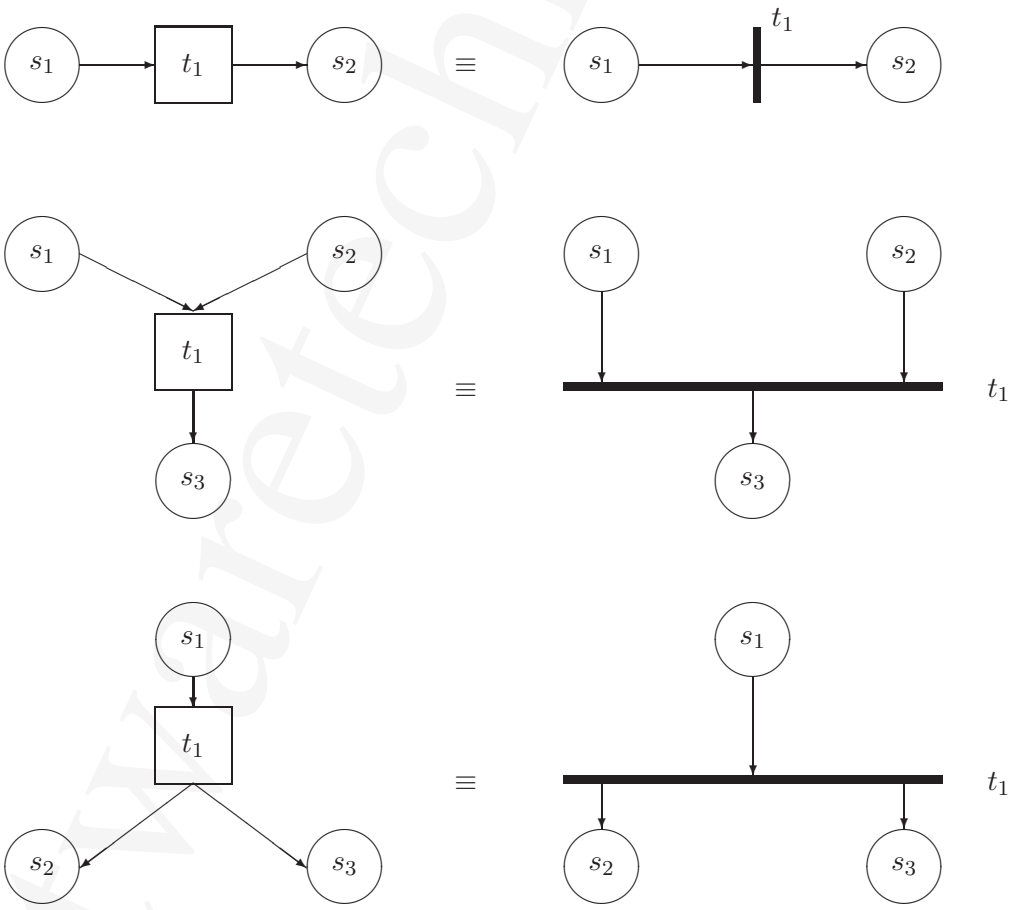


Abbildung 5.5: Zwei Darstellungsformen von Transitionen

5.1.2 Marken und elementare Konstruktionen

Typ nat

Marken[6] können beliebige Objekte symbolisieren. Sie stehen zum Beispiel für einen Beleg, der bearbeitet und weitergereicht wird. Zwei Marken auf einer Stelle bedeuten dann zwei Belege, die auf den Durchlauf durch eine Transition warten. Solche Marken entsprechen dem Datentyp nat (natürliche Zahlen). Netze mit ihnen bezeichnet man als nat-Petri-Netz, oder als Stelle/Transitions-Netz oder Stellen/Transitionen-Netz, kurz als S/T-Netz.

Solche Netze enthalten, wie auch die Bedingungs-Ereignis-Netze, nicht-unterscheidbare Marken. Alle Marken eines Netzes sind gleich. Netze mit unterscheidbaren Marken heißen höhere Petri-Netze und sind als Prädikat/Transitions-Netz [28] oder als gefärbte Petri-Netze [37] bekannt (Näheres Abschnitt 5.3). Das Markenwandern[7] wird durch Beschriftungen an Kanten und Transitionen gesteuert.

Ein Netz setzt sich aus Stellen, Transitionen und Flußrelationen (Kanten, Pfeile) zusammen. Dabei können wir elementare Konstruktionen[8] unterscheiden (vgl. Abbildung 5.6 Seite 135).

Durch Einführung von zusätzlichen Stellen und Transitionen als Hilfsstellen und Hilfstransitionen können komplexe Konstruktionen auf die obigen Primitives zurückgeführt werden. Abbildung 5.7 Seite 136 zeigt ein Beispiel (vgl. [21]). Probleme, die mit solcher Zerlegung verbunden sein können, behandeln wir im Abschnitt 5.4.1 Seite 174.

5.1.3 Nebenläufigkeit und Konflikt

Zwei Vorgänge heißen nebenläufig, wenn sie voneinander unabhängig bearbeitet werden können. Für zwei nebenläufige Vorgänge X und Y ist es gleichgültig, ob erst X und dann Y, oder ob erst Y und dann X, oder ob X und Y zur gleichen Zeit ausgeführt werden.

Für die Nebenläufigkeit[9] einer Menge von Transitionen ist zu fordern, daß genug Marken vorhanden sind, um alle Transitionen nebeneinander [10] zu schalten.

In Abbildung 5.8 liegt bei den Transitionen t_2 und t_3 Nebenläufigkeit vor. Ihr Schalten setzt das Schalten von t_1 voraus.

[6] auch „Token“ genannt

[7] auch Markenspiel genannt

[8] sogenannte Primitives

[9] engl.: concurrency

[10] Begriffe wie „gleichzeitig“ oder „parallel“ bedingen eine Zeitskala. Nebenläufigkeit sei eine Eigenschaft von Mengen von Systemereignissen (Übergängen) im Sinne ihrer gegenseitigen und kollektiven Unabhängigkeit.

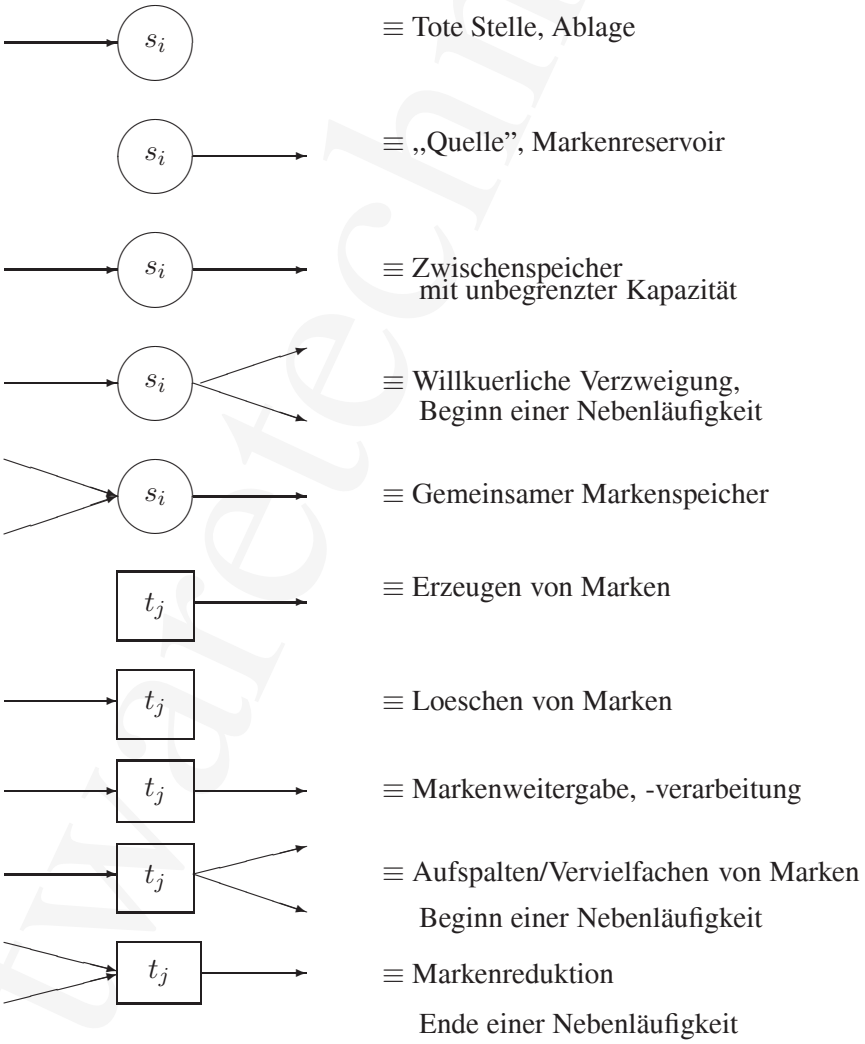
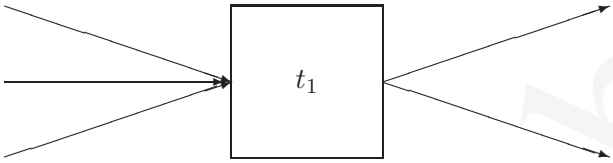
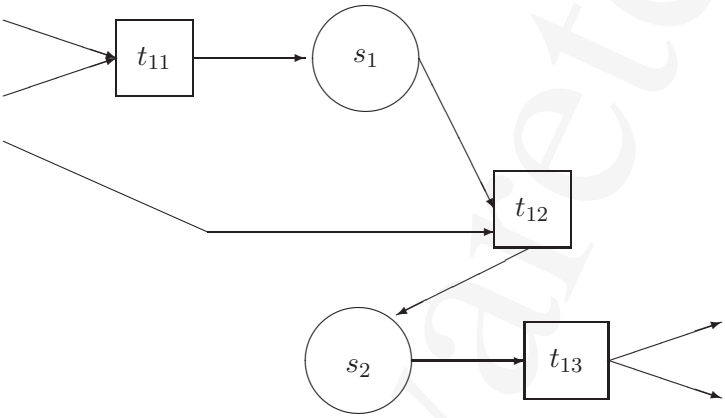


Abbildung 5.6: Elementare Konstrukte (Primitives)

Ausgangskonstruktion:

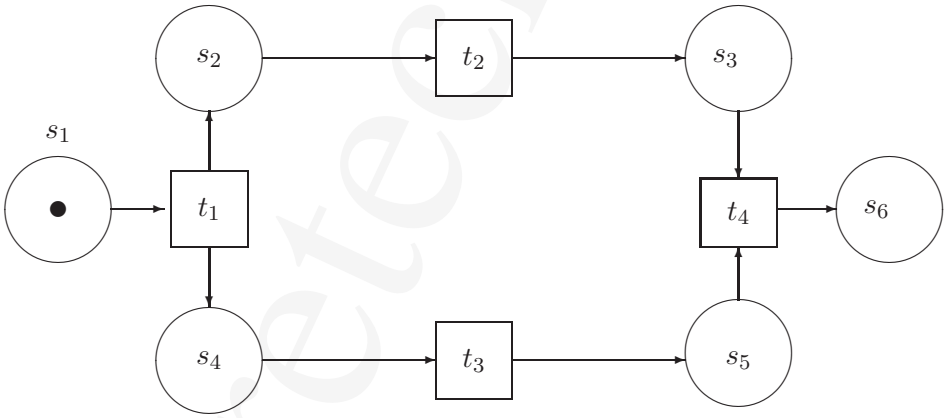


Ersatzkonstruktion:



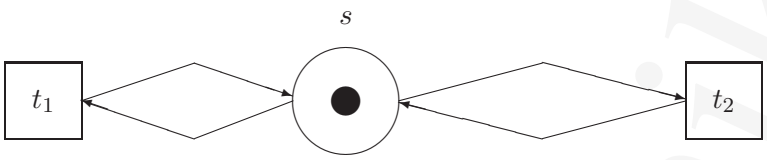
Legende:
Ersatzkonstruktion durch Einführung von neuen Stellen und Transitionen. [Hinweis: Zur markengerechten Verfeinerung vgl. Abschnitt 5.4.1 Seite 174.]

Abbildung 5.7: Beispiel: Zerlegung einer komplexen Konstruktion



Legende:
Nebenläufigkeit bei Transitionen t_2 und t_3

Abbildung 5.8: Beispiel: Nebenläufigkeit



Legende:
Transitionen t_1 und t_2 können in beliebiger Reihenfolge schalten aber nur nacheinander. Sie sind nicht nebenläufig.

Abbildung 5.9: Beispiel: Netz mit Schleifen

Im folgenden Netz (Abbildung 5.9) ist ein Konflikt abgebildet. Die Voraussetzungen zum Schalten der Transitionen t_1 und t_2 bestehen durch die Marke auf s. Beide Transitionen haben Konzession („Berechtigung“) zu schalten. Welche Transition als nächste schaltet, wird vom Netz nicht dokumentiert. Salopp formuliert: Ein zeitliches Verhalten im Sinne von Wer kommt als nächster dran? wird von einem S/T-Netz nicht dokumentiert.

Konzession

5.1.4 Erreichbarkeit

Betrachten wir den Fluß der Marken, dann interessieren uns vielfältige Fragen, zum Beispiel, ob ein Netz bei einer gegebenen Anfangsmarkierung eine vorgegebene Markierung, eine spezifizierte Markenverteilung auf den einzelnen Stellen, erreichen kann.

5.1.5 Lebendigkeit

Neben diesem Erreichbarkeitsproblem stellt sich die Frage, ob ein Netz terminiert. Es kann also nur eine endliche Zahl von Schaltvorgängen geben. Abbildung 5.8 zeigt ein Netz, das terminiert. Die Transitionen in Abbildung 5.9 können beliebig oft schalten, das Netz terminiert nicht.

Interessant ist die Frage, wie oft die Transitionen eines Netzes schalten können. Anders formuliert: Ist jede Transition lebendig? Unterstellen wir eine Startsituation (Anfangsmarkierung) und können wir die Transitionen so schalten, daß eine vorgegebene Transition t_j im weiteren Verlauf nochmals schalten kann, dann ist t_j lebendig; andernfalls ist t_j tot. In Abbildung 5.9 sind t_1 und t_2 lebendig.

Falls es Situationen gibt, bei denen keine Transition schalten kann, wobei allerdings diese Situation bei einer anderen Schaltreihenfolge hätte vermieden werden können, spricht man von Verklemmung. Ein Netz ist verklemmungsfrei („deadlock free “), wenn es keine erreichbare tote

Netzeigenschaften

Mathematische Notation: (ähnlich zum Beispiel [50] oder [30])

1. Das Tripel $\mathcal{N} \equiv (\mathcal{S}, \mathcal{T}, \mathcal{F})$ stellt die Struktur des Netzes $\mathcal{N}$ dar, die aus der Stellenmenge $\mathcal{S}$, der Transitionenmenge $\mathcal{T}$ und der Menge der gerichteten Kanten (Flußrelationen) $\mathcal{F}$ gebildet wird; wobei $\mathcal{F}$ eine Teilmenge von $(\mathcal{S} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{S})$ ist.
2. s_i und t_j bezeichnen einzelne Stellen bzw. Transitionen.
3. $\triangleright s_i$ ist der Vorbereich einer Stelle, $s_i \triangleleft$ ihr Nachbereich.
4. $\triangleright t_j$ ist der Vorbereich einer Transition, $t_j \triangleleft$ ihr Nachbereich.
5. $m_k(s_i)$ bezeichnet die Marken auf einer Stelle s_i
6. M_i ist die Bezeichnung einer Markierung, steht also für die Menge von $m_k(s_i)$.
7. M_0 ist die Anfangsmarkierung des Netzes.
8. $M_i \sqsubset t_j \Rightarrow M_{i+1}$ bedeutet, daß durch das Schalten der Transition t_j die Markierung M_i in die Markierung M_{i+1} überführt wird.
9. Eine Markierung M heißt erreichbar, falls es eine Folge $(M_i)_{i=1,n}$ von Markierungen und eine Folge $(t_j)_{j=1,n}$ von Transitionen gibt, so daß gilt:
   $M_0 \sqsubset t_1 \Rightarrow M_1 \sqsubset t_2 \Rightarrow M_2, \ldots, \sqsubset t_n \Rightarrow M_n = M$.
10. $\mathcal{M} \sqsubset\!\Rightarrow$ bezeichnet die Menge aller erreichbaren Markierungen von einer Markierung M aus.

Tabelle 5.1: Erreichbarkeit

Markierung gibt, das heißt, wenn immer wieder eine Transition schalten kann.

5.1.6 Beschränktheit

Da Transitionen Marken produzieren können, ist zu analysieren, ob auf einer Stelle eine vorgegebene Markenhöchstgrenze überschritten wird. Ein Netz heißt beschränkt, wenn es eine Zahl k gibt, so daß bei jeder erreichbaren Markierung höchsten k Marken auf jeder Stelle liegen. Abbildung 5.10 zeigt ein Netz bei dem die Stellen s_2 und s_4 unbeschränkt sind (vgl. [55]). Wir bezeichnen ein Netz als sicher, wenn es beschränkt ist.

5.1.7 Netzplantechnik ⇔ Petri-Netze

Für das Managen umfangreicher Projekte hat sich die Netzplantechnik[11] bewährt. Ausgehend von einem Startvorgang oder Startereignis (Projektbeginn) werden die zeitfordernden Geschehnisse, die selbst einen definierten Anfang und ein definiertes Ende aufweisen, bis zu einem Endvorgang oder Endereignis in Form eines Knotennetzes dargestellt. Dabei unterscheidet man verschiedene Varianten, zum Beispiel:

CPM Critical Path Method

MPM Metra Potential Method

PERT Program Evaluation and Review Technique

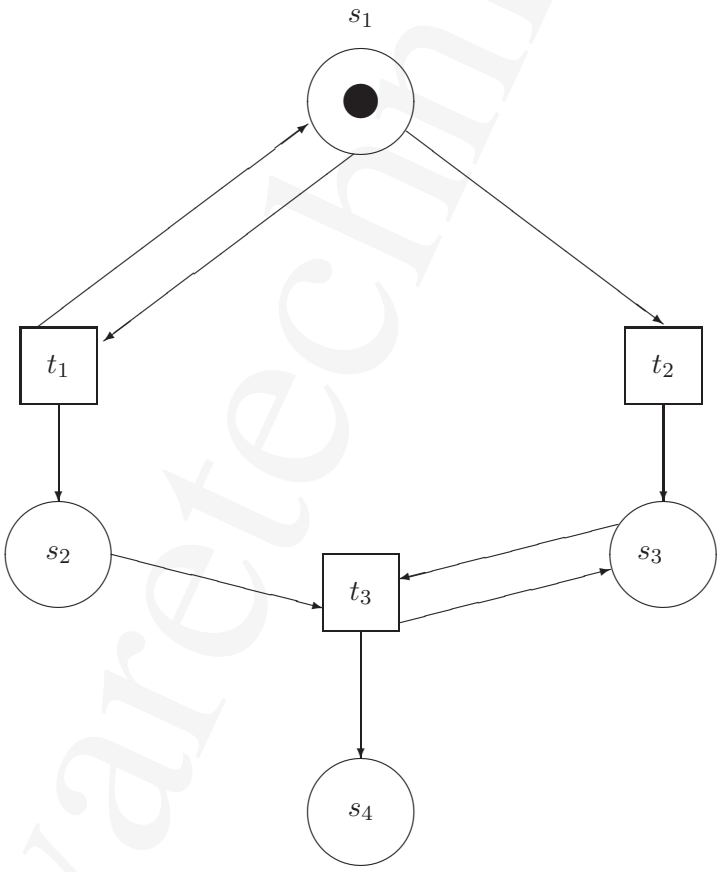
Im Vergleich zu Petri-Netzen arbeiten Netzpläne nur mit einem Knotentyp. Ein einfaches Beispiel eines Netzplanes mit Vorgangsknoten zeigt Abbildung 5.11 Seite 142. Dargestellt ist ein Projektphasenplan entsprechend den „Besonderen Vertragsbedingungen für das Erstellen von DV-Programmen“ in der öffentlichen Verwaltung [20]. Tabelle 5.2 Seite 143 beschreibt die einzelnen Vorgänge.

Führen wir in Abbildung 5.11 weitere Knoten mit der Bedeutung „Der Vorgang V_i darf beginnen!“ und „Der Vorgang V_i ist beendet!“ ein, dann können wir einen Netzplan in ein Petri-Netz umformen (vgl. Abbildung 5.12 Seite 144).

kritische Pfad

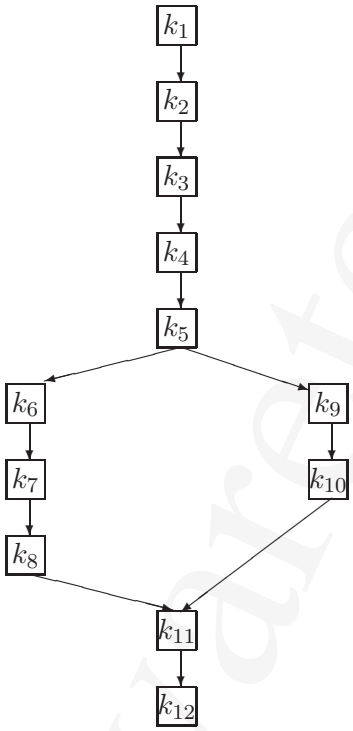
Netzpläne haben als wichtige Aussage die Ermittlung und Darstellung des kritischen Pfades, das heißt der Sequenz von zeitlichen Ge-

[11] Zu den Begriffen vgl. zum Beispiel DIN 69900 Teil 1 und zur Darstellungstechnik DIN 69900 Teil 2 [26].



Legende:
Die Zahl der Marken auf der Stelle s_2 — und damit auch auf s_4 — kann jede vorgegebene Schranke k überschreiten.

Abbildung 5.10: Beispiel: Unbeschränkte Markenanzahl im Netz



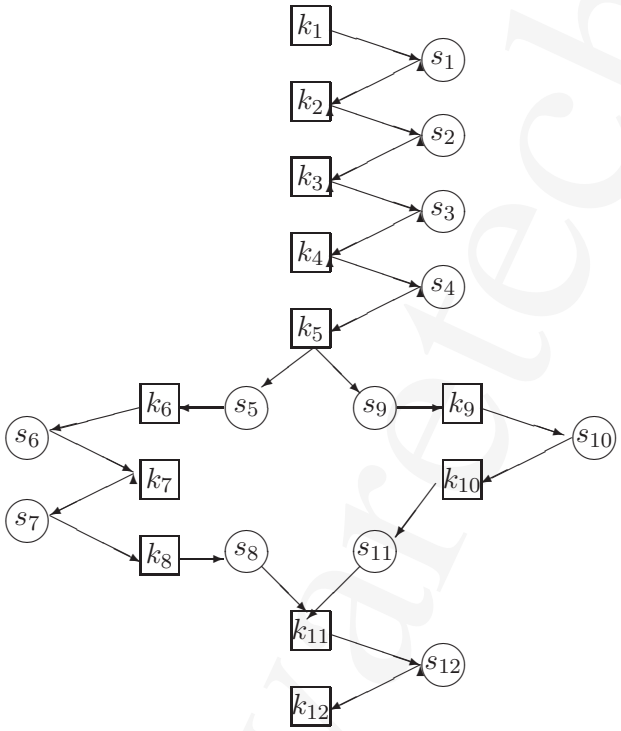
Legende:
Bedeutung der Vorgänge siehe Tabelle 5.2 Seite 143.
Kritische Pfad: $k_1, k_2, k_3, k_4, k_5, k_9, k_{10}, k_{11}, k_{12}$

Abbildung 5.11: Beispiel: Netzplan (vgl. BVB-Erstellung)

Knoten	Vorgang	Dauer [Mann-Tage]	Spätestes Ende
k_1	Verfahrensidee entwickeln	30	
k_2	Ist-Analyse ermitteln	40	
k_3	Forderungen dokumentieren	30	
k_4	Grobkonzept aufstellen	20	
k_5	Fachliches Feinkonzept aufstellen	50	
k_6	DV-technisches Feinkonzept entwickeln	100	
k_7	Programmierung durchführen	90	
k_8	Integrationstest durchführen	20	
k_9	Technische und organisatorische Vorbereitungen treffen	50	
k_{10}	Schulung vollziehen	200	
k_{11}	Verfahrenstest durchführen	30	
k_{12}	Einführung vollziehen	10	1.04.96

Legende:
Vgl. Abbildung 5.11 Seite 142.

Tabelle 5.2: Beispiel: Vorgänge eines Projektphasenplans



Legende:
Bedeutung der Vorgänge vgl. Tabelle 5.2 Seite 143.
Vgl. Netzplan: Abbildung 5.11 Seite 142.

Abbildung 5.12: Beispiel: Netzplan $\Longrightarrow$ Petri-Netz

schehnissen, die das termingerechte Erreichen des Projektendes besonders gefährden. Netzpläne dienen der Planung und Kontrolle der zeitlichen Abläufe. Petri-Netze haben ihren Schwerpunkt eher in der Analyse der Erreichbarkeit von Systemzuständen (vgl. Abschnitt 5.1.4 Seite 138). Sie erlauben zum Beispiel das zyklische Durchlaufen von Knotenfolgen (vgl. zum Beispiel Abbildung 5.9 Seite 138), während ein Netzplan eher auf den einmaligen Durchlauf mit Start-Ziel-Erreichung ausgelegt ist. Ein weiteres Beispiel eines Projektplanes zeigt Abbildung5.42 Seite 189 in Verbindung mit Tabelle 5.9 Seite 188.

5.1.8 Ablaufdiagramme ⇔ Petri-Netze

Ablaufdiagramme[12] sind Graphen, bei denen Anweisungen (Instruktionen) miteinander durch Pfeile verbunden sind. Im allgemeinen bestehen sie aus Sequenzen und Entscheidungen (Verzweigungen) zur Abbildung von Iterationen und („wilden“) Sprüngen. Im Vergleich zu Netzplänen (vgl. Abschnitt 5.1.7 Seite 140) stehen bei Ablaufdiagrammen die Aufspaltung (engl.: split) und die Sammlung (engl.: wait) nicht im Mittelpunkt, sondern erfordern zusätzliche (selten verwendete) Sondersymbole (vgl. Abbildung 5.13 Seite 146).

Bei „normalen“ Ablaufdiagrammen kann bei einem Durchlauf in jedem Fall nur die Ausführung einer nächsten Anweisung auf eine soeben ausgeführte folgen. Bei Ablaufdiagrammen gibt es Verzweigungen (engl.: branch) und ihre inverse Form, die Begegnung (engl.: meet). Mit ihnen sind Iterationen (while-Konstrukt oder until-Konstrukt) darstellbar. Darüberhinaus können auch komplexe, nichtlineare („chaotische“) Strukturen problemlos dokumentiert werden, da beliebige Rücksprünge durch entsprechende Pfeile (Kanten) möglich sind. Ein Beispiel einer nicht-linearen Struktur zeigt Abbildung 4.5 Seite 102.

branch **und** *meet*

In Petri-Netzen lassen sich die Instruktionen als Anschriften eines Knoten dokumentieren, also als beschriftete Stelle oder als beschrifteter Knoten. Eine Sequenz A, B, C eines Ablaufdiagramms bilden wir ab als beschriftetete Transitionen. Als zusätzliche Knoten sind zwischen diesen Transitionen Stellen einzufügen (vgl. Abbildung 5.14 Seite 147).

Die Alternative ist mit Hilfe einer zusätzlichen Kantenbeschriftung abbildbar (vgl. Abbildung 5.15 Seite 148). Man kann auch einen zusätzlichen Knotentyp einführen (vgl. zum Beispiel [60, 61]). Abbildung 5.16 Seite 149 zeigt ein entsprechendes Petri-Netz.

[12] Auch Ablaufpläne, Flußpläne oder Flußdiagramme genannt.

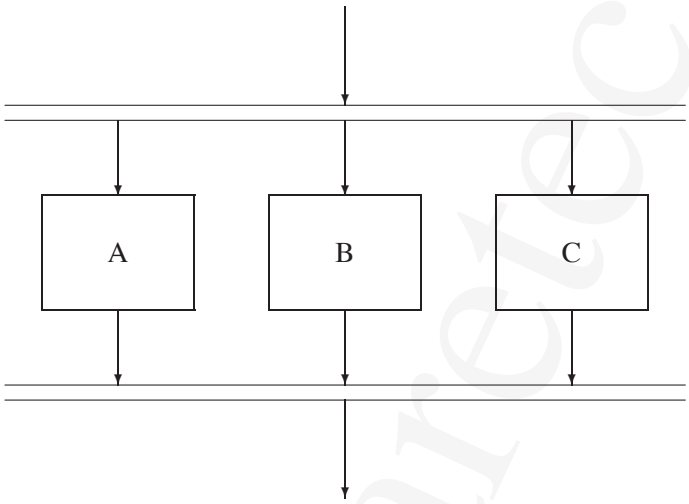


Abbildung 5.13: „split“ -Symbol beim Ablaufdiagramm

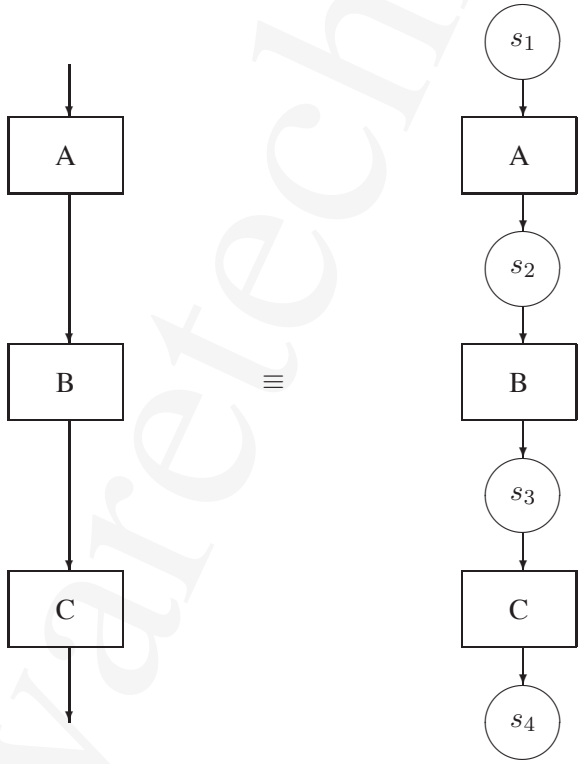


Abbildung 5.14: Sequenz: Ablaufdiagramm $\Longrightarrow$ Petri-Netz

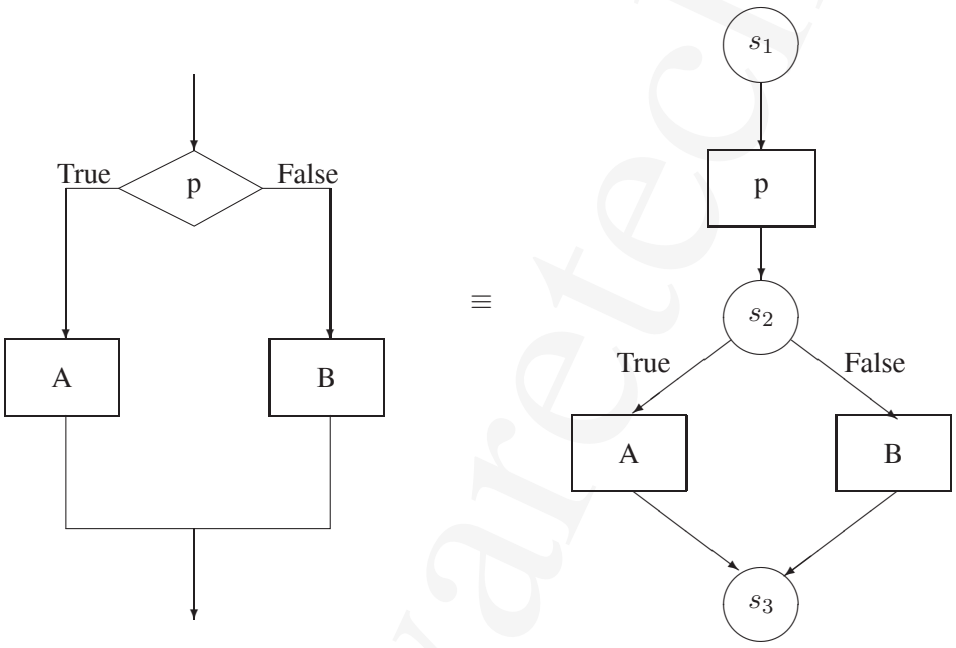
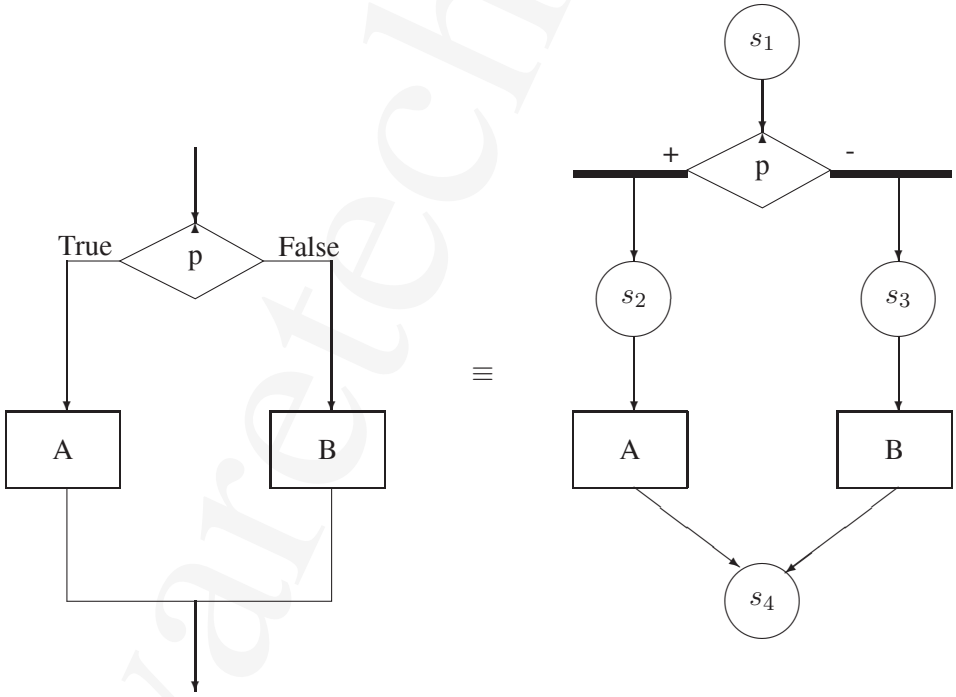


Abbildung 5.15: Alternative: Ablaufdiagramm $\Longrightarrow$ Petri-Netz



Legende:
+ ≡ True
- ≡ False

Abbildung 5.16: Alternative: Zusätzliches Symbol

5.1.9 Kleiner Exkurs Graphentheorie

In der Graphentheorie[13] besteht ein Graph aus einer Menge von Knoten, die teilweise durch Kanten miteinander verbunden sind. Ist den Kanten eine Richtung zugeordnet, spricht man von gerichteten Graphen. Diese können Zyklen enthalten oder zyklusfrei sein.

Halbordnung

Gerichtete zyklusfreie Graphen können eine sogenannte Halbordnung darstellen. Diese ist definiert durch die Relation zwischen den Knoten k:

$$k_i < k_j \tag{5.1}$$

gelesen als „$Knoten_i$ vor $Knoten_j$“ . Bei einer Vollordnung, zum Beispiel bei den natürlichen Zahlen, lassen sich alle Elemente in eine Reihenfolge bringen. Bei einer Halbordnung läßt sich diese Relation nicht für alle Knoten angeben. Ein Beispiel für eine Halbordnung sind komplizierte Verwandschaftsverhältnisse [60]. Bei ihrer Darstellung als Graphen lassen sich für jedes Individum mehrere Abstammungslinien sowohl rückwärts als auch vorwärts angeben, aber nicht für jeden Knoten die Relation 5.1. Ein solche Abstammungslinie wird als gerichteter Weg oder Bahn bezeichnet.

Eine Halbordnung läßt sich als ein Kausalgefüge betrachten. Mit einer endlichen Menge von Bahnen läßt sich ein gerichteter, zyklusfreier Graph überdecken. Erweitern wir die Relation 5.1 um ihre symmetrische Form, dann erhalten wir folgende allgemeine Kausalrelation:

$$caus(k_i, k_j) \equiv (k_i < k_j) \vee (k_j < k_i) \tag{5.2}$$

Kausalkette

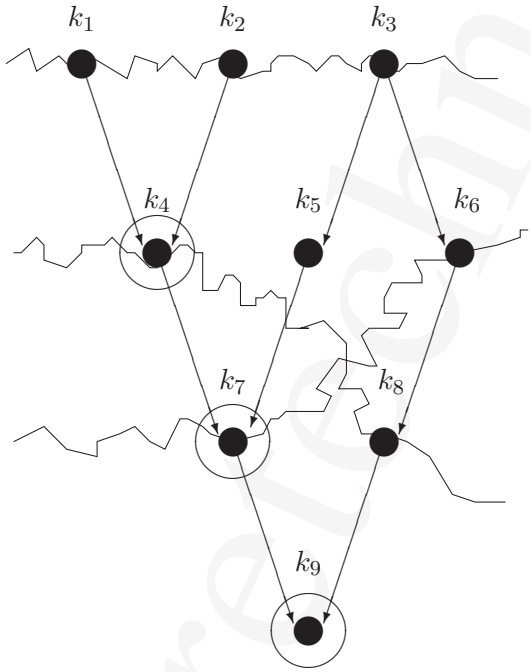
Die Gleichung 5.2 besagt, daß die beiden Knoten k_i und k_j entweder vorwärts oder rückwärts miteinander kausal verbunden sind. In einer Kausalkette gilt diese Relation für zwei beliebige Knoten. Sie bilden innerhalb der Menge der Knoten der Relation $caus$ einen Bezirk oder ken, kurz notiert als $caus(ken)$.

Demgegenüber unterscheiden wir die Relation[14] co, die zwischen zwei Knoten gilt, die nicht zueinander in der Relation 5.2 $caus$ stehen.

Auch innerhalb der Relation co lassen sich Bezirke, also $kens$ bilden, das heißt Teilmengen von Knoten, die in der Relation co stehen. Ein Beispiel zeigt Abbildung 5.17 Seite 151.

[13] Auch kombinatorische Topologie genannt.

[14] Abkürzung für „concurrent“ , deutsch: nebenläufig



Legende:

•	≡	Knoten
$\longrightarrow$	≡	Kante
$\sim$	≡	Alle Knoten auf einem $ken(\sim)$ sind zueinander nebenläufig. (Nur einige $kens$ eingezeichnet!)
◯	≡	Knoten einer Kausalrelation (Nur ein $ken(caus)$ eingezeichnet!)

Die Knoten der $kens(caus)$ bilden eine Vollordnung.
Die Konten der $kens(co)$ unterliegen keiner Ordnung.

Abbildung 5.17: Zyklusfreier Graph: Ordnung

$s_0 \equiv$ Terminal verfügbar
$s_1 \equiv$ Benutzer 1 arbeitet mit Terminal
$s_2 \equiv$ Benutzer 2 arbeitet mit Terminal
$s_3 \equiv$ Benutzer 3 arbeitet mit Terminal

$s_4 \equiv$ Benutzer 1 wartet auf Terminal
$s_5 \equiv$ Benutzer 2 wartet auf Terminal
$s_6 \equiv$ Benutzer 3 wartet auf Terminal

$t_1 \equiv$ Arbeitsbeginn von Benutzer 1
$t_2 \equiv$ Arbeitsbeginn von Benutzer 2
$t_3 \equiv$ Arbeitsbeginn von Benutzer 3

$t_4 \equiv$ Arbeitsende von Benutzer 1
$t_5 \equiv$ Arbeitsende von Benutzer 2
$t_6 \equiv$ Arbeitsende von Benutzer 3

Tabelle 5.3: Beispiel: „knappe Ressource“ (Knoten)

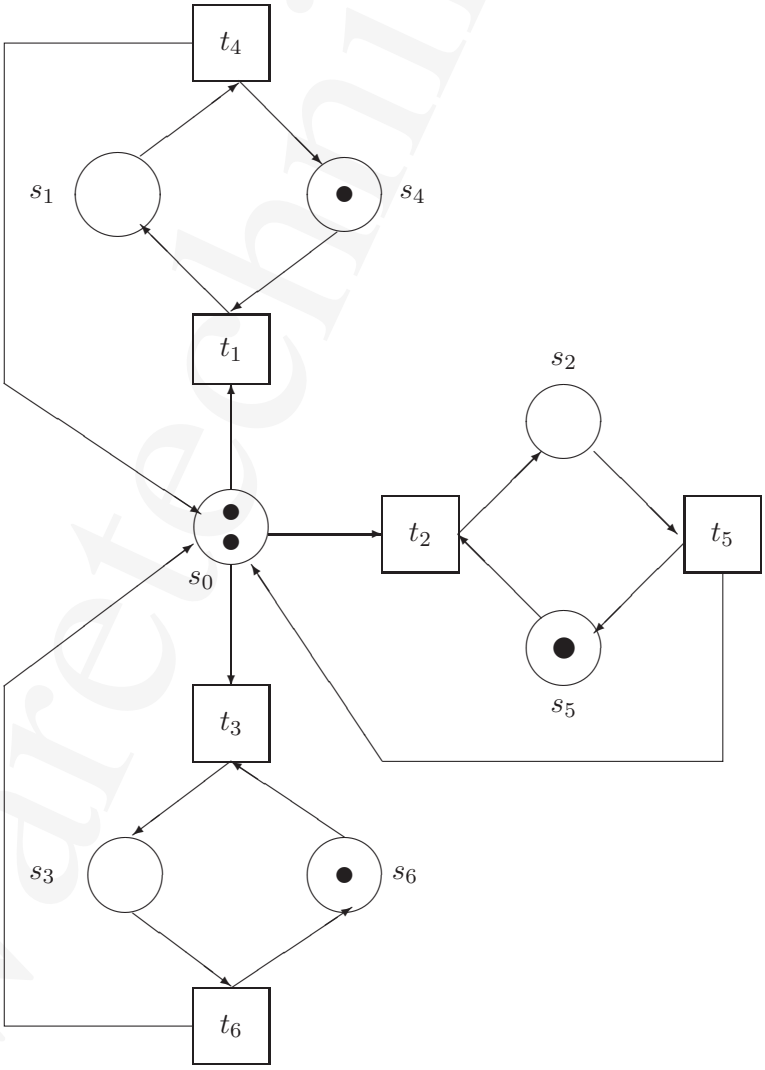
5.2 Stelle/Transitions-Netz

Bisher haben wir Netzbeispiele betrachtet ohne direkten Bezug zu praktischen Anwendungen. In diesem Abschnitt geht es um die Verwaltung begrenzter Betriebsmittel. Dazu nutzen wir Petri-Netze mit nicht unterscheidbaren Marken. Zunächst modellieren wir ein einfaches System mit drei Benutzern, die sich zwei Terminals teilen. Anschließend entwerfen wir ein Reservierungssystem für Sitzplätze in einem Flugzeug. In Abschnitt 5.5 Seite 190 weisen wir Eigenschaften unseres Netzentwurfes nach.

5.2.1 Beispiel: „Knappe Ressource“

Wir nehmen als Erläuterungsbeispiel an, daß sich drei Personalsachbearbeiter (Benutzer) eines rechnergestützten Personalverwaltungssystems zwei Dialogbildschirme teilen. Die knappe Ressource „Terminal“ bilden wir als zwei Marken ab. Zum Übergang von der Stelle Benutzer i wartet auf Terminal zur Stelle Benutzer i arbeitet mit Terminal bedarf es mindestens einer Marke auf der Stelle Terminal verfügbar. Abbildung 5.18 Seite 153 zeigt das entsprechende Netz.

Wir können das Beispiel 3 Benutzer und 2 Terminals als ein System im Sinne der Automatentheorie betrachten (vgl. [55]). Ein System befindet sich stets in einem Zustand, hier zum Beispiel beschrieben durch



Legende:
Vgl. Tabelle 5.3 Seite 152
Dargestellt ist der Anfangszustand: Zwei Terminals verfügbar (Entspricht Zustandsgraph von Tabelle 5.4 Seite 155).

Abbildung 5.18: Drei Benutzer teilen sich zwei Terminals

Benutzer 1 und 2 arbeiten mit Terminals, der durch eine Aktivität, hier zum Beispiel beschrieben durch Benutzer 2 beendet seine Arbeit, in einen neuen Zustand übergeht. In dem Beispiel sind sieben verschiedene Zustände z_i mit $i = 0, \ldots, 6$ abzubilden, die durch sechs verschiedene Aktivitäten t_j mit $j = 1, \ldots, 6$ entstehen. Tabelle 5.4 Seite 155 zeigt die Überführungstabelle.

Die Vorteile der graphischen Darstellung (Abbildung 5.18 Seite 153) gegenüber der tabellarischen Darstellung (Tabelle 5.4 Seite 155) sind offensichtlich. Sie sind besonders prägnant, wenn wir Veränderungen vornehmen. Wir unterstellen jetzt, daß der Benutzer 3 ein Systemprogrammierer ist, der stets 2 Terminals (eines für die Anwendung und eines für die DV-Systemkontrolle) benötigt. Die Transition t_3 kann nur dann eine Schaltberechtigung („Konzession“) haben, wenn s_0 zwei Marken hat. Diese Einschränkung modellieren wir, indem wir einer Kante, hier von s_0 nach t_3, ein Gewicht („Vielfachheit“) zuordnen. Wenn der Systemprogrammier seine Arbeit beendet, hier Transition t_6, dann gibt er beide Terminals zurück. Daher ist das Gewicht der Kante von t_6 nach s_6 ebenfalls 2. Abbildung 5.19 Seite 156 zeigt diese Modifikation.

Gewicht

In Abbildung 5.19 Seite 156 haben wir nur Kanten mit dem Gewicht $\neq 1$ beschriftet. Ein Kantengewicht $= 1$ wird nicht notiert. Die Anzahl der Marken, die eine Stelle maximal haben kann bezeichnet man als Kapazität der Stelle. Sie wird nur dann angegeben, wenn die Gefahr besteht, daß sie überschritten wird. Ohne Kapazitätsangabe hat eine Stelle eine unbeschränkt Kapazität, mathematisch häufig mit „ω“ notiert.

Kapazität

Für ein solches Netz sind die folgenden 5 Angaben charakteristisch: Stellen, Transitionen, Flußrelationen, Gewichte und Anfangsmarkierungen. Tabelle 5.5 Seite 158 dokumentiert ein solches Netz in mathematischer Notation.

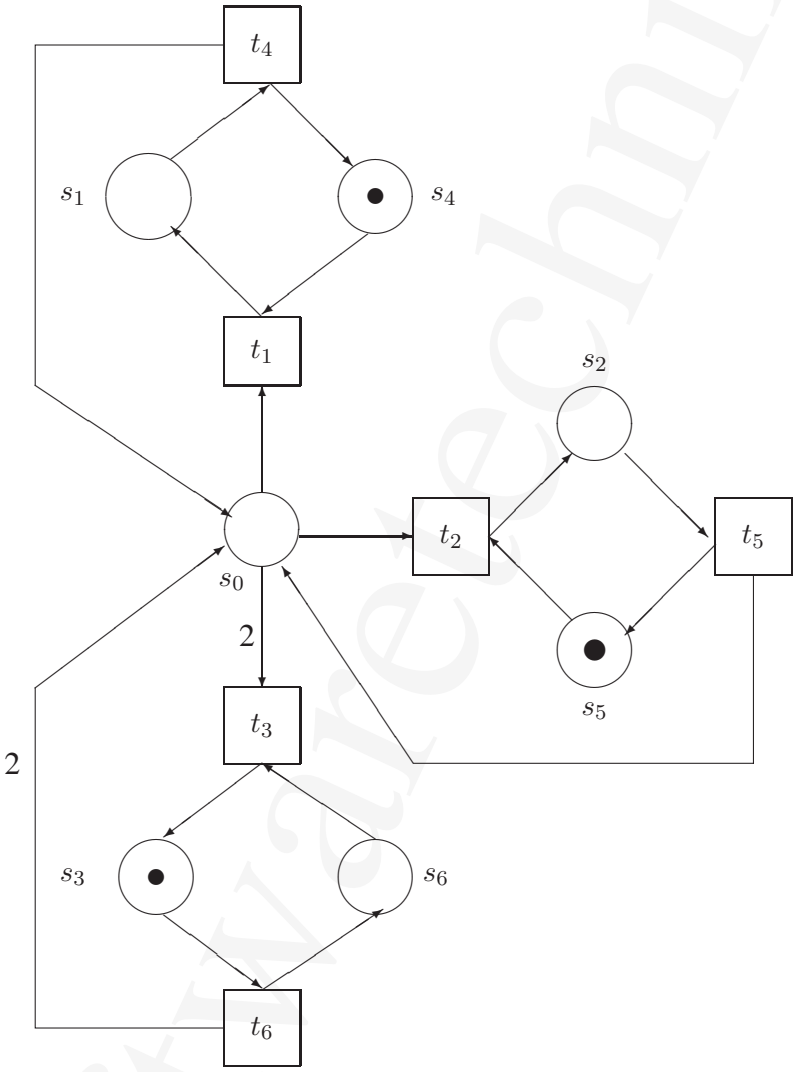
Wir unterstellen nun einmal, es gäbe keine „knappe Resource“, weil zwischenzeitlich genügend Terminals verfügbar sind: Im ersten Fall (Abbildung 5.18 Seite 153) also 3 Terminals im zweiten Fall (Abbildung 5.19 Seite 156) 4 Terminals. Man braucht nun nur eine entsprechende Anzahl von Marken einzuführen. Ein zweckmäßigere Lösung zielt jedoch auf ein einfacheres Modell. Wir streichen die Stelle s_0 mit ihren Kanten, da das Arbeiten der Benutzer 1,2 und 3 nicht mehr von der Anzahl verfügbarer Terminals abhängt. Das Netz zerfällt in drei Teilnetze, die jeweils einen Benutzer modellieren (vgl. Abbildung 5.20 Seite 157).

Aktivi-täten	Zustände						
	z_0	z_1	z_2	z_3	z_4	z_5	z_6
t_1	z_1		z_4	z_5			
t_2	z_2	z_4		z_6			
t_3	z_3	z_5	z_6				
t_4		z_0			z_2	z_3	
t_5			z_0		z_1		z_3
t_6				z_0		z_1	z_2

Legende:
Beispiel: Drei Benutzer teilen sich zwei Terminals (vgl. Abbildung 5.18 Seite 153).

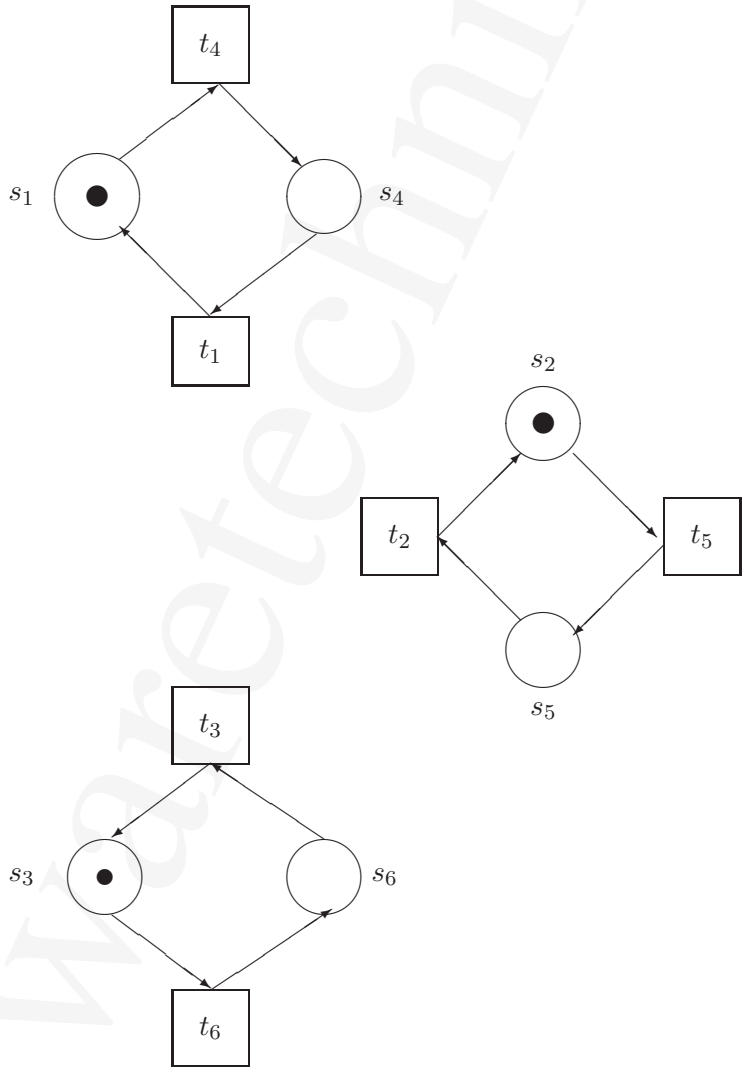
$z_0 \equiv$ Anfangszustand (Zwei freie Terminals)
$z_1 \equiv$ Benutzer 1 arbeitet mit Terminal
$\hookrightarrow s_1$
$z_2 \equiv$ Benutzer 2 arbeitet mit Terminal
$\hookrightarrow s_2$
$z_3 \equiv$ Benutzer 3 arbeitet mit Terminal
$\hookrightarrow s_3$
$z_4 \equiv$ Benutzer 1 $\wedge$ 2 arbeiten mit Terminals
$\hookrightarrow s_1 \wedge s_2$
$z_5 \equiv$ Benutzer 1 $\wedge$ 3 arbeiten mit Terminals
$\hookrightarrow s_1 \wedge s_3$
$z_6 \equiv$ Benutzer 3 $\wedge$ 2 arbeiten mit Terminals
$\hookrightarrow s_3 \wedge s_2$

Tabelle 5.4: Überführungstabelle



Legende:
Vgl. Tabelle 5.3 Seite 152.
Dargestellt ist der Zustand: Benutzer 3 arbeitet mit zwei Terminals

Abbildung 5.19: Benutzer 3 benötigt stets zwei Terminals



Legende:
Vgl. Tabelle 5.3 Seite 152
Dargestellt ist der Zustand: Alle Benutzer arbeiten mit einem Terminal

Abbildung 5.20: Genügend Terminals verfügbar

Mathematische Notation: (vgl. Tabelle 5.1 Seite 139)

1. Das 5 Tupel $\mathcal{N} \equiv (\mathcal{S}, \mathcal{T}, \mathcal{F}, \mathcal{W}, M_0)$ heißt Stellen/Transitionen-Netz (kurz S/T-Netz), falls gilt:
2. $(\mathcal{S}, \mathcal{T}, \mathcal{F})$ ist ein Netz aus Stellen $\mathcal{S}$ und Transitionen $\mathcal{T}$.
3. $\mathcal{W}$ ist eine Abbildung, die jeder Kante f der Flußrelationen $\mathcal{F}$ eine positive natürliche Zahl $\mathcal{W}(f)$ zuordnet.
4. M_0 ist eine Anfangsmarkierung.

Tabelle 5.5: Netz mit Gewichtsangaben

5.2.2 Beispiel: „Reservierung“

Wir entwerfen ein einfaches Buchungsystem mit Hilfe von S/T-Netzen[15]. Es sind Sitzplätze eines Flugzeugs zu verwalten. Unser System TOPI (Akronym für TOtale PlatzInformation") basiert auf folgender verbalen Anforderungsspezifikation (vgl. Tabelle 5.6 Seite 159).

Die Schnittstelle des Systems TOPI zur Umwelt definieren wir in Form der beiden Stellen: „Eingabe“ und „Ausgabe“ . Auf der Stelle „Eingabe“ erwarten wir Marken mit den Angaben über den Kunden und, ob es sich um eine Buchung oder Stornierung handelt. Auf der Stelle „Ausgabe“ legen wir Marken ab, die die jeweilige Nachricht (den Beleg) für den Kunden abbilden.[16] Abbildung 5.21 Seite 160 zeigt ein solches Netz auf oberen Abstraktionsgrad. Ausgehend von dieser Darstellung entsteht durch schrittweises Verfeinern unser TOPI-Entwurf in Form von drei Teilnetzen (Abbildung 5.22, 5.23 und 5.24).

Wir verdeutlichen unsere Lösung zunächst mit drei Teilnetzen, bei denen die Stellen, Transitionen und Kanten beschriftet sind. Das Gesamtnetz überführen wir in Abschnitt 5.5 Seite 190 schrittweise in ein S/T-Netz ohne Stellen- und Transitions-Anschriften, nur mit gewichteten Kanten.

Die Texte auf den Stellen skizzieren Prädikate, die erfüllt sein müssen, damit die jeweilige Transition schalten kann. Die in den TOPI-Netzen beschrifteten Transitionen stellen Anweisungen dar, die beim Schalten der Transition als ein Block vollzogen werden. Der Schaltvorgang passiert quasi unendlich schnell. Eine Beeinflussung von Zuweisungsbefehlen im Schaltvorgang durch andere, nebenläufig schaltende Transitionen ist somit ausgeschlossen.

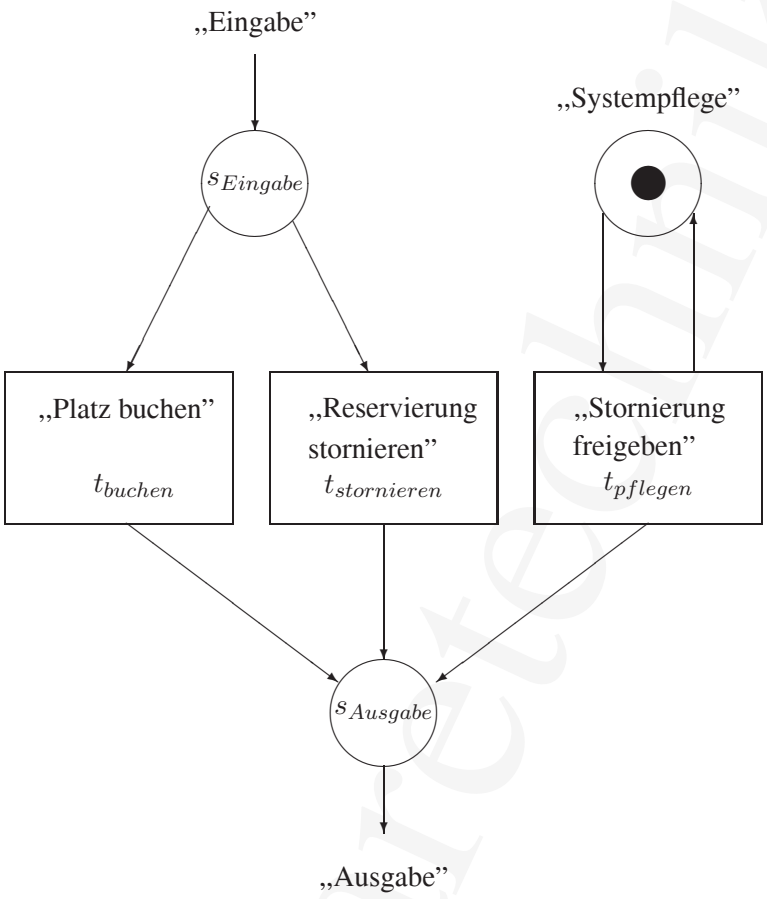
[15] Idee und Lösungsstruktur [50], S. 90 ff

[16] Näheres zu unterscheidbaren Marken behandeln wir im Abschnitt 5.3 Seite 164.

Anforderungen für das TOtale PlatzInformations-System
(kurz: TOPI):
Hinweis: Identifier für eine verbale Anforderung:
$A < level-1 > . < level-2 >, \ldots, < level-i >, \ldots, < level-n >$
mit $< level - i > \equiv$ Gliederungsstufe

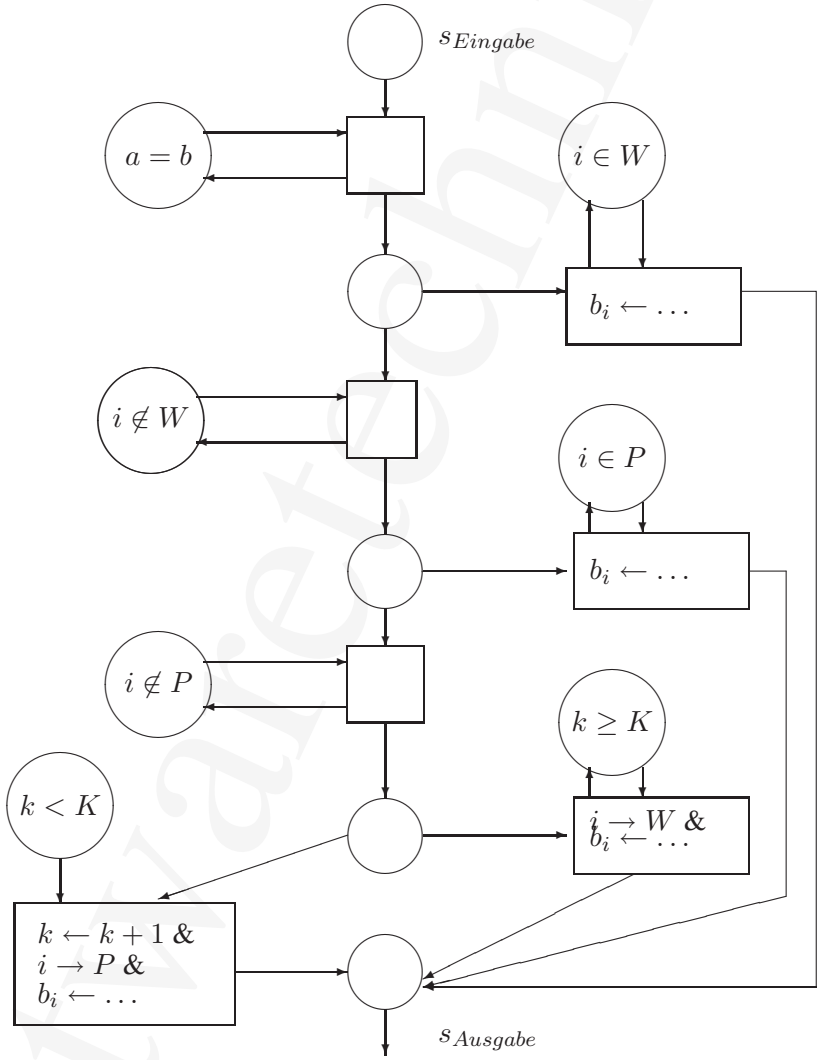
A1	TOPI reserviert Sitzplätze in einem Flugzeug.
A2	Auf TOPI können verschiedene voneinander unabhängige Reisebüros zugreifen und
A2.1	einen Platz buchen oder eine
A2.2	Reservierung stornieren.
A3	TOPI führt jeden Buchungsauftrag aus, indem es den
A3.1	Kunden in die Passagierliste aufnimmt, falls diese nicht voll ist, anderenfalls
A3.2	wird der Kunde in eine Warteliste aufgenommen.
A4	TOPI streicht bei einem Stornierungsauftrag den Kunden aus
A4.1	der Passagierliste oder
A4.2	der Warteliste.
A5	TOPI erstellt in jedem Fall eine Nachricht für den Kunden. Dies gilt auch für unausführbare Aufträge wie zum Beispiel
A5.1	wiederholte Buchung desselben Kunden oder
A5.2	Stornierung eines nicht gebuchten Platzes.
A6	Ein Aktualisierungs-Programm ermöglicht, stornierte Plätze
A6.1	für Kunden auf der Warteliste zu reservieren, sie entsprechend zu benachrichtigen oder
A6.2	bei leerer Warteliste solche Plätze zur Buchung direkt freizugeben.
A7	TOPI bearbeitet Buchungs- und Stornierungsaufträge so weit wie möglich nichtsequentiell, um einen hohen Durchsatz zu erreichen.

Tabelle 5.6: TOPI-Anforderungsspezifikation



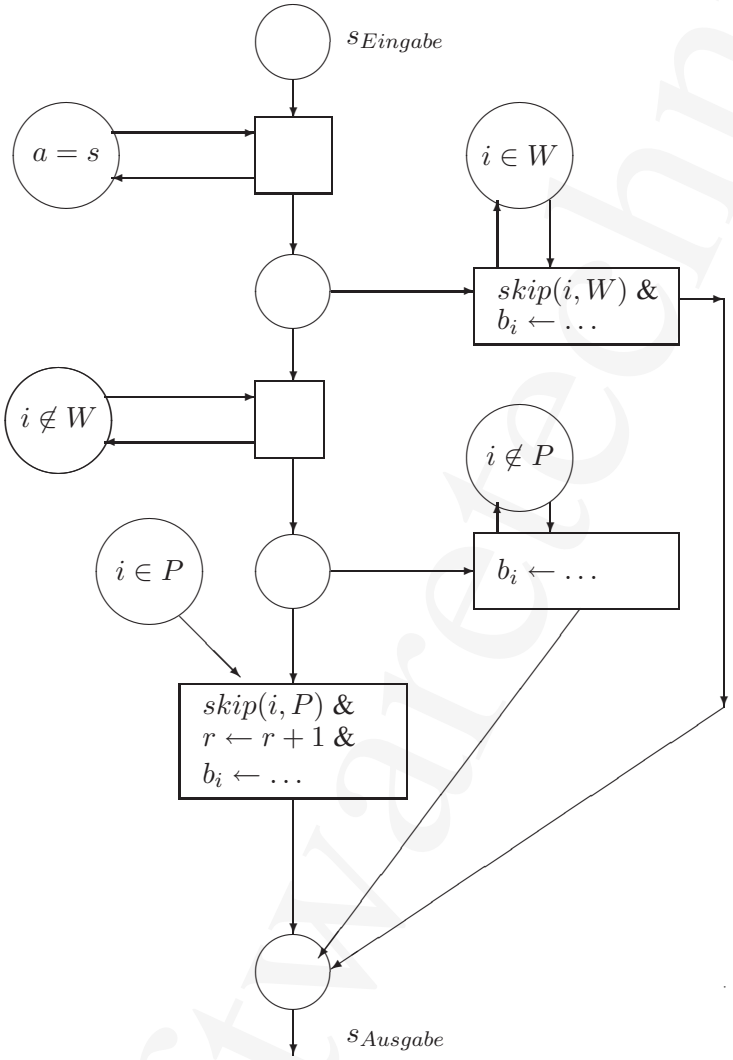
Legende:
Vgl. Tabelle 5.6 Seite 159
Stellen S:
„Eingabe" ; vgl. A1, A7
„Ausgabe" ; vgl. A5
„Systempflege" ; vgl. A6
Transitionen T:
„Platz buchen" ; vgl. A2.1, A3, A5.1
„Reservierung stornieren" ; vgl. A2.2, A4, A5.2
„Stornierungen freigeben" ; vgl. A6

Abbildung 5.21: TOPI als Kontextnetz (obere Abstraktionsgrad)



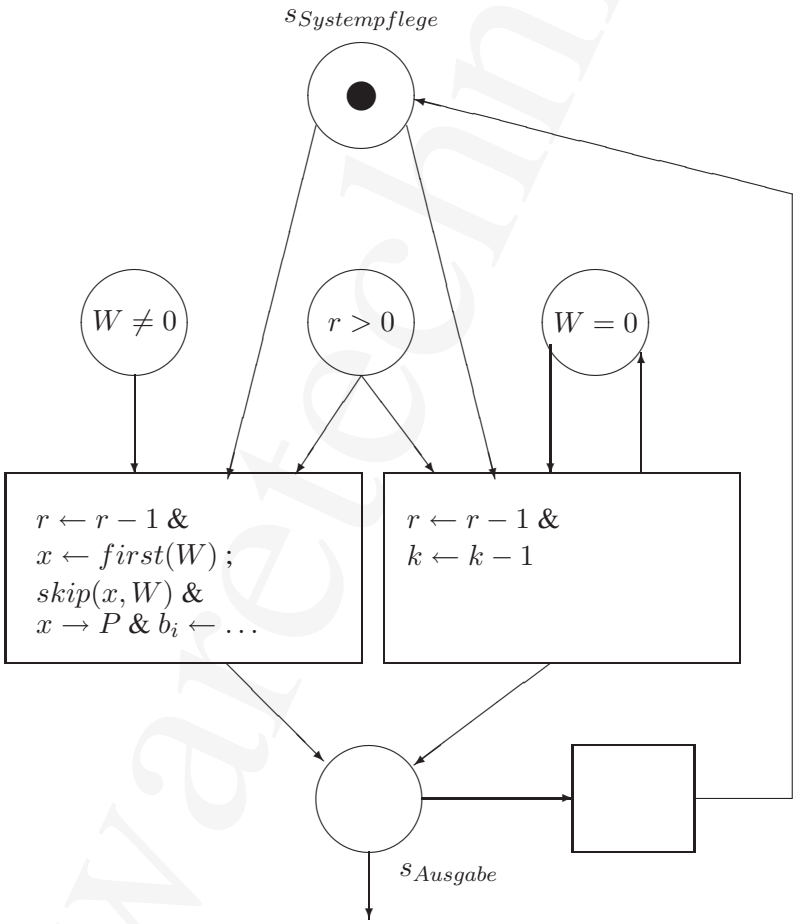
Legende:
Vgl. Tabelle 5.7 Seite 164.

Abbildung 5.22: TOPI-Teilnetz: „Platz buchen“



Legende:
Vgl. Tabelle 5.7 Seite 164.

Abbildung 5.23: TOPI-Teilnetz: „Reservierung stornieren“



Legende:
Vgl. Tabelle 5.7 Seite 164.

Abbildung 5.24: TOPI-Teilnetz: „Systempflege“

i $\equiv$ Kundenidentifizierung
a $\equiv$ Auftragsart:
$b \equiv$ buchen
$s \equiv$ stornieren
P $\equiv$ Passagierliste
K $\equiv$ Zahl der Plätze (Kapazität von P)
k $\equiv$ Zahl der in P reservierten Plätze
W $\equiv$ Warteliste
r $\equiv$ stornierte, aber noch nicht gelöschte
Reservierungen (aktuelle Zahl)
b_i $\equiv$ Beleg (Nachricht) an das Reisebüro des Kunden i

Tabelle 5.7: Beschreibung der TOPI-Komponenten

Es gilt folgende Notation für die Junktoren:

- „& “ $\equiv$ nebenläufige Ausführung von Anweisungen
- Semikolon $\equiv$ sequentielle Ausführung von Anweisungen

Die Beschriftung der Stellen sind Bedingungen (Prädikate), die zu erfüllen sind, um ein Schalten der Transition zu ermöglichen.

Die Passagierliste P und die Warteliste W sei nach dem Prinzip First-In-First-Out (FIFO) organisiert, so daß wir folgende Notation nutzen können:

- $first(W) \equiv$ erstes Element von W
- $skip(x, W) \equiv$ löscht x aus der Liste W
- $x \rightarrow W \equiv$ setzt das Element x an den Schluß der Liste W
- $b_i \leftarrow \ldots \equiv$ steht für die jeweils geeignete Nachricht für den Kunden i.

Die Eigenschaften dieses Netzes analysieren wir in Abschnitt 5.5 Seite 5.5.

5.3 Netze mit unterscheidbaren Marken

Zunächst haben wir Stellen und Transitionen mit zusätzlichen Informationen (Bedingungen oder Anweisungen) beschriftet. Die Kanten trugen

gegebenenfalls Gewichtsangaben (vgl. Abbildung 5.19 Seite 156). Eine nächster Schritt sind Netze mit Anschriften an Kanten. Bei solchen Netzen unterscheiden wir die einzelnen Marken. Längs einer beschrifteten Kante können nur diejenigen Marken fließen, die der Beschriftung entsprechen.

Kantenanschrift

Die Kantenanschrift kann sich auf einzelne (konkrete) Objekte beziehen oder im Sinne einer Variablen auf Objekttypen. Im ersten Fall muß die Marke genau das genannte Objekt sein. Im zweite Fall muß die Marke ein möglicher Wert der genannten Variablen sein, so daß alle Anschriften erfüllt werden. Wir skizzieren im folgenden anhand von Beispielen einige Möglichkeiten bei Unterscheidung der einzelnen Marken.

5.3.1 Konstante Objekte als Kantenanschrift

Ellipsen

Um die einzelnen Marken unterscheiden zu können, stellen wir sie nicht mehr als Punkt dar, sondern geben jede Marke mit ihrem eindeutigen Bezeichner (Identifier) an. Im folgenden verwenden wir dazu Großbuchstaben. Damit wir die Bezeichner leichter in den Stellen notieren können, „entarten“ unsere Kreise häufig zu Ellipsen.

Marken repräsentieren jetzt verschiedene Objekte. Welches Objekt (welche „Marke“) von einer Stelle oder auf eine Stelle fließen kann, bestimmt die Kantenanschrift. Weist eine Kante einen Objektbezeichner auf, dann kann nur dieses Objekt entlang der Kante „fließen“. Ist die Kante nicht beschriftet kann jedes Objekt „fließen“.

Eine Transition t_j ist schaltberechtigt, wenn jede Stelle s_i aus ihrem Vorbereich dasjenige Objekt aufweist, das die jeweilige Kantenanschrift von s_i nach t_j bezeichnet. Die Transition t_j schaltet, indem

1. jeder Stelle s_i das Objekt entnommen wird, das die Kante von s_i nach t_j nennt, und

2. jede Stelle s_k aus dem Nachbereich von t_j das Objekt erhält, das die jeweilige Kante von t_j nach s_k bezeichnet.

Mit dieser Schaltregel kann die Transition t_j jedes genannte Objekt einer Kantenanschrift „vernichten“ bzw. „erzeugen“ . Abbildung 5.25 Seite 166 verdeutlicht ein solches Schalten.

Die Vorteile einer Kantenbeschriftung verdeutlicht das folgende Beispiel. Es geht um drei Programmierer, die bei einem Softwarepaket abwechselnd Modifikationen vornehmen. Zunächst ist das Beispiel mit nichtunterscheidbaren Marken dargestellt (Abbildung 5.26 Seite 168). Dabei ist nicht sichtbar, welcher der drei Programmierer gerade durch

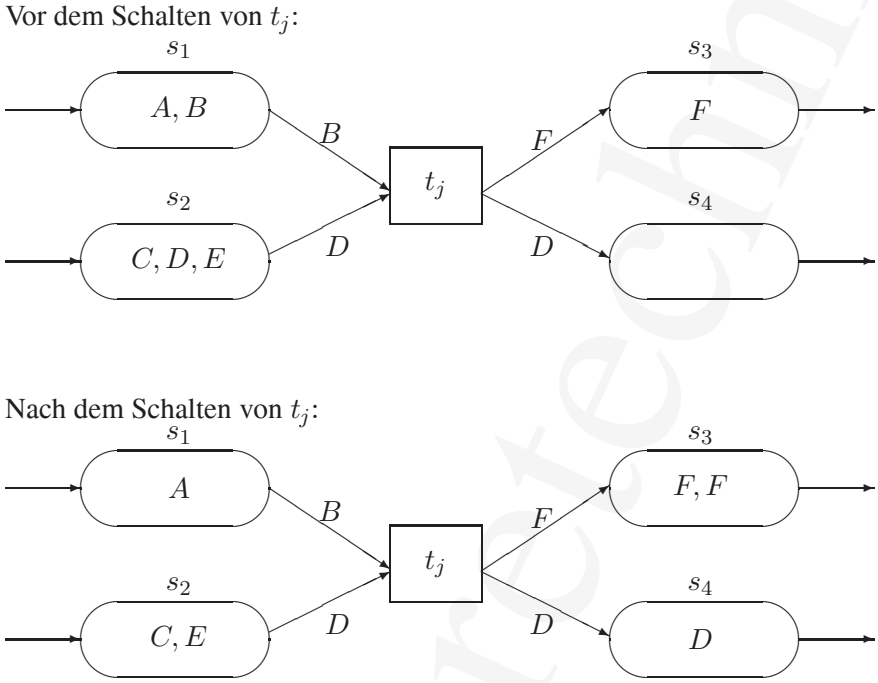


Abbildung 5.25: Beispiel: Konstante Objekte als Kantenanschrift

die Marke auf s_4 repräsentiert wird. Abbildung 5.27 Seite 169 zeigt die Programmierer als Marken P_1, P_2 und P_3. Die Modifikationen sind als Marken M_1 und M_2 dargestellt. Auf der Stelle s_4 befindet sich die Marke P_2. Sichtbar ist nun, daß ein ganz bestimmter Programmierer die Modifikation vollzogen hat. Auf der Stelle s_2 liegt die Marke M_2, das heißt die Modifikation M_2 befindet sich in der Produktion.

Abbildung 5.27 Seite 169 zeigt, daß bei mehreren Objekten das Netz schnell wächst und (im Vergleich zu Abbildung 5.26 Seite 168) unübersichtlich wird. Die Information, welcher Programmierer welche Modifikation vollzieht, ist einfacher darstellbar, wenn die Kantenanschriften sich nicht auf das einzelne Objekt beziehen, sondern auf Objekttypen. Bei den Kanten wollen wir nur unterscheiden, daß entweder Modifikationen oder Programmierer betroffen sind. Es geht daher um den Objekttyp im Sinne einer Variablen. Diese hat als Wert das jeweilige Objekt. Wir erweitern somit unser Netz zu einem mit variablen Objekten als Kantenanschrift (vgl. Abbildung 5.29 Seite 171).

5.3.2 Variable Objekte als Kantenanschrift

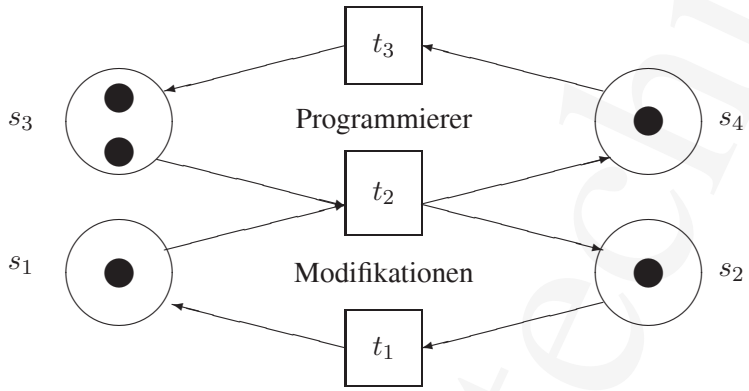
Die Schaltregel bei Variablen als Kantenanschriften basiert auf der widerspruchsfreien Ersetzung der einzelnen Variablen durch individuelle Objekte. Die Ersetzung im Vorbereich und im Nachbereich einer Transition t_j muß widerspruchsfrei möglich sein, sonst kann t_j nicht schalten. Dazu müssen die Stellen des Vorbereiches passende Objekte des Typs haben, die durch die Variable an den Eingangskanten von t_j bestimmt werden. Abbildung 5.28 Seite 170 zeigt ein entsprechendes Beispiel.

Mit Variablen läßt sich das obige Beispiel 3 Programmierern und 2 Modifikationen stark vereinfacht darstellen (Abbildung 5.29 Seite 171). Jetzt kann die Zahl der beteiligten Programmierer oder der Modifikationen bei gleichem Netz problemlos verändert werden.

Je nach Bedarf sind die Kantenanschriften im Zusammenhang mit einer passenden Schaltregel erweiterbar. So können wir die Gewichtsangabe mit der Objekttypangabe kombinieren. Es „fießen“ dann entlang einer Kante zum Beispiel „$2 * x$“ Objekte. Auch die Begrenzung auf einen Objekttyp pro Kante können wir aufgeben. Eine Kante kann dann zum Beispiel die Anschrift „$2 * x + y$“ aufweisen.

5.3.3 Prädikat/Transitions-Netz

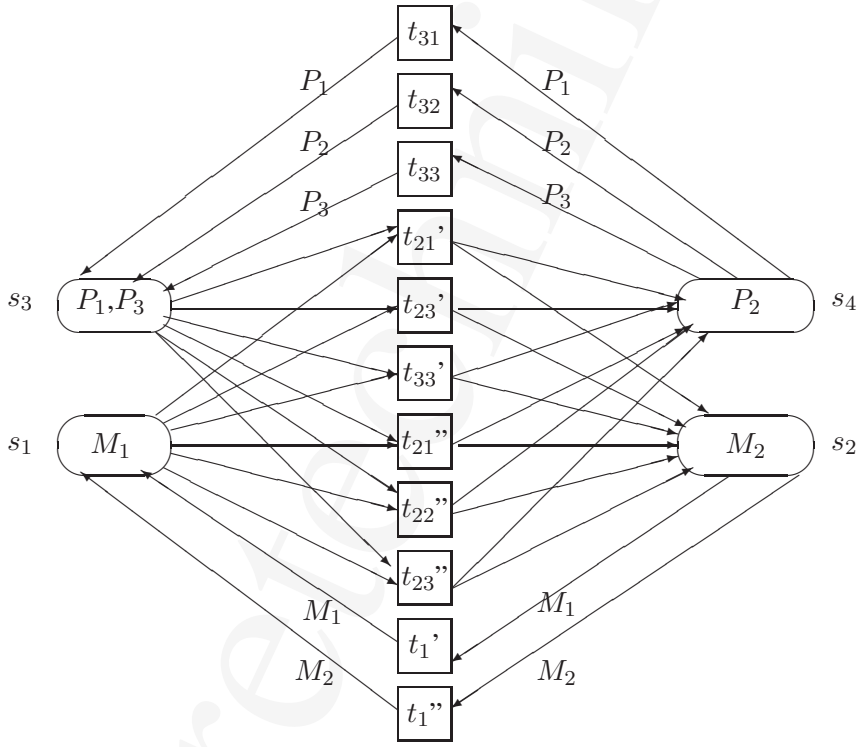
Es mag ausreichen, Objekte durch ein einfaches Merkmal zu unterscheiden. Wir können uns dann gefärbte Marken als Repräsentation der einzelnen Objekte vorstellen. Haben wir es mit einem einfachen Objekttyp



Legende:
3 Programmierer und 2 Modifkationen

$s_1 \equiv$ spezifizierte Softwaremodifikationen
$s_2 \equiv$ Modifikation in Produktion
$t_1 \equiv$ Produktionsprobleme machen Softwaremodifikation erforderlich
$t_2 \equiv$ Modifikation durchführen
$s_3 \equiv$ Sachkundiger Programmierer
$s_4 \equiv$ Modifikationsauftrag vollzogen
$t_3 \equiv$ Programmierer schulen

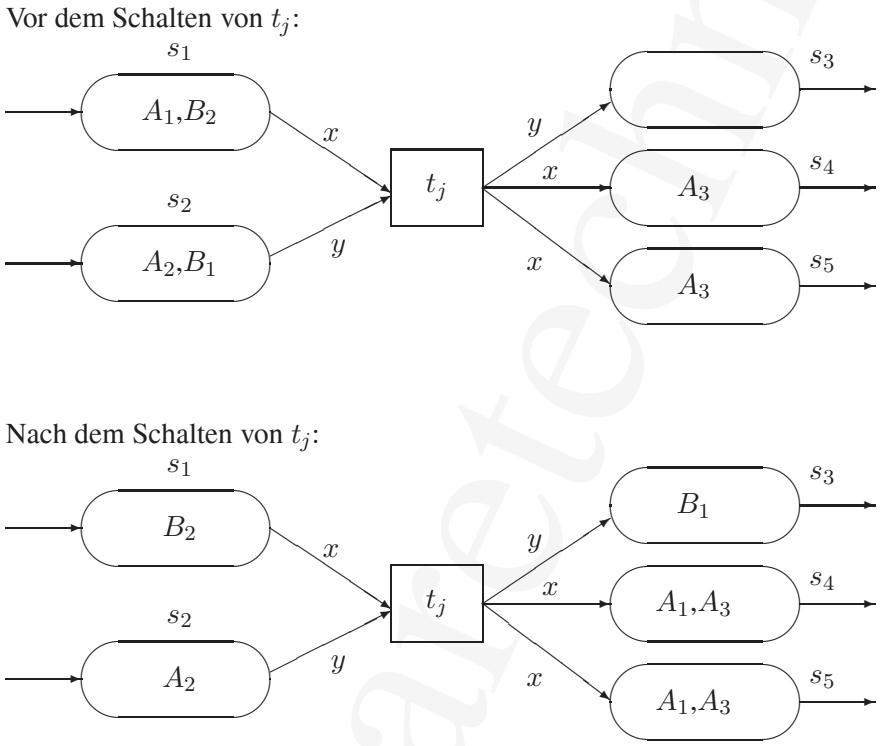
Abbildung 5.26: Beispiel: Nichtunterscheidbare Marken



Legende:
3 Programmierer und 2 Modifkationen
siehe Abbildung 5.26

P_1, P_2, P_3 $\equiv$ Programmierer
M_1, M_2 $\equiv$ Softwaremodifikationen

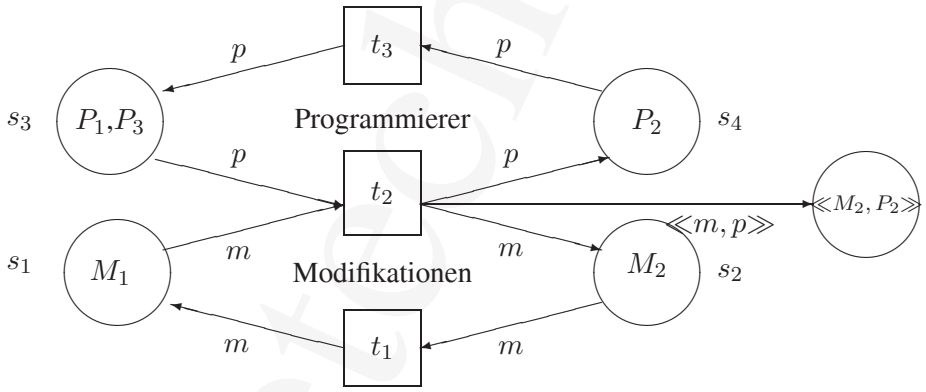
Abbildung 5.27: Beispiel: Unterscheidbare Marken



Legende:

$x \equiv$ Variable mit Wertebereich: $A_1, A_2, \ldots, A_n$
$y \equiv$ Variable mit Wertebereich: $B_1, B_2, \ldots, B_m$

Abbildung 5.28: Beispiel: Variable Objekte als Kantenanschrift



Legende:
3 Programmierer und 2 Modifkationen vgl. Abbildung 5.26 Seite 168.

$p \equiv$ Programmierer mit Wertebereich: P_1, P_2, P_3
$m \equiv$ Modifikationen mit Wertebereich: M_1, M_2

Abbildung 5.29: Beispiel: Programmierer & Modifikationen

PrT-Netz

zu tun, dann bietet sich ein Netz mit gefärbten Marken an. Für jedes einzelne Objekt steht eine andersfarbige Marke. Häufig jedoch sind Marken zu verwenden, die nicht nur durch ein einfaches Merkmal (ihre Farbe) zu unterscheiden sind, sondern in ihrer Struktur. In solchen Fällen geht man zu Prädikat/Transition(s)-Netzen (kurz: PrT-Netze) über.

Konstruktion von Marken

Zu einem PrT-Netz gehört eine Sprache, in der man die Konstruktion der verwendeten Marken definieren kann. Die mathematische Logik (Algebra) stellt eine solche Termsprache bereit (Näheres vgl. [14]). Mit der strengen Syntax einer Beschreibungssprache gewinnen wir die Möglichkeit, die Anschriften zu präzisieren.

Abbildung 5.30 Seite 173 zeigt ein Beispiel (vgl. [27]). Die Schaltregel ist offensichtlich:

1. die logische Formel der Transition, hier: „$y < z$“, ist zu erfüllen,
2. die Eingangsprädikate, hier: s_1 und s_2, haben genügend passende Marken, und
3. die Ausgangsprädikate, hier: s_3 und s_4, haben genügend Kapazität (kein Überlauf!).

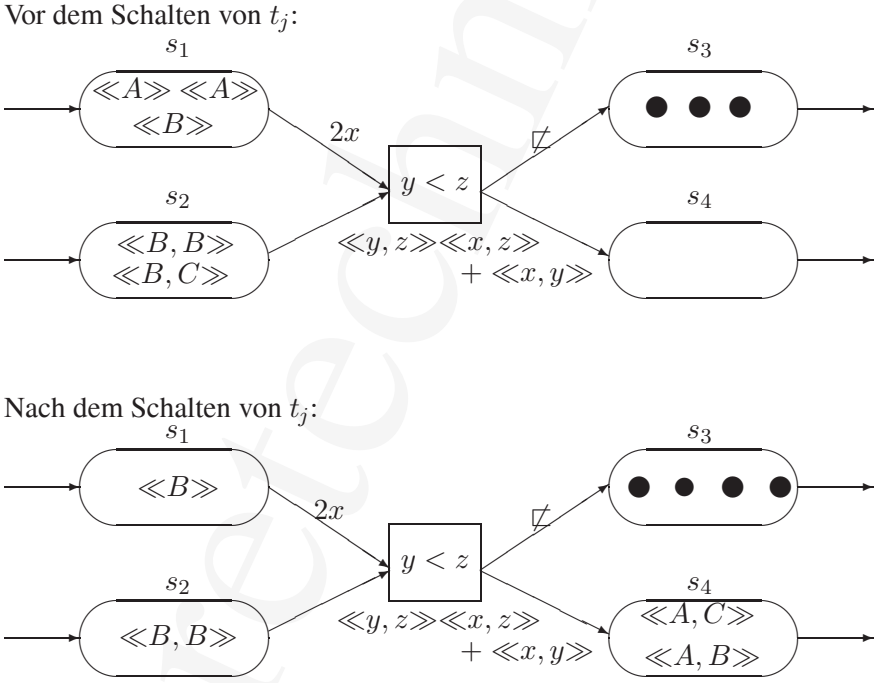
5.4 Modellieren

Zum Modellieren eines konkreten Problems benötigen wir Netzdarstellungen auf verschiedenen Abstraktionsebenen. Dazu sind einerseits grobe Netze mit wenigen Knoten in mehrere Teilnetze zu verfeinern. Andererseits gilt es, im Sinne von Fertigteilen oder Halbfertigteilen schon existierende Netze miteinander zu einem neuen Netz zu verknüpfen.

Verfeinerungsproblem

Intuitiv erscheint es einfach, eine Transition t_j durch ein ganzes Netz zu ersetzen. Selbstverständlich beachten wir die offensichtliche Plausibilität: Die Zahl und die Beschriftung (zum Beispiel das Gewicht) der Eingangs- und Ausgangs-Kanten von t_j bleibt unverändert. Damit ist das Verfeinerungsproblem allerdings nicht gelöst. Wir fordern, daß der Markenfluß im Eingangs- und Ausgangsbereich von t_j sich nicht ändert. Mit dieser Forderung nach gleichem Markenverhalten (auch„Markentreue“ genannt, vgl. zum Beispiel [51]) ist das Verfeinern und das Kombinieren von Netzen keine triviale Aufgabe.

Im folgenden sind anhand weniger Beispiele das Verfeinern und das Kombinieren im Ansatz skizziert.



Legende:

$<$	$\equiv$ lexikalische Ordnung in $y < z$
$\not\sqsubset$	$\equiv$ Fluß einer anonymen Marke
$\ll$A$\gg$	$\equiv$ Beispiel einer unterscheidbaren Marke
$\ll$B,B$\gg$	$\equiv$ Beispiel einer strukturierten Marke
$\ll$x,y$\gg + \ll$x,y$\gg$	$\equiv$ Beispiel einer Konstruktionsvorschrift

Abbildung 5.30: Beispiel: Prädikat/Transitions-Netz

5.4.1 Verfeinern

Wir nehmen ein einfaches S/T-Netz mit einer Transition t_j an. Ihr Vorbereich besteht aus den Stellen s_1 und s_2. Die Stelle s_1 sei markiert (Abbildung 5.31 Seite 175). Die Transition t_j hat keine Konzession. Sie kann nicht schalten, da s_2 keine Marke hat. Ersetzt man nun t_j durch ein S/T-Netz, dann soll nicht durch eine Konzession die Marke von s_1 „abfließen“ können. Anders formuliert: Das Ersatznetz soll ebenfalls nur Konzession haben, wenn beide Stelle s_1 und s_2 markiert sind. Diese Forderung erreichen wir durch beibehalten einer gemeinsamen Eingangstransition t_{j1} (vgl. Abbildung 5.31 Seite 175). Ohne eine solche gemeinsame Eingangstransition, können wir leicht ein Ersatznetz entwerfen, das kein „markengerechtes“ Verfeinern darstellt (Abbildung 5.32 Seite 176).

Markengerecht!

Analog zum Vorbereich von t_j ist beim Verfeinern der Nachbereich von t_j zu beachten. Auch hier fordern wir ein markengerechtes Verhalten. Eine einfache Lösung ist eine gemeinsame Ausgangstransition, hier: t_{j4} in Abbildung 5.31 Seite 175 für die Stellen s_3, s_4 und s_5.

Abbildung 5.33 Seite 177 zeigt eine Lösung mit Rückkopplung. Diese verhindert fehlende Marken auf den Stellen s_3, s_4 und s_5, wenn Marken auf s_1 und s_2 ein mehrfaches Schalten der Eingangstransition t_{j0} ermöglicht. In diesem Fall könnte zum Beispiel t_{j1} auch mehrfach schalten, wenn es nicht die Rückkopplungsstellen s_{r1}, s_{r2} und s_{r3} gäbe. Diese zusätzlichen Stellen erzwingen die Schaltreihenfolge $t_{j1}, t_{j2}, t_{j3}, t_{j1}$ etc.

Rückkopplung

5.4.2 Kombinieren

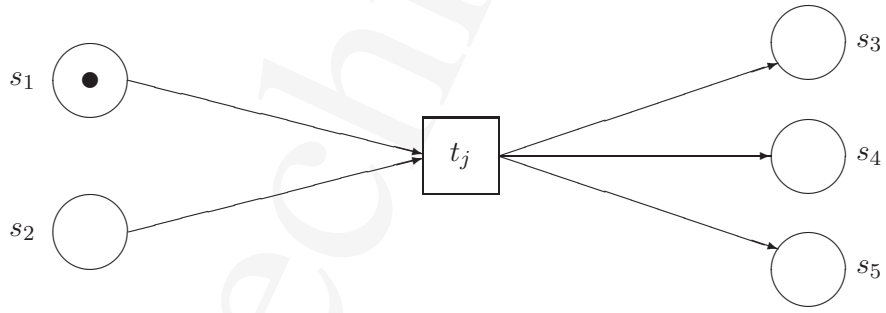
Analog zum Verfeinern kann es beim Kombinieren von zwei Teilnetzen Probleme („Nebeneffekte“) geben. Kombinieren wir zum Beispiel ein lebendiges, konfliktfreies S/T-Netz (Abbildung 5.34 Seite 178 Teilnetz I) mit einem anderen konfliktfreien S/T-Netz (Abbildung 5.34 Seite 178 Teilnetz II) dann kann im Gesamtnetz ein Konflikt auftreten (Fall 2). Kurz und plakativ formuliert: Auf einen Integrationstest können wir auch bei getesteten Teilnetzen nicht verzichten.

Integration

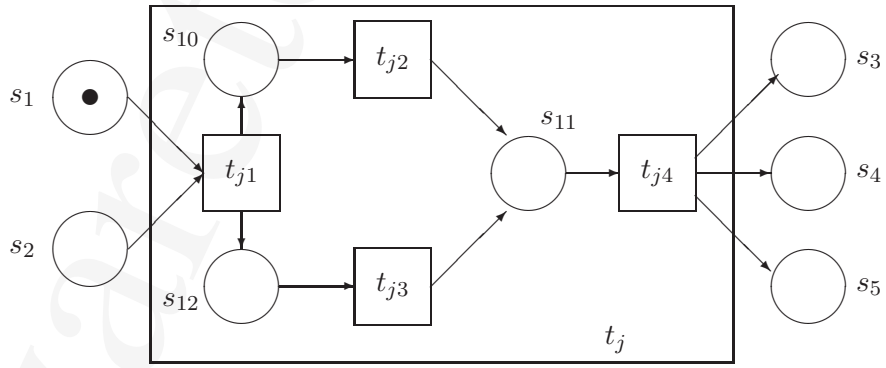
5.4.3 Beispiel: „Versionsmanagement“

- IST-Aufnahme: Ein Team produziert Software. Dabei kann ein Entwicklungsergebnis (zum Beispiel ein Modul) in verschiedenen Versionen existieren. Die Versionen entstehen zwangsläufig im Zusammenhang mit der notwendigen Fortentwicklung, Wartung und Fehlerbehebung. So können zum Beispiel zu einem Zeitpunkt

Grobes Netz:



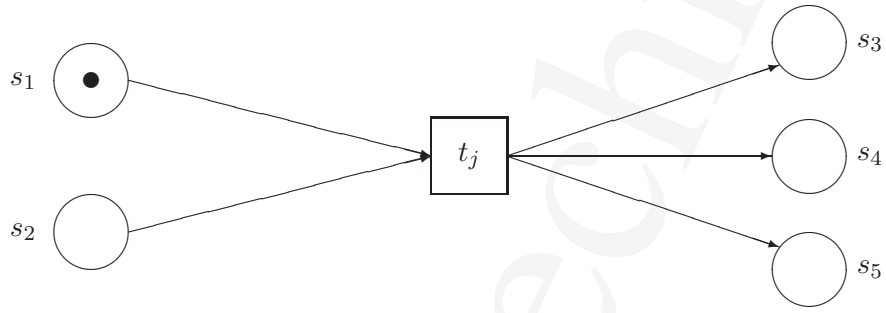
Verfeinertes Netz:



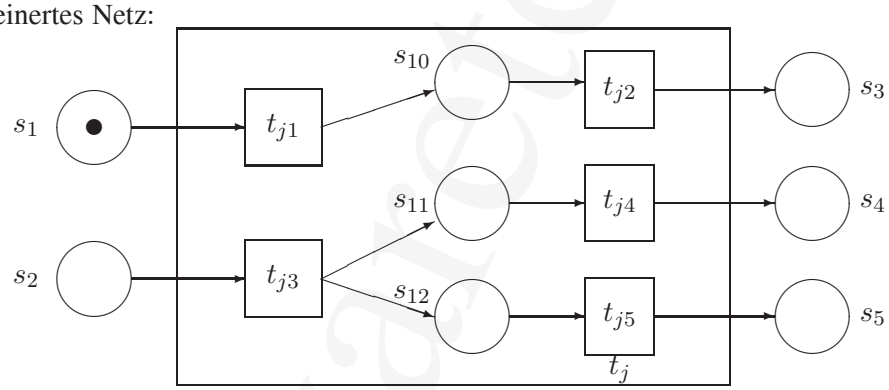
Legende:
t_j und t_{j1} haben keine Konzession.

Abbildung 5.31: Beispiel: Verfeinern der Transition t_j

Grobes Netz:



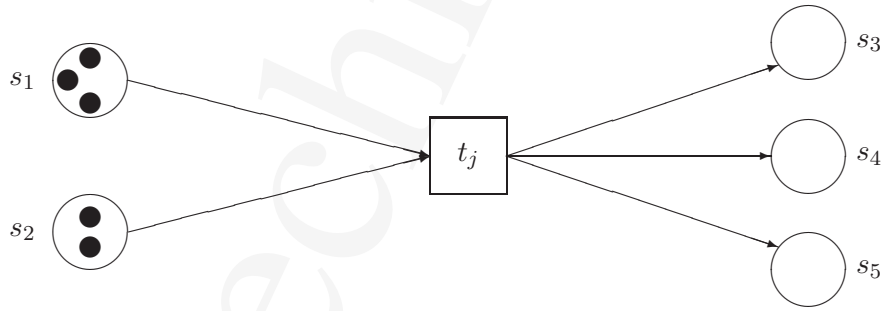
Verfeinertes Netz:



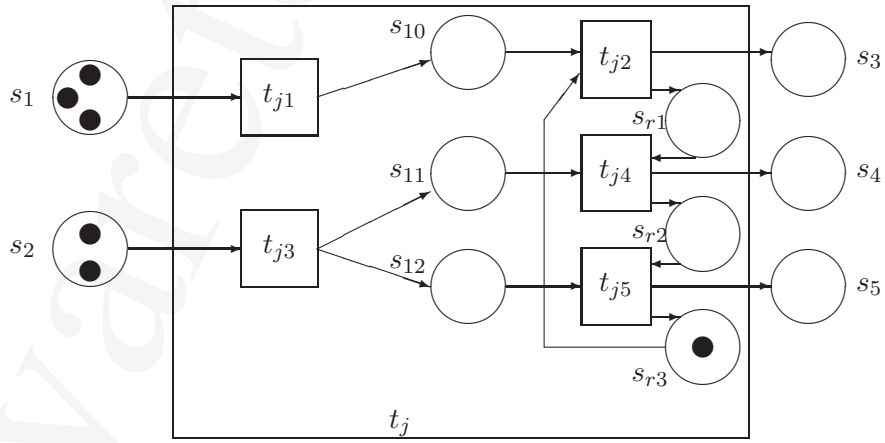
Legende:
t_j hat keine Konzession während t_{j1} Konzession hat.

Abbildung 5.32: Beispiel: Kein „markengerechtes“ Verfeinern für t_j

Grobes Netz:

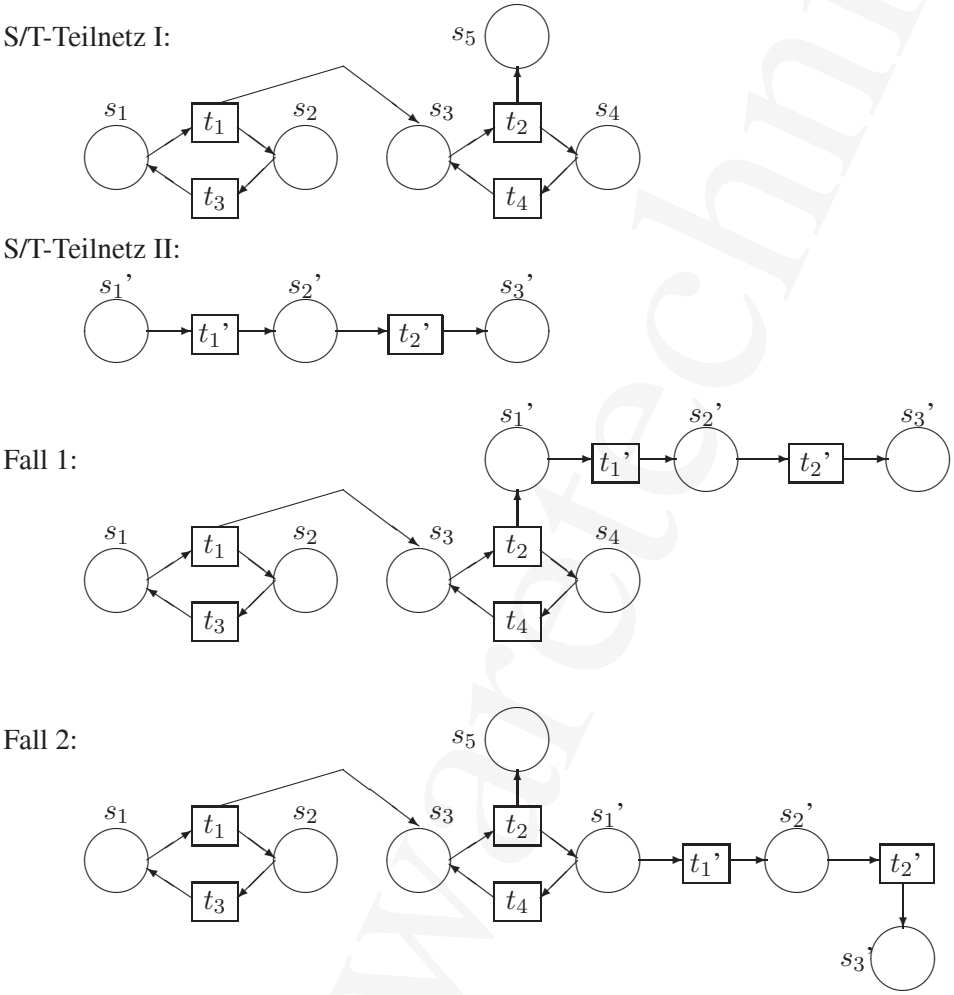


Verfeinertes Netz:



Legende:
Mehrfaches Schalten von t_{j1}.

Abbildung 5.33: Beispiel: Rückkopplung beim Verfeinern von t_j



Legende:
Fall 1: s_5 durch s_1' ersetzt.
Fall 2: s_4 durch s_1' ersetzt.

Abbildung 5.34: Beispiel: Kombinieren von Teilnetzen

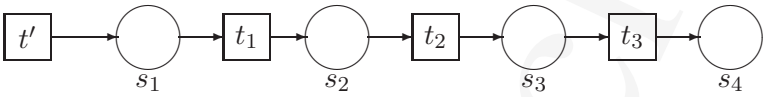
sich Versionen sowohl in der Produktion, in der Fehlerbehandlung und in der Wartung befinden. Aufgabe ist ein zweckmäßiges „Versionsmanagement“ (kurz: VM).

- VM-Schwachstellenanalyse: Die Änderungen im Rahmen der Fortentwicklung und Wartung passen nicht zu den Änderungen aufgrund der Fehlerbeseitung. Es kommt zu Inkonsistenzen zwischen Wartung und Fehlerbehandlung.

- VM-SOLL-Konzept und -Verbesserung: Es ist zu gewährleisten, daß von demselben Entwicklungsergebnis maximal zwei Versionen existieren. Die Bearbeitung dieser Versionen verläuft koordiniert, wobei die Fehlerbehandlung Priorität hat.

- VM-Modellentwicklung: Zur Darstellung einer Lösung nutzen wir ein Bedingungs-/Ereignis-System (bool-Petri-Netz). Das Schalten erfordert erfüllte Vorbedingungen und nicht erfüllte Nachbedingungen (vgl. Abschnitt 5.1.1 Seite 129).

1. Im ersten Schritt entwerfen wir die zwei Teilnetze:
   - Ergebniserarbeitung (Abbildung 5.35 Seite 180) und
   - Fehlerbehandlung (Abbildung 5.36 Seite 181).
2. Im zweiten Schritt kombinieren wir die Teilnetze zu einem Netz. Ansatzpunkt ist die gemeinsame Stelle: Produktionsübernahme erfolgt (Abbildung 5.37 Seite 182).
3. Im dritten Schritt führen wir eine Bearbeitungsfreigabe ein. Damit verdeutlichen wir, daß eine Archivierung erfolgt sein muß, bevor die nächste Version entstehen kann. Die Marke auf der Eingangsbedingung Bearbeitung erlaubt kann entweder das Netz Ergebnisbearbeitung oder das Netz Fehlerbehebung aktivieren (Abbildung 5.38 Seite 183).
4. Ist ein Ergebnis in Bearbeitung, das heißt, es entsteht eine neue Version neben der archivierten, und ist ein Fehler zu beheben, dann soll die Fehlerbehandlung Priorität haben. In Abbildung5.38 Seite 183 sind dann die Stellen s_{10} und s_2 markiert, während die Stelle s_{20} keine Marke mehr hat. Da s_{20} nicht markiert ist, kann t_{10} nicht schalten. Die Fehlerbehebung kann nicht vollzogen werden. Wir führen daher eine Bearbeitungssperre ein, die das Übernehmen des Ergebnisses in die Produktion stoppt und erst den Fehler in der alten Version behebt. Ist dies erfolgt, dann

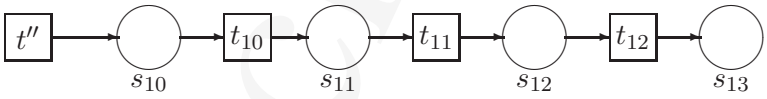


Legende:
Bool-Petri-Netz

t' $\equiv$ Ereignis tritt ein – generiert M_0-Teilmarkierung

$1 \leq$ Identifier-Nummer < 10
s_1 $\equiv$ Ergebnis ist zu bearbeiten
t_1 $\equiv$ Ergebnisbearbeitung vorbereiten
s_2 $\equiv$ Ergebnis in Bearbeitung
t_2 $\equiv$ Ergebnis in Produktion übernehmen
s_3 $\equiv$ Produktionsübernahme erfolgt
t_3 $\equiv$ Archivierung durchführen
s_4 $\equiv$ Archiviert

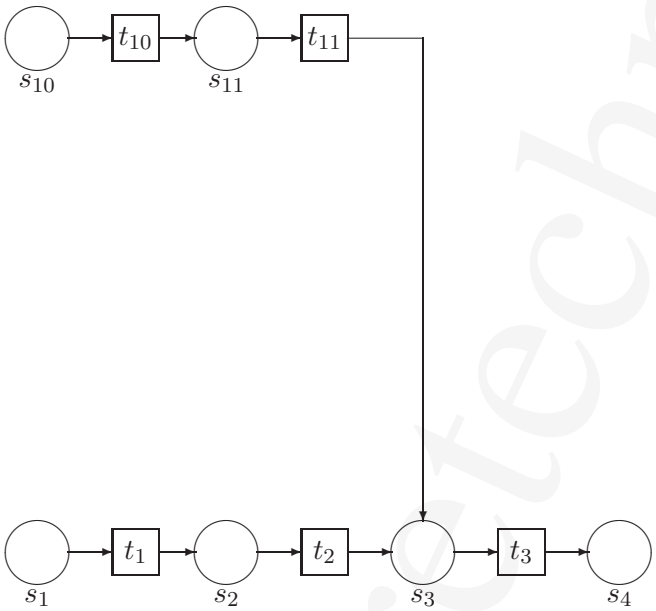
Abbildung 5.35: Teilnetz I: Ergebniserarbeitung



Legende:
Bool-Petri-Netz

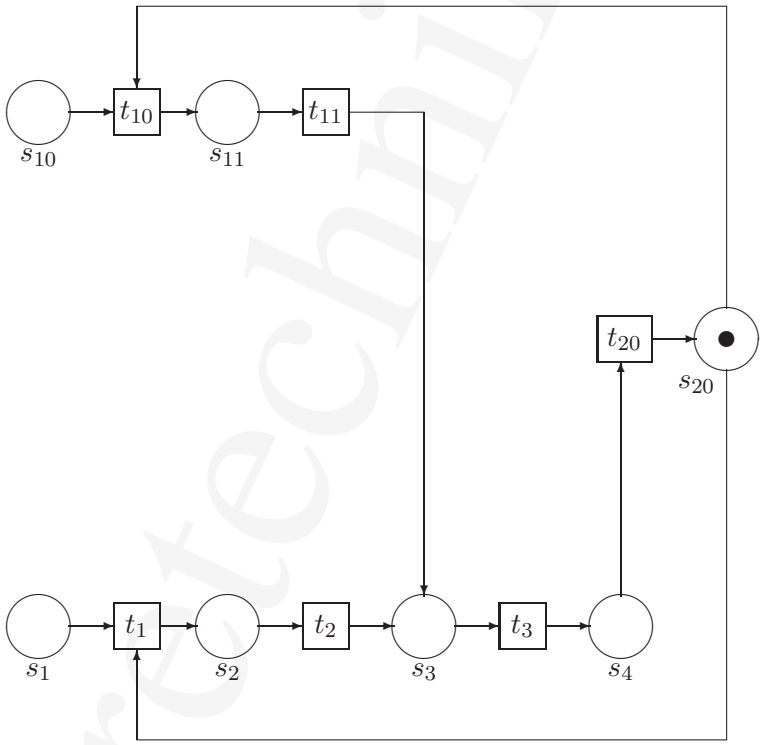
t'' $\equiv$ Ereignis tritt ein – generiert M_0-Teilmarkierung
$10 \leq$ Identifier-Nummer < 20
s_{10} $\equiv$ Fehler ist zu beheben
t_{10} $\equiv$ Fehlerbehebung vorbereiten
s_{11} $\equiv$ Ergebnis in Fehlerbehebung
t_{11} $\equiv$ Ergebnis in Produktion übernehmen
s_{12} $\equiv$ Produktionsübernahme erfolgt
t_{12} $\equiv$ Archivierung durchführen
s_{13} $\equiv$ Archiviert

Abbildung 5.36: Teilnetz II: Fehlerbehebung



Legende:
Bool-Petri-Netz
vgl. Abbildung 5.35 und Abbildung 5.36
$s_{12} = s_3$

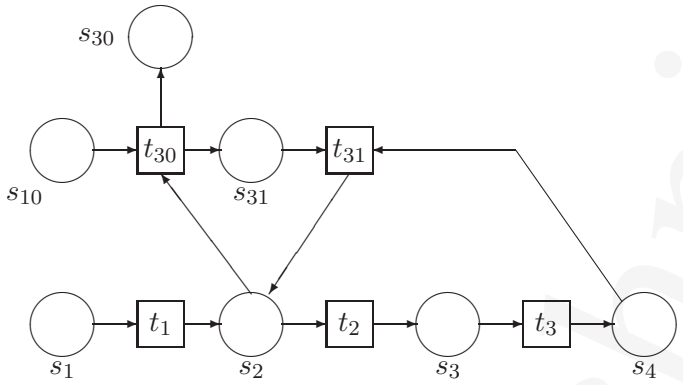
Abbildung 5.37: 1. Netzentwurf: Versionsmanagement



Legende:
Bool-Petri-Netz
vgl. Abbildung 5.35 und Abbildung 5.36

$20 \leq$ Identifier-Nummer < 30
s_{20} $\equiv$ Bearbeitung erlaubt
t_{20} $\equiv$ Bearbeitung freigegeben

Abbildung 5.38: 2. Netzentwurf: Versionsmanagement



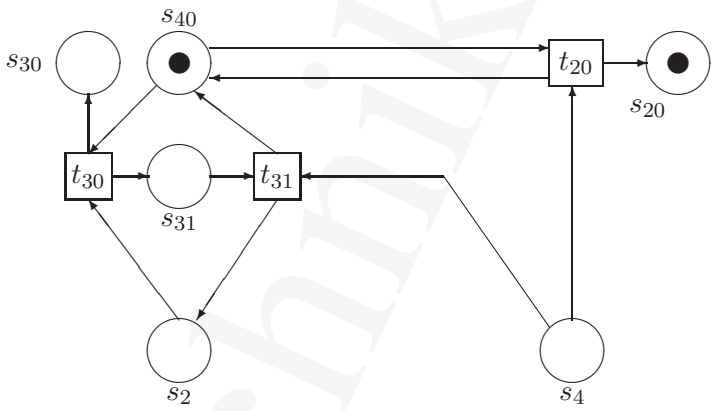
Legende:
Bool-Petri-Netz
vgl. Abbildung 5.35 und Abbildung 5.36

$30 \leq$ Identifier-Nummer < 40
s_{30} $\equiv$ Bearbeitung gesperrt
t_{30} $\equiv$ Bearbeitung sperren
s_{31} $\equiv$ Bearbeitungssperre aktiv
t_{31} $\equiv$ Korrektur bekannt gegeben

Abbildung 5.39: Netzverfeinerung: Priorität für Fehlerbehebung

wird die vollzogene Fehlerkorrektur bekanntgegeben, so daß sie bei der Ergebnisbearbeitung berücksichtigt werden kann. Die Bearbeitung ist so lange zu sperren, bis die Korrektur bekannt gegeben wurde. Dies setzt die Archivierung der korrigierten Version voraus. Abbildung 5.39 Seite 184 zeigt diesen Netzausschnitt.

5. Bei unseren bisherigen Überlegungen kann nach der Archivierung, das heißt, ist s_4 markiert, die Bearbeitung freigegeben werden. Dann hat t_{20} Konzession, so daß eine neue Bearbeitung erlaubt würde, und s_{20} eine Marke erhält. Zunächst ist jedoch die Korrektur bekannt zu geben, das heißt, es soll erst t_{31} schalten.



Legende:
Bool-Petri-Netz
vgl. Abbildung 5.35 Seite 180 und Abbildung 5.36 Seite 181
$40 \leq$ Identifier-Nummer < 50
$s_{40} \equiv$ Bearbeitungssperre inaktiv

Abbildung 5.40: Netzverfeinerung: Korrekturbekanntgabe erzwingen

Wir bauen daher für t_{20} eine Vorbedingung s_{40} ein, die durch das Schalten von t_{31} eine Marke erhält. Die Stelle s_{40} ist das Komplement zur Stelle Bearbeitungssperre aktiv (Abbildung 5.40 Seite 185).

6. Die Stelle s_{30} verdeutlicht, daß wir die Ergebnisbearbeitung gesperrt haben und mit der Fehlerbehandlung beginnen können. Intuitiv erscheint eine Kante von s_{30} nach t_{10} angebracht. Die Kante wäre jedoch falsch: Damit würde s_{30} zur Vorbedingung von t_{10}, also auch für den Fall, daß nur ein Fehler zu beheben ist und keine betroffene Version gerade bearbeitet wird. Wir vermeiden dies

Bool-Petri-Netz

s_1	$\equiv$ Ergebnis ist zu bearbeiten
t_1	$\equiv$ Ergebnisbearbeitung vorbereiten
s_2	$\equiv$ Ergebnis in Bearbeitung
t_2	$\equiv$ Ergebnis in Produktion übernehmen
s_3	$\equiv$ Produktionsübernahme erfolgt
t_3	$\equiv$ Archivierung durchführen
s_4	$\equiv$ Archiviert
s_{10}	$\equiv$ Fehler ist zu beheben
t_{10}	$\equiv$ Fehlerbehebung vorbereiten
s_{11}	$\equiv$ Ergebnis in Fehlerbehebung
t_{11}	$\equiv$ Ergebnis in Produktion übernehmen
s_{20}	$\equiv$ Bearbeitung erlaubt
t_{20}	$\equiv$ Bearbeitung freigeben
s_{30}	$\equiv$ Bearbeitung gesperrt
t_{30}	$\equiv$ Bearbeitung sperren
s_{31}	$\equiv$ Bearbeitungssperre aktiv
t_{31}	$\equiv$ Korrektur bekannt gegeben
s_{40}	$\equiv$ Bearbeitungssperre inaktiv
t_{40}	$\equiv$ Fehlerbehebung vorbereiten

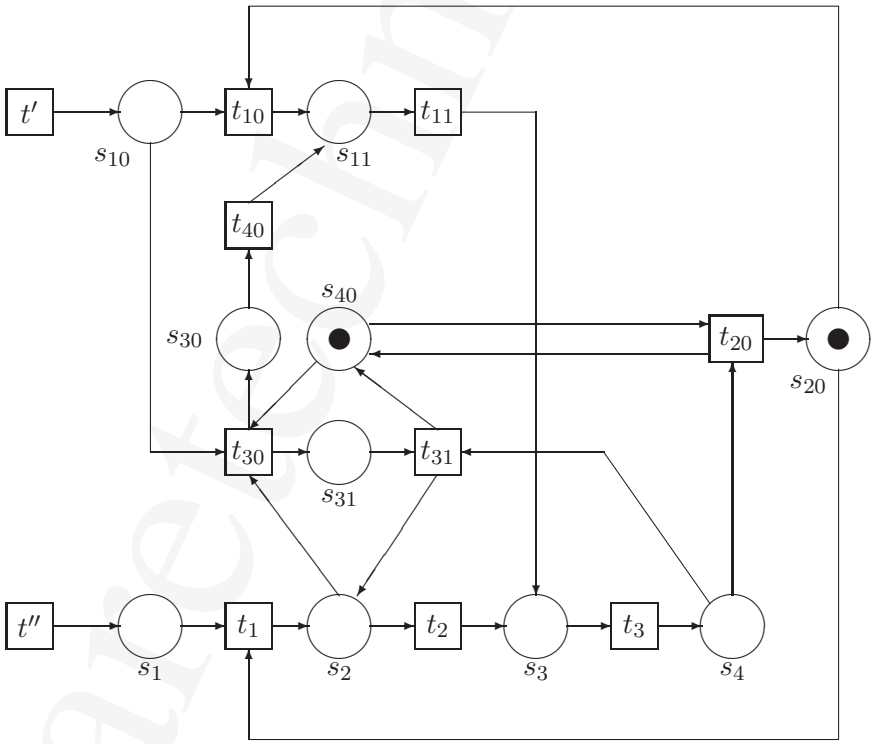
Tabelle 5.8: Legende zum Versionsmanagement

durch Duplizieren der Transition t_{10}. Die entstandene Transition t_{40} hat als Nachbereich die Stelle s_{11}. Abbildung 5.41 Seite 187 zeigt unser Modell.

5.4.4 Beispiel: „Projektplan“

Wenn es um die Darstellung der zeitlichen Abläufe eines Projektes geht, dann empfiehlt es sich, einen Netzplan oder zumindest ein Balkendiagramm aufzustellen. Im Mittelpunkt stehen die Schätzungen von optimistischen, normalen und pessimistischen Bearbeitungsdauern, die benötigten Kapazitäten und die darauf aufbauende Berechnung des *kritischen Pfades*. Späteste und früheste Termine sowie Pufferzeiten sind mittels Netzplantechnik bestimmbar. Solche Netze haben im Gegensatz zu Petri-Netzen im allgemeinen nur einen Typ von Knoten. Bei Petri-Netzen unterscheiden wir zwei Typen von Knoten: Stellen und Transitionen, Bedingungen und Ereignisse oder Prädikate und Transitionen.

Sind nur die Arbeitsschritte und die dabei entstehenden Dokumente zu benennen und die Abhängigkeit der Dokumente untereinander zu verdeutlichen, dann kann auch ein bool-Petri-Netz genutzt werden. Dieses Netz gibt dann nur einen Überblick über die logischen Abhängikeiten.



Legende:
vgl. Tabelle 5.8 Seite 186.

Abbildung 5.41: Gesamtnetz: Versionsmanagement

Bool-Petri-Netz

t_1	≡ SWTVDA-Schnittstellen ermitteln
t_2	≡ SWTVDA-Entwicklungsmethodik ermitteln
t_3	≡ Zielsetzung ermitteln
t_4	≡ SWTVDA-Informationsstruktur ermitteln
t_5	≡ DAKEU-Schnittstellen ermitteln
t_6	≡ DAKEU/SWTVDA-Entwicklungsabschnitte ermitteln
t_7	≡ DAKEU/SWTVDA-Informationsstruktur ermitteln
t_8	≡ Technische Kompatibilität ermitteln
t_9	≡ Einsatzbereiche von SWTVDA in DAKEU ermitteln
t_{10}	≡ Informationelle Kompatibilität ermitteln
t_{11}	≡ Kompatibilitätsbericht zusammenstellen
t_{12}	≡ Weitere Vorgehensweise abstimmen
t_{13}	≡ Weitere Vorgehensweise planen
t_{14}	≡ Projekt einstellen
s_1	≡ SWTVDA-Schnittstellen-Spezifikation
s_2	≡ SWTVDA-Entwicklungsmethodik-Bericht
s_3	≡ SWTVDA-Informationsstruktur-Bericht
s_4	≡ DAKEU-Schnittstellen-Spezifikation
s_5	≡ DAKEU/SWTVDA-Entwicklungsabschnitte-Bericht
s_6	≡ Zielsetzungs-Bericht
s_7	≡ DAKEU/SWTVDA-Informationsstruktur-Bericht
s_8	≡ Technische Kompatibilitäts-Bericht
s_9	≡ Bericht: Einsatzbereiche von SWTVDA in DAKEU
s_{10}	≡ Bericht: Informationelle Kompatibilität
s_{11}	≡ Kompatibilitätsbericht
s_{12}	≡ Abstimmungsergebnis
s_{13}	≡ Projektplan: Phase Systemplanung
s_{14}	≡ Einstellungsbegründung

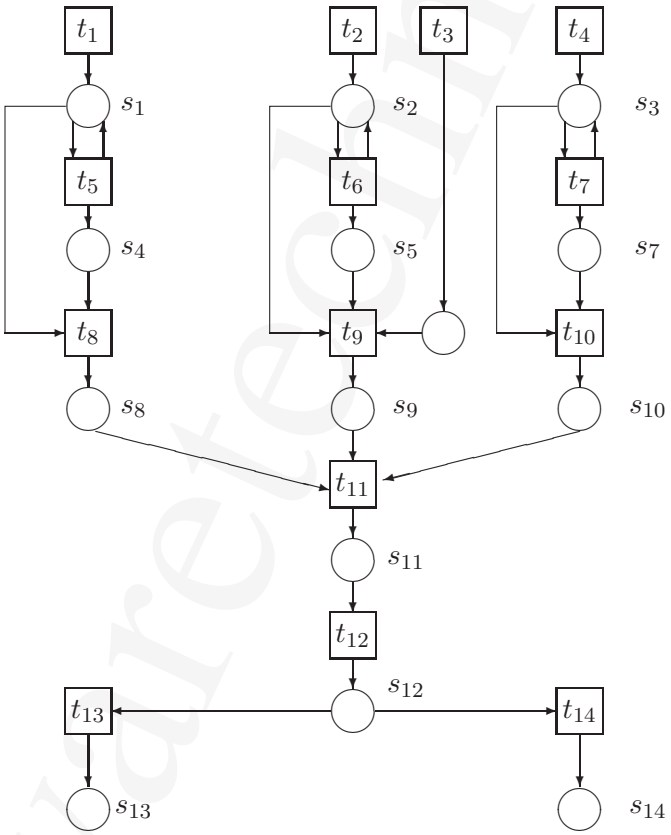
Tabelle 5.9: Legende zum Projektplan

Es kann daher als Studie für die Erstellung eines zeitlich-orientierten Netzplanes dienen.

Wir nehmen als Beispiel an, daß ein Bericht zu erstellen sei, aus dem ersichtlich ist, ob ein bisher genutztes Softwarewerkzeug[17] in eine moderne Entwicklungsumgebung[18] paßt. Abbildung 5.42 Seite 189 zeigt in Verbindung mit Tabelle 5.9 Seite 188 einen solchen Projektplan.

[17] Zum Beispiel SWTVDA von der Firma SWT: Vorgangsorientierte Dialoganwendungen.

[18] Zum Beispiel DAKEU [13].



Legende:
Vgl. Tabelle 5.9 Seite 188.

Abbildung 5.42: Netz: Projektplan

5.5 Netzanalyse

Mit Hilfe der Netztheorie können Eigenschaften eines vorgegebenen Netzes nachgewiesen werden. Wir können zum Beispiel für unser System TOPI (vgl. Tabelle 5.6 Seite 159) folgende bedeutsame Zusicherungen überprüfen:

- Z_1: Die Passagierliste kann nicht überbucht werden.
- Z_2: Ein Kunde wird nur dann in die Warteliste aufgenommen, wenn die Passagierliste voll ist.
- Z_3: Nur wenn die Warteliste leer ist, wird ein Kunde in die Passagierliste aufgenommen; stornierte Plätze bekommen bevorzugt Kunden der Warteliste.

Es stellen sich damit die Fragen nach der Erreichbarkeit (vgl. Tabelle 5.1 Seite 139), Lebendigkeit und der Beschränktheit dieses Netzes. Bedeutsam sind die Zusammenhänge zwischen P (Passagierliste), k (Zahl reservierte Plätze), W (Warteliste) und r (öffeneSStornierungen). Da die Nachrichten an den Kunden darauf keinen Einfluß haben, konzentrieren wir uns auf den für diese Fragen relevanten Netzausschnitt (vgl. Abbildung 5.43 Seite 192).

5.5.1 Relevanter Netzausschnitt

Erläuterung der relevanten Transitionen $t_1, \ldots, t_6$ (Abbildung 5.43 Seite 192):

- Die Transition t_1 stellt die Aufnahme eines Kunden i in die Passagierliste P dar. Sie erfolgt nur unter der Bedingung, daß $k < K$ ist, wie die Eingangstelle von t_1 dokumentiert.
- Die Transition t_2 bildet die Aufnahme von i in die Warteliste ab. Dies setzt voraus, daß die Zahl der reservierten Plätze k größer als die Kapazität K der Passagierliste P ist. Es gilt $k \geq K$. Da die Aufnahme von i in W keinen Einfluß auf k hat, also die Bedingung weiterhin wahr sein soll, geben wir beim Schalten von t_2 wieder eine Marke zurück. Das Einspeichern in W beeinflußt P nicht. Für die Eingangsstelle von t_1 ist keine Schleife zu konstruieren, da beim Schalten der Wert von k verändert wird und damit die Bedingung $k \leq K$ nicht mehr zutreffen muß.

- Die Transition t_3 entfernt den ersten Kunden auf der Warteliste und bucht diesen in die Passagierliste. Dazu sind zwei Bedingungen Voraussetzung: Die Warteliste darf nicht leer sein, und es gibt stornierte Plätze in der Passagierliste. Es gilt $r > 0$.

- Die Transition t_4 bildet den Fall ab, daß es keine Einträge in der Warteliste gibt, aber stornierte Buchungen in der Passagierliste sind noch freizugeben.

- Die Transitionen t_5 und t_6 stellen Stornierungen dar. Bei t_5 ist der Kunde in der Passagierliste eingetragen, bei t_6 in der Warteliste.

Als ersten Schritt für eine formale Netzanalyse ersetzen wir die Anweisungen in den Transitionen durch zusätzliche Stellen und Flußrelationen. Zum Beispiel führen wir für die Anweisung „$i \rightarrow P$“ eine neue Stelle P zusammen mit Flußrelationen ein. Die Stelle P ist Ausgangsstelle von t_1. Statt „$i \rightarrow P$“ auszuführen, wandert jetzt eine Marke auf P. Die Marken von P bilden die gebuchten Plätze ab. Da bei t_3 mit „$x \rightarrow P$“ ein Kunde aus der Warteliste in die Passagierliste aufgenommen wird, ist ein zusätzlicher Pfeil von t_3 nach P erforderlich. Mit dieser Flußrelation entfällt die Anweisung „$x \rightarrow P$“ . Abbildung 5.44 Seite 193 zeigt diese Modifikation durch Einführung der neuen Stelle P.

neue Stellen

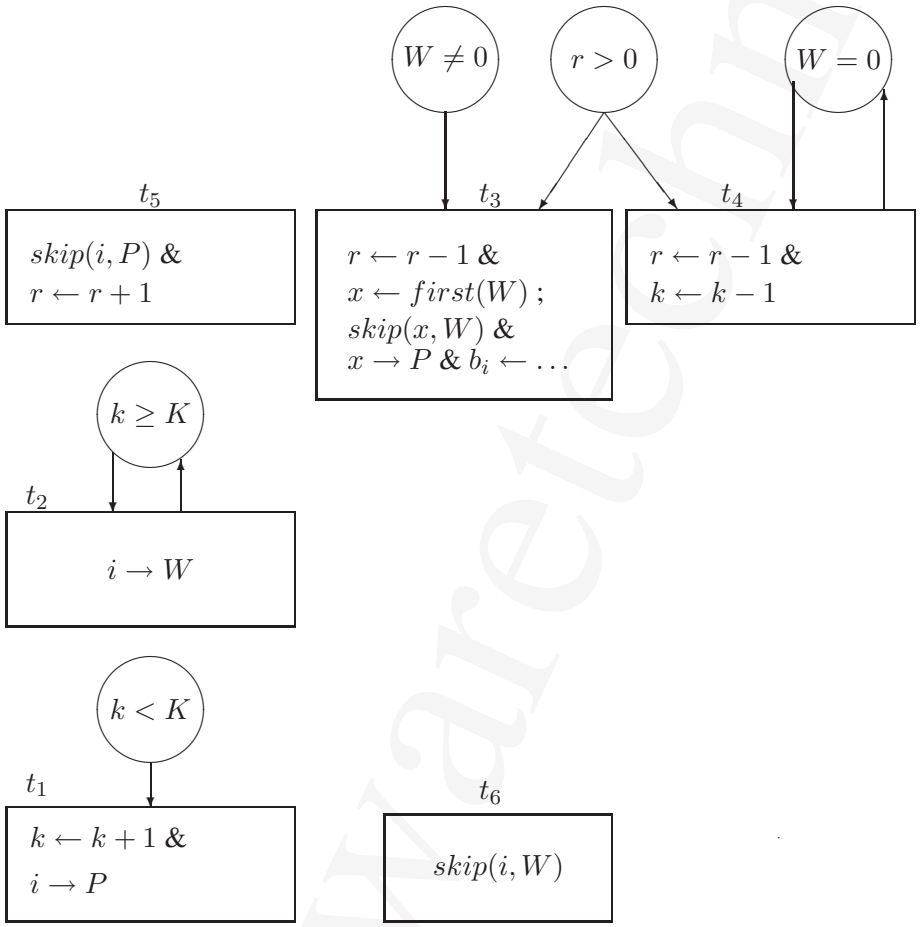
Analog zur Einführung der Stelle P konstruieren wir eine Stelle W für die Warteliste. Das Ergebnis zeigt Abbildung 5.45 Seite 194.

Auch die Warteliste hat eine begrenzte Kapazität, die wir hier mit L bezeichnen. Bei mehr als L Einträgen müssen weitere Buchungen von TOPI abgewiesen werden. Diese Kapazität L bilden wir mit dem Komplement von W ab, das heißt mit einer Stelle $\neg W$, die zunächst L Marken aufweist. Jede Transition, die eine Marke auf W setzt, muß als Eingangsstelle $\neg W$ haben, das heißt, sich eine Marke aus dem Vorrat L „holen“ . Die Bedingung $W = 0$ ist interpretierbar als keine Marke auf der Stelle W und L Marken auf der Stelle $\neg W$. Anders formuliert: Wenn wir L Marken von der Stelle $\neg W$ „holen“ können, dann ist keine Marke auf der Stelle W. Den Einbau der Stelle $\neg W$ in unser S/T-Netz zeigt Abbildung 5.46 Seite 195.

Komplement

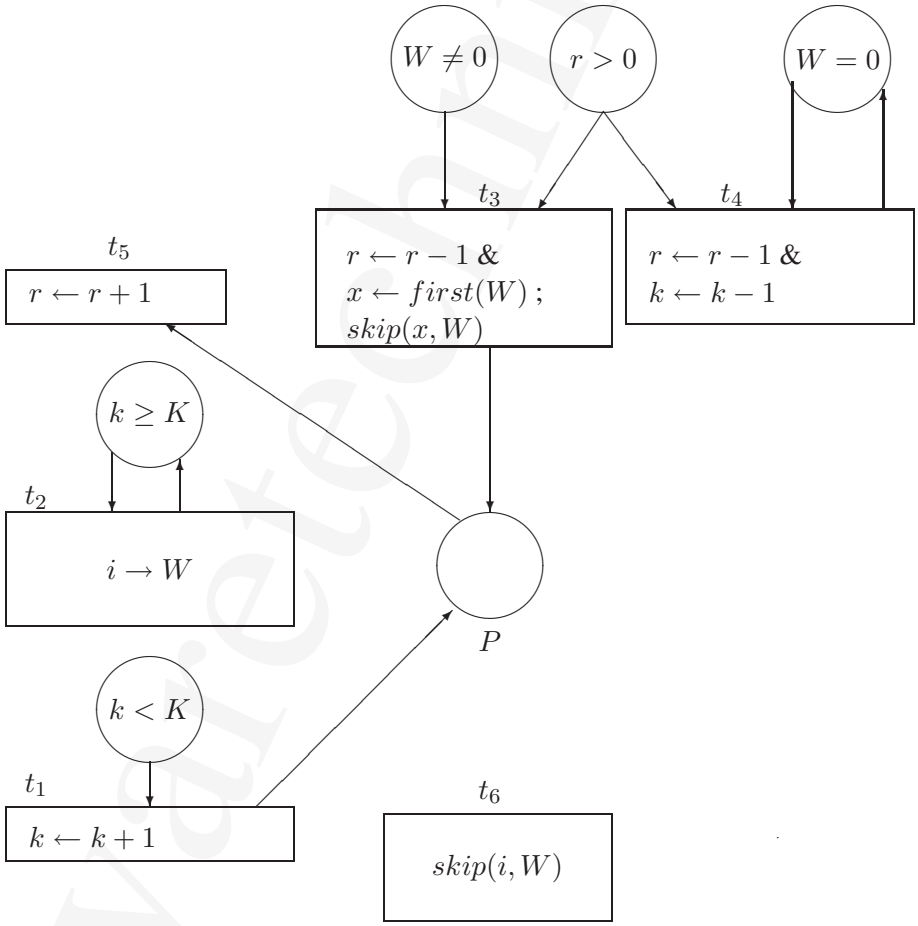
Dieses S/T-Netz weist noch Anweisungen mit r („Zahl der nicht rückgebuchten Stornierungen“) und k („Zahl der Reservierungen“) auf. Für beide sind entsprechend der für P bzw. W erläuterten Vorgehensweise neue Stellen einzubauen. Das Ergebnis zeigt Abbildung 5.47 Seite 196.

Diese Abbildung zeigt sogenannte Schlingen, wobei eine Stelle s_i Eingangs- und Ausgangsstelle derselben Transition t_j ist. Eine solche



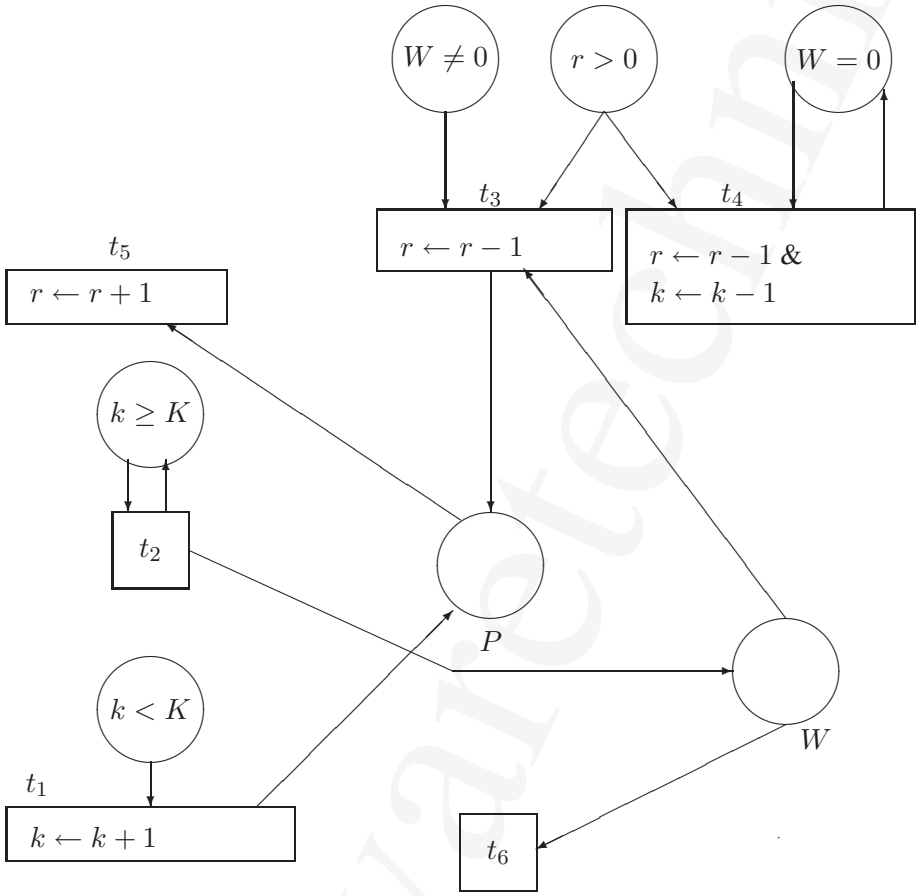
Legende:
vgl. Abbildungen 5.22, 5.23 und 5.24

Abbildung 5.43: Relevante Transitionen des TOPI-Netzes



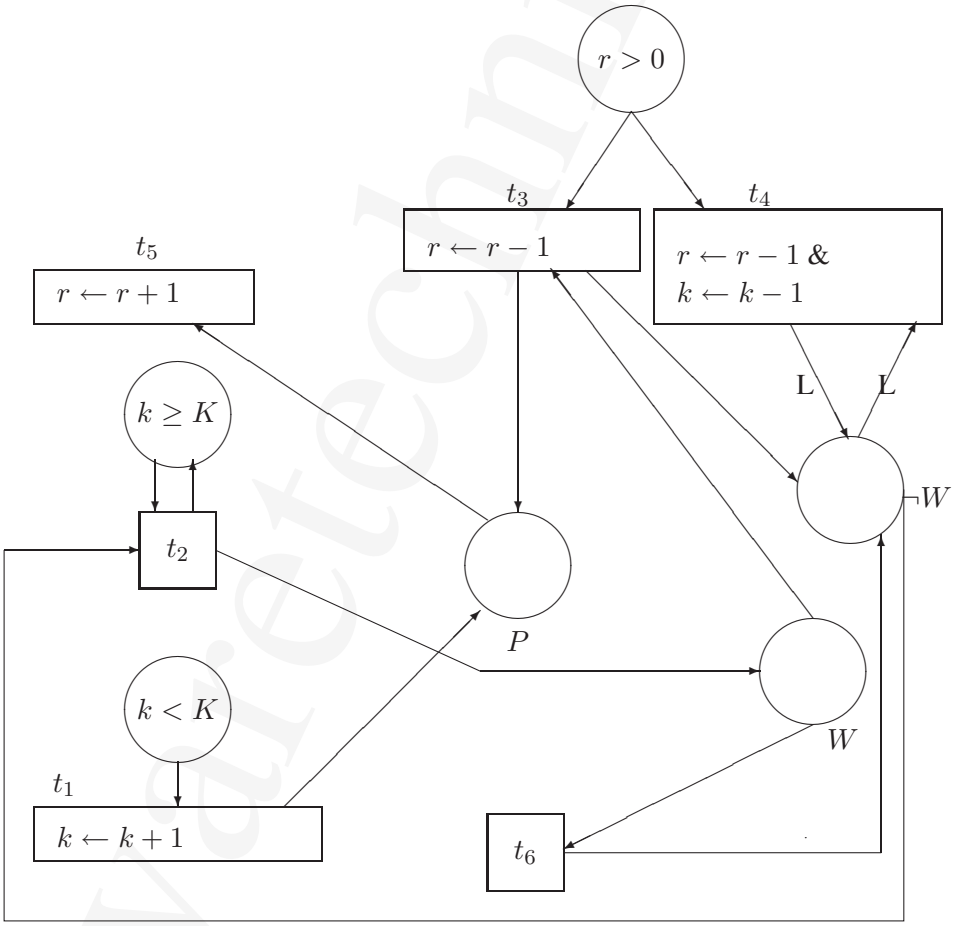
Legende:
vgl. Abbildung 5.43 Seite 192
Die aktuellen Einträge in der Passagierliste sind durch Marken der neuen Stelle P repräsentiert. Die betroffenen Anweisungen in den Transitionen entfallen.

Abbildung 5.44: Passagierliste als Stelle P abgebildet



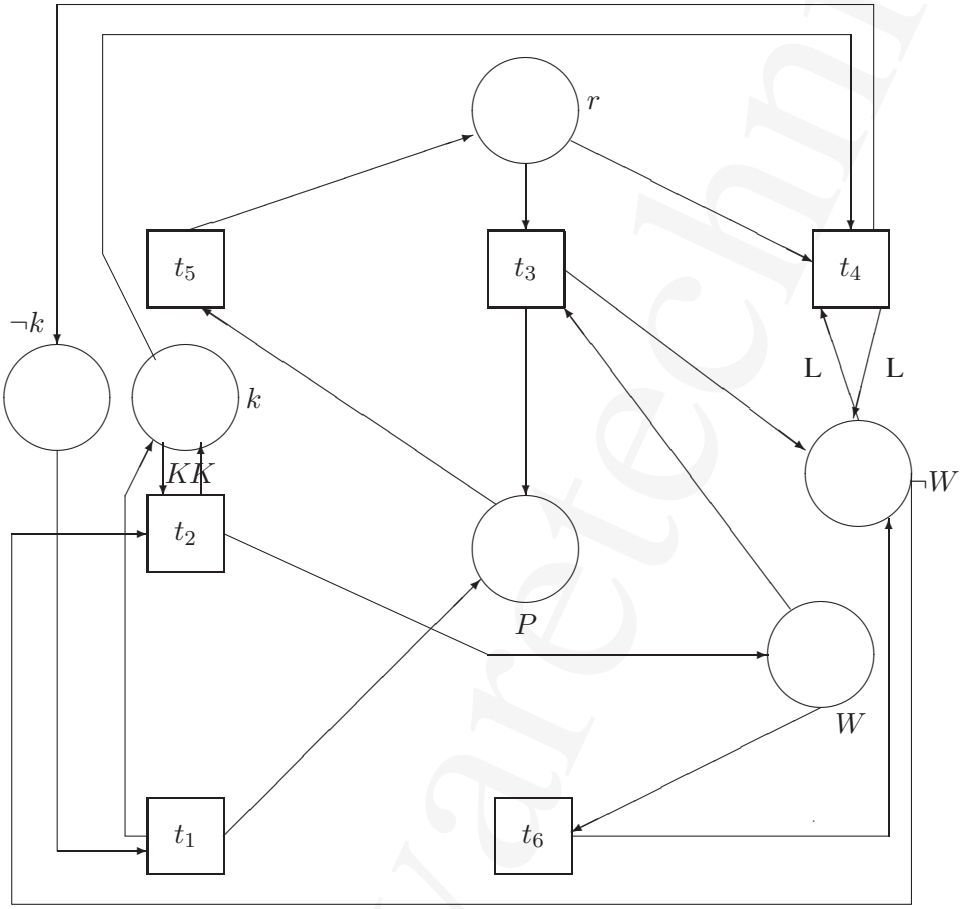
Legende:
Vgl. Abbildung 5.44 Seite 193.
Die aktuellen Einträge in der Warteliste sind durch Marken der neuen Stelle W repräsentiert. Die betroffenen Anweisungen in den Transitionen entfallen.

Abbildung 5.45: Warteliste als Stelle W abgebildet



Legende:
vgl. Abbildung 5.45 Seite 194.
Die Warteliste hat eine begrenzte Kapazität, hier L . Ist L erschöpft können keine Buchungen mehr akzeptiert werden. Dies wird dem Komplement $\neg W$ zur Stelle W abgebildet.

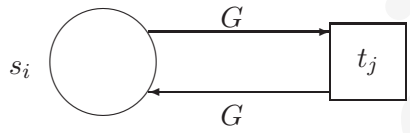
Abbildung 5.46: Komplement $\neg W$ zur Stelle W eingebaut



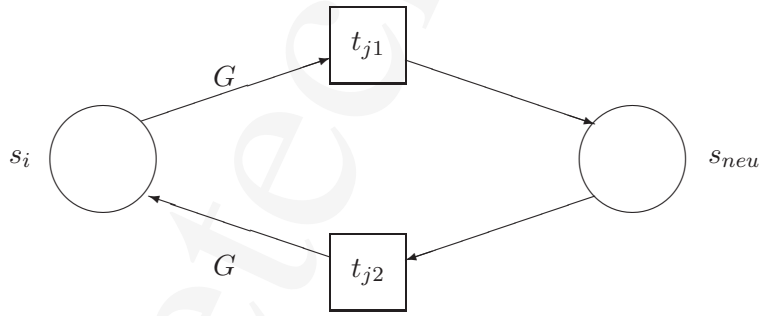
Legende:
Vgl. Abbildung 5.46 Seite 195.
Zum Ersetzen der Anweisungen mit r und k sind weitere Stellen mit der Anfangsmarkierungen $m_0(r) = 0$ und $m_0(k) = 0$ eingeführt. Zur Abbildung der Kapazität K ist das Komplement zu k mit der Anfangsmarkierung $m_0(k) = K$ eingeführt.

Abbildung 5.47: Stellen für r und k eingebaut

Schlinge:



Aufgeloeste Schlinge:



Legende:
$G \equiv$ Gewicht

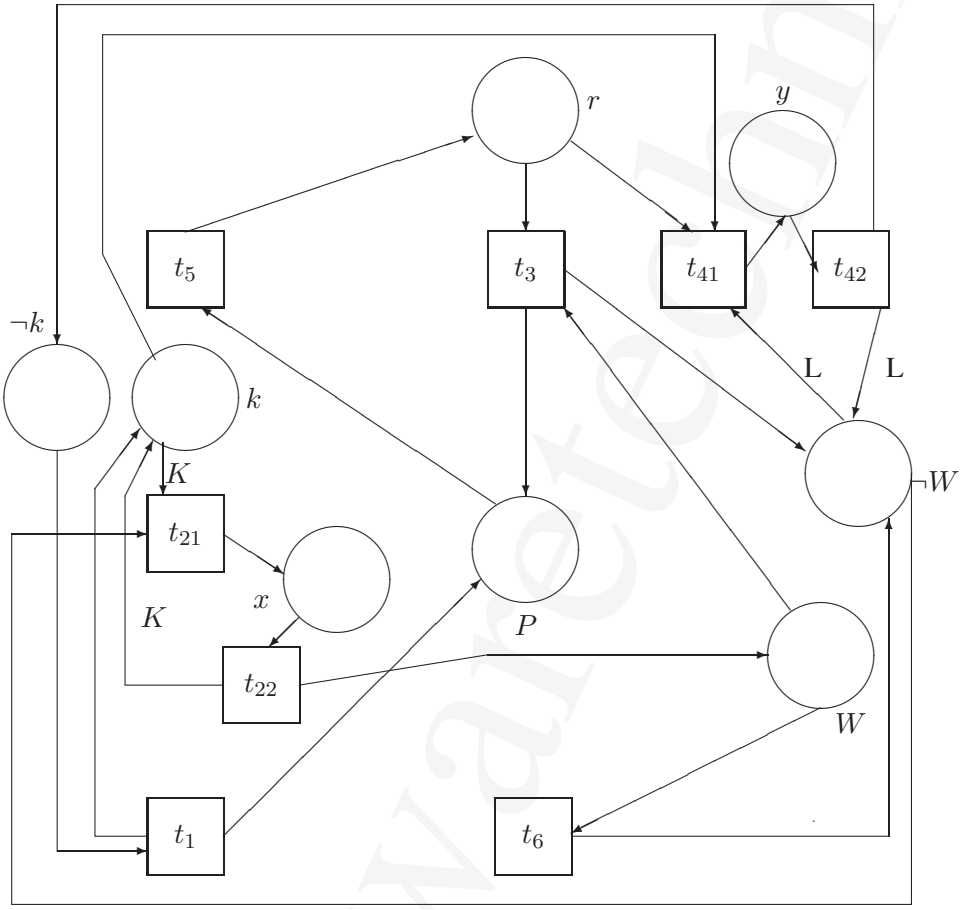
Abbildung 5.48: Auflösung einer Schlinge (Schleife)

Schlinge ist durch eine zusätzliche Stelle und eine Teilung der Transition auflösbar (vgl. Abbildung 5.48 Seite 197).

5.5.2 Invarianz

Abbildung 5.49 Seite 198 zeigt das vom Ausgangsentwurf (Abbildungen 5.22, 5.23 und 5.24) abgeleitete S/T-Netz für die formale Analyse der gewünschten Eigenschaften (hier: Zusicherungen Z_1, Z_2 und Z_3). Dazu suchen wir diejenigen Stellenkombinationen, bei denen sich die jeweilige Markengesamtzahl beim Schalten von Transitionen nicht verändert. Bleibt die Markengesamtzahl bei einer Menge von Stellen konstant, dann bezeichnen wir sie als Invariante des Systems. Der Vorteil solcher Invarianten liegt darin, daß mit ihnen Systemeigenschaften nachweisbar sind, die auf dem Verhältnis verschiedener Stellen untereinander beruhen. Für solche Nachweise sind Invarianten besonders nützlich, die nur positive Elemente enthalten; ihre Stellen werden als

konstante Markenmenge



Legende:
Vgl. Abbildung 5.47 Seite 196.
Schlingen mit neuen Stellen x und y aufgelöst.

Abbildung 5.49: S/T-Netz: Grundlage der formalen TOPI-Analyse

	t_1	t_{21}	t_{22}	t_3	t_{41}	t_{42}	t_5	t_6
k	1	$-K$	K		-1			
$\neg k$	-1					1		
W			1	-1				-1
$\neg W$		-1		1	$-L$	L		1
P	1			1			-1	
r				-1	-1		1	
x		1	-1					
y					1	-1		

Tabelle 5.10: Matrix der Markenveränderung

mit positiven S-Invarianten überdeckt bezeichnet.

Zum Auffinden solcher Invarianten drehen wir unsere bisherige Blickrichtung um. Wir betrachten nacheinander alle Transitionen, hier $t_1, t_{21}, t_{22}, \ldots, t_6$ und ermitteln die Veränderung der Markenzahl auf jeder Stelle, hier von $k, \neg k, W, \neg W, P, r, x$ und y.

Die Veränderungen der Marken auf einer Stelle halten wir in einer Matrix fest, wobei die Zeilen gleich die Stellen und die Spalten gleich die Transitionen abbilden. Eine solche Matrix bezeichnet man als Inzidenzmatrix oder auch als Akzidenzmatrix. Fließt beim Schalten von t_j eine Marke von einer Stelle s_i „ab“ , dann tragen wir im entsprechenden Matrixfeld m_{ij} den Wert -1 ein. Beim Zufluß weist das Matrixfeld den Wert +1 auf. Gibt es bei s_i keine Änderung ist der Wert 0, den wir aus Gründen der Übersichtlichkeit nicht vermerken. Hat die Flußrelation ein Gewicht, entspricht der Eintrag dieser Gewichtsangabe. Tabelle 5.10 zeigt die Matrix für das S/T-Netz von Abbildung 5.49.

Wenn sich die Markengesamtzahl einer Stellenmenge nicht ändert, dann entspricht sie der Markengesamtzahl der Anfangsmarkierung dieser Stellen. In unserem S/T-Netz bilden die Stellen $\neg k, P, r$ und y eine solche Invariante. Es gilt daher:

$$m(\neg k)+m(P)+m(r)+m(y) = m_0(\neg k)+m_0(P)+m_0(r)+m_0(y) = K \tag{5.3}$$

mit: $m(s) \equiv$ aktuelle Markierung der Stelle s
$m_0(s) \equiv$ Anfangsmarkierung der Stelle s

Wir notieren eine solche Stellenmenge als einen Vektor, der für jede Stelle des S/T-Netzes ausweist, ob sie zu dieser Stellenmenge gehört oder nicht. Betrachten wir die Stellen in der Reihenfolge: $k, \neg k, W, \neg W, P, r, x, y$

dann hat unser Vektor folgende Eintragungen:

$$i_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

Betrachten wir eine mögliche Markenmenge ebenfalls als Vektor M, dann gilt aufgrund der Invarianz:

$$\begin{aligned} &Anfangsmarkierung \\ &\quad \star Invariante \quad = \\ &\qquad\qquad Markierung\ nach\ geschalteten\ Transitionen \\ &\qquad\qquad \star Invariante \end{aligned} \tag{5.4}$$

Wir notieren bezogen auf die obige Invariante i_1 für eine durch Schaltungen entstandene Markierung M_1:

$$[M_0] * [i_1] = [M_1] * [i_1] \tag{5.5}$$

Durch Umformung der Gleichung 5.5 erhalten wir:

$$0 = [M_1] * [i_1] - [M_0] * [i_1] \tag{5.6}$$

Nehmen wir z.B an, daß $[M_1]$ durch das Schalten der Transition t_1 entstanden ist, dann gilt:

$$[M_1] = [M_0] + [Veraenderungen\ durch\ Schalten\ von\ t_1] \tag{5.7}$$

Die Veränderungen durch Schalten von t_1 stehen in der Spalte t_1 der obigen Matrix (vgl. Tabelle 5.10 Seite 199). Setzen wir Gleichung 5.7 in Gleichung 5.6 ein, dann gilt:

$$0 = ([M_0] + [Spalte\ t_1]) * [i_1] - [M_0] * [i_1] \tag{5.8}$$

Damit erhalten wir:

$$0 = [Spalte\ t_1] * [i_1] \tag{5.9}$$

Nehmen wir nun an, daß die Markierung $[M_1]$ nicht durch Schalten von t_1 sondern z.B von t_5 entstanden wäre, dann ist die Gleichung 5.9 wie folgt zu notieren:

$$0 = [Spalte\ t_5] * [i_1] \tag{5.10}$$

Für alle Transitionen unseres S/T-Netzes können wir dann kurz notieren:

$$0 = \text{Matrix Tabelle 5.10} * [i_1] \tag{5.11}$$

oder allgemein formuliert:

$$0 = [C] * [i_1] \text{ mit: } C \equiv Inzidenzmatrix \tag{5.12}$$

Da es mehr als einen Stellenvektor i_1 geben kann, bei dem die Markengesamtzahl invariant ist, können wir allgemein formulieren:

$$\begin{aligned} 0 &= [C] * [I] \\ \text{mit: } C &\equiv Inzidenzmatrix \\ I &\equiv Invarianten \end{aligned} \tag{5.13}$$

Formel 5.14 stellt ein Gleichungssystem dar, das zur Bestimmung der Invarianten dient. Mit Hilfe der (linearen) Algebra ist dieses Gleichungssystem (häufig) lösbar.

Tabelle 5.11 Seite 202 zeigt die Invarianten $i_1, \ldots, i_4$ und die Anfangsmarkierung unseres S/T-Netzes. Aufgrund der Gewichte der Flußrelationen, hier K und L, haben unsere Invarianten nicht nur die Werte 0 (weggelassen), +1 und -1. Im Sinne der Gleichung 5.5 können wir gemäß Tabelle 5.11 Seite 202 für diese Invarianten notieren:

$$\begin{aligned} i_1 &\equiv m(\neg k) + m(P) + m(r) + m(y) \\ &= m_0(\neg k) + m_0(P) + m_0(r) + m_0(y) \\ &= K \end{aligned} \tag{5.14}$$

$$\begin{aligned} i_2 &\equiv -m(k) + m(P) + m(r) - K * m(x) \\ &= -m_0(k) + m_0(P) + m_0(r) - K m_0(y) \\ &= 0 \end{aligned} \tag{5.15}$$

$$\begin{aligned} i_3 &\equiv m(k) + m(\neg k) + K * m(x) + m(y) \\ &= m_0(k) + m_0(\neg k) + K * m_0(x) + m_0(y) \\ &= K \end{aligned} \tag{5.16}$$

$$\begin{aligned} i_4 &\equiv m(W) + m(\neg W) + m(x) + L * m(y) \\ &= m_0(W) + m_0(\neg W) + m_0(x) + L * m_0(y) \\ &= L \end{aligned} \tag{5.17}$$

	t_1	t_{21}	t_{22}	t_3	t_{41}	t_{42}	t_5	t_6	i_1	i_2	i_3	i_4	m_0
k	1	$-K$	K		-1					-1	1		
$\neg k$	1					1			1		1		K
W			1	-1				-1				1	
$\neg W$		-1		1	$-L$	L		1				1	L
P	1			1			-1		1	1			
r				-1	-1		1		1	1			
x		1	-1							$-K$	K	1	
y					1	-1			1		1	L	

Legende:
Vgl. Abbildung 5.49 Seite 198.

Tabelle 5.11: Invarianten $i_1, \ldots, i_4$

Die Zusicherung Z_1 („Die Passagierliste kann nicht überbucht werden“) ist mit i_1 nachgewiesen, wenn man die Gleichung 5.14 nach $m(P)$ auflöst:

$$\begin{aligned} m(\neg k) + m(P) + m(r) + m(y) &= K \\ m(P) &= K - m(r) - m(\neg k) - m(y) \\ &< K \qquad (5.18) \\ &\text{Keine Überbuchung von P} \end{aligned}$$

Die Zusicherung Z_2 („Ein Kunde wird nur dann in die Warteliste aufgenommen, wenn die Passagierliste voll ist.“) bedingt, daß x genau dann markiert ist, wenn TOPI einen Buchungsauftrag mit der Aufnahme in die Warteliste beantwortet. Aus Gleichung 5.15 folgt dann:

$$\begin{aligned} -m(k) + m(P) + m(r) - K * m(x) &= 0 \\ m(p) + m(r) &= m(k) + K * m(x) \\ &= m(k) + K \\ &> K \qquad (5.19) \\ &\text{Passagierliste ausgeschöpft!} \end{aligned}$$

Die Zusicherung Z_3 („Nur wenn die Warteliste leer ist, wird ein Kunde in die Passagierliste aufgenommen; stornierte Plätze bekommen bevorzugt Kunden der Warteliste“) ist wie folgt nachgewiesen:

Aus Gleichung 5.19 folgt, daß bei einem Buchungsauftrag eine Aufnahme in die Warteliste eine Markierung m_1 bedingt, für die folgendes

gilt:

$$\begin{aligned} m_1(k) &= K \text{ vgl. 5.19} \\ m(k) + m(\neg k) + K * m(x) + m(y) &= K \end{aligned} \tag{5.20}$$

Da die Invariante i_3 auch den Fall $m_1(k)$ einschließt, gilt:

$$K + m_1(\neg k) + K * m_1(x) + m_1(\neg k) = K \tag{5.21}$$

Da in Gleichung 5.21 m_1 stets positiv ist, gilt:

$$m_1(\neg k) = 0 \tag{5.22}$$

Die Aktualisierungsroutine kann einzelne Plätze zur erneuten Reservierung freigeben und die Beantwortung eines Buchungsauftrages mit der Aufnahme in P (Schalten von t_1) und durch Markierung von $\neg k$ ermöglichen. Dies geschieht durch Schalten von t_{42} und bedingt, daß y dann markiert war (Markierung m_2).

$$m(W) + m(\neg W) + m(x) + L * m(y) = L \tag{5.23}$$

$$m_2(W) + m_2(\neg W) + m_2(x) + L = L \tag{5.24}$$

$$\text{damit folgt: } m_2(W) = 0 \tag{5.25}$$

Gleichung 5.25 zeigt das die Warteliste leer ist.

5.6 Fazit

Auch ohne einen mathematischen Definitionsversuch vermittelt das in Abschnitt 5.5 skizzierte Verfahren zum Nachweis von vorgegebenen Eigenschaften einen ersten Eindruck über Vor-/Nachteile und Aufwand von Petri-Netzen. Offensichtlich ist die Berechnung (zum Beispiel von Invarianten) bei komplexen Netzen nicht trivial. Der Systemanalytiker muß sich daher auf den jeweils relevanten Ausschnitt konzentrieren, um das Netz klein, überschaubar und mit vertretbarem Aufwand berechenbar zu halten.

Problemtransparenz

Unstrittig ermöglicht eine graphische Netzdarstellung bei wenigen Netzknoten eine bessere Problemsicht. Die Vorteile im Vergleich zu anderen Darstellungsmitteln wie zum Beispiel Entscheidungstabellen, Struktogrammen (Nassi/Shneiderman-Diagrammen), Ablaufplänen, Entscheidungsbäumen, HIPO, Jackson-Diagramme und vieles mehr liegen in der Handhabung nebenläufiger Prozesse.

Ein Vorteil von Netzen ist ihre mathematische Fundierung. Passend zur Problemstellung kann aus dem Netz-Grundtyp ein zweckmäßiger Netztyp entwickelt werden, ohne die Optionen der Netztheorie zu verlieren. Anpassen können wir zum Beispiel:

- die Art der Marken (zum Beispiel unterscheidbar: farbig, strukturiert, . . .)
- die Art der Beschriftung der Netzelemente (zum Beispiel Gewicht für Flußrelationen, Schaltausdrücke für Transitionen, . . .)
- die Art der Bewertung von Netzelementen (zum Beispiel Kapazitäten für Stellen, Prioritäten für Transitionen,. . .)
- die Schaltregel und
- die Schaltstrategie.

Mit der Hand können wir nur relativ kleine Netze zeichnen und berechnen. Mit rechnergestützten Produktionshilfen („tools“) ist die beherrschbare Knotenzahl wesentlich größer. Benötigt werden zumindest (vgl. [55]):

- Editor,
- Simulator,
- Analysator
- Reduktor und
- Ergebnisinterpreter

Der Editor ermöglicht die interaktive graphische Netzerstellung. Der Simulator präsentiert den Ablauf („Markenfluß“). Der Analysator nutzt die netztheoretischen Verfahren (berechnet zum Beispiel Invarianten), wobei ein Reduktor zur Vereinfachung des Netzes (zum Beispiel in Hinblick auf Invarianten) beiträgt. Der Ergebnisinterpreter („Expertensystem“ ?) zieht aus den nachgewiesenen Netzeigenschaften Schlußfolgerungen. Leider sind häufig nur Editoren und Simulatoren verfügbar.

Im Sinne einer abschließenden Daumenregel plakativ und holzschnittartig formuliert: Nutze Petri-Netze immer dann, wenn vermutbar ist, daß wirklich bedeutsame Eigenschaften sich auf wenige Knoten konzentrieren lassen.

5.7 Übungen

5.7.1 Aufgabe: Analyse von Netzeigenschaften

Netz I

Analysieren Sie das Netze in Abbildung 5.50 Seite 205.

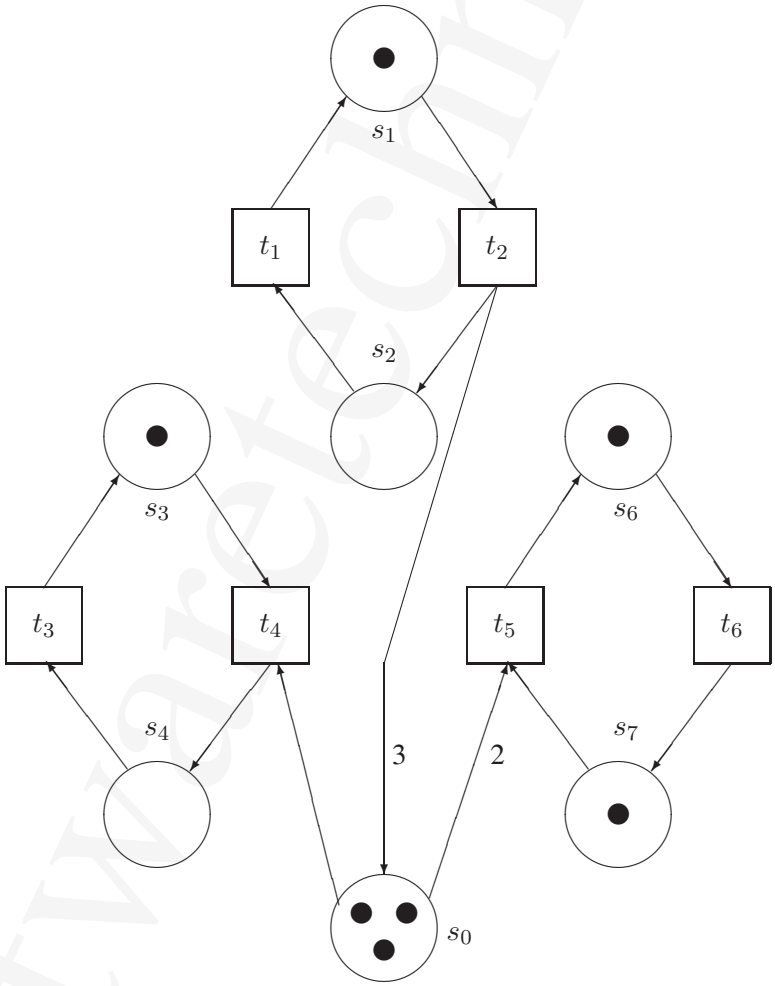


Abbildung 5.50: Netz I

Netz II

Analysieren Sie das Netze in Abbildung 5.51 Seite 207 bei folgenden Anfangsmarkierungen:

Fall 1: $M_0 \equiv s_3 + s_4 + s_7 + s_{10}$
Fall 2: $M_0 \equiv s_2 + s_4 + s_7 + s_{10}$
Lösungen:
Fall 1: Schaltfolge: $t_3, t_1, t_5, t_2, t_4, \ldots$
Man erreicht wieder M_0.
Fall 2: Schaltfolge: t_1 danach „tot“ .

Werkzeugmagazin

Verfeinern Sie die Transitionen des Netzes in Abbildung 5.52 Seite 208. Dokumentieren Sie Vorgänge in der Werkzeugausgabe und Werkzeugrückgabe.

5.7.2 Aufgabe: Briefbeförderung

Abbildung 5.53 Seite 208 zeigt ein Netz, bei dem es um die Beförderungsstationen eines Briefes geht.[19] Verfeinern Sie diese Kette aus zwei Stellen und einer Transition. Die erste bzw. letzte Transition Ihres verfeinerten Netzes möge dabei zum Beispiel bedeuten: „Einwurf in den Post- bzw. Hausbriefkasten“ . Fassen Sie anschließend Ihre Verfeinerung wieder zu gröberen Transitionen bzw. Stellen zusammen. Abbildung 5.54 Seite 209 zeigt eine mögliche Lösung.

5.7.3 Aufgabe: Warenlager

Abbildung 5.55 Seite 210 zeigt ein Netz, bei dem es um die Abfertigung von Lastkraftwagen bei einem Warenlager geht. Auf den ersten Blick können mehrere Lastkraftwagen abgefertigt werden. Jeder Lastkraftwagen wird durch eine Marke abgebildet. Bei genauerer Analyse zeigt sich folgendes Problem: Wenn zum Beispiel jeweils eine Marke auf s_4 und s_5 ist, müßte der Gabelstapler gleichzeitig an beiden Orten sein - unmöglich! Zur Problemlösung führen Sie neue Stellen ein. Abbildung 5.56 zeigt die Lösung. Dort verhindert die Stelle s_7 eine Markierung von s_3 und s_4; stets kann nur s_3 oder s_4 eine Marke aufweisen (oder beide keine). Die Stellen s_6 und s_8 haben ähnliche Aufgaben für den einzelnen Schritt.

[19] Idee vgl. [3].

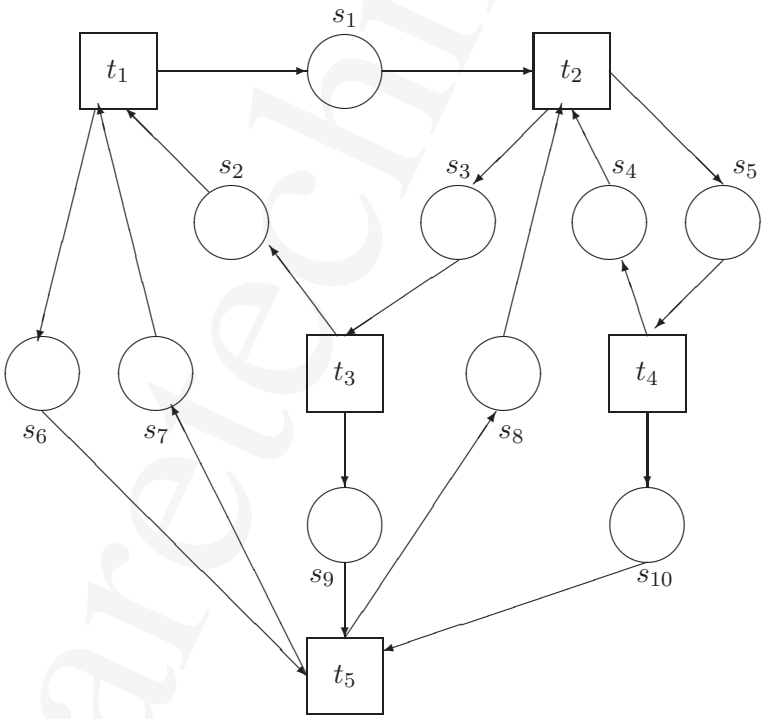


Abbildung 5.51: Netz II

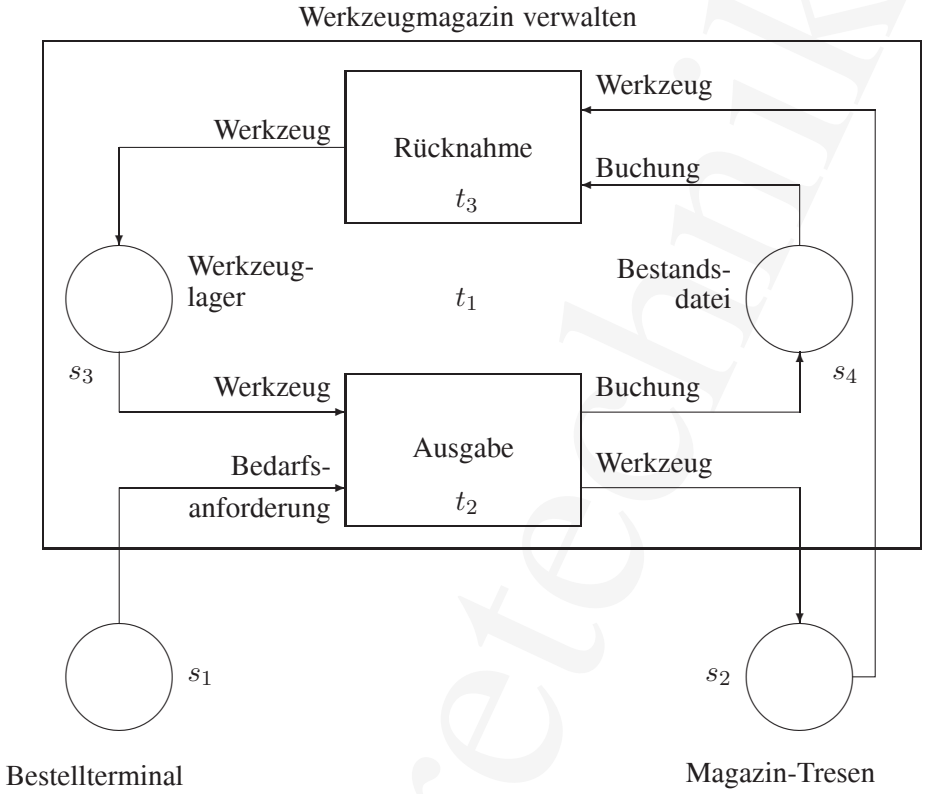


Abbildung 5.52: Netz: Werkzeugmagazin

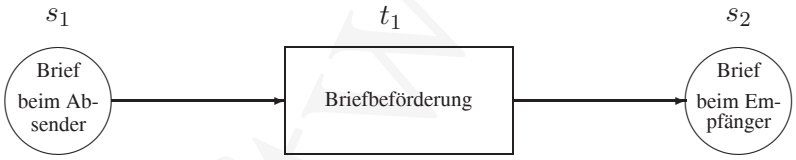
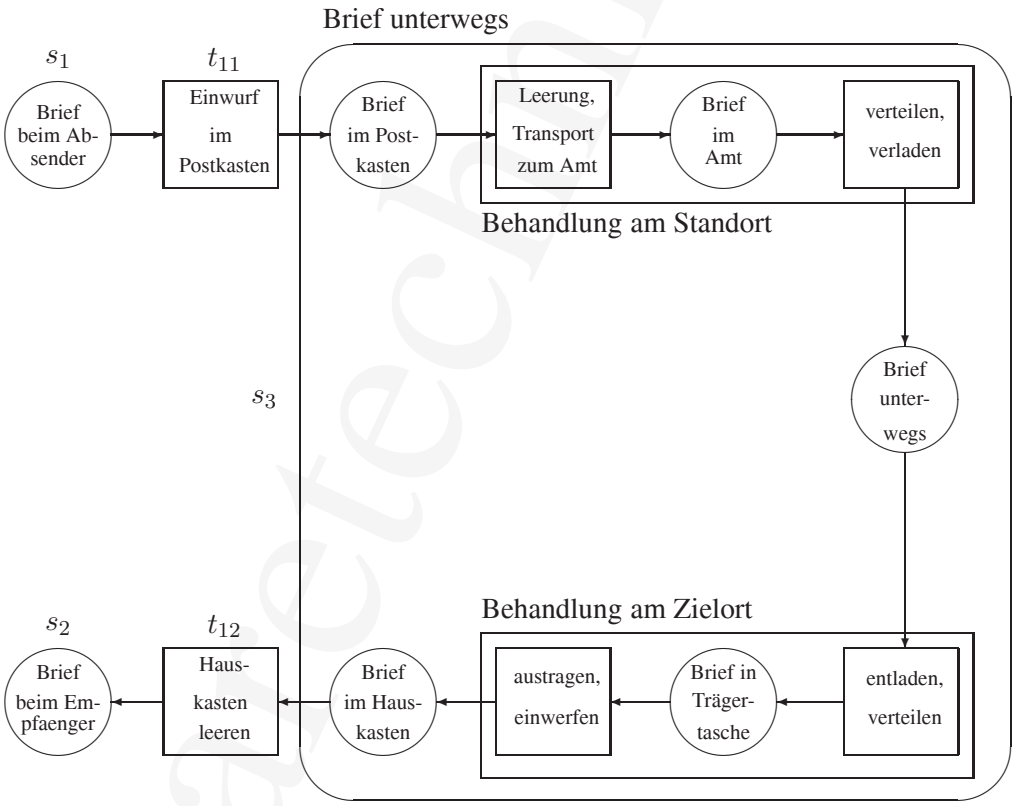


Abbildung 5.53: Briefbeförderung (Version 1)

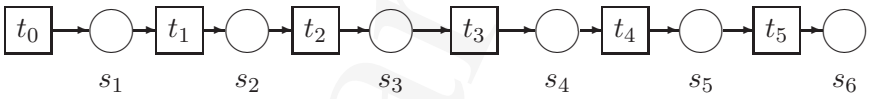


Legende:
Vgl. Abbildung 5.53 Seite 208
Vgl. ([3] S. 63.)

Abbildung 5.54: Briefbeförderung (Version 2)

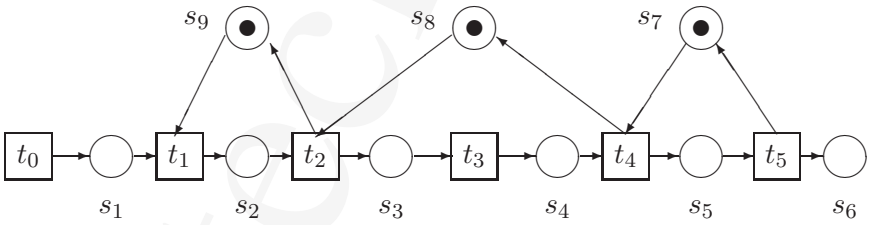
t_0	$\equiv$	Lastwagen (Lkw) kommt an
s_1	$\equiv$	Lkw ist an der Eingangsschranke.
t_1	$\equiv$	Eingangskontrolle: Papiere ausfertigen; Lkw fährt zur Laderampe.
s_2	$\equiv$	Kontrollierter Lkw wartet an Laderampe.
t_2	$\equiv$	Der Gabelstapler fährt leer zur Laderampe ins Lager.
s_3	$\equiv$	Der Gabelstapler ist im Lager.
t_3	$\equiv$	Der Gabelstapler wird beladen und fährt zur Laderampe.
s_4	$\equiv$	Der Gabelstapler ist an der Laderampe.
t_4	$\equiv$	Der Gabelstapler wird entladen.
s_5	$\equiv$	Ladung ist auf dem Lkw.
t_5	$\equiv$	Ausgangskontrolle: Ladung und Papiere überprüfen.
s_6	$\equiv$	Lkw auf Auslieferungsfahrt.

Tabelle 5.12: Knotenbeschreibung „Warenlager“



Legende:
Vgl. Tabelle 5.12 Seite 210.

Abbildung 5.55: Abfertigung beim Warenlager (Version 1)



Legende:
vgl. Tabelle 5.12 Seite 210

Abbildung 5.56: Abfertigung beim Warenlager (Version 2)

t_1	$\equiv$	Antragsteller gibt Sachbearbeiter Antrag.
t_2	$\equiv$	Sachbearbeiter bearbeitet Antrag.
t_3	$\equiv$	Sachbearbeiter übergibt Antragsteller den Bescheid.
t_4	$\equiv$	Antragsteller betritt Dienstzimmer.
t_5	$\equiv$	Antragsteller verläßt Dienstzimmer.
s_1	$\equiv$	Sachbearbeiter ist frei.
s_2	$\equiv$	Sachbearbeiter hat den Antrag.
s_3	$\equiv$	Antrag ist bearbeitet.
s_4	$\equiv$	Antragsteller wartet im Dienstzimmer.
s_5	$\equiv$	Dienstzimmer ist gesperrt.
s_6	$\equiv$	Dienstzimmer ist frei.
s_7	$\equiv$	Antragsteller ist abgefertigt.

Tabelle 5.13: Knotenbeschreibung „Auftragsbearbeitung“

Modellieren Sie den Vorgang, daß der Beschäftigte an der Ein- gangskontrolle t_1 (vgl. Abbildung 5.56 Seite 211) seine Zustimmung durch Drücken eines Knopfes gibt, der auch die Schranke betätigt. Abbildung 5.57 zeigt diesen Zusatz.

5.7.4 Aufgabe: Antragsbearbeitung im Büro

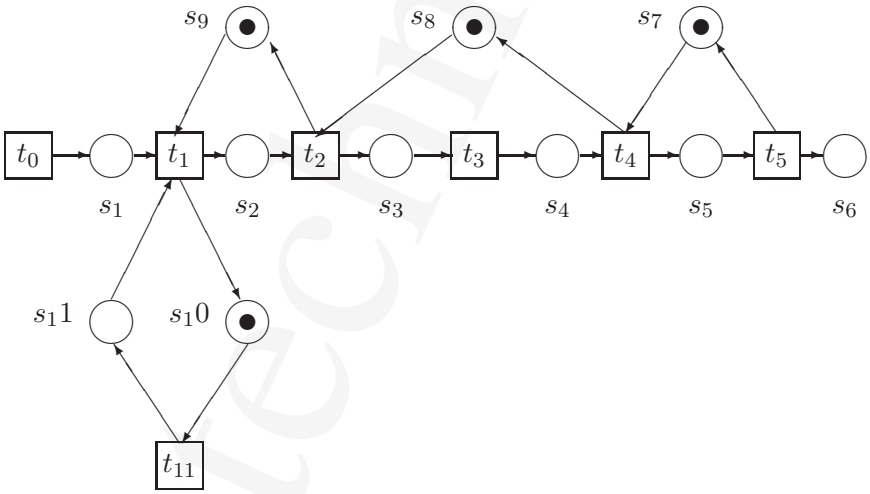
Das Dienstzimmer eines Sozialamtes ist mit einem Sachbearbeiter besetzt, der für die Bearbeitung eines Antrages allein zuständig ist. Die Bearbeitung erfolgt in Anwesenheit des Antragstellers. Aus Datenschutzgründen darf stets nur ein Antragsteller das Dienstzimmer betreten. (Idee für dieses Beispiel vgl. ([52] Seite 14)).

Entwerfen Sie ein Netz, daß diese Antragsbearbeitung abbildet. Abbildung 5.58 Seite 214 zeigt eine Lösung und Abbildung 5.59 Seite 214 eine Alternative dazu. Die Definition der Stellen und Transitionen enthält Tabelle 5.13 Seite 212.

5.7.5 Aufgabe: Kommando-Abarbeitung

Abbildung 5.60 Seite 215 skizziert die Abarbeitung von Kommandos bei der *Korn-Shell* eines UNIX-Systems. Erkennt die Shell ein Kommando, dann setzt sie (in der Regel) einen Kindprozeß ab, der versucht, das Kommando auszuführen. Der Kindprozeß selbst kann wiederum einen Kindprozeß generieren, wenn das Kommando aus primitiveren Kommandos zusammengesetzt ist.

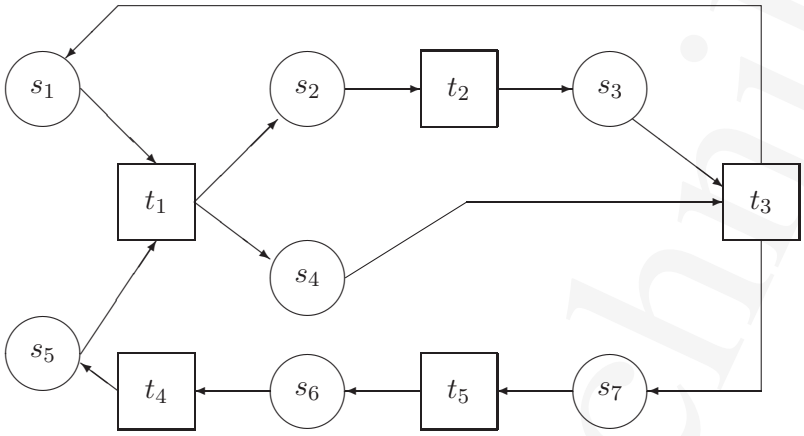
Entwerfen Sie ein Netz, daß die Abarbeitung eines Kommandos dar-



Legende:
Vgl. Tabelle 5.12 Seite 210.

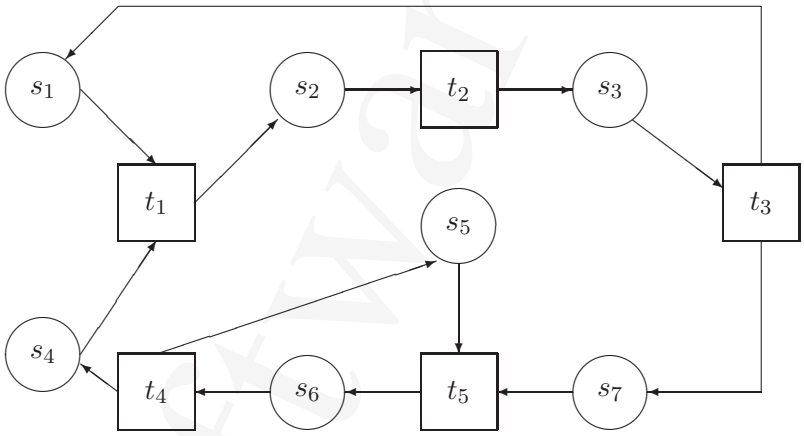
s_{10} ≡ Knopf kann gedrückt werden
t_{11} ≡ Kontrolleur drückt Knopf
s_{11} ≡ Lkw hat freie Fahrt

Abbildung 5.57: Abfertigung beim Warenlager (Version 3)



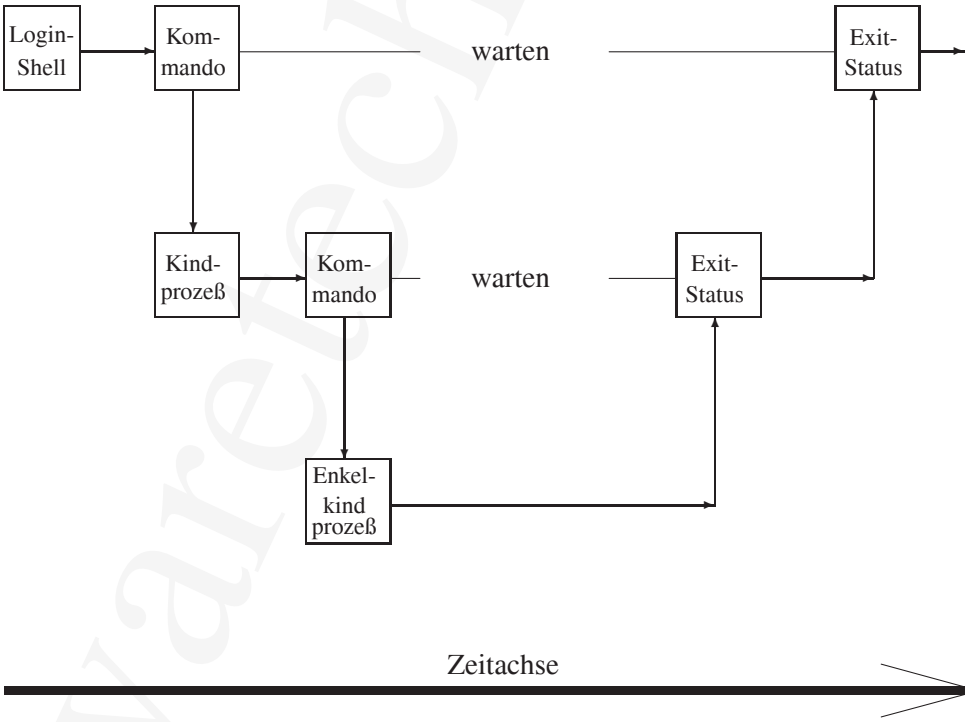
Legende:
Vgl. Tabelle 5.13 Seite 212.

Abbildung 5.58: Auftragsbearbeitung im Büro (Version 1)



Legende:
Vgl. Tabelle 5.13 Seite 212.

Abbildung 5.59: Auftragsbearbeitung im Büro (Version 2)



Legende:
Nähres dazu zum Beispiel [11].

Abbildung 5.60: Skizze: Korn-Shell Kommando-Abarbeitung

stellt. Abbildung 5.61 Seite 217 zeigt eine mögliche Lösung[20].

5.7.6 Aufgabe: $\wedge$-, $\vee$-Verknüpfung

Gegeben sei folgende logische Verknüpfung:
$input \wedge (process_i \vee process_j) \wedge output$
Interpretieren Sie diesen Ausdruck und entwerfen Sie ein entsprechendes Petri-Netz. Eine Lösung zeigt Abbildung 5.62 Seite 218.

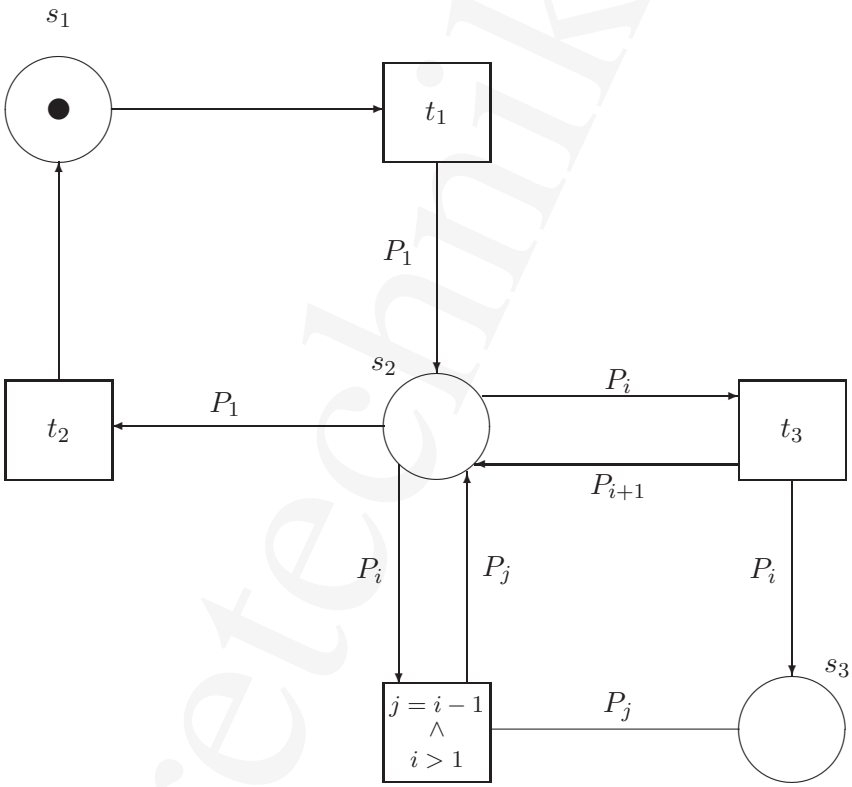
5.7.7 Aufgabe: Interpreter-Zyklus

Ein Interpreter, wie zum Beispiel ein LISP-System[21] arbeitet üblicherweise in einem Zyklus mit den Phasen: `READ`, `EVAL` und `PRINT`. Entwerfen Sie ein entsprechendes Petri-Netz.

Abbildung 5.63 Seite 219 zeigt eine Lösung.

[20] Diese Lösung wurde im Team von Herrn Roland Seen im Sommersemester 1993 entworfen.

[21] Nähres dazu vgl. zum Beispiel [10].



Legende:
Vgl. Abbildung 5.60 Seite 215.

M_0 $\equiv$ Marke auf s_1
s_1 $\equiv$ `login` Bereitschaft
s_2 $\equiv$ Prozeß P_i aktiv
s_3 $\equiv$ Prozesse im Wartestand
mit $i = 1, \ldots, n$
und $j = 1, \ldots, n-1$
t_1 $\equiv$ `login` erzeugt (Mutter-)Prozeß P_1
t_2 $\equiv$ `logout` entfernt (Mutter-)Prozeß P_1
t_3 $\equiv$ aktiver Prozeß P_i benötigt Kindprozeß P_{i+1}
t_4 $\equiv$ aktiver Prozeß P_i wird (als Kindprozeß) beendet und der (Mutter-)Prozeß wird aktiviert

Abbildung 5.61: Petri-Netz: Korn-Shell Kommando-Abarbeitung

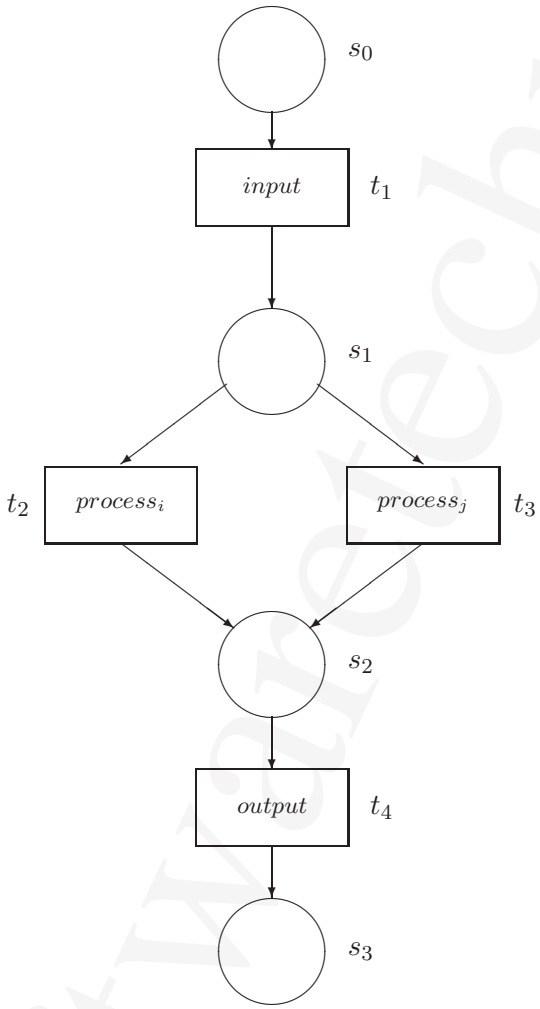


Abbildung 5.62: Petri-Netz für logische Verknüpfung

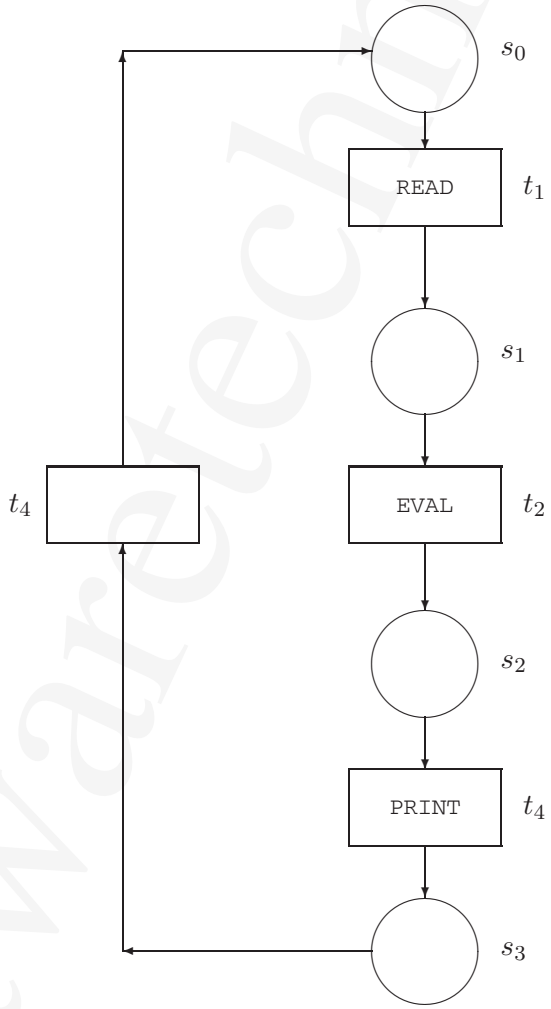


Abbildung 5.63: Petri-Netz: Interpreter-Zyklus

Kapitel 6

Weitere Boxologie

Dieses Kapitel befaßt sich mit (weiteren) Darstellungsmitteln in der Ausprägung, wie sie in heute marktüblichen CASE-Tools[1] häufig vorkommen.

Anhand eines Beispiels werden die Unterschiede zur Darstellung mittels Petri-Netzen, Programmablaufplänen und Struktogrammen verdeutlicht.

Wegweiser

Der Abschnitt *Weitere Boxologie* erläutert:

- ein Zustandsübergangsdiagramm in der Notation von Ward & Mellor,
  ↪Seite 222 . . .

- und Tabellen für die Abbildung von Zuständen und Aktionen.
  ↪Seite 224 . . .

---

[1] Eine Klassifikation und Berwertung der CASE-Tools ist nicht Ziel dieser Ausführungen. Dazu vgl. zum Beispiel [12].

6.1 Zustandsübergangsdiagramm

Im Zusammenhang mit CASE-Tools gewinnen Zustandsübergangsdiagramme in der Notation von Ward & Mellor [59] an Bedeutung. Um diese Notation zu erläutern, gehen wir von einem einfachen Beispiel aus:

> Beispiel „Stehlampe“:
> Eine Stehlampe aus Großmutters Zeiten hat zwei Glühbirnen. Eine mit 50W und eine mit 100W. Der Zugschalter steuert die Lampe so, daß eine Folge: kein Licht, 50W Gesamtleistung, 100W Gesamtleistung, 150W Gesamtleistung, kein Licht usw. entsteht.

Für dieses Beispiel zeigt Abbildung 6.1 Seite 223 ein Zustandsübergangsdiagramm. Die Konstrukte in diesem Diagramm haben folgende Bedeutung:

- Zustand ≡ gestrecktes Rechteckt
- Übergang ≡ Pfeil
- Übergangsbedingung ≡ Text direkt neben dem Übergang und oberhalb der Linie
- Übergangsaktion ≡ Text direkt neben dem Übergang und unterhalb der Linie.

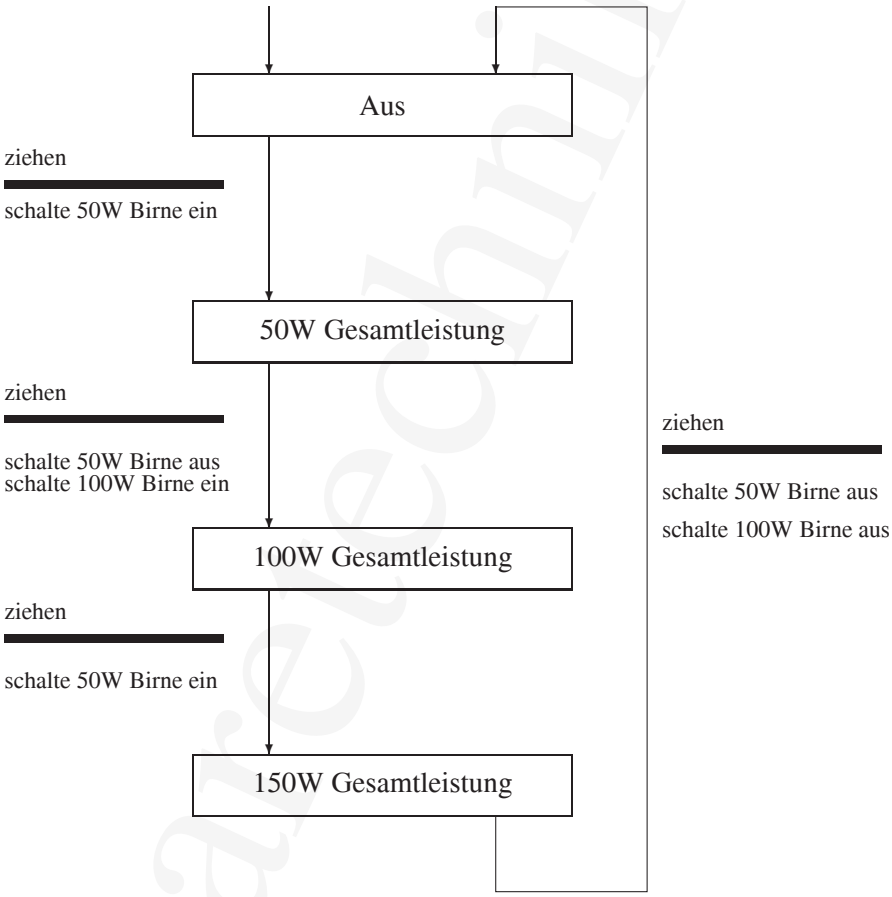
Ein Zustand repräsentiert einen extern zu beobachtenden Verhaltensmodus. Der Name des Zustandes charakterisiert das Verhalten des Systems. Die Transformationen können sich nur in einem Zustand zu einer gegebenen Zeit befinden. Einer der Zustände des Diagramms ist der Anfangszustand. Dieser wird durch einen darauf weisenden Pfeil erkennbar, wobei dieser Pfeil von keinem anderen Zustand kommt. Bedingungen und/oder Aktionen können, müssen aber nicht vorhanden sein. Ein oder mehrere Zustände können Endzustand oder Endzustände sein. Ein Endzustand ist durch ankommende Übergangspfeile gekennzeichnet und das Fehlen von weiterführenden Pfeilen. Bildlich formuliert stellt er den „Tod“ im Verhalten eines „lebendigen“ Systems dar.

Anfangszustand

Endzustand

Zustandsübergänge repräsentieren den Wechsel von einem in den nächsten Zustand. Sie können zwischen beliebigen Zuständen auftreten – also auch vom Zustand, von dem sie kommen. Ein Übergang von einem Zustand in denselben Zustand steht für ein Systemverhalten, das

Übergänge



Legende:
(ähnlich [59], S. 71)

Gestrecktes Rechteck	≡ Zustand
Pfeil	≡ Übergang
Übergangsbedingung	≡ Text direkt neben dem Übergang und oberhalb der Line
Übergangsaktion	≡ Text direkt neben dem Übergang und unterhalb der Line

Abbildung 6.1: Zustandsübergangsdiagramm: „Stehlampe“

vor und nach dem Auftreten eines Zustandsüberganges gleich ist. Mehrfache Zustandsübergänge von und zu einem bestimmten Zustand sind zulässig. Zustandsübergänge sind mit Bedingungen und Aktionen verknüpft.

Bedingung

Aktion

Die Bedingungen veranlassen das System zum Zustandsübergang. Die Bedingung steht über über der Linie, die die Bedingungen und Aktionen eines Zustandsübergangs trennt. Die Aktionen werden ausgeführt, wenn der Zustandsübergang erfolgt. Mehrere unabhängige Aktionen können mit einem Zustandsübergang verbunden sein. Von ihnen wird unterstellt, daß sie ohne Zeitbedarf und nebenläufig stattfinden, falls nicht ausdrücklich eine Sequenz angegeben wird. Statt von einer Ausführung ohne Zeitbedarf zu sprechen, kann man auch annehmen, daß beim Zustandswechsel nichts außer den genannten Aktionen selbst stattfinden kann.

Diese Ward & Mellor Notation für Zustandsübergangsdiagramme stellt einen endlichen Automaten mit Ausgängen[2] dar.

Als Alternative zu den Zustandsübergangsdiagramme kommen zum Beispiel

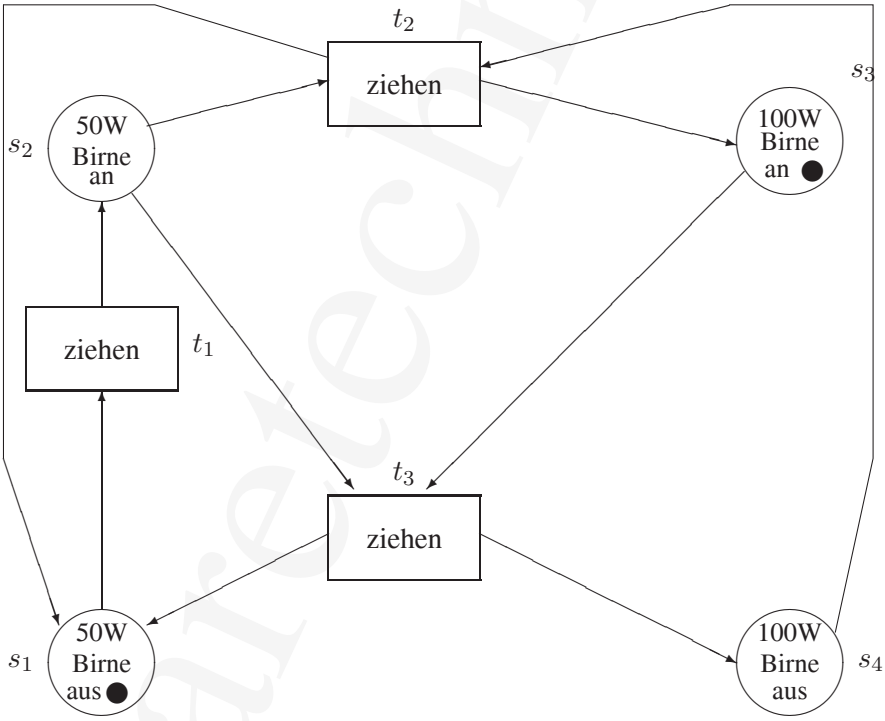
- Petri-Netze (vgl. Abbildung 6.2 Seite 225),
- Ablaufpläne (vgl. Abbildung 6.3 Seite 226),
- Struktogramme (vgl. Abbildung 6.4 Seite 227),
- Entscheidungstabellen (vgl. Abbildung 6.1 Seite 228) oder
- Matrixdarstellungen (Tabellen, vgl. Abschnitt 6.2 Seite 224)

in Betracht.

6.2 Zustands- & Aktionstabelle

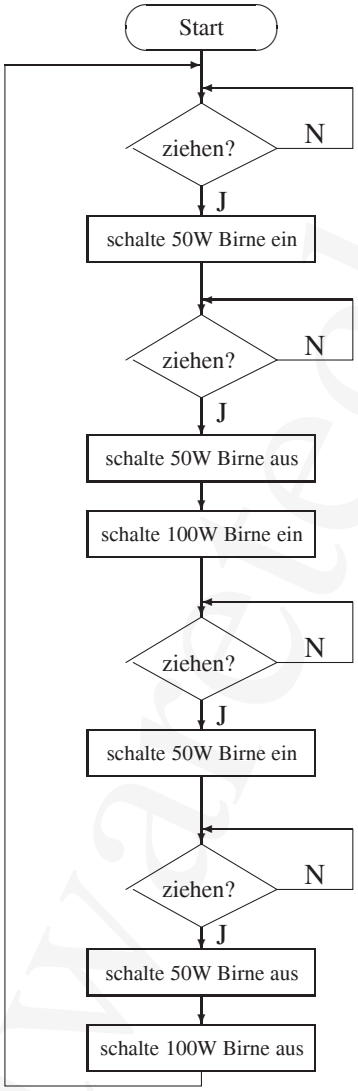
Ein Zustandsübergangsdiagramm läßt sich mit Hilfe einer Kombination von Zustandsübgangstabelle und Aktionstabelle notieren. Welche Darstellung zweckmäßiger ist, hängt von dem Verhältnis der Zahl der Zustände zur Zahl der Zustandsübergänge ab. Sind viele Zustandsübergänge zwischen wenigen Zuständen abzubilden, dann wird das Diagramm überladen und kaum durchschaubar. Im umgekehrten Fall sind die Tabellen zu groß und haben viele leere Felder.

[2] Es handelt sich um einen sogenannten Mealy-Automat, bei dem jeder Zustandsübergang – also jede Kombination von Zustand und Eingangssymbol – mit einem Ausgangssymbol verknüpft ist. Näheres dazu vgl. zum Beispiel [34].



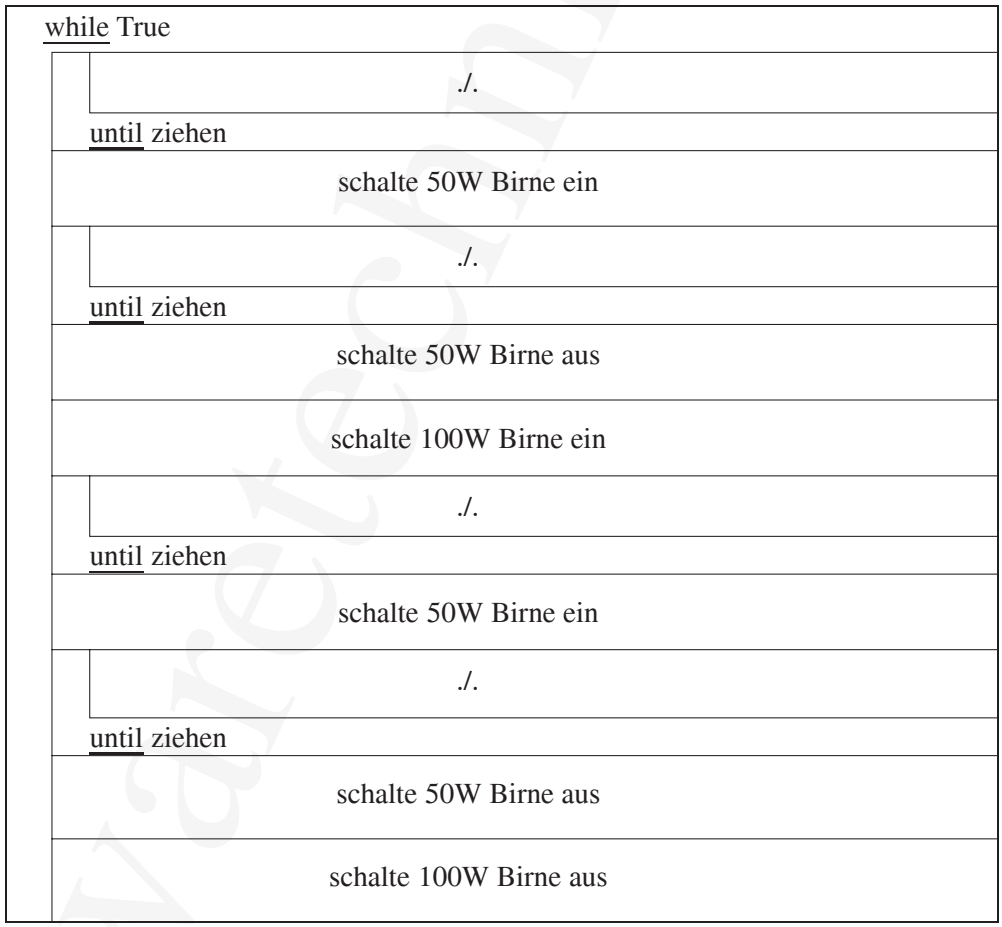
Legende:
Vgl. Abbildung 6.1 Seite 223
Statt im Zustandsübergangsdiagramm ist hier der Sachverhalt als Petri-Netz notiert. Dargestellt ist der Systemzustand, daß nur die 100W Birne brennt.

Abbildung 6.2: Bedingungs/Ereignis-Netz: „Stehlampe“



Legende:
Vgl. Abbildung 6.1 Seite 223
Statt im Zustandsübergangsdiagramm ist hier der Sachverhalt als Ablaufplan notiert.
Der aktuelle Systemzustand ist nicht erkennbar.

Abbildung 6.3: Ablaufplan: „Stehlampe“



Legende:
Vgl. Abbildung 6.1 Seite 223
Statt im Zustandsübergangsdiagramms ist hier der Sachverhalt als Struktogramm (vgl. DIN 66261 [25]) notiert. Der aktuelle Systemzustand ist nicht erkennbar.

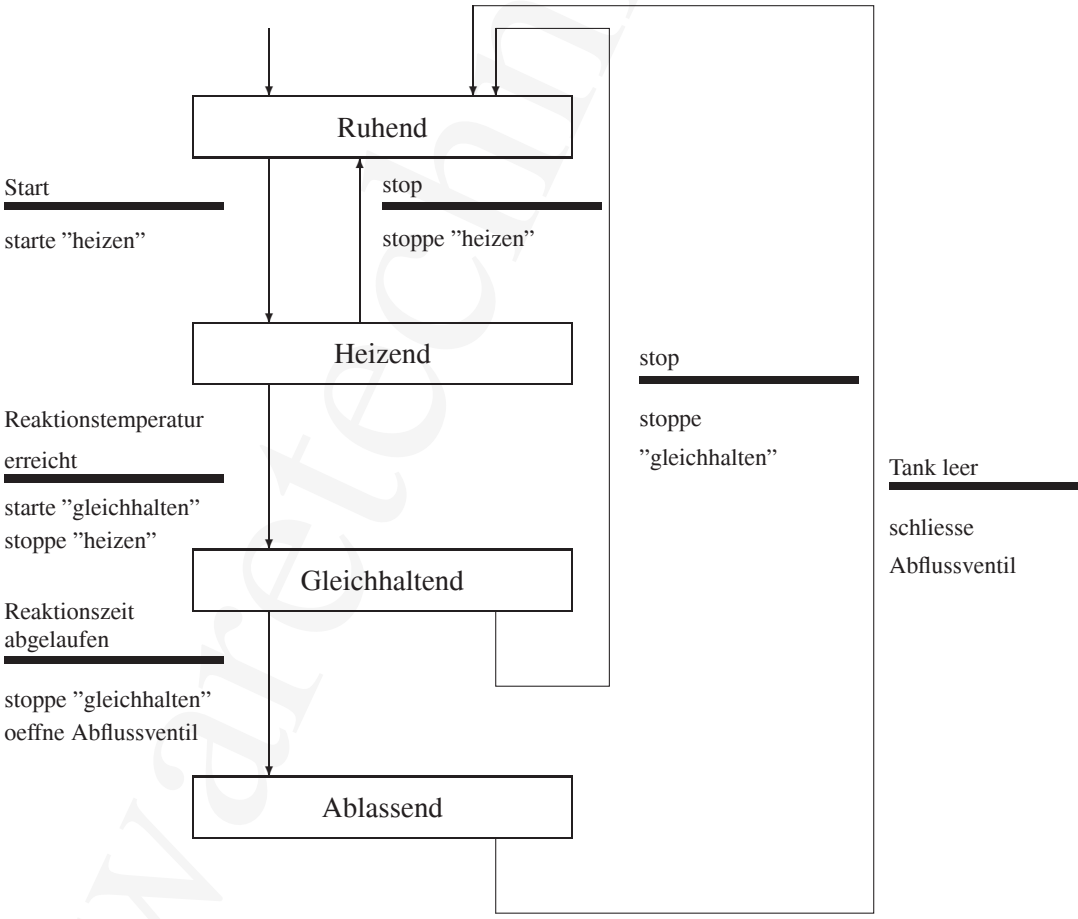
Abbildung 6.4: Struktogramm: „Stehlampe“

ET-Stehlampe		R1	R2	R3	R4
B1	50W Gesamtleistung ?	J	N	N	N
B2	100W Gesamtleistung ?	-	J	N	N
B3	150W Gesamtleistung ?	-	-	J	N
A1	Ziehen	X	X	X	X
A2	Schalte 50W Birne ein.		X		X
A3	Schalte 50W Birne aus.	X		X	
A4	Schalte 100W Birne ein.	X			
A5	Schalte 100W Birne aus.			X	
A6	Wiederhole ET-Stehlampe	X	X	X	X

Legende:
Vgl. Abbildung 6.1 Seite 223
Statt im Zustandsübergangsdiagramm ist hier der Sachverhalt als begrenzte, komplexe ET notiert. Der aktuelle Systemzustand ist nicht erkennbar.

Tabelle 6.1: ET: „Stehlampe“

Zum Vergleich ist ein Zustandsdiagramm, das einen chemischen Reaktionsprozeß als Beispiel hat (vgl. Abbildung 6.5 Seite 229), auch als Zustandsübergangstabelle (vgl. Abbildung 6.2 Seite 230) und als Aktionstabelle (vgl. Abbildung 6.3 Seite 231) notiert.



Legende:
(vgl. [59], S. 74)
Notation vgl. Abbildung 6.1 Seite 223

Abbildung 6.5: Diagramm: „Reaktionsprozeß“

Zustände		**Bedingungen**				
		Start I	Stop II	Tank leer III	Reaktionstemperatur erreicht IV	Reaktionszeit abgelaufen V
1	Ruhend	Heizend				
2	Heizend		Ruhend		Gleichhaltend	
3	Gleichhaltend		Ruhend			Ablassend
4	Ablassend			Ruhend		

Legende:
Zustandsübergangstabelle entsprechend Abbildung 6.5 Seite 229
Vgl. auch Aktionstabelle 6.3 Seite 231

Tabelle 6.2: Zustandstabelle: „Reaktionsprozeß“

Zustände		**Bedingungen**				
		Start I	Stop II	Tank leer III	Reaktionstemperatur erreicht IV	Reaktionszeit abgelaufen V
1	Ruhend	starte ”heizen”				
2	Heizend		stoppe ”heizen”		stoppe ”heizen”, starte ”gleichhalten”	
3	Gleichhaltend		stoppe ”gleichhalten”			stoppe ”gleichhalten”, öffne Abflussventil
4	Ablassend			schliesse Abflussventil		

Legende:
Aktionstabelle entsprechend Abbildung 6.5 Seite 229
Vgl. auch Zustandsübgergangstabelle 6.2 Seite 230

Tabelle 6.3: Aktionstabelle: „Reaktionsprozeß“

Anhang A

Literaturverzeichnis

Literaturverzeichnis

[1] Helmut Balzert (Hrsg.); Moderne Software-Entwicklungssysteme und -werkzeuge, Band 44 Reihe Informatik, herausgegeben von K. H. Böhling / U. Kulisch / H. Maurer, Mannheim Wien Zürich (Bibliographisches Institut) 1985.

[2] Friedrich L. Bauer / Gerhard Goos; Informatik - Eine einführende Übersicht, Erster Teil, Berlin u.a. (dritte Auflage, völlig neu bearbeitet und erweitert von F. L. Bauer) 1982. [Hinweis: Kurze Einführung in ET.]

[3] Bernd Baumgarten; Petri-Netze, Grundlagen und Anwendungen, Mannheim Wien Zürich (Bibliographisches Institut) 1990. [Hinweis: Leicht verständliche, praxisnahe Einführung in die Petri-Netz-Technik.]

[4] Fevzi Belli / K.-E. Grosspietsch; Specification of Fault-Tolerant System - Issues by Predicate/- Transition Nets and Regular Expressions - Approach and Case Study, Universität Paderborn, Technical Report, 1990/3 (June 1990), erscheint in: IEEE Transactions On Software Engineering, June 1991. [Hinweis: Erweitertes Petri-Netz zur Modellierung eines Hochregallagers.]

[5] Carl Böhret; Grundriß der Planungspraxis – Mittelfristige Programmplanung und angewandte Planungstechniken, Opladen (Westdeutscher Verlag) 1975. [Hinweis: Planungstechniken für politische Maßnahmen.]

[6] Hinrich Bonin; Neue Impulse für die Verwaltungsautomation?, in: ÖVD/Online Heft 7, 1983, S. 91–95. [Hinweis: Eine Diskussion des Ansatzes von [48].]

[7] Hinrich Bonin; Prototyping, in: ÖVD/Online, Heft 5, 1984, S. 74–78. [Hinweis: Ein Erläuterung der unterschiedlichen Vorgehensweisen.]

[8] Hinrich Bonin; Kontrollstrukturen, Arbeitsberichte des Fachbereichs 2 der Hochschule Bremerhaven, ISSN 0176-8158, Version 1.1, Skript, 1986/4.

[9] Hinrich Bonin; Die Planung komplexer Vorhaben der Verwaltungsautomation, Heidelberg (R. v. Decker & C. F. Müller) 1988. [Hinweis: Eine Kritik des „Wasserfall-Modells“ und ein Plädoyer für das Denkmodell „lernendes System“.]

[10] Hinrich Bonin; Software-Konstruktion mit LISP, Berlin u.a. (Walter de Gruyter) 1991. [Hinweis: Konzepte der systematischen Konstruktion mittels Konstruktoren, Selektoren, Prädikaten und Mutatoren bei unterschiedlichen Programmierparadigmen.]

[11] Hinrich Bonin; The Joy of Computer Science – Skript zur Vorlesung EDV –, FINAL, 3. Jahrgang, Heft 3, September 1993 (ISSN 0939-8821).

[12] Hinrich Bonin / Andreas Engel; CASE-Methoden in der Verwaltungspraxis, in: GI Softwaretechnik-Trends, Band 12, Heft 2, Mai 1992, S. 59 –65 (Mitteilungen der Fachgruppe „Software-Engineering“ der Gesellschaft für Informatik e. V. ISSN 0720-8928) [Hinweis: Berichte vom Workshop „Case-Methoden in der Verwaltungspraxis“ – Analyse der heutigen CASE-Tools.]

[13] G. Bollmann / J. Knowles; DAKEU: DAK - Entwicklungsumgebung, Benutzerhandbuch, AG: Methoden, Verfahren, Richtlinien, (DAK Hamburg) 1990.

[14] W. Brauer (ed.); Petri Nets Central Models and Their Properties, Lecture Notes in Computer Science, Vol. 255, Berlin Heidelberg New York (Springer-Verlag) 1987. [Hinweis: Aktueller Stand der Petri-Netz-Diskussion.]

[15] Hans Brinckmann; Zweierlei Experten für die gleiche Aufgabe: Das stabile Mißverständnis zwischen DV-Fachleuten und Verwaltungsfachleuten, in: Informatik-Fachberichte, herausgegeben von W. Brauer im Auftrag der GI, Band 44 herausgegeben von H. Reinermann u.a., Organisation informationstechnik-gestützter öffentlicher Verwaltungen, Fachtagung Speyer Oktober 1980, Berlin u.a. (Springer Verlag) 1981, S. 340–351. [Hinweis: Analyse des Spannungsfeldes Anwender ⇔ Informatiker.]

[16] Frederick Brooks, Jr.; The Mythical Man-Month, Reading Massachusetts, Menlo Park California (Addison-Wesley) 1975. [Hinweis: Die klassische Kritik der Planungsdimension „LOC“.]

[17] Manfred Broy / Ralf Steinbrüggen; Modellbildung in der Inforamtik, Berlin u. a. (Springer-Verlag) 2004, Xpert.press, ISBN 3-540-44292-8. [Hinweis: Modellierung wird als durchgängiges Thema für unterschiedliche Facetten der Informatik betrachtet.]

[18] R. Budde / K. Kuhlenkamp / L. Mathiassen / H. Züllinghoven (Herausgeber); Approaches to Prototyping, Berlin u.a. (Springer-Verlag). [Hinweis: Eine umfassende Darstellung der Thematik in Form von Konferenz-Beiträgen. Besonders informativ sind die Beiträge von Christiane Floyd und Anker Helms Jörgensen.]

[19] Rudolf Buechi; Entscheidungstabellen - Ein Lernprogramm, Giessen 1976. [Hinweis: Einfach lesbare, leichte Einfhrung in ET.]

[20] BVB-Erstellung; Besondere Vertragsbedingungen für das Erstellen von DV-Programmen, 20. Dezember 1985, Der Bundesminister des Innern (Bundesanzeiger). [Hinweis: Handlungsanweisung für die Bundesverwaltung.]

[21] Volker Claus / Andreas Schwill (Bearbeiter); Schüler-Duden: Die Informatik, Mannheim Wien Zürich (Bibliographisches Institut - Dudenverlag) 1986 (Stichwort: Petri-Netz). [Hinweis: Kurze prägnante Übersichtsdarstellung.]

[22] E. W. Dijkstra; GoTo Statement Considered Harmful, in: Communications of the ACM, 11(1968). [Hinweis: Klassische Kritik am GoTo-Statement.]

[23] Tom DeMarco; Structured Analysis and Design Specification, Englewood Cliffs, N. J. (Prentice-Hall) 1979.

[24] DIN 66241, Deutsches Institut für Normung e.V.; Entscheidungstabelle, Berlin Köln (Beuth Verlag) 1979. [Hinweis: Notation und ET-Verbundsysteme.]

[25] DIN 66261, Deutsches Institut für Normung e.V.; Sinnbilder nach Nassi-Shneiderman und ihre Anwendung in Programmablaufplänen, Berlin Köln (Beuth Verlag) 1984. [Hinweis: Notation von PAP.]

[26] DIN 69900; Netzplantechnik, Berlin Köln (Beuth Verlag) 1979. [Hinweis: Teil 1 Begriffe, Teil 2 Darstellungsformen.]

[GoRa99] Michel Goossens / Sebastian Rahtz / Eitan Gurari / Ross Moore / Robert Sutor; The LaTeXWeb Companion, Integrating TeX, HTML, and XML — Tools and Techniques for Computer Typesetting, ISBN 0-201-43311-7, (Addision-Wesley), 1999. {Hinweis: Ein hervorragendes Buch zur Frage wie in Zukunft ein Web-Dokument erzeugt werden kann. Befaßt sich zum Beispiel ausführlich mit DSSSL. CSS, XSL.}

[27] H. J. Genrich; Predicate/Transition Nets, in: [14] pp. 207-247. [Hinweis: Theoriearbeit.]

[28] H. J. Genrich / K. Lautenbach; S-invariance in Predicate/Transition Nets; in: Applications and Theory of Petri Nets, Informatik-Fachberichte Nr. 66, Heidelberg u.a. (Springer Verlag) 1982, pp. 98-111. [Hinweis: Theoriearbeit.]

[29] Lee L. Gremillion / Philip Pyburn; Breaking the systems development bottleneck, in: Harvard Business Review, 2/1983, S. 130 – 137. [Hinweis: Prototyping als ökonomisches Vorgehen dargestellt.]

[30] Carl Georg Hartung; Programmierung einer Klasse von Multiprozessorsystemen mit höheren Petri-Netzen, Heidelberg (Hüthig Verlag) 1987. [Hinweis: Dissertation; zum Verstehen ist fundiertes mathematisches Rüstzeug erforderlich.]

[31] Wolfgang Hesse / Hans Keutgen / Alfred L. Luft / Dieter H. Rombach; Ein Begriffsystem für die Softwaretechnik – Vorschlag zur Terminologie, in: Informatik-Spektrum, Heft 7, 1984, S. 200 – 213. [Hinweis: Begriffsbestimmungen vom Arbeitskreis der GI-Fachgruppe „Software Engineering“; leicht verständliche, praxisgerechte Definitionen.]

[32] Helmut Hofstetter; Organisationspsychologische Aspekte der Softwareentwicklung, in: Heinz Schelle / Peter Molzberger (Hrsg.); Psychologische Aspekte der Software-Entwicklung, München Wien (R. Oldenbourg Verlag) 1983, S. 25 – 62. [Hinweis: Kritische Auseinandersetzung mit einer technikbezogenen Sicht, die den Mensch vernachläßigt.]

[33] Helmut Hofstetter; Psychologische Probleme und Verfahren bei der Software-Entwicklung, in: GI Softwaretechnik Trends (Mitteilungen der Fachgruppe „Software Engineering“ der Gesellschaft für Informatik e.V., Bonn), Heft 5–2, Juni 1985, S. 63–72. [Hinweis: Übersichtsartikel.]

[34] J. D. Hopcroft / J. D. Ullman; Introduction to Automata Theory, Languages, and Computation, Reading (Addison-Wesley), 1979.

[35] Marion L. Hughes / Richard M. Shank / Elinor Svendsen Stein; Decision Tables, New York St. Louis u.a. (McGraw- Hill) 1968.[Hinweis: Leicht lesbare historische Arbeit über ET.]

[36] M. A. Jackson; Grundsätze des Programmentwurfs, 7. Auflage 1986, Darmstadt (Orginaltitel: Principles of Program Design, Übersetzung von R. Schaefer und G. Weber). [Hinweis: Klassische Arbeit über strukturierte Programmierung.]

[37] K. Jensen; High Level Petri Nets; in: Applications and Theory of Petri Nets, Informatik-Fachberichte Nr. 66, Heidelberg u.a. (Springer Verlag) 1982, pp. 166 – 180. [Hinweis: Theoriearbeit.]

[38] Bela von Jokuthy / Wulf Schnupp; Anwendung der Entscheidungstabellen-Technik, Berlin München, 1976.

[39] Gerald Jüttner / Hardy Feller; Entscheidungstabellen und wissensbasierte Systeme – Anwendungen in der Arbeitsplanung – Eine Studie der Nixdorf Computer AG am FAW Ulm, München Wien (R. Oldenbourg) 1989. [Hinweis: Beispiele aus der Arbeitsplanung ohne Einhaltung der normierten ET-Notation.]

[40] Tracy Kidder; Die Seele einer neuen Maschine, aus dem Amerikanischen übersetzt von Tony Westmayr, Basel Bosten Stuttgart (Birkhäuser Verlag) 1982 (Orginalausgabe: The Soul of a New Maschine, Boston Toronto (Atlantic Monthly Press and Little Brown Company, 1981). [Hinweis: Spannende Schilderung der Entstehung eines Computers.]

[41] Georg Klaus / Heinz Liebscher (Hrsg.); Wörterbuch der Kybernetik, überarbeitete Neuauflage, Frankfurt am Main (Fischer Taschenbuch Verlag) 1979. [Hinweis: Erste Auflage April 1967, Berlin – Das umfassende Kybernetik Nachschlagewerk in der ehemaligen DDR; theoretisch fundiert, allerdings mit vielen sozialistischen Passagen.]

[42] Richard C. Linger / Harlan D. Mills / Bernard I. Witt; Structured Programming: Theory and Practice, Reading, Massachusetts (Addison-Wesley) 1979. [Hinweis: Ausführliche Erläuterung von Kontrollstrukturen anhand von PDL.]

[43] Jochen Ludewig; Modelle der Software-Entwicklung – Abbilder oder Vorbilder? in: GI (Gesellschaft für Informatik) Software Trends, Band 9 Heft 3, Oktober 1989, S. 1 – 12. [Hinweis: Überblick über Modelle zur Vorgehensweise.]

[44] Manfred Nagl; Softwaretechnik - Methodisches Programmieren im Groaen, Berlin u.a. (Springer-Verlag) 1990. [Hinweis: Umfassende Einführung für Praktiker mit Schwerpunkt Modularisierung.]

[45] John Naisbitt; Megatrends – 10 Perspektiven, die unser Leben verändern werden, Deutsche Übersetzung von Günter Hehemann, München (Heyne) Sachbuch Nr. 01/7235, Orginalausgabe Megatrends 1982. [Hinweis: Populärwissenschaftliche Prognose über die zukünftige Gesellschaft.]

[46] I. Nassi / B. Shneiderman; Flowchart Techniquies for Structured Programming, in: SIGPLAN Notices 8 (1973) 8, p. 12 – 26. [Hinweis: Orginalarbeit über Struktogramme.]

[47] Carl Adam Petri; Kommunikation mit Automaten, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Schrift Nr. 2, Bonn 1962. [Hinweis: Grundlegende Arbeit vom „Erfinder" dieser Netze.]

[48] Heinrich Reinermann; Brauchen wir eine „Bauhaus-Bewegung" für die Verwaltungsautomation?, in: ÖVD/Online Heft 2 1983, S. 67 – 72. [Hinweis: Ein Plädoyer für eine Reform von Verwaltungsvorgängen im Zusammenhang mit ihrer Automation.]

[49] Heinrich Reinermann; Neue Technologien in der öffentlichen Verwaltung, in: IBM Nachrichten, Heft 277, Juni 1985, S. 7 – 13. [Hinweis: Zustandsskizze und Strategieempfehlungen für die Verwaltungsautomation.]

[50] Wolfgang Reisig; Petrinetze - Eine Einführung, Berlin Heidelberg New York (Springer-Verlag) 1982. [Hinweis: Mathematisch geprägte Einführung in die netztheoretischen Grundbegriffe und Denkweisen. Es sind fundierte Kenntnisse der Mengentheorie erforderlich.]

[51] Wolfgang Reisig; Systementwurf mit Netzen, Belin Heidelberg u.a. (Springer-Verlag) 1985. [Hinweis: Leicht verständliche Einführung ohne mathematische Voraussetzungen.]

[52] Bernd Rosenstengel / Udo Winand; Petri-Netze, Eine anwendungsorientierte Einführung, (Programm Angewandte Informatik, Herausgeber Paul Schmitz, Norbert Szyperski) Braunschweig Wiesbaden (Vieweg & Sohn) 1982. [Hinweis: Darstellung eines umfangreichen Fall-Beispiels.]

[53] Peter Schefe; Informatik – Eine konstruktive Einführung – LISP, PROLOG und andere Konzepte der Programmierung, Band 48 Reihe Informatik, herausgegeben von K. H. Böhling / U. Kulisch / H. Maurer, Mannheim Wien Zürich (Bibliographisches Institut) 1985. [Hinweis: Eine Alternative zu einer imperativgeprägten Einführung in die Informatik – auf einem theoretisch fundierten Niveau.]

[54] Peter Schnupp / Christiane Floyd; Software – Programmentwicklung und Projektorganisation, 2., durchgesehene Auflage, Berlin New York (Walter de Gruyter) 1979. [Hinweis: Ein Klassiker (praxisnahes Werk) im Bereich „Software Engineering".]

[55] Peter H. Starke; Analyse von Petri-Netz-Modellen, Stuttgart (B. G. Teubner) 1990. [Hinweis: Hauptanliegen ist die Analyse von Netzen mit dem Ziel Eigenschaften zu beweisen, daher ist mathematisches Rüstzeug notwendig.]

[56] Ian Sommerville; Software Engineering, third edition, Wokingham, England u.a. (Addison-Wesley) 1989. [Hinweis: Eines der hervorragenden Standardwerke im Bereich Softwaretechnik.]

[57] W. Strunz; Entscheidungstabellentechnik - Grundlagen und Anwendungsmöglichkeiten bei der Gestaltung rechnergestützter Informationssysteme, München Wien 1977. [Hinweis: Umfassendes Werk über ET.]

[58] Gerhard Versteegen; Projektmanagement mit dem Rational Unified Process, unter Mitarbeit von Philippe Kruchten und Barry Boehm, Berlin Heidelberg (Springer), 2000, ISBN 3-540-66755-5. [Hinweis: Leicht verständliches Werk mit starker Ausrichtung auf die Softwareprodukte von Rational – the e-development company.]

[59] Paul T. Ward / Stephan J. Mellor; Strukturierte Systemanalyse von Echtzeit-Systemen, München, Wien (Hanser) 1991 (amerikanische Orginalausgabe: Structured Development for Real-Time Systems, (Prentice-Hall) 1985).[Hinweis: Praxisnahe Einführung in die Darstellung von Prozessen mittels selbst konzipierter Notation.]

[60] Konrad Zuse; Petri-Netze aus der Sicht des Ingenieurs, Braunschweig Wiesbaden (Vieweg & Sohn) 1980. [Hinweis: Praxisnahe Petri-Netzerläuterung primär anhand der Schaltalgebra und der formalen Logik.]

[61] Konrad Zuse; Anwendungen von Petri-Netzen, Braunschweig Wiesbaden (Vieweg & Sohn) 1982. [Hinweis: Erläuterung von zellularen Automaten als Petri-Netze.]

Anhang B

Abkürzungen und Akronyme

BVB	Besondere VertragsBedingungen des öffentlichen Sektors
CASE	Computer Aided Software Engineering
CMM	Capability Maturity Model for Software
COBOL	Common Business-oriented Language
CPM	Critical Path Method
CPN	Coloured Petri Net
DIN	Deutsches Institut für Normung e. V.
DTD	Document Type Definition
GSPN	Generalised Stochastic Petri Net
MPM	Metra Potential Method
PDL	Process Design Language
PERT	Program Evaluation and Review Technique
RM&E	Requirements Management & Engineering
SA	Structured Analysis
SEI	Software Engineering Institute
S/T	Stelle/Transitions-Netz
UML	Unified Modeling Language
XML	Extensible Markup Language

Anhang C

Software-Werkzeuge

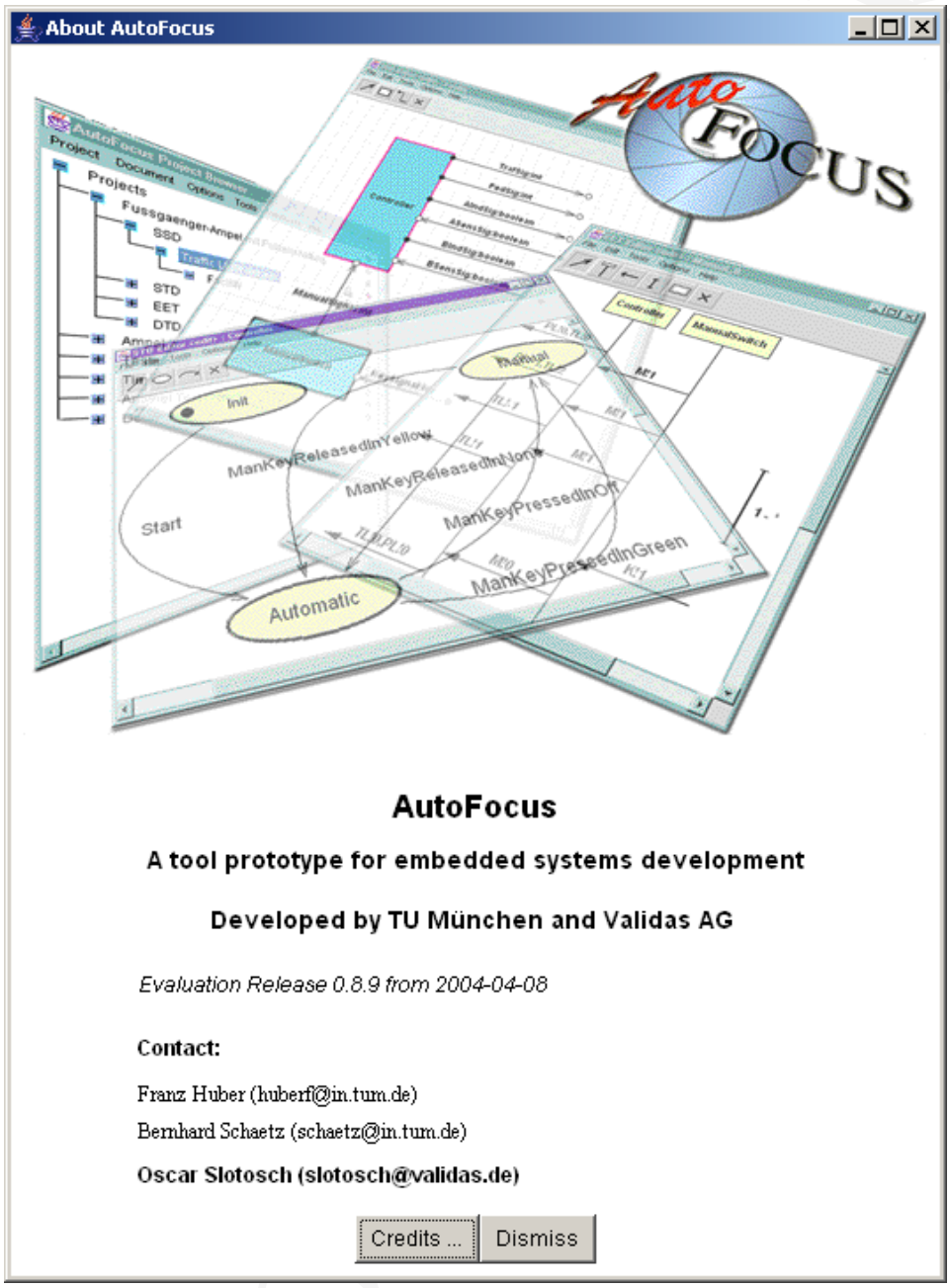
C.1 AutoFocus — *Embedded Systems Development*

AutoFocus will mit Hilfe von graphischen Darstellungsmitteln ein „ein möglichst intuitives Arbeiten ermöglichen.“[1] Der Kern des Werkzeuges ist ein Projektbrowser, der Spezifikationsdokumente verschiedener Projekte sowie verschiedene Editoren für die einzelnen Spezifikationsdokumente verwaltet. AutoFocus unterstützt den Entwurf der Spezifikation durch:

- eine verteilte, plattformunabhängige Entwicklung,
- eine Versionsverwaltung (inclusive Sperroptionen) auf der Basis
- von hierarchisch-geprägten Beschreibungsmitteln.

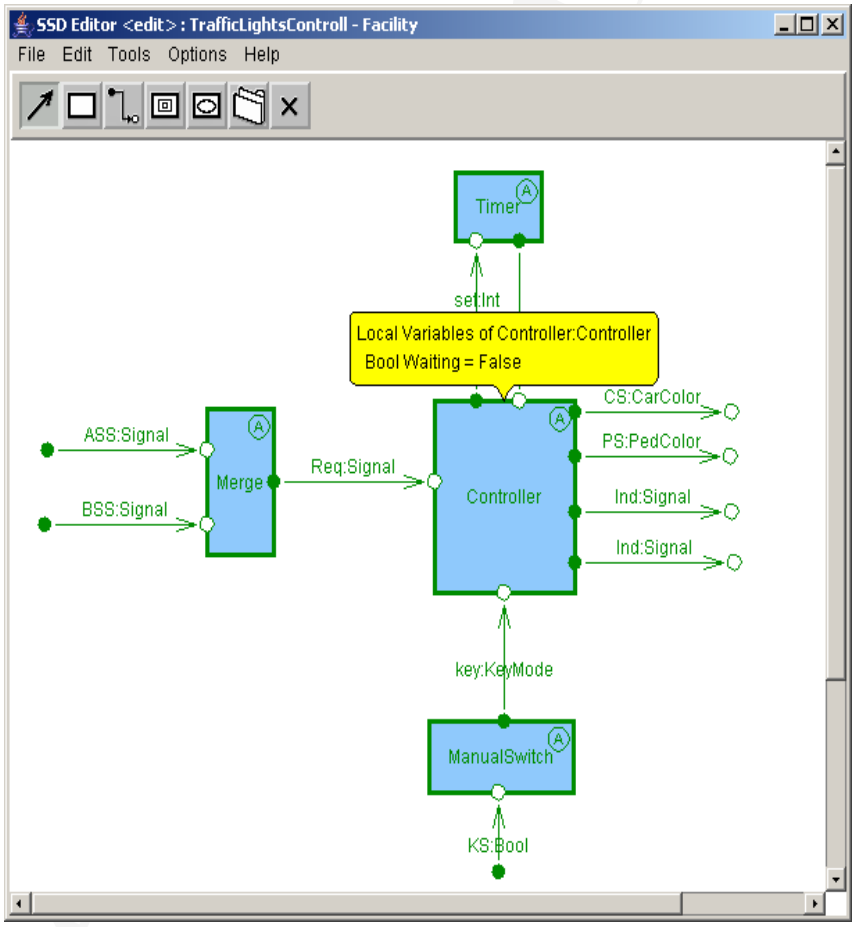
Die Beschreibung des Verhaltens des Gesamtsystems oder seiner Komponenten aus mehreren Blickwinkeln ist das Konzept, um ein transparentes Gesamtbild des Systems zu gewinnen. AutoFocus bietet mit den Systemstrukturdiagrammen, den Automatendiagrammen und den Erweiterten Ereignisdiagrammen drei graphische Beschreibungstechniken (↪ Abbildung C.2 S. 245). Je nach Grad der Detaillierung sind Komponenten oder Verhaltensbausteine atomar oder sind selbst wieder aus Unterkomponenten bzw. Unterbausteinen zusammengesetzt. AutoFocus unterstützt mit den hierarchisch-geprägten Beschreibungsmitteln den Wechsel zwischen unterschiedlich detaillierten Sichten. AutoFocus, ein Werkzeug der Technischen Universität München und der Validas AG konzentriert sich auf die Entwicklung von sogenannten „Eingebetteten Systemen“, also von System im Bereich der Steuerung von Maschinen. AutoFocus ist unter:

[1] ↪ http://autofocus.informatik.tu-muenchen.de/Infos/afinfo-de.html (Zugriff 30-Jun-2004)



Legende:
Werkzeug der TU München und der Validas AG

Abbildung C.1: `Autofocus`— Help/About-Fenster



Legende:
Werkzeug der TU München und der Validas AG

Abbildung C.2: Autofocus— SSD-Beispiel

`http://autofocus.informatik.tu-muenchen.de/`
(Zugriff 28-Jun-2004)

frei verfügbar. Die folgende Prokolldatei dokumentiert die Java-Basis diese Werkzeuges.

Protokolldatei `AutoFocus.log`

```
C:\Programme\AutoFocus>startaf
Applic~1\Quest\sim.jar;Applic~1\Quest\sableutil.jar;
  Applic~1\Quest\log4j.jar;Applic~1\Quest\jai_core.jar;
  Applic~1\Quest\jai_codec.jar;.
WARN  autofocus.box.browser.PlugIns  - class
  quest.codegen.ada.Generator for Ada Code
  not found in classpath
WARN  autofocus.box.browser.PlugIns  - extension
  Ada Code not installed
FRED is coming up... please wait!
no version control (vc=off)
Local FRED is up!!!
C:\Programme\AutoFocus>
```

C.2 DaNAMiCS — Petri-Netz-Tool

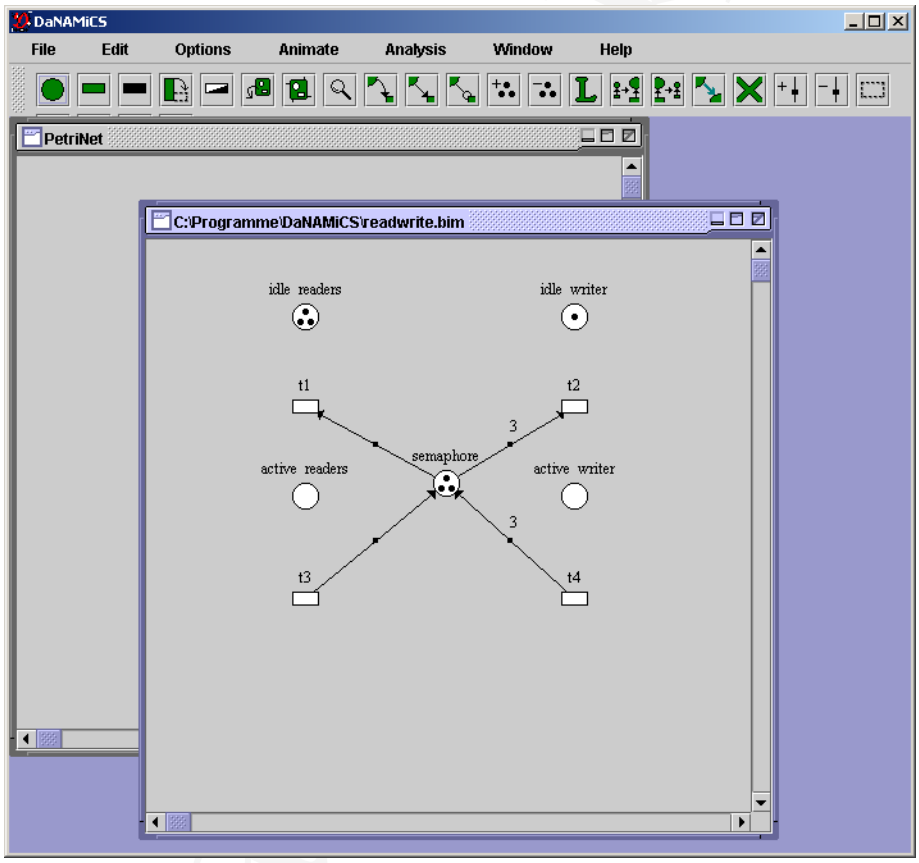
DaNAMiCS (*Data Network Architecture: Modelling Concurrent Systems*) ist ein Java-basierter Editor für Petrinetze. Eine Demo-Version ist unter:

`http://www.cs.uct.ac.za/Research/DNA/DaNAMiCS/DownLoads.html`
(Zugriff 05-Apr-2003)

frei verfügbar. DaNAMiCS verwendet Petrinetze als Formalismus um ein Modell zu spezifizieren. Solche Modelle werde im Hinblick auf Korrektheit und Performance analysiert. Zum Beispiel werden *Deadlock*-Situationen erkannt. Das Werkzeug hat drei Hauptkomponenten:

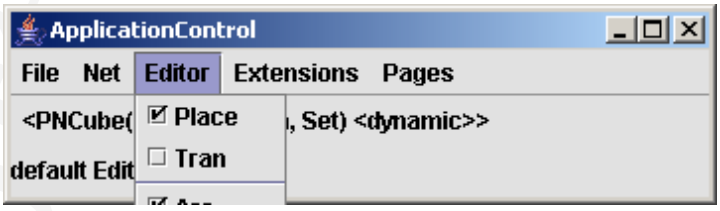
- *Editor*,
- *Analyser* und
- *Results Viewer*.

DaNAMiCS unterstützt *Generalised Stochastic Petri nets* (GSPNs) und *Coloured Petri Nets* (CPNs).



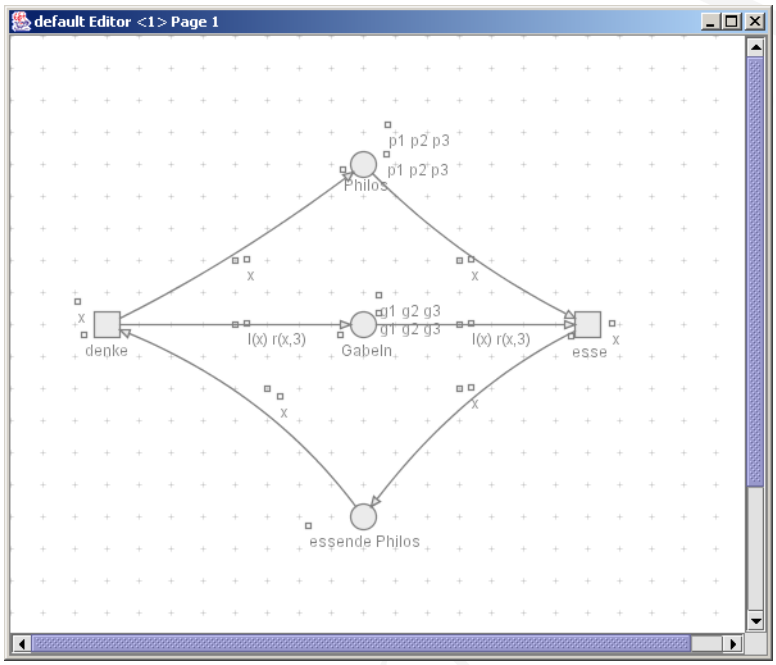
Legende:
Beispiel mit dem Petri-Netz-Werkzeug DaNAMiCS, Demo-Version

Abbildung C.3: DaNAMiCS-Beispiel



Legende:
Beispiel mit dem Petri-Netz-Werkzeug PNK Java-Version 2.2

Abbildung C.4: PNK — *Application Control*



Legende:
Beispiel mit dem Petri-Netz-Werkzeug PNK Java-Version 2.2

Abbildung C.5: PNK-Beispiel

C.3 Petri Net Kernel — Humboldt Uni Berlin

Das Werkzeug PNK (*Petri Net Kernel*), der Humboldt Universität zu Berlin, Informatikfakultät, Lehrstuhl Theorie der Programmierung, stellt eine Infrastruktur bereit, um die Ideen der Analyse, Simulation und Verifikation von Petri Netzen zu unterstützen. Ursprünglich wurde das Werkzeug in Python[2] implementiert. Daraus abgeleitet wurde eine Java-Implementation. Beide sind unter:

`http://www.informatik.hu-berlin.de/top/pnk/index.html`
(Zugriff 05-Apr-2003)

frei verfügbar. Die folgende Prokolldatei dokumentiert die Java-Basis diese Werkzeuges. Die Abbildung C.4 S. 247 skizziert die Steuerung der Anwendung.

Protokolldatei `PNK2.log`

```
C:\Programme\PNK2.2>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

C:\Programme\PNK2.2>java -jar PNK2.jar
..... NewNetAction... Thread[Thread-1,6,main]
  de.huberlin.informatik.pnk.kernel.Net@1ce669e
  de.huberlin.informatik.pnk.editor.
    GruenLookAndFeel/icons/Question.gif
not found.

C:\Programme\PNK2.2>
```

[2] Programmiersprache Python ↪`http://www.python.org/` (Zugriff 05-Apr-2003)

Anhang D

Leistungsnachweis

Dieses Kapitel dient zum selbständigen Arbeiten an einer etwas größeren Aufgaben.

Es werden als Lernziele angestrebt:

- das Umsetzen von Anforderungen in eine zweckmäßige und adressatengerechte Dokumentation und
- das Sammeln von konkrete Erfahrungen beim „Programmieren im Kleinen“

Wegweiser

Der Abschnitt *Leistungsnachweis* stellt die Design- und Implementationsaufgaben:

- Archivierungsprogramm und
  ↪Seite 252 . . .
- Software für die Unterstützung eines Fahrradkäufers.
  ↪Seite 253 . . .

D.1 Aufgabe: Archivierungsprogramm

D.1.1 Mindestanforderungen

R01 Getestetes Programm

Realisieren Sie das Programm ARCHIV, das folgende Anforderungen erfüllt:[1]

A1 ARCHIV druckt nicht formatierte Textdateien, zum Beispiel Quellcodelisten.

A2 ARCHIV erfaßt im Dialog das Titelblatt mit den Merkmalen:

- A2.1 Titel umfaßt maximal 100 Zeichen.
- A2.2 Kurztitel umfaßt maximal 30 Zeichen.
- A2.3 Autorangabe umfaßt maximal 30 Zeichen.
- A2.4 Zweckangabe umfaßt maximal 200 Zeichen.

A3 ARCHIV ermittelt das Tagesdatum und druckt

A4 dieses auf jeder Seite in der Kopfzeile.

A5 ARCHIV druckt in einer Fußzeile die Seitenzahl in folgender Form: Seite i von n Seiten.

E1 ARCHIV ist in einer standardisierten Programmiersprache erstellt.

E2 ARCHIV nutzt keine Fertigbausteine.

R02 Zweckmäßige Dokumentation

D.1.2 Abnahmekriterien für *ARCHIV*

1. ARCHIV-Dokumentation
   - Erfüllung der obigen funktionalen Anforderungen
   - Zweckmäßigkeit des logischen Entwurfes
     - Begründung der Modularisierungsentscheidungen
     - Beschreibung der Einheiten des Funktionsbaumes
     - Beschreibung der Datenstrukturierung
   - Aussagekraft der Testprotokolle (Verifikations- und Validationsunterlagen)

[1] A ≡ Anforderung; E ≡ Entwurf(srestriktion); T ≡ Testfall; zum Vorschlag für diese Abkürzungen vgl. [10].

- Quellcodeliste mit Verweisen zu den Anforderungen in Form von Kommentaren und zeckmäßiger, adressatengerechter Kommentierung.

2. Erfolgreiche ARCHIV-Präsentation auf einem Rechner Ihrer Wahl.

D.2 Aufgabe: Hilfe für Fahrradkäufer

D.2.1 Mindestanforderungen

Konzipieren und realisieren Sie das Softwarepaket RADVORSCHLAG zur Beratung von Kunden, die ein Fahrrad erwerben wollen. Die Aufgabe umfaßt insbesondere folgende Arbeiten:

R01 Requirements Engineering

- Ermitteln und Analyse von Anforderungen
- Kompromißfindung zwischen konkurrierenden Anforderungen

R02 Begründung und Wahl des Designs

- Diskussion des Grobentwurfes anhand von Alternativlösungen
- Ableitung des Feinentwurfes aus dem gewählten Grobentwurf

R03 Programmierung & Implementation

R04 Validation & Verifikation

D.2.2 Abnahmekriterien für RADVORSCHLAG:

1. RADVORSCHLAG-Dokumentation – insbesondere Begründung Ihrer Analyse- & Design-Entscheidungen.
2. Erfolgreiche RADVORSCHLAG-Präsentation auf einem Rechner Ihrer Wahl

D.2.3 Optionen

- Das Kundenberatungssystem kann sich auf ein anderes Produkt beziehen, zum Beispiel auf ein Radio.
- Die Dokumentation kann mit einem beliebigen (Satz-)System geschrieben werden — es muß nicht mein „Liebling“ TEX & LATEX oder XML-basiert in `DocBook` sein.

Anhang E

(Klausur-)Aufgaben

Dieses Kapitel enthält einige Klausuraufgaben, die im Rahmen des Faches *Anwendungsentwicklung* im Studiengang *Wirtschaftsinformatik* des Fachbereichs Wirtschaft der Fachhochschule Nordostniedersachsen im Zeitraum von 1990–2002 gestellt wurden. Bei einer vierstündigen Klausur waren 100 Punkte zu erreichen, bei einer zweistündigen 50. Zum Bestehen mußte mindestens die Hälfte der erreichbaren Punkte erzielt werden. Die Punktergebnisse wurden wie folgt in die üblichen Noten umgerechnet:

- $\geq 95\% \equiv$ Note: $1,0$
- $\geq 92\% \equiv$ Note: $1,3$
- $\geq 89\% \equiv$ Note: $1,7$
- $\geq 80\% \equiv$ Note: $2,0$
- $\geq 77\% \equiv$ Note: $2,3$
- $\geq 74\% \equiv$ Note: $2,7$
- $\geq 65\% \equiv$ Note: $3,0$
- $\geq 62\% \equiv$ Note: $3,3$
- $\geq 59\% \equiv$ Note: $3,7$
- $\geq 50\% \equiv$ Note: $4,0$
- $< 50\% \equiv$ Note: nicht bestanden

Musterlösungen zu den einzelnen Aufgaben finden Sie im KapitelF Seite 285 ff. Eine dort skizzierte Lösung schließt andere, ebenfalls richtige Lösungen, nicht aus.

E.1 Aufgabe: PAP-Linearisierung

Die Abbildung E.1 Seite 256 zeigt einen Programmablaufplan (PAP). [Hinweis: T $\equiv$ True; F $\equiv$ False].

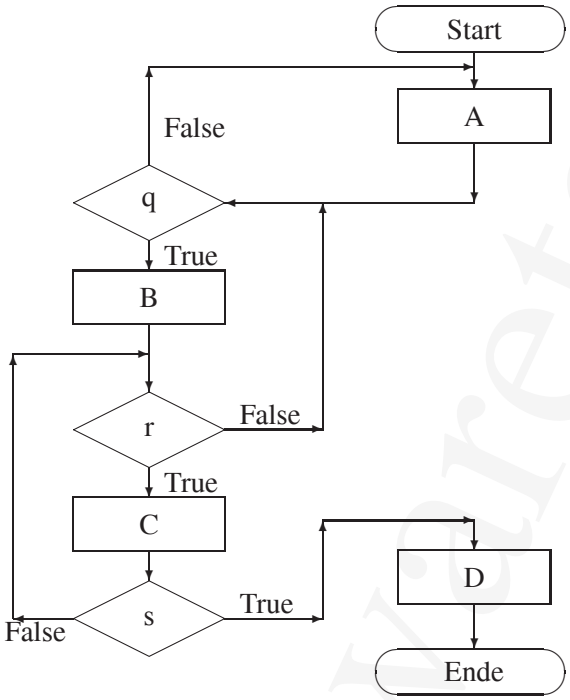


Abbildung E.1: PAP mit nichtlinearer Kontrollstruktur

E.1.1 PAP-Umkonstruktion (10 Punkte)

Transformieren Sie diese PAP in einen gleichwertigen PAP, der nur aus Bausteinen vom Typ:

- Sequenz,
- while ... do ... od
  und / oder
- if ... then ... fi

konstruiert ist.[1]

E.1.2 Struktogramm (4 Punkte)

Geben Sie Ihren umkonstruierten PAP als Struktogramm (Nassi/Shneiderman-Notation) an.

E.2 Aufgabe: Rekursion & Struktogramm

Bei einer rekursiv definierten Folge kann das n-te Glied aus den vorangehenden Gliedern berechnet werden. Die Gleichung E.1 gibt die Rekursionsvorschrift an:

$$a_n = \begin{cases} 1 & \text{für } n = 0 \\ (\frac{1}{2} * a_{n-1}) + \frac{1}{a_{n-1}} & \text{für } n > 0 \end{cases} \tag{E.1}$$

E.2.1 Struktogramm (3 Punkte)

Geben Sie ein Struktogramm (Nassi/Snneiderman-Notation) für die Berechnung des n-ten Glieds an. Verwenden Sie dabei die Notation entsprechend Abbildung E.2 Seite 258.

E.2.2 Wert: $n \leftarrow 3$ (3 Punkte)

Skizzieren Sie anhand Ihres Struktogrammes die Berechnung des Wertes für den Wert: $n \leftarrow 3$.

E.3 Aufgabe: ET Konsolidierung

Tabelle E.1 Seite 258 zeigt eine Entscheidungstabelle (ET).

[1] Bei Verwendung einer zusätzlichen Ablaufsteuerungsgröße erhalten Sie für Ihre richtige Lösung nur 5 Punkte.

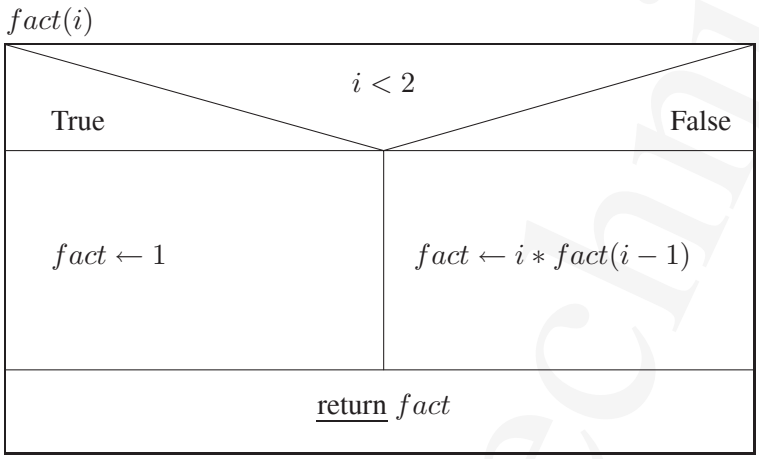


Abbildung E.2: Rekursives Struktogramm: Fakultätsfunktion

ET-Z		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16
B1	$p?$	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N
B2	$q?$	J	J	J	J	N	N	N	J	J	J	J	J	N	N	N	N
B3	$r?$	J	J	N	N	J	J	N	N	J	J	N	N	J	J	N	N
B4	$s?$	J	N	J	N	J	N	J	N	J	N	J	N	J	N	J	N
A1	A	X	X							X							
A2	B			X	X				X		X						
A3	C				X	X	X	X	X			X					
A4	D												X				
A5	E													X	X	X	X

Tabelle E.1: Begrenzte Eintreffer-ET

E.3.1 Typangabe (2 Punkte)

Geben Sie den Typ dieser ET an.

E.3.2 Widerspruchsprüfung (4 Punkte)

Enthält diese ET einen formalen Widerspruch? Wenn ja, geben Sie die betroffenen Regelbezeichner an und lösen Sie den Widerspruch durch zusätzliches Markieren von Aktionen auf.

E.3.3 Vollständigkeitsprüfung (2 Punkte)

Prüfen Sie diese ET auf Vollständigkeit.

E.3.4 Konsolidierung (6 Punkte)

Konsolidieren Sie Ihre korrigierte ET.

E.4 Aufgabe: Label Structure Program

Die Abbildung E.3 Seite 260 zeigt einen Programmablaufplan (PAP). [Hinweis: T ≡ True; F ≡ False]

E.4.1 Label Structure Program (12 Punkte)

Überführen Sie diesen in ein reduziertes label structure program. Fassen Sie vorher elementare Bausteine zusammen.

E.4.2 Struktogramm (8 Punkte)

Zeichnen Sie Ihr reduziertes label structure program als Struktogramm (Nassi/Shneiderman-Diagramm). (Hinweis: Umkonstruktionen sind nicht erforderlich.)

E.5 Aufgabe: Programmentwurf

Ein Tankstellenbesitzer setzt für die Verwaltung seines relativ kleinen Ersatzteillagers das weltweit verbreitete Standardpaket RM7 (Realtime Materialverwaltung Version 7) auf seinem PC ein. RM7 verwaltet die Artikelstammdatei ARTIKEL-ST. In ARTIKEL-ST sind die aktuellen Umsätze pro Artikel bezogen auf die Zeiträumen „Monat“, „Jahr“und „letzte 5 Jahre“gespeichert.

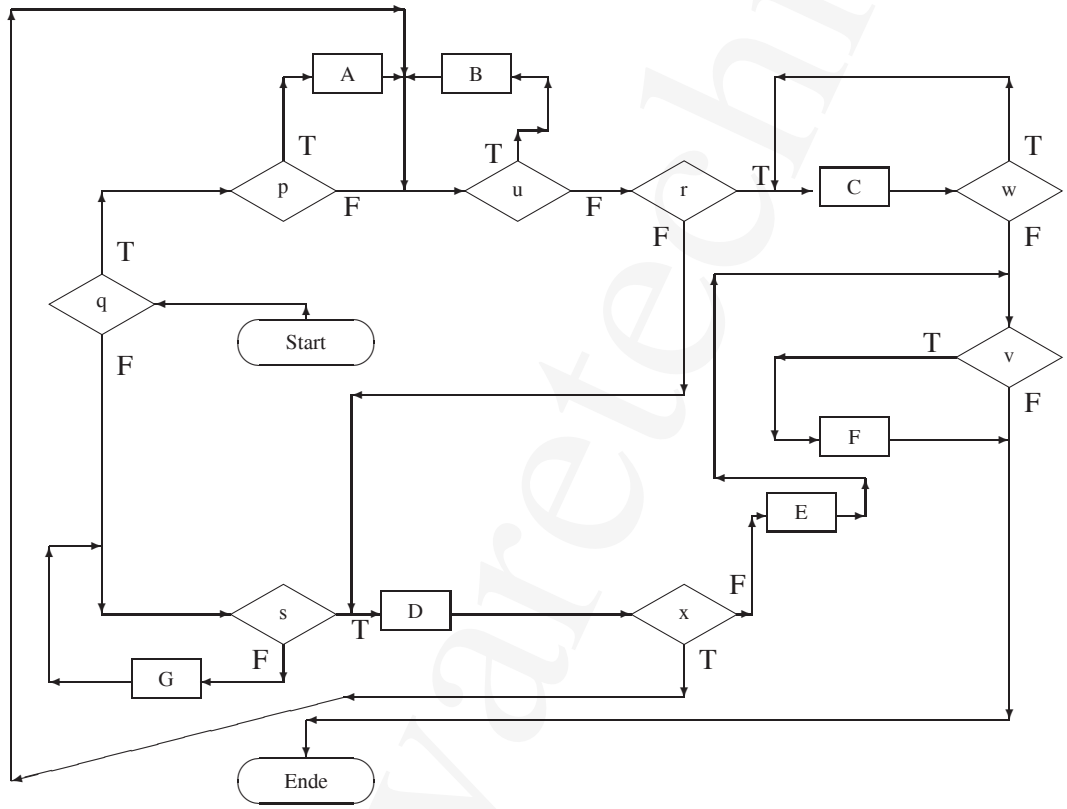


Abbildung E.3: Chaotische Kontrollstruktur

Der Tankstellenbesitzer will über die Artikel entscheiden, mit denen er bisher den geringsten Umsatz gemacht hat. Er stellt sich daher ein zusätzliches Auswertungsprogramm ATÜ (Artikelüberwachung) vor, das die zehn Artikel mit den geringsten Umsätzen selektiert und deren Bezeichnungen mit den Umsätzen anzeigt. Bei jedem dieser angezeigten Artikel will er durch Eingabe markieren, ob er diesen Artikel weiterhin führen will oder nicht. Beim erneuten Start von ATÜ sollen schon so markierte Artikel nicht noch einmal berücksichtigt werden.

E.5.1 Entwurf mit SA (14 Punkte)

Skizzieren Sie einen Entwurf für ATÜ unter der Berücksichtigung, daß ARTIKEL-ST nicht von ATÜ verändert werden kann. Dokumentieren Sie Ihre Skizze mit Hilfe von SA (Structured Analysis).

E.5.2 Verbesserungsvorschläge (4 Punkte)

Geben Sie Verbesserungsvorschläge für den Leistungsumfang von ATÜ an.

E.5.3 Entwicklungsumgebung (2 Punkte)

Mit welcher Softwareentwicklungsumgebung kann ATÜ zweckmäßig realisiert werden. Begründen Sie Ihren Vorschlag.

E.6 Aufgabe: ET-Verbundsystem

Die Abbildung E.4 auf Seite 262 zeigt die Kontrollstruktur eines Programms in PDL-Notation (Process Design Language - auch Pseudocode genannt). Diese Formulierung nutzt die drei Konstrukte: Sequenz, Alternative und Iteration. Beginn und Ende eines Programms sind mit den Kennworten proc (Procedure) und corp markiert. Das Kennwort run bezeichnet den Aufruf eines Programms.

E.6.1 ET-Verbund-Aufstellung (15 Punkte)

Transformieren Sie die Kontrollstruktur der Abbildung E.4 Seite 262 in Entscheidungstabellen vom Typ „Eintreffer“.

E.6.2 Konsolidierungsprüfung (5 Punkte)

Können Ihre Entscheidungstabellen konsolidiert werden? Wenn ja, führen Sie die Konsolidierung durch.

```
proc MODUL-X
  if
      p
    then
      if
          q
        then
          do
              A
            until
              r
          od
          B
        else
          C
      fi
    else
      run MODUL-Y
      D
  fi
E
corp
```

```
proc MODUL-Y
  if
      (s ∧ t)
    then
      F
    else
      if
          (¬t ∨ ¬u)
        then
          G
        else
          H
      fi
      K
  fi
corp
```

Abbildung E.4: Kontrollstruktur in PDL-Notation

```
LOOP.
        I ← (A + B)/2
        IF TAB(I) = C THEN GOTO TREFFER.
        IF TAB(I) < C THEN A ← I + 1.
        IF TAB(I) > C THEN B ← I − 1.
        IF (B − A) > 1 THEN GOTO LOOP.
        IF TAB(A) = C THEN GOTO TREFFER.
        IF TAB(B) = C THEN GOTO TREFFER.
        E ← 1.
        GOTO FERTIG.
TREFFER.
        E ← 2.
FERTIG.
        ENDE.
```

Abbildung E.5: Programmfragment in „Computer Esperanto“

E.6.3 Vollständigkeitsprüfung (2 Punkte)

Prüfen Sie, ob Ihre Entscheidungstabellen formal vollständig sind.

E.7 Aufgabe: Programmanalyse

Die Abbildung E.5 Seite 263 zeigt ein unstrukturiertes Programmfragment, das in einem allgemeinverständlichen „Computer Esperanto“geschrieben ist.

PAP-Erstellung (8 Punkte)
Analysieren Sie dieses Programmfragment. Erstellen Sie einen Programmablaufplan (PAP) für dieses Programmfragment.

E.8 Aufgabe: Programmentwurf

Die Samtgemeinde *Schönhausen* plant das Neubaugebiet *Heideweg* mit insgesamt 84 Bauplätzen. Aus Erfahrung geht man dabei von ca. 280 ernsthaften Interessenten aus. Zur Reduzierung des Verwaltungsaufwandes ist die Vergabe der Bauplätze mittels Rechnereinsatz zu unterstützen. Grundlage für das VERS (Vergabesystem für Bauplätze) ist eine Punkteregelung, wobei der Interessent, der die höchste Punktzahl erreicht, als erster den Zuschlag für seinen ausgewählten Bauplatz erhält. Ent-

sprechend der Reihenfolge der erzielten Punktzahl erfolgt die weitere Vergabe. Bei Punktgleichheit und gleichem ausgesuchten Bauplatz entscheidet das Los. Der Rat hat am 1. April 1993 folgende Punkteregelung beschlossen:

- 5 Punkte, wenn der Interessent Einwohner von *Schönhausen* im Sinne der Gemeindeordnung (GO) ist
- 3 Punkte, wenn der Interessent Familienangehörige in *Schönhausen* hat und selbst nicht Einwohner im Sinne der Gemeindeordnung ist
- 2 Punkte pro Kind des Interessenten im Sinne der Kindergeldregelung
- 3 Punkte pro behindertes Familienmitglied im Haushalt des Interessenten
- 4 Punkte, wenn der Interessent seine Arbeitsstelle in *Schönhausen* hat.

Die Verwaltung von *Schönhausen* verfügt über ein modernes LAN mit 8 Personalcomputern, das über eine Standleitung mit dem kommunalen Rechenzentrum (KRZ) *Wollburg* verbunden ist. Das Vermessungswesen ist mit dem landeseinheitlichen Standardverfahren ($VW2000$) weitgehend automatisiert. Die Grundstückskarte *Heideweg* ist daher im KRZ elektronisch verfügbar.

E.8.1 Leistungen von VERS (6 Punkte)

Präzisieren Sie die von VERS zu erbringenden Leistungen. [Hinweis: Bei fehlenden Informationen oder Interpretationsunsicherheit notieren Sie Ihre Annahmen.]

E.8.2 Entwurf mit SA (14 Punkte)

Skizzieren Sie einen Entwurf für Ihr VERS. Dokumentieren Sie Ihre Skizze mit Hilfe von **SA** (*Structured Analysis*).

E.8.3 Wiederverwendbarkeit (2 Punkte)

Skizzieren Sie Maßnahmen damit VERS auch für andere Neubaugebiete einsetzbar ist.

E.8.4 Entwicklungsumgebung (3 Punkte)

Mit welcher Softwareentwicklungsumgebung kann VERS zweckmäßig realisiert werden. Begründen Sie Ihren Vorschlag.

E.9 Aufgabe: ET Analyse

Tabelle E.2 Seite 266 zeigt eine begrenzte Eintreffer-Entscheidungstabelle (ET).

E.9.1 Redundanzprüfung (2 Punkte)

Enthält diese ET redundante Regeln? Wenn ja, dann geben Sie die Regelbezeichner an und beseitigen Sie die Redundanz durch das Streichen von einer oder mehreren Regel(n).

E.9.2 Widerspruchsprüfung (2 Punkte)

Enthält diese ET einen formalen Widerspruch? Wenn ja, geben Sie die betroffenen Regelbezeichner an und streichen Sie die Regel mit der kleineren Regelbezeichnernummer.

E.9.3 Vollständigkeitsprüfung (2 Punkte)

Prüfen Sie Ihre gegebenenfalls korrigierte ET auf formale Vollständigkeit.

E.9.4 Konsolidierung (4 Punkte)

Konsolidieren Sie Ihre gegebenenfalls korrigierte ET falls möglich.

E.9.5 ET-Vergleich (2 Punkte)

Tabelle E.3 Seite 266 zeigt eine Eintreffer-Entscheidungstabelle mit der ELSE-Regel (Regelbezeichner $\equiv$ R3). Vergleichen Sie diese ET mit Ihrer ET (von E.9.4). Geben Sie Unterschiede an, falls welche bestehen.

E.10 Aufgabe: Bedingungs-/Ereignisnetz

Abbildung E.6 Seite 267 zeigt ein Bedingungs-/Ereignisnetz.

ET-S		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16
B1	u ?	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N
B2	v ?	J	J	J	J	N	N	N	N	J	J	J	J	N	N	N	N
B3	w ?	J	J	N	N	N	J	N	N	J	J	N	N	J	N	N	N
B4	x ?	J	N	J	N	J	N	J	N	J	N	J	N	J	N	J	N
A1	A														X		X
A2	B		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
A3	C		X	X	X	X	X	X	X	X	X	X	X	X	X	X	
A4	D		X	X	X	X	X	X	X	X	X	X	X	X		X	
A5	E	X															
A6	F	X															

Tabelle E.2: Begrenzte Eintreffer-ET

ET-Alles		R1	R2	R3
B1	u ?	J	N	E
B2	v ?	J	N	L
B3	w ?	J	N	S
B4	x ?	J	N	E
A1	A		X	
A2	B		X	
A3	C			X
A4	D			X
A5	E	X		
A6	F	X		

Tabelle E.3: Eintreffer-ET mit ELSE-Regel

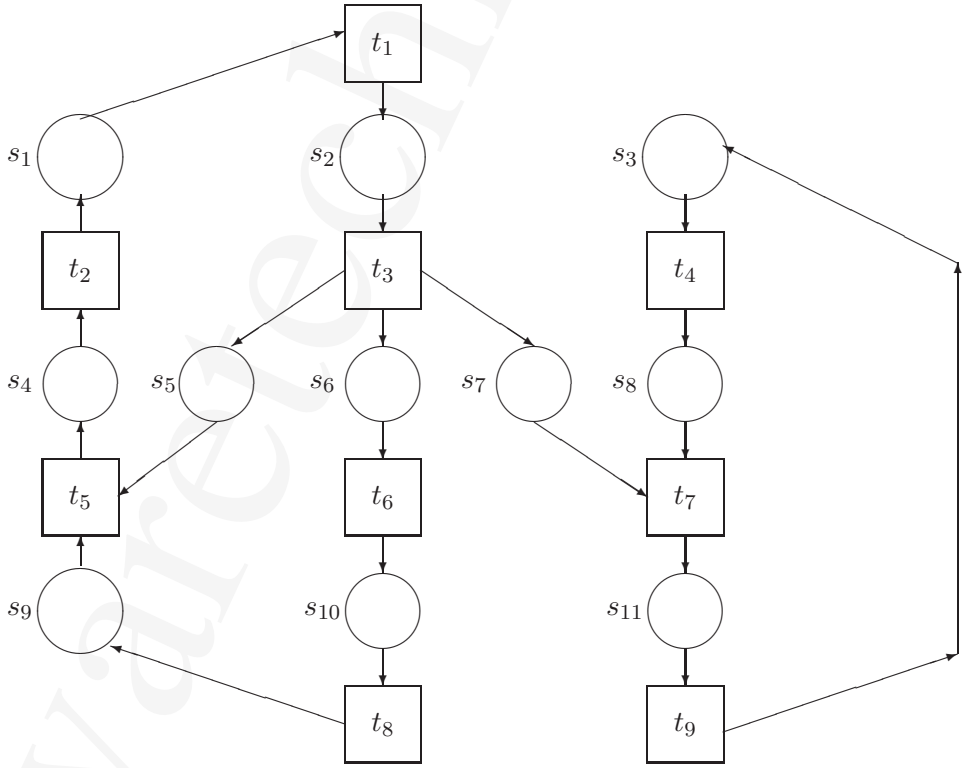


Abbildung E.6: Petri-Netz (Typ $\equiv$ Bedingungs-/Ereignisnetz)

E.10.1 Schaltregel (2 Punkte)

Notieren Sie die Schaltregel für ein Bedingungs-/Ereignis-Netz.

E.10.2 Schaltfolge (6 Punkte)

Geben Sie mögliche Schaltfolgen bei folgenden Anfangsmarkierungen an:
Fall 1: $M_0 \equiv s_2$
Fall 2: $M_0 \equiv s_2 + s_3$

E.11 Aufgabe: Petri-Netz-Analyse

Abbildung E.7 Seite 269 zeigt ein Petri-Netz mit der Anfangsmarkierung Markierung M_0.

E.11.1 Schaltkonzession von t_4 (3 Punkte)

Beschreiben Sie warum die Transition t_4 Schaltkonzession hat oder nicht hat.

E.11.2 Marken auf $s_{4...6}$ (6 Punkte)

Wie oft kann die Transition bei der vorgegebenen M_0-Markierung schalten. Dokumentieren Sie die Marken auf den Stellen $s_{4...6}$, wenn t_4 keine Schaltkonzession mehr hat.

E.12 Aufgabe: Kontrollstruktur-Linearisierung

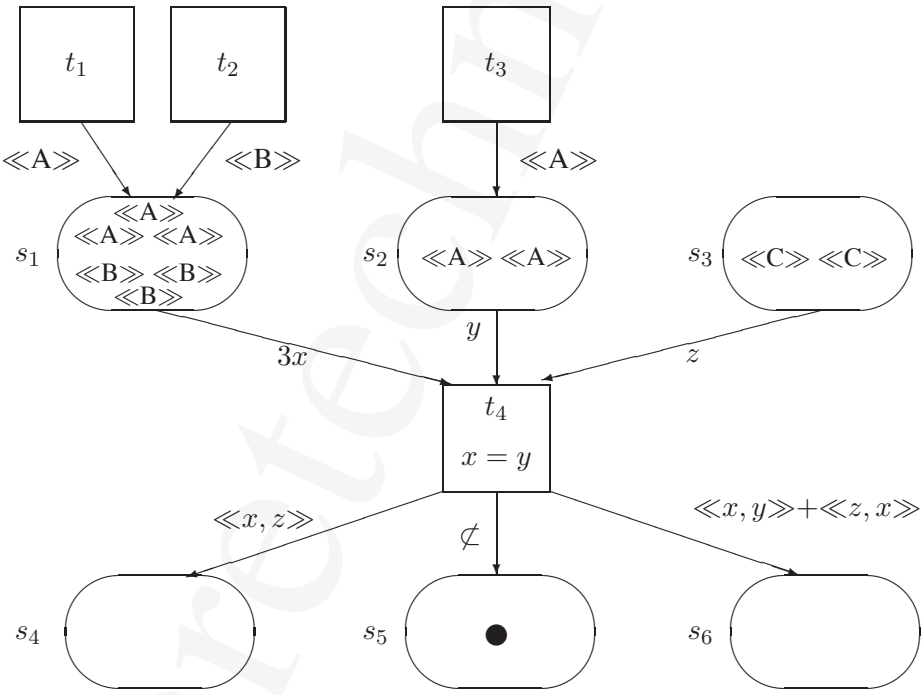
Die Abbildung E.8 Seite 270 zeigt einen Programmablaufplan. [Hinweis: T $\equiv$ True; F $\equiv$ False].

E.12.1 Label Structure Program (12 Punkte)

Überführen Sie diesen PAP in ein reduziertes *label structure program*. Fassen Sie vorher elementare Konstrukte zusammen.

E.12.2 Struktogramm (8 Punkte)

Zeichen Sie Ihr reduziertes *label structure program* als Struktogramm (Nassi/Shneiderman-Diagramm).



Legende:

$s_{1...6}$	≡ Stellen
$t_{1...4}$	≡ Transitionen
$=$	≡ Gleichheit
$\not\subset$	≡ Fluß einer anonymen Marke
≪A≫	≡ Beispiel einer unterscheidbaren Marke
≪A,B≫	≡ Beispiel einer strukturierten Marke
≪x,y≫ + ≪z,x≫	≡ Beispiel einer Konstruktionsvorschrift

Abbildung E.7: Petri-Netz „höherer Ordnung“

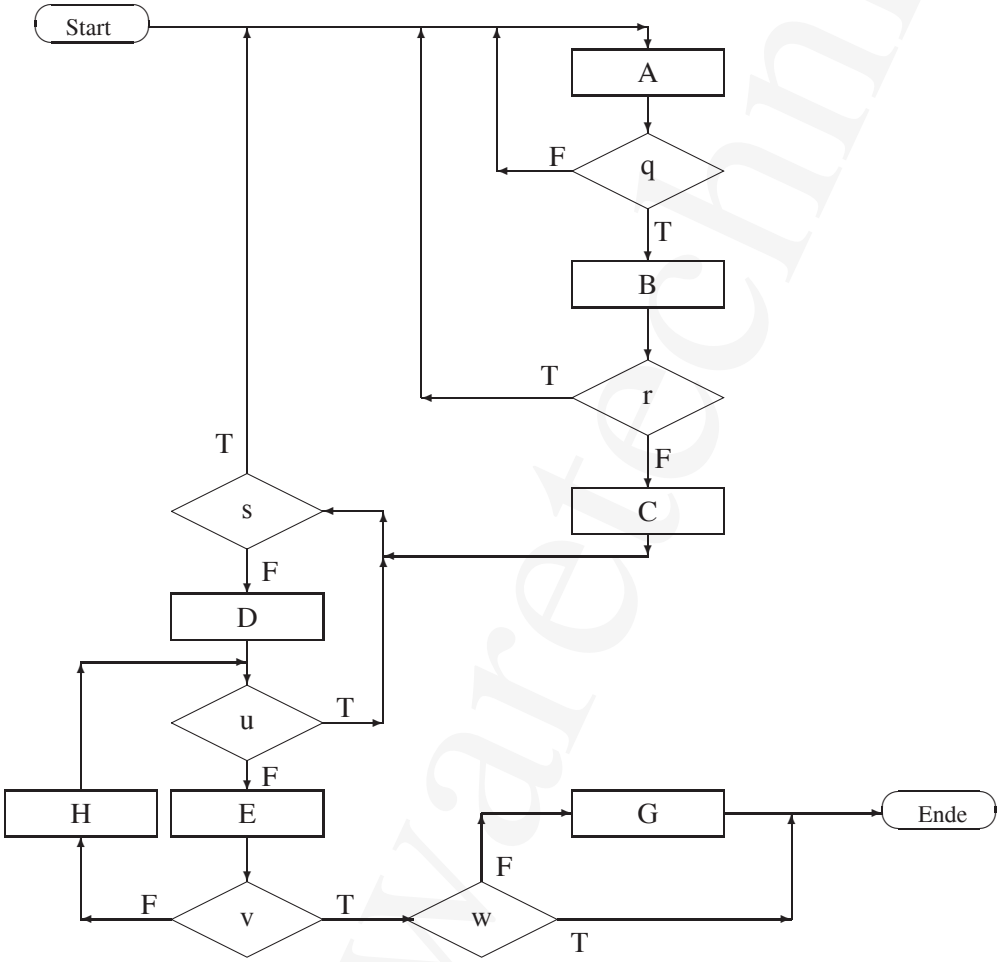


Abbildung E.8: Nichtlineare Kontrollstruktur

E.13 Aufgabe: COBOL-Quellcodeanalyse

Analysieren Sie das COBOL-Programm EAN anhand der folgenden Quellcodeliste (IBM AIX VS COBOL).

```
* IBM AIX VS COBOL Compiler/6000 LP                    30-Jun-93 11:07 Page
*                                    ean.cbl
* Options: int(ean.int)
      *    PROGRAMM EAN (EUROPAEISCHE ARTIKELNUMMER) ERZEUGT
      *    DRUCKAUFBEREITETE DATEI MIT UMSATZ PRO LIEFERANT
      *    -- REVIEW EXAMPLE.
       IDENTIFICATION DIVISION.
       PROGRAM-ID.
           EAN.
       AUTHOR.
           H.E.G.BONIN.
       INSTALLATION.
           FH NORDOSTNIEDERSACHSEN D-2120 LUENEBURG.
       DATE-WRITTEN.
           30-Jun-93.
       DATE-COMPILED. 30-Jun-93 11:07.
       SECURITY.
           LOW.

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.
           IBM-RISC-6000.
       OBJECT-COMPUTER.
           IBM-RISC-6000.
       SPECIAL-NAMES.
           C01 IS TO-PAGE-START.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT EAN-INPUT      ASSIGN TO "ean.in"
           ORGANIZATION IS LINE SEQUENTIAL.
           SELECT EAN-REPORT     ASSIGN TO "ean.rpt"
           ORGANIZATION IS LINE SEQUENTIAL.

       DATA DIVISION.
       FILE SECTION.

       FD  EAN-INPUT.
       01  EAN-RECORD.
           05  BUCHUNGSZEITPUNKT.
               10  INDUSTRIE-TAG       PIC 999.
               10  UHRZEIT.
                   15  STUNDEN         PIC 99.
                   15  MINUTEN         PIC 99.
                   15  SEKUNDEN        PIC 99.
           05  LIEFERANT               PIC 99.
           05  EAN-NUMBER.
```

```
               10  NATION              PIC 99.
               10  BETRIEBS-NUMBER     PIC 9(5).
               10  ARTIKEL             PIC 9(5).
               10  PRUEFZIFFER         PIC 9.
           05  MENGE                   PIC 99.
           05  BETRAG                  PIC 9(5)V99.

       FD  EAN-REPORT.
       01  REPORT-RECORD               PIC X(80).

       WORKING-STORAGE SECTION.

       01  FLAGS.
```



```
           05  MORE-DATA-FLAG          PIC XXX VALUE 'YES'.
               88  MORE-DATA                   VALUE 'YES'.
               88  NO-MORE-DATA                VALUE 'NO'.

       01  LAYOUT.
           05 MAX-LINE-NUMBER          PIC S99 VALUE +60.

       01  AUXILIARY.
           05 LINE-NUMBER              PIC S99  VALUE +1.
           05 PAGE-NUMBER              PIC S999 VALUE +1.
           05 VORIGE-LIEFERANT         PIC 99.

       01  UMSATZ.
           05 LIEFERANT-UMSATZ         PIC S9(6)V99 VALUE ZERO.
           05 GESAMT-UMSATZ            PIC S9(7)V99 VALUE ZERO.

       01  BODY-LINE.
           05  VORSCHUB-ZEICHEN        PIC X.
           05  BUCHUNGSZEITPUNKT-OUT.
               10  INDUSTRIE-TAG-OUT   PIC 999.
               10  FILLER              PIC X.
               10  UHRZEIT.
                   15  STUNDEN-OUT     PIC 99.
                   15  MINUTEN-OUT     PIC 99.
                   15  SEKUNDEN-OUT    PIC 99.
                   15  FILLER          PIC X.
           05  EAN-OUT.
               10  NATION-OUT          PIC 99.
               10  FILLER              PIC X.
               10  BETRIEBS-NUMBER-OUT PIC 9(5).
               10  FILLER              PIC X.
               10  ARTIKEL-OUT         PIC 9(5).
               10  FILLER              PIC X.
               10  PRUEFZIFFER-OUT     PIC 9.
           05  FILLER                  PIC X.
           05  LIEFERANT-OUT           PIC 99.
```

```
           05  FILLER                  PIC X.
           05  MENGE-OUT               PIC 99.
           05  FILLER                  PIC X.
           05  BETRAG-OUT              PIC ZZ,ZZ9.99.
           05  LIEFERANT-UMSATZ-OUT    PIC ZZZ,ZZ9.99.
           05  GESAMT-UMSATZ-OUT       PIC Z,ZZZ,ZZ9.99.
           05  FILLER                  PIC X(6).

       01  HEAD-LINE.
           05  VORSCHUB-ZEICHEN        PIC X.
           05  FILLER                  PIC XXX   VALUE 'Tag'.
           05  FILLER                  PIC XXX   VALUE ' hh'.
           05  FILLER                  PIC XX    VALUE 'mm'.
           05  FILLER                  PIC XXX   VALUE 'ss '.
           05  FILLER                  PIC X(16)
               VALUE '      EAN      '.
           05  FILLER                  PIC XXX   VALUE 'LF.'.
           05  FILLER                  PIC XXX   VALUE 'MG.'.
           05  FILLER                  PIC X(10)
               VALUE '    Betrag'.
           05  FILLER                  PIC X(10)
               VALUE '   Umsatz '.
* IBM AIX VS COBOL Compiler/6000 LP                    30-Jun-93 11:07 Page
*                                        ean.cbl
           05  FILLER                  PIC X(12)
               VALUE 'Gesamtumsatz'.
           05  FILLER                  PIC XXX   VALUE ' S.'.
           05  PAGE-NUMBER-OUT         PIC ZZ9.

       PROCEDURE DIVISION.

       EAN-MAIN.
           OPEN INPUT  EAN-INPUT
                OUTPUT EAN-REPORT.
           PERFORM EAN-RECORD-READ.
           IF MORE-DATA
               MOVE LIEFERANT TO VORIGE-LIEFERANT
               PERFORM EAN-RECORD-PROC
                   UNTIL NO-MORE-DATA
               PERFORM GESAMT-UMSATZ-PROC.
           CLOSE EAN-INPUT
                 EAN-REPORT.
           STOP RUN.

       EAN-RECORD-READ.
           READ EAN-INPUT
               AT END MOVE 'NO' TO MORE-DATA-FLAG.

       EAN-RECORD-PROC.
           IF LIEFERANT IS NOT EQUAL TO VORIGE-LIEFERANT
               PERFORM LIEFERANT-UMSATZ-PROC.
```

```
           PERFORM BODY-LINE-LOAD.
           PERFORM LINE-OUTPUT.
           ADD BETRAG TO LIEFERANT-UMSATZ
                         GESAMT-UMSATZ.
           PERFORM EAN-RECORD-READ.

       GESAMT-UMSATZ-PROC.
           PERFORM LIEFERANT-UMSATZ-PROC.
           MOVE SPACES TO BODY-LINE.
           MOVE GESAMT-UMSATZ TO GESAMT-UMSATZ-OUT.
           PERFORM LINE-OUTPUT.

       LIEFERANT-UMSATZ-PROC.
           MOVE SPACES TO BODY-LINE.
           MOVE VORIGE-LIEFERANT TO LIEFERANT-OUT.
           MOVE LIEFERANT-UMSATZ TO LIEFERANT-UMSATZ-OUT.
           PERFORM LINE-OUTPUT.
           MOVE LIEFERANT TO VORIGE-LIEFERANT.
           MOVE ZERO TO LIEFERANT-UMSATZ.

       LINE-OUTPUT.
           IF LINE-NUMBER = 1
                MOVE PAGE-NUMBER TO PAGE-NUMBER-OUT
                WRITE REPORT-RECORD FROM HEAD-LINE
                    AFTER ADVANCING TO-PAGE-START
                MOVE SPACES TO REPORT-RECORD
                WRITE REPORT-RECORD AFTER ADVANCING 2 LINES
                MOVE 4 TO LINE-NUMBER
                ADD 1 TO PAGE-NUMBER.
           WRITE REPORT-RECORD FROM BODY-LINE AFTER
                ADVANCING 1 LINES.

* IBM AIX VS COBOL Compiler/6000 LP                  30-Jun-93 11:07 Page   4
*                                        ean.cbl
           IF LINE-NUMBER = MAX-LINE-NUMBER
                MOVE 1 TO LINE-NUMBER
              ELSE
                ADD 1 TO LINE-NUMBER.

       BODY-LINE-LOAD.
           MOVE SPACES          TO BODY-LINE.
           MOVE INDUSTRIE-TAG   TO INDUSTRIE-TAG-OUT.
           MOVE STUNDEN         TO STUNDEN-OUT.
           MOVE MINUTEN         TO MINUTEN-OUT.
           MOVE SEKUNDEN        TO SEKUNDEN-OUT.
           MOVE NATION          TO NATION-OUT.
           MOVE BETRIEBS-NUMBER TO BETRIEBS-NUMBER-OUT.
           MOVE ARTIKEL         TO ARTIKEL-OUT.
           MOVE PRUEFZIFFER     TO PRUEFZIFFER-OUT.
           MOVE LIEFERANT       TO LIEFERANT-OUT.
           MOVE MENGE           TO MENGE-OUT.
           MOVE BETRAG          TO BETRAG-OUT.
```

```
      *
      * ******** END OF PROGRAM FILE **********************

* IBM AIX VS COBOL Compiler/6000 LP
* Micro Focus COBOL/2                v1.1 revision 001 Compiler
* Copyright (c) 1984, 1989 Micro Focus Ltd.    URN AXUCJ/ET0/00001H
*                                               REF CNB-001047057D4-AXUKD18
*
* Total Messages:     0
* Data:        1340     Code:        659     Dictionary:        2571
* Licensed Material - Property of IBM
```

E.13.1 EAN-Kontrollstruktur (12 Punkte)

Dokumentieren Sie die Kontrollstruktur zum Beispiel in Form eines PAP (Programmablaufplanes) oder eines Struktogrammes (Nassi/Shneiderman-Diagrammes). [Hinweise: Die Formulierungen
`PERFORM` *prozedurname* `UNTIL` *bedingung*
`PERFORM` *prozedurname* `WITH TEST BEFORE UNTIL` *bedingung*
sind funktionsgleich. Bei Verwendung der `AFTER`-Option in der `WRITE`-Anweisung wird erst ein Vorschub ausgeführt und danach ein Datensatz geschrieben. Im Paragraphen `SPECIAL-NAMES` bewirkt `C01 IS` <mnemonic-name> für eine Anweisung `ADVANCING` <mnemonic-name> einen Vorschub zum Seitenanfang.]

E.13.2 EAN Output-Skizze (8 Punkte)

Skizzieren Sie das Ergebnis `ean.rpt`, wenn EAN ausgeführt wurde.

E.13.3 EAN Input-Skizze (3 Punkte)

In welche Sortierung sollte die Datei `ean.in` vorliegen?

E.13.4 Verbesserungsvorschläge (3 Punkte)

Kann EAN verbessert werden, zum Beispiel in Hinblick auf bessere Lesbarkeit von `ean.rpt`? Nennen Sie Ihre Vorschläge und skizzieren Sie, wie diese realisiert werden können.

E.14 Aufgabe: Formalisierung

Der Lagerverwalter der Elektronik GmbH & Co KG, Herr Franz Krause, beobachtet an seinem Arbeitsplatz folgenden Sachverhalt:

Von einem Startzeitpunkt $t_0 = 0$ bis zu einer Zeitspanne t_s ist die Lagermenge m_l im Zwischenlager „ZLAGER-II“ proprotional zur Zeit. Zum Zeitpunkt t_s ist $m_l = m_{max}$. In der Zeitspanne von t_s bis $2t_s$ sinkt die Lagermenge auf den Wert $m_l = \frac{1}{2} * m_{max}$. Zum Zeitpunkt $2t_s$ fällt sie schlagartig wieder auf 0. Dieses Verhalten wiederholt sich in der Zeitspanne von $2t_s$ bis $4t_s$ usw.

E.14.1 Funktionsskizze (2 Punkte)

Skizzieren Sie den Verlauf der Lagermenge m_l in Abhängigkeit von der Zeit in einem xy-Koordinatensystem.

E.14.2 Struktogramm aufstellen (6 Punkte)

Bilden Sie diesen Sachverhalt mit einem Struktogramm (Nassi/Shneiderman-Diagramm) ab.

E.15 COBOL Programmdokumentation

Das COBOL-Programm AUFTRAG dient zur Erfassung von Auftragsdaten. Nach dem Compilieren mit dem „IBM AIX VS COBOL“ Compiler auf einer RS/6000-Workstation erhält man folgende Liste.

```
* IBM AIX VS COBOL Compiler/6000 LP                    11-Jan-94 09:07 Page   1
*                                    auftrag.cbl
* Options: int(auftrag.int)
      *    PROGRAMM AUFTRAG ERFASST AUFTRAEGE VON KUNDEN
      *    UND SCHREIBT DIESE IN DIE DATEI AUFTRAG.DAT
      *    -- SEHR EINFACHES PROGRAMM FUER SCHULUNGSZWECKE --.
      *
       IDENTIFICATION DIVISION.
       PROGRAM-ID.
           AUFTRAG.
       AUTHOR.
           H.E.G.BONIN.
       INSTALLATION.
           FH NORDOSTNIEDERSACHSEN D-21339 LUENEBURG.
       DATE-WRITTEN.
           07-SEP-93.
       DATE-COMPILED. 11-Jan-94 09:07.
       SECURITY.
           LOW.
      *
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.
```

```
           IBM-RISC-6000.
       OBJECT-COMPUTER.
           IBM-RISC-6000.
       SPECIAL-NAMES.
           CONSOLE IS CRT.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT AUFTRAG-INPUT ASSIGN TO "auftrag.dat"
           ORGANIZATION IS LINE SEQUENTIAL.
      *
       DATA DIVISION.
       FILE SECTION.
      *
       FD  AUFTRAG-INPUT.
       01  AUFTRAG-RECORD.
           05  KUNDE.
               10  KUNDE-ID              PIC 9(5).
           05  ARTIKEL.
               10  ARTIKEL-ID            PIC 9(2).
               10  MENGE                 PIC 9(4).
      *
       PROCEDURE DIVISION.
      *
       VORLAUF.
           OPEN EXTEND AUFTRAG-INPUT.
      *
       MAIN.
           PERFORM WITH TEST AFTER UNTIL KUNDE-ID = ZERO
             DISPLAY SPACES
             DISPLAY
               "A U F T R A G S E R F A S S U N G"
                                        AT 0810 UPON CRT-UNDER
               "Kunden-ID/00000: [     ]"  AT 1110
               "Artikel-ID:         [  ]"  AT 1310
               "Menge in Stueck:  [    ]"  AT 1510
             ACCEPT KUNDE-ID              AT 1128
             IF KUNDE-ID NOT=ZERO
* IBM AIX VS COBOL Compiler/6000 LP                    11-Jan-94 09:07 Page
*                                 auftrag.cbl
               THEN
                 MOVE ZERO TO ARTIKEL-ID MENGE
                 ACCEPT ARTIKEL-ID        AT 1331
                 ACCEPT MENGE             AT 1529
                 WRITE AUFTRAG-RECORD
               ELSE CONTINUE
             END-IF
           END-PERFORM.
      *
       NACHLAUF.
           CLOSE AUFTRAG-INPUT.
           STOP RUN.
```

```
      *
      * ******** END OF PROGRAM FILE *********************

* IBM AIX VS COBOL Compiler/6000 LP
* Micro Focus COBOL/2                v1.1 revision 001 Compiler
* Copyright (c) 1984, 1989 Micro Focus Ltd.    URN AXUCJ/ET0/00001H
*                                               REF CNB-001047057D4-AXUKD18
*
* Total Messages:     0
* Data:         948     Code:         386     Dictionary:         536
* Licensed Material - Property of IBM
```

E.15.1 Dokumentation (10 Punkte)

Skizzieren Sie die Dokumentation für das Programm AUFTRAG. Adressat Ihrer Dokumentation ist der Softwareingenieur, der das Programm AUFTRAG pflegt und weiterentwickelt. [Hinweis: Geben Sie auch das Inhaltsverzeichnis der Dokumentation an.]

E.15.2 Kontrollstruktur (6 Punkte)

Ihre Dokumentation (von E.15.1) sollte die Kontrollstruktur von AUFTRAG verdeutlichen. Bilden Sie diese Kontrollstruktur in Form eines geschlossenen Entscheidungstabellen-Verbundsystems ab. Bewerten Sie die Aussage:

> „Ein Entscheidungstabellen-Verbundsystem kann diese Kontrollstruktur im Vergleich zu Struktogrammen (Nassi/Shneiderman-Diagrammen) nur relativ kompliziert abbilden."

E.15.3 Input/Output-Beschreibung (4 Punkte)

Ihre Dokumentation sollte auch einen Beispielsdatensatz der Datei auftrag.dat aufweisen. Skizzieren Sie kurz eine Nutzung des Programms AUFTRAG, die einen solchen Beispielsdatensatz erzeugt. Geben Sie eine Möglichkeit im AIX-Umfeld an, sich den Inhalt der Datei auftrag.dat auf dem Bildschirm anzeigen zu lassen.

E.15.4 Verbesserungsvorschläge (4 Punkte)

Nennen Sie Erweiterungen und Verbesserungen für das COBOL-Programm AUFTRAG.

ET-FOO		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16
B1	p ?	J	J	J	J	J	J	J	J	N	N	N	N	J	N	N	N
B2	q ?	J	J	J	J	N	N	N	N	J	J	J	J	N	N	N	N
B3	r ?	J	J	N	N	J	J	N	N	J	J	N	N	J	J	N	N
B4	s ?	J	N	J	N	J	N	J	N	J	N	J	N	J	N	J	N
A1	A														X		X
A2	B	X													X		X
A3	C		X	X	X	X	X	X	X	X	X	X	X	X		X	
A4	D		X	X	X	X	X	X	X	X	X	X	X	X		X	
A5	E	X	X	X	X	X	X	X	X	X	X	X	X	X			
A6	F	X															

Tabelle E.4: Begrenzte Eintreffer-ET

E.16 Aufgabe: ET Analyse

Tabelle E.4 Seite 279 zeigt eine begrenzte Eintreffer-Entscheidungstabelle (ET).

E.16.1 Redundanzprüfung (2 Punkte)

Enthält diese ET redundante Regeln? Wenn ja, dann geben Sie die Regelbezeichner an und beseitigen Sie die Redundanz durch das Streichen von einer oder mehreren Regel(n).

E.16.2 Widerspruchsprüfung (2 Punkte)

Enthält diese ET einen formalen Widerspruch? Wenn ja, geben Sie die betroffenen Regelbezeichner an und streichen Sie die Regel mit der kleineren Regelbezeichnernummer.

E.16.3 Vollständigkeitsprüfung (2 Punkte)

Prüfen Sie Ihre gegebenenfalls korrigierte ET auf formale Vollständigkeit.

E.16.4 Konsolidierung (4 Punkte)

Konsolidieren Sie Ihre gegebenenfalls korrigierte ET, falls möglich.

E.16.5 ET-Vergleich (2 Punkte)

Tabelle E.5 Seite 280 zeigt eine Eintreffer-Entscheidungstabelle mit der ELSE-Regel (Regelbezeichner $\equiv$ R3). Vergleichen Sie diese „ET-Alles“ mit Ihrer „ET-FOO“ (von E.16.4). Geben Sie Unterschiede an, falls welche bestehen.

ET-Alles		R1	R2	R3
B1	p ?	J	N	E
B2	q ?	J	N	L
B3	r ?	J	N	S
B4	s ?	J	N	E
A1	A		X	
A2	B	X	X	
A3	C			
A4	D			X
A5	E	X		X
A6	F	X		

Tabelle E.5: Eintreffer-ET mit ELSE-Regel

E.17 Aufgabe: Kontrollstruktur-Linearisierung

Die Abbildung E.9 Seite 281 zeigt einen Programmablaufplan. [Hinweis: T $\equiv$ True; F $\equiv$ False].

E.17.1 Label Structure Program (12 Punkte)

Überführen Sie diesen PAP in ein reduziertes *label structure program*. Fassen Sie vorher elementare Konstrukte zusammen.

E.17.2 Struktogramm (8 Punkte)

Zeichen Sie Ihr reduziertes *label structure program* als Struktogramm (Nassi/Shneiderman-Diagramm).

E.18 Aufgabe: Petri-Netzanalyse

Abbildung E.10 Seite 282 zeigt ein S/T-Netz mit folgender Schaltregel:

> Die Transition t_i hat Schaltkonzession, wenn der Vorbereich $\triangleright t_i$ markiert ist. Der Nachbereich $t_i \triangleleft$ kann beliebig viele Marken aufnehmen.

E.18.1 Schaltfolge (6 Punkte)

Geben Sie mögliche Schaltfolgen bei folgenden Anfangsmarkierungen an:
Fall 1: $M_0 \equiv s_1 + s_5$
Fall 2: $M_0 \equiv s_1 + s_4 + s_6$

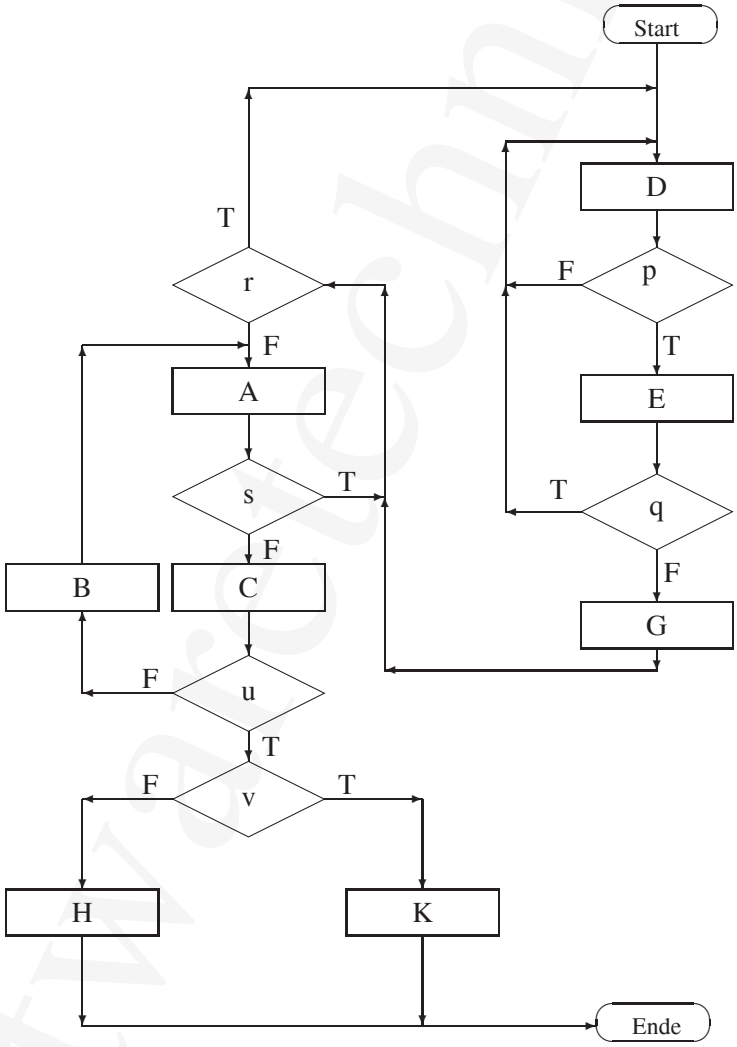


Abbildung E.9: Nichtlineare Kontrollstruktur

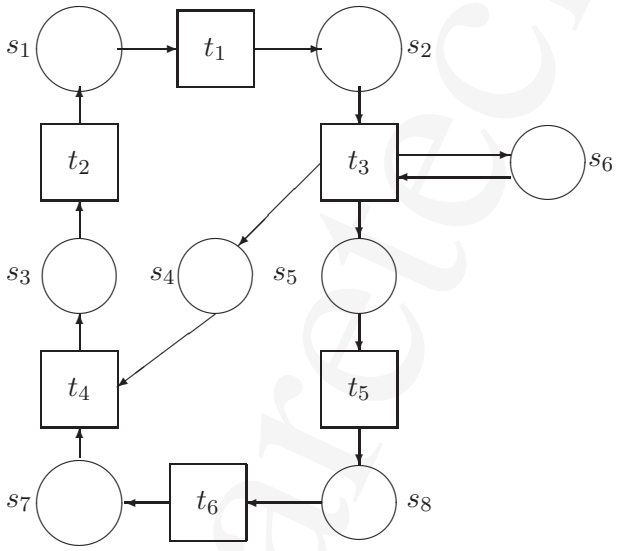


Abbildung E.10: Petrinetz (Typ $\equiv$ S/T-Netz)

E.18.2 Lebendigkeit (2 Punkte)

Ist das S/T-Netz bei einer Anfangsmarkierung von $M_0 \equiv s_2 + s_5 + s_6$ lebendig, wenn folgende geänderte Schaltregel unterstellt wird:

> Die Transition t_i hat Schaltkonzession, wenn der Vorbereich $\triangleright t_i$ markiert und der Nachbereich $t_i \triangleleft$ nicht markiert ist.

E.19 Aufgabe: Petri-Netzmodellierung

Ein Interpreter arbeitet üblicherweise in einem Zyklus mit den drei Phasen:

1. Konstrukt einlesen („READ“)
2. Konstrukt auswerten („EVAL“)
3. Konstruktergebnis ausgeben („PRINT“)

E.19.1 Petri-Netzentwurf (4 Punkte)

Skizzieren Sie diesen „READ-EVAL-PRINT“-Zyklus als ein Petrinetz. Sie können dazu die folgenden Stellen $s_{0...4}$ und die folgenden Transitionen $t_{0...3}$ verwenden.

s_0	$\equiv$ Konstrukte liegen vor
s_1	$\equiv$ Einlesebereitschaft gegeben
s_2	$\equiv$ Konstrukt ist eingelesen
s_3	$\equiv$ Konstrukt ist ausgewertet
s_4	$\equiv$ Konstruktergebnis ist ausgegeben
t_0	$\equiv$ Konstrukterzeugung
t_1	$\equiv$ READ
t_2	$\equiv$ EVAL
t_3	$\equiv$ PRINT

Nennen Sie den Typ ihres Petrinetzes und beschreiben Sie die Schaltregel. Geben Sie eine M_0-Markierung an, so daß die Transition Schaltkonzession hat, welche die „READ“-Phase (t_1) abbildet,

E.19.2 Petri-Netzverfeinerung (4 Punkte)

Die „EVAL“-Transistion (t_2) erkennt das Konstrukt „EXIT“, um den „READ-EVAL-PRINT“-Zyklus zu verlassen. Verfeinern Sie Ihr Petri-

netz (von E.19.1) entsprechend. Hat sich Ihr Petri-Netztyp geändert? Wenn ja, erläutern Sie die zugehörende Schaltregel.

E.20 Aufgabe: Systementwurf

Die Fitness-Studiokette „Workout and Fitness Generators Comp.“ plant die neue Studiogeneration „CF “ (*controlled fitness*). Die „CF“-Studios basieren primär auf einer stärkeren Computerunterstützung als die bisherigen Studios. Der massive Computereinsatz ermöglicht einen wesentlich besseren Trainingsservice — zum Beispiel:

- Für den Kunden kann ein individuell optimiertes Trainingsprogramm erstellt und laufend angepaßt werden.
- Der Kunde kann sich jederzeit über seinen Trainingsfortschritt informieren.
- Der Kunde kann durch vielfältige Anreize (Gebührenreduktion, öffentliche Bildschirmanzeige einer aktuellen„Ranking“-Liste, automatisches Foto bei persönlicher Bestleistung und vieles mehr) zur intensiveren Benutzung des Studios motiviert werden.

E.20.1 Leistungen von CF (6 Punkte)

Beschreiben Sie Ihre Vorschläge zum „CF“-Leistungsumfang in Form eines Pflichtenheftes, das die notwendige Software und Hardware spezifiziert.

E.20.2 Entwurf mit SA (14 Punkte)

Skizzieren Sie einen Entwurf für Ihr „CF“-Studio. Dokumentieren Sie Ihre Skizze mit SA (*Structured Analysis*).

Anhang F

Lösungen

Lösung Aufgabe E.1:
E.1.1:
Abbildung F.1 Seite 286 zeigt den mittels Funktionsduplizierung linearisierten PAP.

E.1.2:
Abbildung F.2 Seite 287 zeigt das Struktogramm.

Lösung Aufgabe E.2:
E.2.1:
Abbildung F.3 Seite 288 zeigt das rekursive Struktogramm.

E.2.2:

Aufruf:	**Berechnung:**
Entering:	
1. $a(3)$	$b \leftarrow a(2)$
	$a \leftarrow \frac{1}{2} * b + \frac{1}{b}$
2. $a(2)$	$b \leftarrow a(1)$
	$a \leftarrow \frac{1}{2} * b + \frac{1}{b}$
3. $a(1)$	$b \leftarrow a(0)$
	$a \leftarrow \frac{1}{2} * b + \frac{1}{b}$
4. $a(0)$	return 1
Leaving:	
5. $a(1)$	$b \leftarrow 1$
	return $\frac{3}{2}$
6. $a(2)$	$b \leftarrow \frac{3}{2}$
	return $\frac{3}{4} + \frac{2}{3} = \frac{17}{12}$

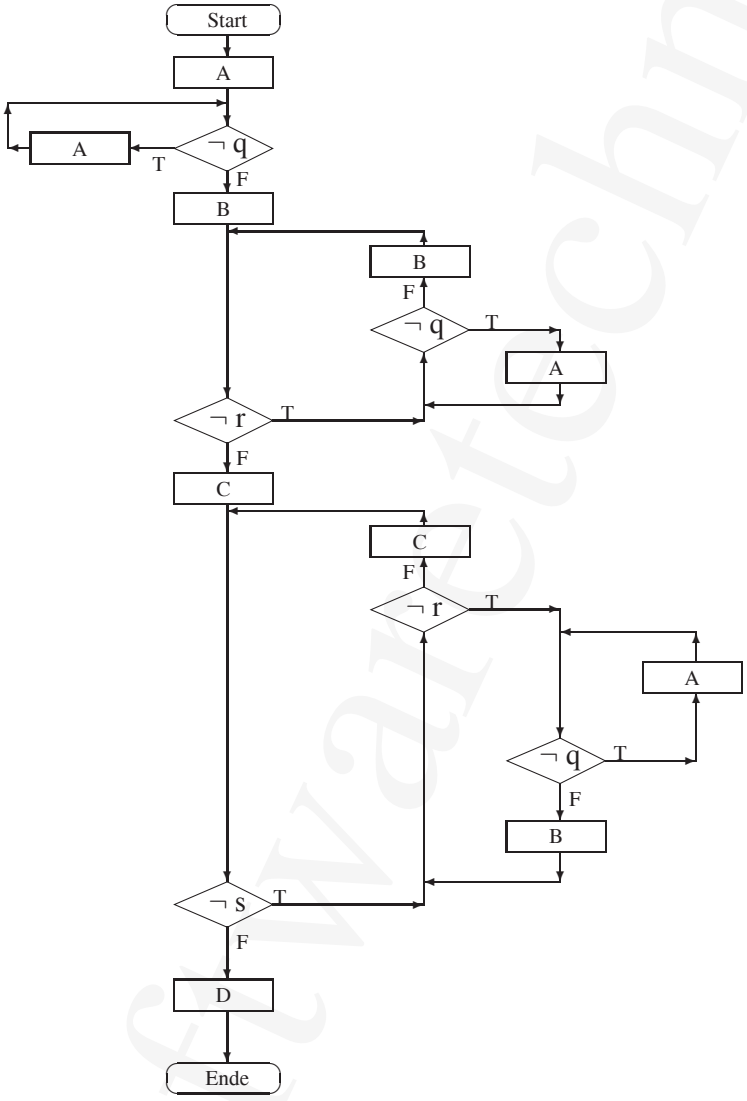
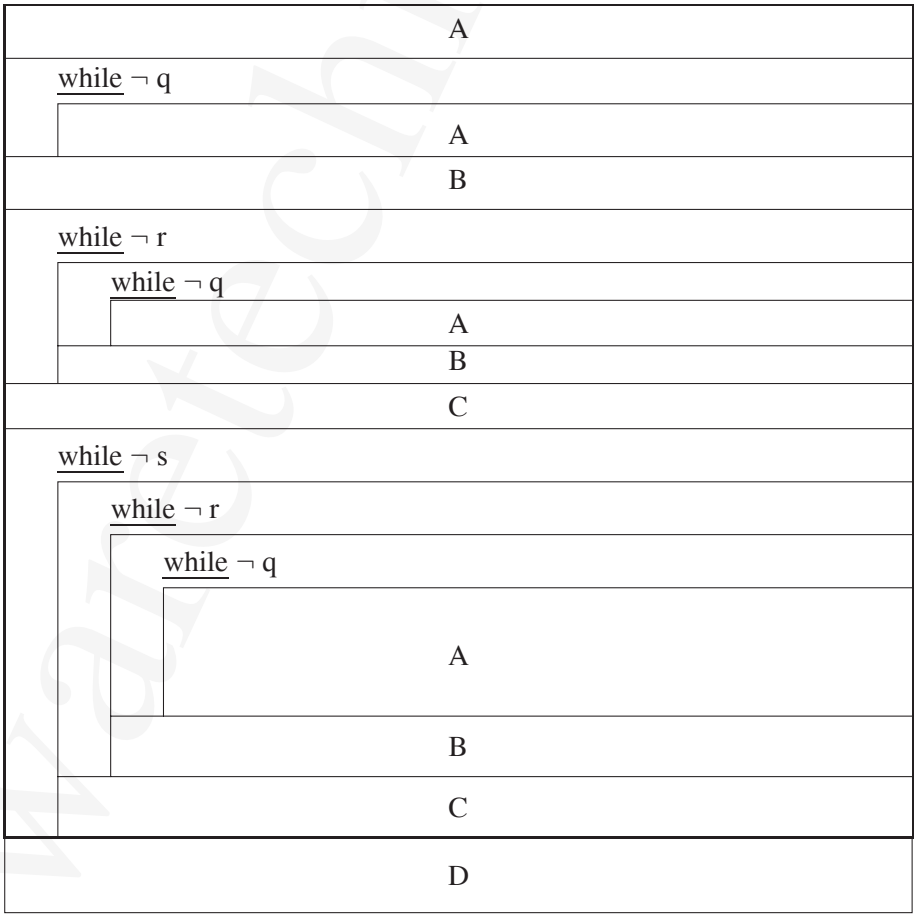


Abbildung F.1: Linearisierter PAP



Legende:
Vgl. Abbildung F.1 Seite 286

Abbildung F.2: PAP ⇒ Struktogramm

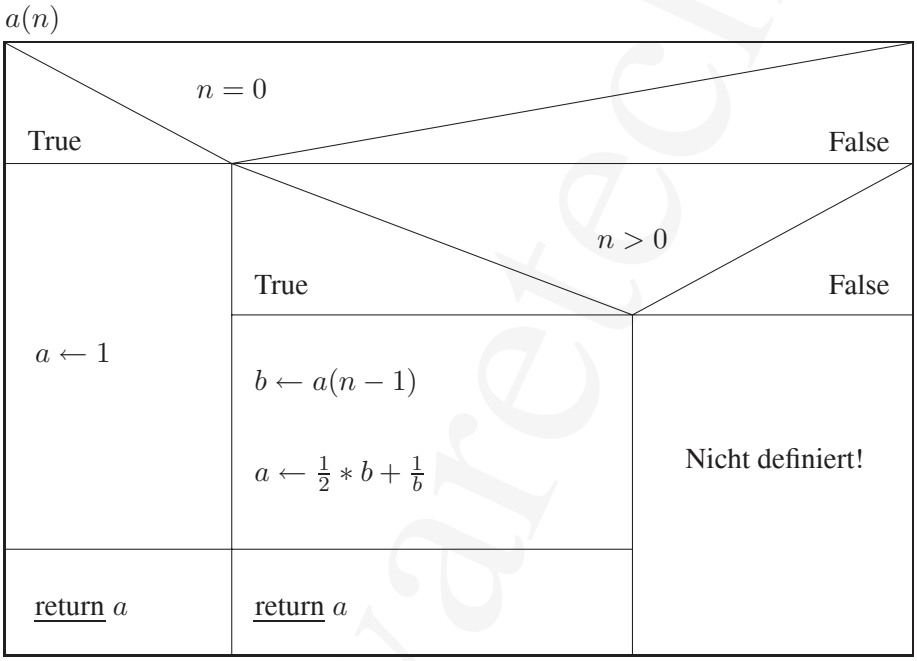


Abbildung F.3: Rekursives Struktogramm: Folge $a(n)$

7. $a(3)$ $b \leftarrow \frac{17}{12}$
return $\frac{17}{24} + \frac{12}{17} = \frac{577}{408} = 1.41421568627451$

Lösung Aufgabe E.3:

E.3.1:

Es handelt sich um eine begrenzte, einfache Eintreffer-ET. Sie ist bis auf Regel R8 in kanonischer Normalform notiert.

E.3.2:

Die Bedingungsanzeigerfolgen der Regeln R4 und R8 sind gleich, die Aktionsanzeigerfolgen nicht. R4 und R8 sind widersprüchlich. Der Widerspruch ist gelöst, wenn in R4 Aktion A3 und in R8 Aktion A2 markiert wird. Danach kann R8 (oder R4) gestrichen werden, da beide Regeln gleich sind. Für die folgende Betrachtung gilt R8 als gestrichen.

E.3.3:

4 Bedinginungen $\Rightarrow 2^4 = 16$ Regeln.
Nach Streichung der Redundanz enthält die ET nur noch 15 Regeln, sie ist daher unvollständig. Es fehlt die Regel mit der Bedingungsanzeigerfolge „J N N N“.

E.3.4:

1. Konsolidierungsdurchlauf:

$R1 \widehat{+} R2 \Rightarrow R1'$ mit B4 = - .
$R5 \widehat{+} R6 \Rightarrow R6'$ mit B4 = - .
$R13 \widehat{+} R14 \Rightarrow R13'$ mit B4 = - .
$R15 \widehat{+} R16 \Rightarrow R15'$ mit B4 = - .

2. Konsolidierungsdurchlauf:

$R13' \widehat{+} R15' \Rightarrow R13$“mit B3 = - .

Tabelle F.1 Seite 290 zeigt die konsolidierte ET.

Lösung Aufgabe E.4:

E.4.1:

Zunächst fassen wir die lineraren Teile des PAP zusammen (vgl. Abbildung F.4 Seite 291) und führen für diese Bezeichnungen mit einer pragmatischen Notation[1] Kennnummer (Zusammenfassungsschritt plus laufende Nummer) „/“ Anzahl der zusammengefaßten Knoten ein. In den PAP mit diesen Zusammenfassungen notieren wir Knotennummern und zwar so, daß der Endknoten die Nummer 0 hat (vgl. Abbildung F.5 Seite 292). Mit Hilfe einer zusätzlichen Variable L (*label*) geben wir

[1] Vgl. [42] S. 128.

ET-Z		R1'	R3	R4	R5'	R7	R9	R10	R11	R12	R13"
B1	$p?$	J	J	J	J	J	N	N	N	N	N
B2	$q?$	J	J	J	N	N	J	J	J	J	N
B3	$r?$	J	N	N	J	N	J	J	N	N	-
B4	$s?$	-	J	N	-	J	J	N	J	N	-
A1	A	X					X				
A2	B		X	X				X			
A3	C			X	X	X			X		
A4	D									X	
A5	E										X

Legende:
Vgl. Tabelle E.1 Seite 258.

Tabelle F.1: Konsolidierte ET

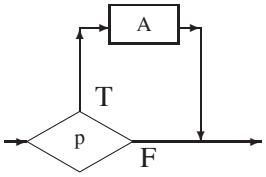
die Nummer des Nachfolgeknotens an (vgl. Abbildung F.6 Seite 293). Die L-Werte, die nicht mehrfach vorkommen, werden durch ihren „Verarbeitungszweig" ersetzt (Ausnahme bildet der Einstiegsknoten in das Netz, hier $L \leftarrow 1$.). Das so entstandene Diagramm ist das reduzierte *Label Structure Program* (vgl. Abbildung F.7 Seite 294).
E.4.2:
Da der PAP in Abbildung F.7 Seite 294 eine lineare Struktur darstellt, ist er direkt in ein Struktogramm übertragbar. Abbildung F.8 Seite 295 zeigt die Lösung.

Lösung Aufgabe E.5:
E.5.1:
Gemäß [23] bedingt *Structured Analysis* (SA):

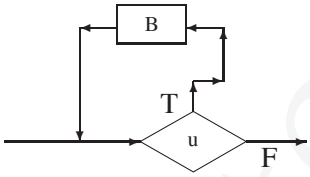
1. ein Kontextdiagramm,
2. eine Hierarchie von Datenflußdiagrammen, bis die einzelnen Prozesse mittels ihrer Minispezifikation hinreichend definiert sind, und
3. ein Datenlexikon.

Der ATÜ-Entwurf stellt sich damit in folgenden Abbildungen und Tabellen dar:

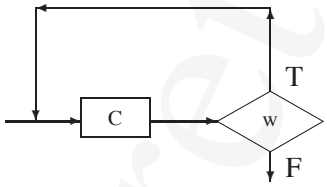
- ATÜ Kontextdiagramm vgl. Abbildung F.9 Seite 297
- ATÜ $Level_0$-Diagramm vgl. Abbildung F.10 Seite 298
- ATÜ $Level_1$-Diagramm vgl. Abbildung F.11 Seite 299
  - Prozeß 1.1: Selektieren-Artikel vgl. Abbildung F.12 Seite 300



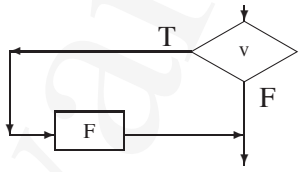
100/2 if p then A fi



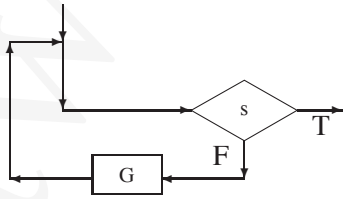
101/2 while u do B od



102/2 do C until ¬ w od



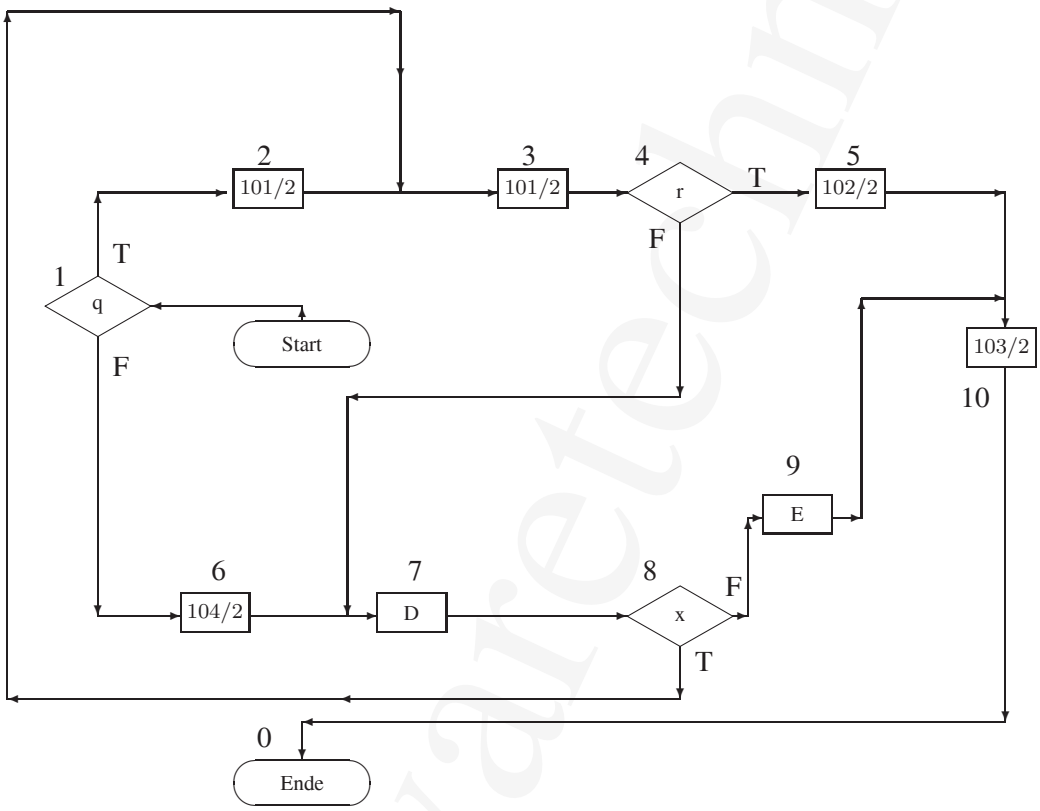
103/2 if v then F fi



104/2 while ¬ s do G od

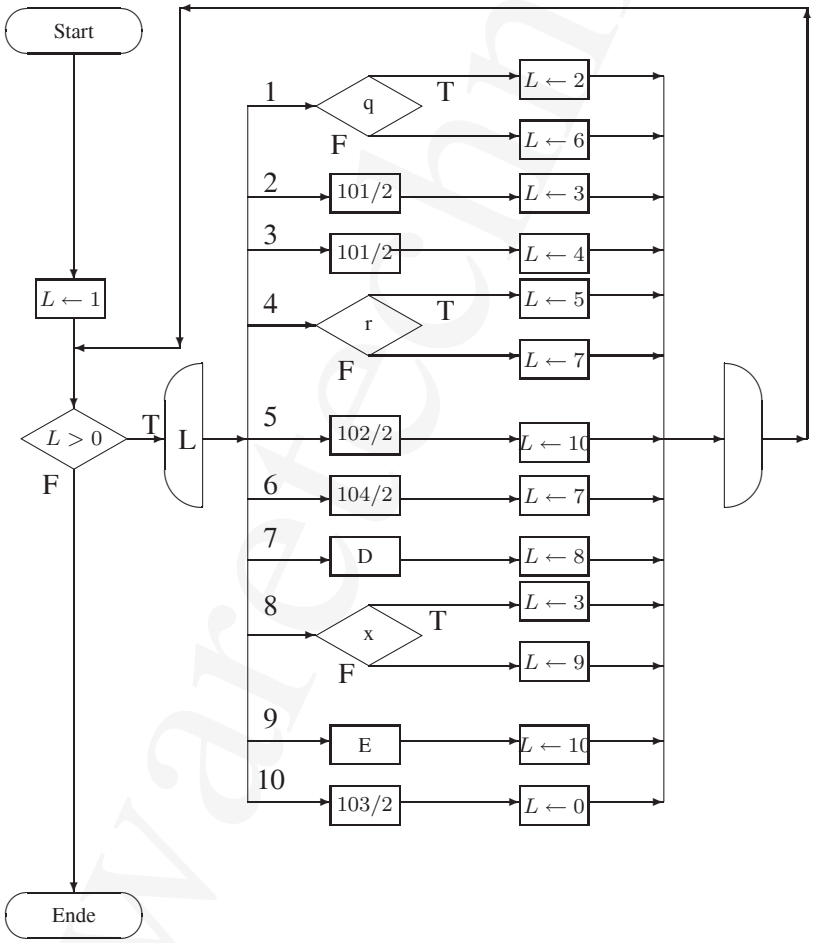
Legende:
Vgl. Abbildung E.3 Seite 260

Abbildung F.4: PAP: Zusammenfassung linearer Anteile



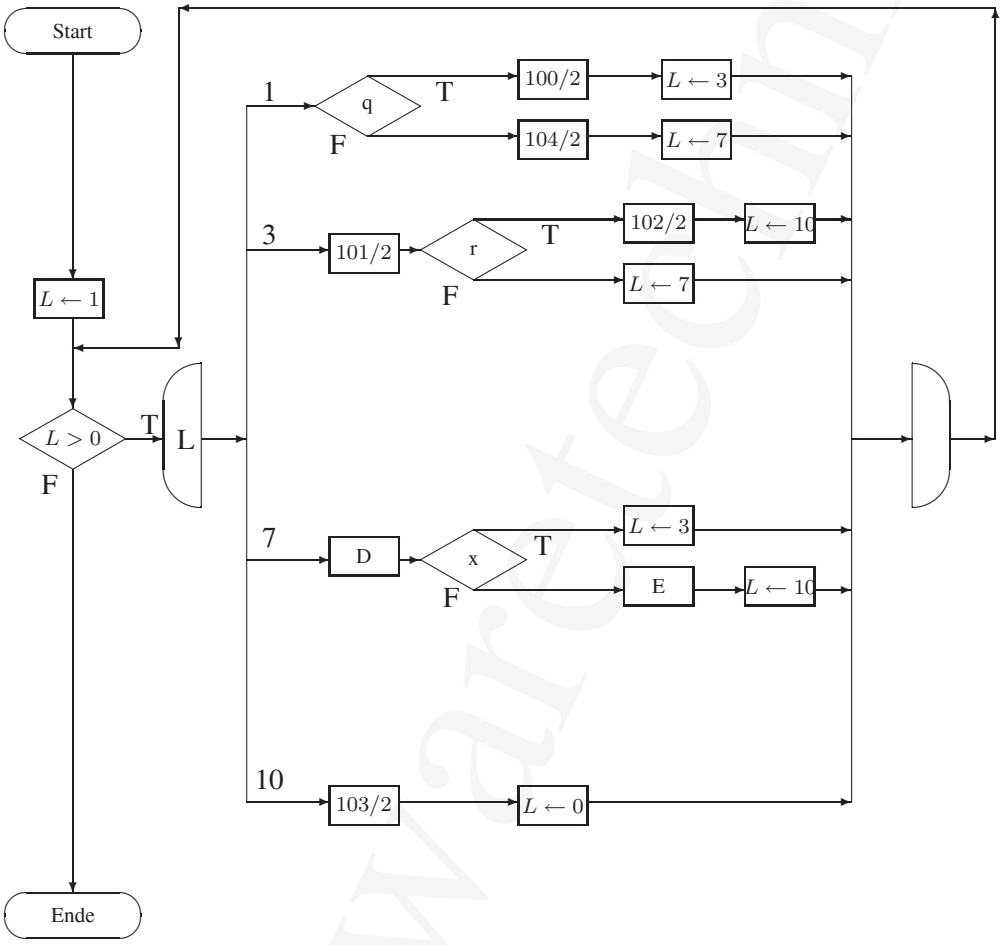
Legende:
Vgl. Abbildung E.3 Seite 260 und Abbildung F.4 Seite 291

Abbildung F.5: PAP: Einführung von Knotennummern



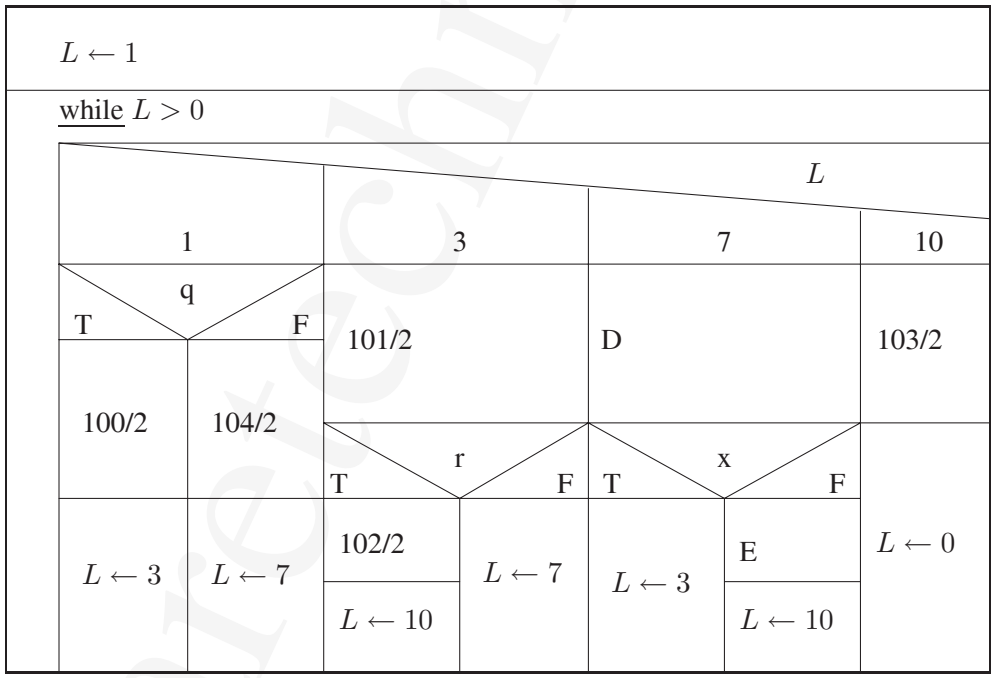
Legende:
Vgl. Abbildung E.3 Seite 260, Abbildung F.4 Seite 291 und Abbildung F.5 Seite 292.

Abbildung F.6: PAP: *Label Structure Program*



Legende:
Vgl. Abbildung F.6 Seite 293
Reduzierbar sind die L-Werte: 2,4,5,6,8,9

Abbildung F.7: PAP: Reduziertes *Label Structure Program*



Legende:
Vgl. Abbildung F.7 Seite 294

Abbildung F.8: PAP ⇒ Struktogramm

Nr.	**Identifier**	**Beschreibung**	**Typ/Werte**	**Beispiel**
1	ARTIKEL-ST	Artikelstammdatei	vgl. RM7	
2	LADENHUETER-ST	Ladenhüter-stammdatei		
2.1	ARTIKEL-NR	Artikelnummer	vgl. RM7	4711
2.2	MARK-ZEIT-M	Zeitraum Monat	alpha	F
2.3	MARK-ZEIT-J	Zeitraum Jahr	alpha	A
2.4	MARK-ZEIT-5J	Zeitraum 5 Jahre	alpha	A
3.	MARKIERUNG	Entscheidung des Händlers	alpha F ≡ Fortführen A ≡ Aufgeben	F
4.	ZEITRAUM	Wahl des Betrach-tungszeitraumes	alpha M ≡ Monat J ≡ Jahr L ≡ 5 Jahre	M
5.	U-SCHWACHE-ARTIKEL	Umsatzschwache Artikel		
5.1	ARTIKEL-NR	Artikelnummer	vgl. RM7	4711
5.2	ARTIKEL-NAME	Bezeichnung des Arikels	vgl. RM7	
5.3	U-ZEIT	Umsatz pro Zeiteinheit	numerisch	1203.12
6.	START	Aufruf von ATÜ		

Tabelle F.2: ATÜ: Datenlexikon

  - Prozeß 1.2: Markieren-Artikel vgl. Abbildung F.13 Seite 300

- ATÜ Datenlexikon vgl. Tabelle F.2 Seite 296

E.5.2:
Verbesserungsvorschläge:

1. ATÜ pflegt die Datei LADENHUETER-ST und ermöglicht damit die Korrektur einer vollzogenen Markierung.
2. ATÜ erstellt eine Liste aller markierten Artikel, sortiert nach Markierung und Umsatz.
3. ATÜ zeigt zusätzlich zum Umsatz die Absatzmengen und die Stückpreise (Einkauf & Verkauf) an.
4. ATÜ bietet die Möglichkeit die unterschiedlichen Bezugszeiträume zu gewichten (zum Beispiel 50% für Monats-, 30% für Jahres- und 20% für 5 Jahreswert) und berechnet eine Gesamtentscheidung.
5. Option einbauen (Parameter) für die Anzahl der selektierten umsatzschwächsten Artikel.

E.5.3:

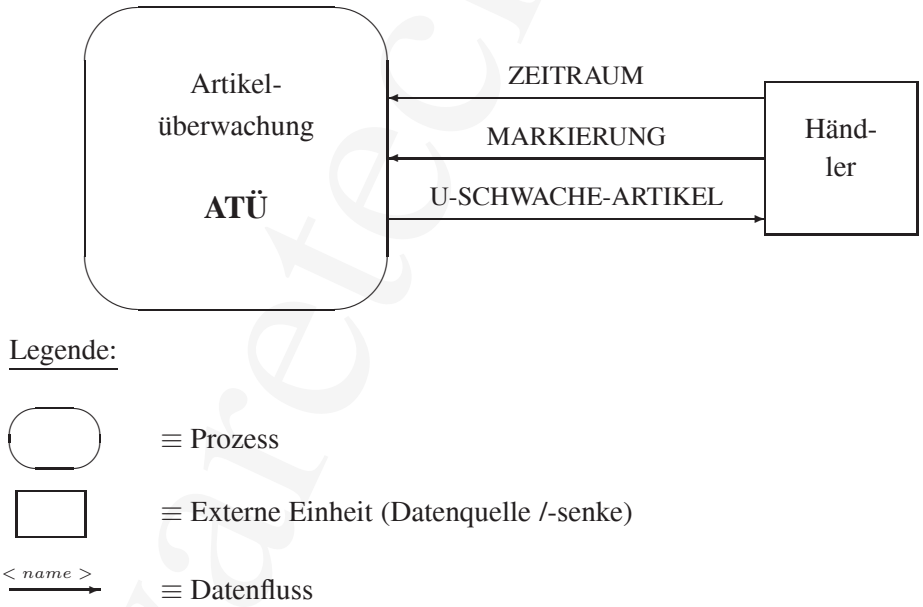
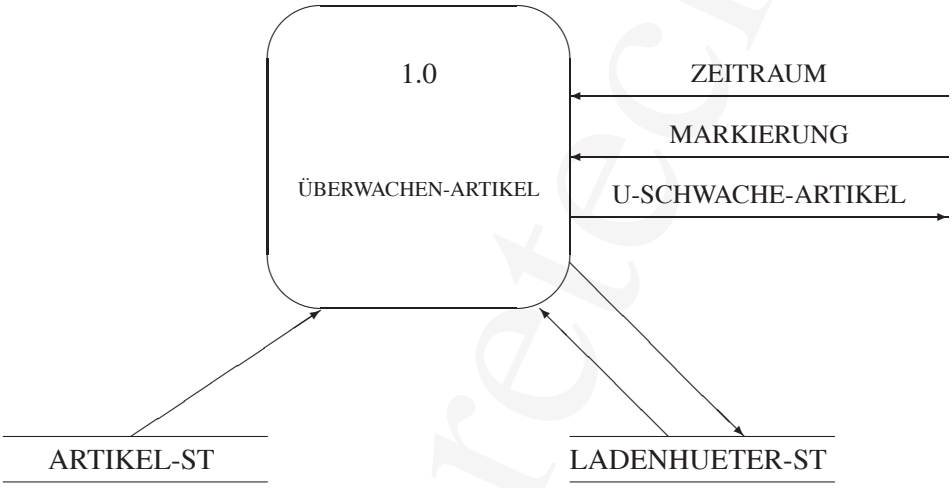
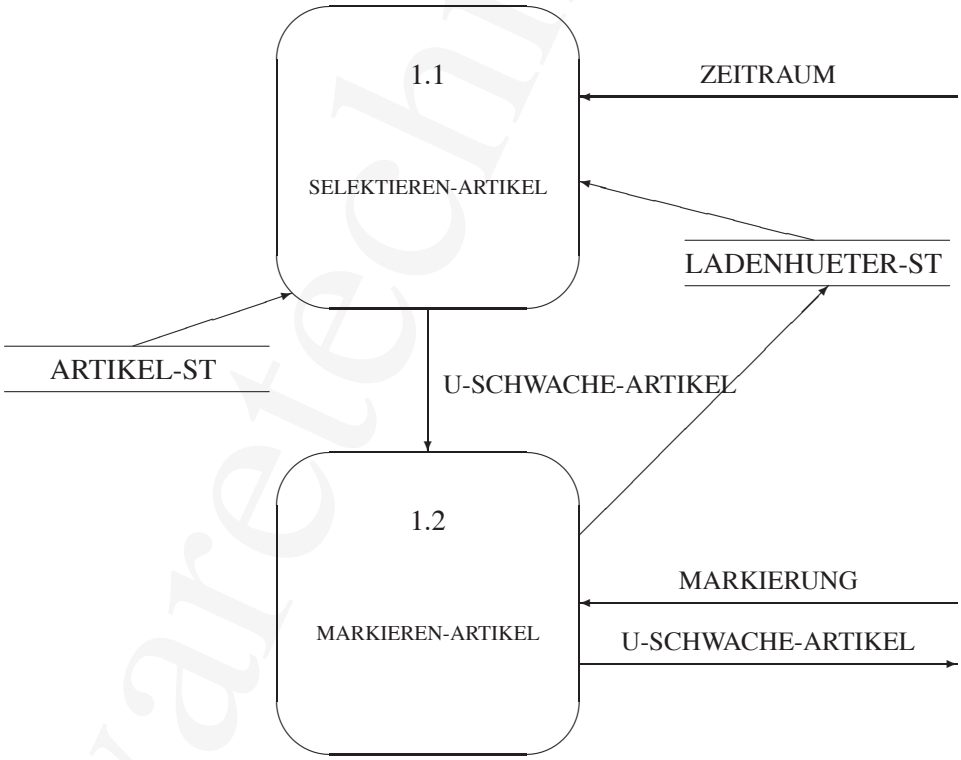


Abbildung F.9: ATÜ: Kontextdiagramm



Legende:
Vgl. Abbildung F.9 Seite 297.

Abbildung F.10: ATÜ: $Level_0$-Diagramm



Legende:
Vgl. Abbildung F.10 Seite 298.

Abbildung F.11: ATÜ: $Level_1$-Diagramm

SELEKTIEREN-ARTIKEL

Abfragen von ZEITRAUM
Sequentielles Lesen von ARTIKEL-ST, bis Tabelle U-SCHWACHE-ARTIKEL mit 10 Artikel, die nicht im ZEITRAUM in LADENHUETER-ST markiert sind, geladen ist.
Sortieren von U-SCHWACHE-ARTIKEL
while weitere Artikel in ARTIKEL-ST
Update Tabelle U-SCHWACHE-ARTIKEL durch umsatzschwächere Artikel.

Legende:
Vgl. Abbildung F.11 Seite 299.

Abbildung F.12: ATÜ: Prozeß 1.1 (Selektieren-Artikel)

MARKIEREN-ARTIKEL

Zeigen U-SCHWACHE-ARTIKEL
Abfragen der MARKIERUNG
Udate LADENHUETER-ST

Legende:
Vgl. Abbildung F.11 Seite 299.

Abbildung F.13: ATÜ: Prozeß 1.2 (Markieren-Artikel)

Die Sotware-Entwicklungsumgebung für ATÜ ist abhängig von der RM7-Implementation der Datei ARTIKEL-ST, da darauf häufig zuzugreifen ist.

- Wird ARTIKEL-ST mit Hilfe eines „relationalen“ Datenbankmanagementsystems verwaltet, dann bietet sich zunächst dessen 4GL an. In Betracht kommen dann auch Programmiersprachen wie C und COBOL, die Embedded SQL ermöglichen.
- Da nur der Händler auf die ARTIKEL-ST zugreift, könnte diese auch als einfache ISAM-Datei geführt werden. Dann bietet sich COBOL (ANSI 85) an. In COBOL sind die ATÜ-Datenstrukturen direkt abbildbar. Darüber hinaus existieren leistungsfähige Werkzeuge für die Dialogführung (Maskengenerator).

Lösung Aufgabe E.6:

E.6.1:

Tabelle F.3 Seite 302 zeigt das geschlossene ET-Verbundsystem.

E.6.2:

Konsolidierbar sind:

- ET-MODUL-X: $R3 \widehat{+} R4 \Rightarrow R3'$ mit B2 = - .
- ET-MODUL-Y:
  $R1 \widehat{+} R2 \Rightarrow R1'$ mit B3 = - .
  $R3 \widehat{+} R4 \Rightarrow R3'$ mit B3 = - .
  $R7 \widehat{+} R8 \Rightarrow R7'$ mit B3 = - .
  $R3' \widehat{+} R7' \Rightarrow R7$“mit B1 = - .

Tabelle F.4 Seite 302 zeigt die konsolidierte ET-MODUL-Y.

E.6.3:

Vollständigkeitsprüfung:

- ET-MODUL-X in konsolidierter Form:
  2 einwertige Regeln (R1, R2) $\Longrightarrow$ 2 Fälle
  1 zweiwertige Regell (R3') $\Longrightarrow$ 2 Fälle
  $\hookrightarrow$ 4 Fälle $\equiv 2^2 \Longrightarrow$ formal vollständig
- ET-MODUL-X-II: 2 einwertige Regeln (R1, R2) $\Longrightarrow$ 2 Fälle
  $\hookrightarrow$ 2 Fälle $\equiv 2^1 \Longrightarrow$ formal vollständig
- ET-MODUL-Y in konsolidierter Form:
  2 einwertige Regeln (R5, R6) $\Longrightarrow$ 2 Fälle

ET-MODUL-X		R1	R2	R3	R4
B1	p	J	J	N	N
B2	q	J	N	J	N
A1	A	X			
A2	B	X			
A3	run ET-MODUL-X-II	X			
A4	C		X		
A5	run ET-MODUL-Y			X	X
A4	D			X	X
A4	E	X	X	X	X

ET-MODUL-X-II		R1	R2
B1	r	J	N
A1	A		X
A2	run ET-MODUL-X-II		X
A3	return	X	

ET-MODUL-Y		R1	R2	R3	R4	R5	R6	R7	R8
B1	s	J	J	J	J	N	N	N	N
B2	t	J	J	N	N	J	J	N	N
B3	u	J	N	J	N	J	N	J	N
A1	F	X	X						
A2	G			X	X		X	X	X
A3	H					X			
A4	K			X	X	X	X	X	X
A5	return	X	X	X	X	X	X	X	X

Legende:
Vgl. Abbildung E.4 Seite 262

Tabelle F.3: PDL-Kontrollstruktur ⇒ Eintreffer-ET-Verbundsystem

ET-MODUL-Y		R1'	R5	R6	R7"
B1	s	J	N	N	-
B2	t	J	J	J	N
B3	u	-	J	N	-
A1	F	X			
A2	G			X	X
A3	H		X		
A4	K		X	X	X
A5	return	X	X	X	X

Legende:
Vgl. Tabelle F.3 Seite 302

Tabelle F.4: Konsolidierte ET-MODUL-Y

1 zweiwertige Regell (R1') $\Longrightarrow$ 2 Fälle
1 vierwertige Regell (R7“) $\Longrightarrow$ 4 Fälle
$\hookrightarrow$ 8 Fälle $\equiv 2^3 \Longrightarrow$ formal vollständig

Lösung Aufgabe E.7:
Die Abbildung F.14 Seite 304 zeigt den PAP.

Lösung Aufgabe E.8:
E.8.1:
Benutzt wird hier die alphanumerische Identifizierung Lnn der einzelnen Leistungen mit $nn = 01, \ldots, 99$.

$L01$ VERS verwaltet jeden Interessenten (IT) mit Namen, Adresse, eindeutiger Identifizierungs-Nummer (I-NR) und gewünschten Bauplätze (vgl. $L06$,$L07$) in Form von:

- $L01.1$ Neuaufnahme
- $L01.2$ Änderungen (Update)
- $L01.3$ Rücktritt (Löschen)

$L02$ VERS erfaßt im Dialog für jeden IT die punktrelevanten Kriteren:

- $L02.1$ Ortsansässigkeit $\rightarrow P_O$
- $L02.2$ Kinderanzahl $\rightarrow P_K$
- $L02.3$ Behinderung $\rightarrow P_B$
- $L02.4$ Arbeitsstelle $\rightarrow P_A$

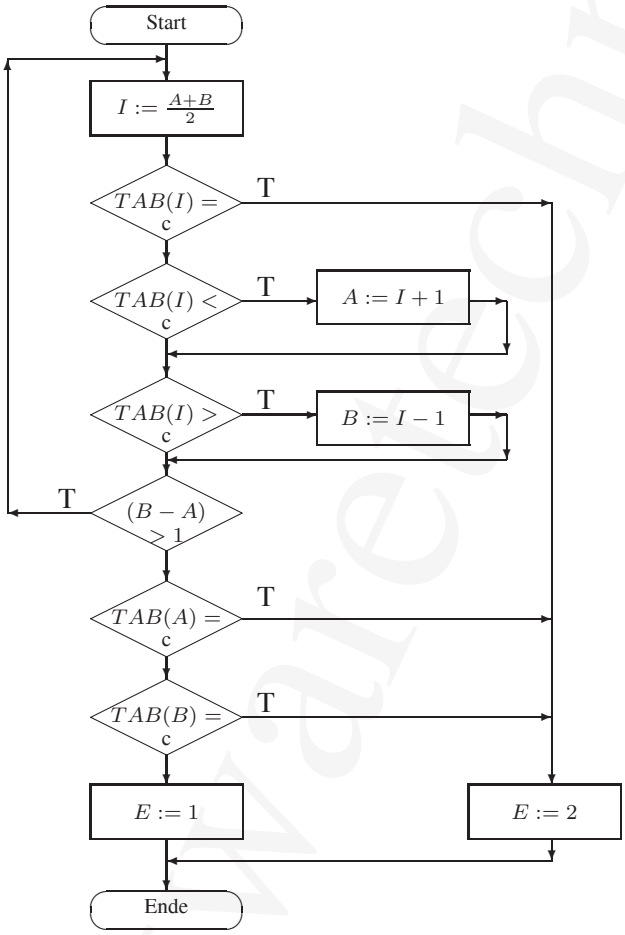
und berechnet die Punktsumme P_S.

$L03$ Punktsumme: $P_S = P_O + P_K + P_B + P_A$

$L04$ VERS bildet die Punktregelung wie folgt ab:

$L04.1$ Ortsansässigkeit:

ET-Ortsansässigkeit		R1	R2	R3
B1	IT = Einwohner gemäß der GO	J	N	N
B2	IT hat Familienangehörige in *Schönhausen*	–	J	N
A1	$P_O \leftarrow 0$			X
A2	$P_O \leftarrow 3$		X	
A3	$P_O \leftarrow 5$	X		



Legende:
Vgl. Abbildung E.5 Seite 263

Abbildung F.14: Programmfragment ⇒ PAP

$L04.2$ Kinderzahl:
$P_K = 2*$ `ANZAHL-DER-KINDER`

$L04.3$ Behinderung:
$P_B = 3*$ `ANZAHL-DER-BEHINDERTEN-FAMILIENMITGLIEDER`

$L04.4$ Arbeitsstelle:

ET-Arbeitsstelle		R1	R2
B1	IT hat Arbeitsstelle in *Schönhausen*	J	N
A1	$P_A \leftarrow 0$		X
A2	$P_A \leftarrow 4$	X	

$L05$ VERS selektiert aus $VW2000$ die Identifier der Bauplätze *Heideweg* $\rightarrow$ BP-ID.

$L06$ VERS speichert für jeden IT die ihn interessierenden Bauplätze mit der BP-ID.

$L07$ Ein IT kann sich für ein, mehrere oder alle Bauplätze interessieren (bewerben).

$L08$ VERS bildet das Losverfahren durch eine Zufallszahlfunktion (RANDOM--Prozedure) ab.

$L08.1$ VERS ermittelt bei gleicher Punktsumme P_S für eine BP-ID pro betroffenen IT eine eigene, bisher nicht vorkommende Zufallszahl Z_{Zufall}.

$L08.2$ Der Z_{Zufall}-Wert bestimmt die Zuschlagsrangreihenfolge RANG, wobei der kleinste Wert den 1. Rang repräsentiert.

$L09$ VERS erstellt eine Zuschlagsliste $L_{Zuschlag}$ sortiert nach:

1. BP-ID aufsteigend,
2. P_S absteigend
3. RANG aufsteigend

$L10$ VERS erstellt eine Liste der freien Bauplätze $L_{freie-BP-ID}$.

$L11$ VERS erstellt eine Interessentenliste L_{IT} sortiert nach:

1. Name, Adresse
2. BP-ID
3. P_S absteigend

proc 1.0 ERMITTLE-BP-ID
READ BAUGEBIET
VERBINDE MIT VW2000
SELEKTIERE BP-ID
WRITE BAUPLÄTZE
corp

Tabelle F.5: VERS-Prozeß 1.0: ERMITTLE-BP-ID

proc 2.0 VERWALTE-INTERESSENT
ERFRAGE MODUS ; neu, update, löschen
ERFRAGE, PRÜFE und WRITE IT-STAMMDATEN
BERECHNE P_S
ERFRAGE, PRÜFE und WRITE GEWÄHLTE-BP
WRITE GEBUCHT in BAUPLÄTZE
corp

Tabelle F.6: VERS-Prozeß 2.0: VERWALTE-INTERESSENT

4. RANG aufsteigend

$L12$ VERS erstellt für jeden IT einen Bescheid $L_{Bauplaetze}$ mit dem Vergabeergebnis.

$L13$ VERS arbeitet auf einem Rechner im LAN.

E.8.2:
VERS ist entsprechend *Structured Analysis* dokumentiert mittels:

- VERS Kontextdiagramm vgl. Abbildung F.15 Seite 307
- VERS $Level_0$-Diagramm vgl. Abbildung F.16 Seite 308
- VERS Miniprozeß-Definitionen vgl. Tabellen F.5 . . . F.8 Seite 306 . . . 308
- VERS Datenlexikon vgl. Tabelle F.9 Seite 309

E.8.3:
Analog zu $L02.1 \ldots L02.5$ nimmt VERS jetzt Kriterien an, die einsatzfallabhängig formulierbar sind und deren Punkte der Sachbearbeiter direkt pro IT eingeben kann. Vor dem VERS-Einsatz sind in der Datei

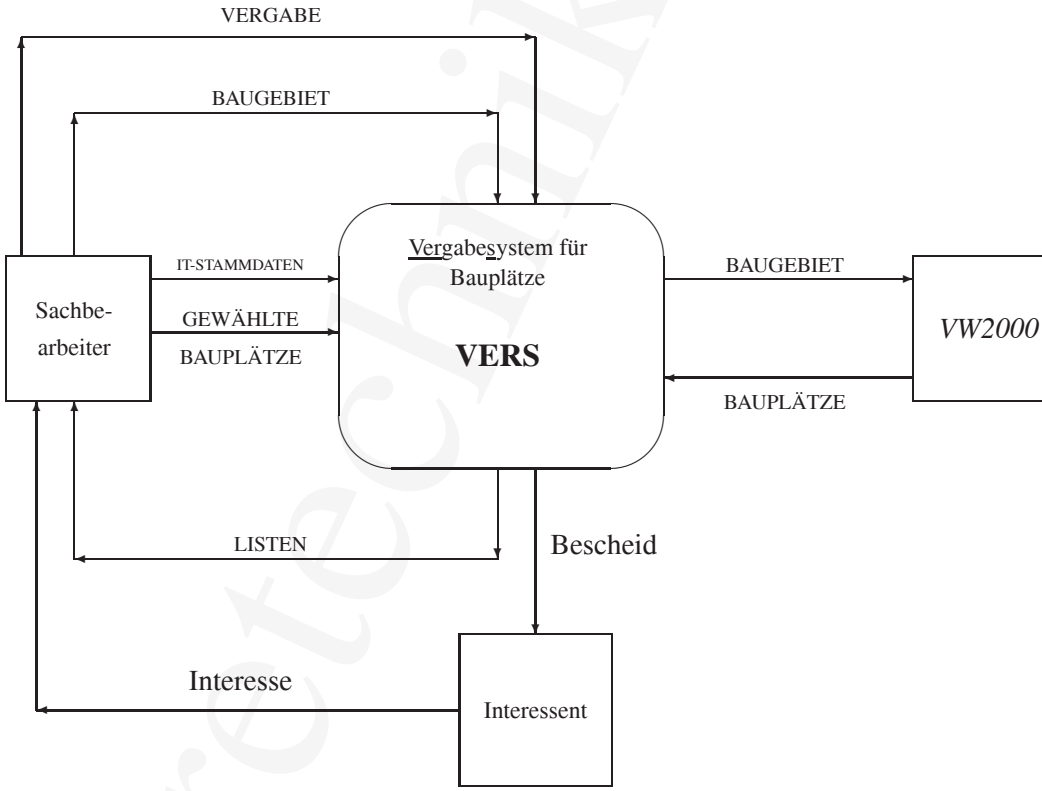


Abbildung F.15: VERS: Kontextdiagramm

proc 3.0 `ERMITTLE-ZUORDNUNG`
`READ GEWÄHLTE-BP` und `IT-STAMMDATEN`
`WRITE ZUORDNUNG`
`SORT ZUORDNUNG BP-ID`,P_S oder äquivalenter DB-Zugriff
`GRUPPENVERARBEITUNG`
if mehrere Sätze mit gleicher `BP-ID`,P_S pro Satz
then `ERMITTLE` neue Z_{Zufall} pro Satz
`KONVERTIERE` Z_{Zufall} in `RANG` pro Satz
`UPDATE ZUORDNUNG`
fi
corp

Tabelle F.7: VERS-Prozeß 3.0: ERMITTLE-ZUORDNUNG

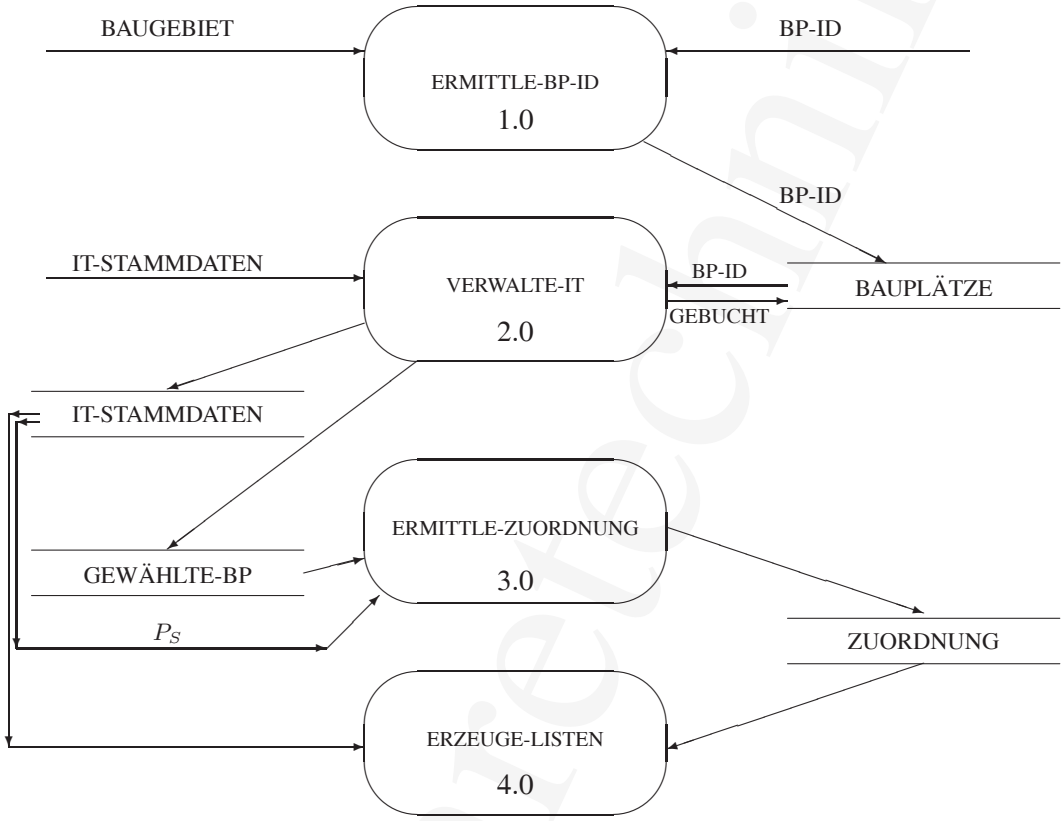


Abbildung F.16: VERS: $Level_0$-Diagramm

proc 4.0 ERZEUGE-LISTEN
READ ZUORDNUNG, IT-STAMMDATEN
ERSTELLE LISTE L_{IT}
ERSTELLE BESCHEID $L_{Bauplaetze}$
READ BAUPLÄTZE
ERSTELLE LISTE $L_{freie-BP-ID}$
corp

Tabelle F.8: VERS-Prozeß 4.0: ERZEUGE-LISTEN

Nr.	**Idenitfier**	**Beschreibung**	**Typ/Werte**	**Beispiel**
1	BAUGEBIET	Grundstückskarte	vgl. *VW2000*	*Heideweg*
2	IT-STAMMDATEN	Interessent		
2.1	I-NR	Identifizierungsnr.	999	203
2.1	NAME	Zu- und Vornamen	X(50)	Meyer, Karl
2.2	ADRESSE	Postanschrift	X(50)	PF12 Reppenstedt D-21391
2.3	KRITERIEN	punktrelevante Kriterien		
2.3.1	P_O	vgl. $L04.1$	9	5
2.3.2	P_K	vgl. $L04.2$	99	02
2.3.3	P_B	vgl. $L04.3$	99	06
2.3.4	P_A	vgl. $L04.4$	9	4
2.4	P_S	vgl. $L03$	99	07
3	BAUPLÄTZE	Bauplätze		
3.1	BP-ID	KEY	999	101
3.2	GEBUCHT	Anzahl der IT	999	002
4	GEWÄHLTE-BP	Ausgesuchte Bauplätze		
4.1	I-NR	vgl. 2.1		
4.2	BP-ID	vgl. 3.1		
5	ZUORDNUNG	Zugeordnete Bauplätze		
5.1	BP-ID	vgl. 3.1		
5.2	P_S	vgl. 2.4		
5.3	RANG	vgl. $L08.2$	99	02
5.4	I-NR	vgl. 2.1		
6	VERGABE	Start der Zuordnung		

Tabelle F.9: VERS: Datenlexikon

FAKT.TXT die einzelnen Fakten in jeweils einem Satz abzulegen. Die Kriterien sind stets disjunkt zu formulieren, damit die Punkteangabe unabhängig von der vorherigen erfolgen kann (– anders als bei $L04.1$ –). Pro FAKT-Satz und IT gibt es Punkte, deren Summe P_S (analog zu $L03$) von VERS berechnet wird. VERS zeigt die FAKT-Sätze beim Prozeß 2.0 AUFNEHMEN-INTERESSENT am Bildschirm.

E.8.4:
Da die üblichen Leistungen einer Stammdatenverwaltung dominieren, ist eine datenbankorientierte Enwicklungsumgebung zweckmäßig, die ein angepaßtes CASE-Tool mit umfaßt — zum Beispiel ORACLE (hier auf einem Server im LAN).

Das kleine Projekt VERS kann jedoch nicht die Beschaffungskosten eines solchen „Werkzeuges“ tragen. Es hat sich daher nach der vorgefundenen Softwareentwicklungsumgebung zu richten. Ist keine vorhanden, dann ist die Beschaffung eines kostengünstigen PC-(DB)-Systems – zum Beispiel CLIPPER – angebracht.

Da ein Zugriff auf *VW2000* erforderlich ist, kommt auch die *VW2000*-Entwicklungsumgebung in Betracht, damit zumindest die Zusammenarbeit zwischen VERS und *VW2000* leicht abbildbar ist.

Lösung Aufgabe E.9:

E.9.1:
R5 und R7 haben gleiche Bedingungs- und Aktionsanzeigerfolgen → Redundanz. R7 wird gestrichen.

E.9.2:
R14 und R16 haben gleiche Bedingungsanzeigerfolgen und unterschiedliche Aktionsanzeigerfolgen → Widerspruch. Gemäß Aufgabentext ist die Regel mit der kleineren Regelbezeichnernummer zu streichen → R14 ist zu streichen.

E.9.3:
In Bezug zur kanonischen Normalform fehlen die Regeln mit den Bedingungsanzeigerfolgen:

B1	J	N
B2	N	N
B3	J	J
B4	J	N

E.9.4:
1. Konsolidierungsdurchlauf:
$\widehat{R2 + R6} \Rightarrow R2'$ mit B2 = -

ET-S		R1	R2'	R3"	R9"	R13'	R16
B1	u ?	J	J	J	N	N	N
B2	v ?	J	-	-	J	N	N
B3	w ?	J	J	N	-	-	N
B4	x ?	J	N	-	-	J	N
A1	A						X
A2	B		X	X	X	X	X
A3	C		X	X	X	X	
A4	D		X	X	X	X	
A5	E	X					
A6	F	X					

Legende:
Vgl. Aufgabe E.9.4 Seite 265

Tabelle F.10: Konsolidierte ET

$R3\widehat{+}R4 \Rightarrow R3'$ mit B4 = -
$R5\widehat{+}R8 \Rightarrow R5'$ mit B4 = -
$R9\widehat{+}R10 \Rightarrow R9'$ mit B4 = -
$R11\widehat{+}R12 \Rightarrow R11'$ mit B4 = -
$R13\widehat{+}R15 \Rightarrow R13'$ mit B3 = -

2. Konsolidierungsdurchlauf:
$R3'\widehat{+}R5' \Rightarrow R3''$ mit B2 = -
$R9'\widehat{+}R11' \Rightarrow R9''$ mit B3 = -
Tabelle F.10 Seite 311 zeigt die konsolidierte ET-Scheck. Tabelle F.11 Seite 312 zeigt eine Tabelle mit noch weniger Regeln. Sie nutzt die ELSE-Regel.
E.9.5:
Die ELSE-Rege der ET-Alles hat nicht die Aktion A2 markiert. Außerdem deckt sie auch die fehlenden Regeln:

- B1=J, B2=N, B3=J, B4=J und
- B1=N, B2=N, B3=J, B4=N

ab. Darüber enthält ET-Scheck jedoch keine Aussage.

Lösung Aufgabe E.10:
E.10.1:
Wenn die Stellen des Vorbereichs der Transition t_j markiert und die des Nachbereichs nicht markiert sind, hat t_j Schaltkonzession.

ET-S		R1	R'_{fehlt}	R''_{fehlt}	R16	R_{else}
B1	u ?	J	J	N	N	E
B2	v ?	J	N	N	N	L
B3	w ?	J	J	J	N	S
B4	x ?	J	J	N	N	E
A1	A				X	
A2	B				X	X
A3	C					X
A4	D					X
A5	E	X				
A6	F	X				
A7	nicht definierter Zustand		X	X		

Legende:
Vgl. Aufgabe E.9.4 Seite 265

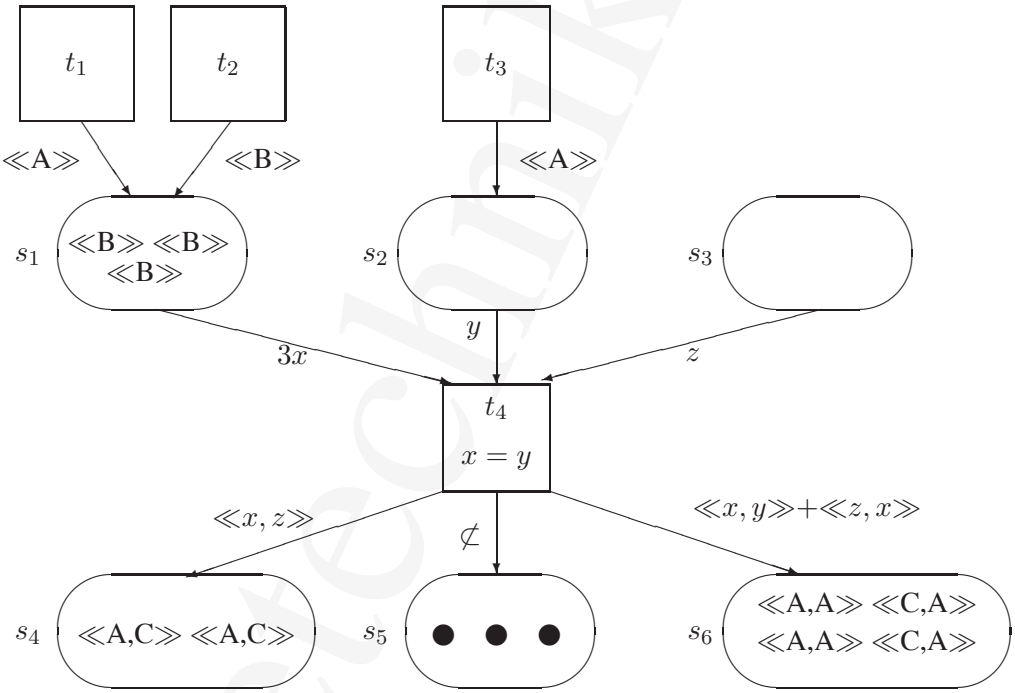
Tabelle F.11: ET mit ELSE-Regel

if
Vorbereich t_j markiert $\wedge$ Nachbereich t_j $\neg$markiert
then
Schaltkonzession t_j
fi

E.10.2:

- Fall 1 ($M_0 \equiv s_2$):
  $t_3, t_6, t_8, t_5, t_2, t_1 \rightarrow$ Netz ist „tot“
- Fall 2 ($M_0 \equiv s_2 + s_3$):
  $t_4, t_3, t_7, t_6, t_8, t_5, t_2, t_1 \rightarrow M_0$
  Die durch Nebenläufigkeit möglichen anderen Schaltfolgen sind hier nicht genannt. Sie alle führen jedoch zu M_0.

Lösung Aufgabe E.11:
E.11.1:
Angenommen wird eine Schaltregel, die nur vom Vorbereich der Transition abhängig ist. Die Kanalanschriften des Vorbereiches von t_4 sind erfüllt, wenn x und y mit $\ll$A$\gg$ und z mit $\ll$C$\gg$ belegt sind. Dann ist auch die Bedingung $x = y$ in t_4 erfüllt. t_4 hat Schaltkonzession.



Legende:
Vgl. Abbildung E.7 Seite 269.

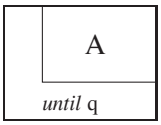
Abbildung F.17: Petri-Netz – nach zweimaligem Schalten

E.11.2:
Die Transition t_4 schaltet genau zweimal, wobei angenommen wird, daß t_1 mindestens noch dreimal geschaltet hat. Das Ergebnis zeigt Abbildung F.17 Seite 313.

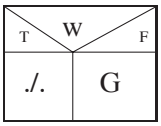
Lösung Aufgabe E.12:
E.12.1:
Zur Zusammenfassen der linearen Teile des PAP (vgl. Abbildung F.18 Seite 314) nutzen wir eine pragmatische Notation für die Bezeichnung der entstehenden Knoten:

Knotennummer = Zusammenfassungschritt plus zweistellige laufende Nummer, Zeichen „/“und anschließend Anzahl der zusammengefaßten Knoten

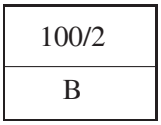
Abbildung F.19 zeigt den PAP mit so zusammengefaßten Knoten. Mit



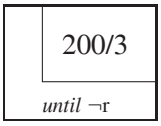
≡ 100/2 *do* A *until* q *od*



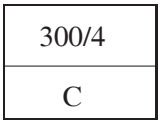
≡ 101/2 *if* ¬w *then* G *fi*



≡ 200/3 100/2 B



≡ 300/4 *do* 200/3 *until* ¬r *od*



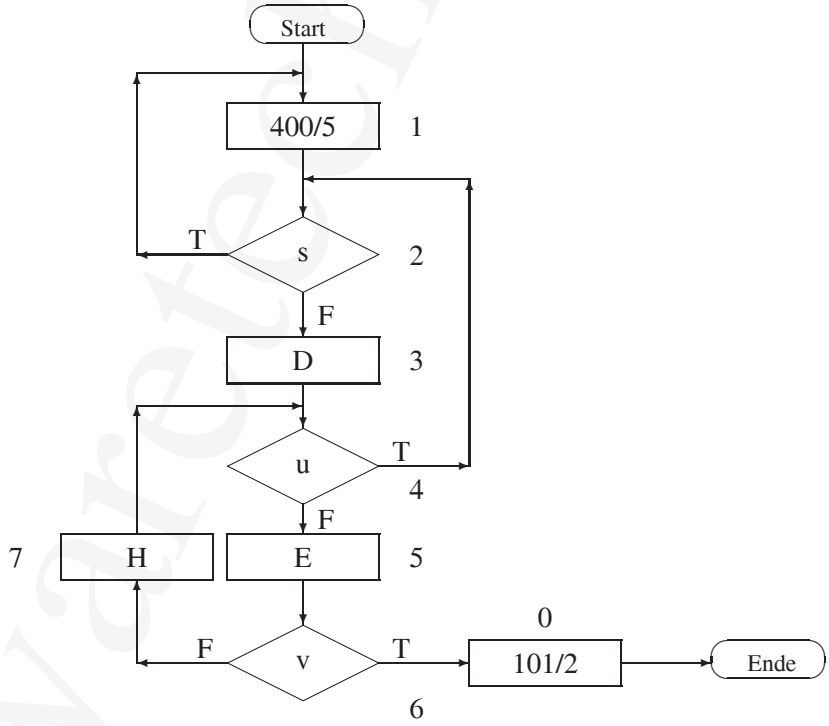
≡ 400/5 300/4 C

Legende:
Vgl. Abbildung E.8 Seite 270

Abbildung F.18: PAP: Zusammenfassung linerarer Anteile

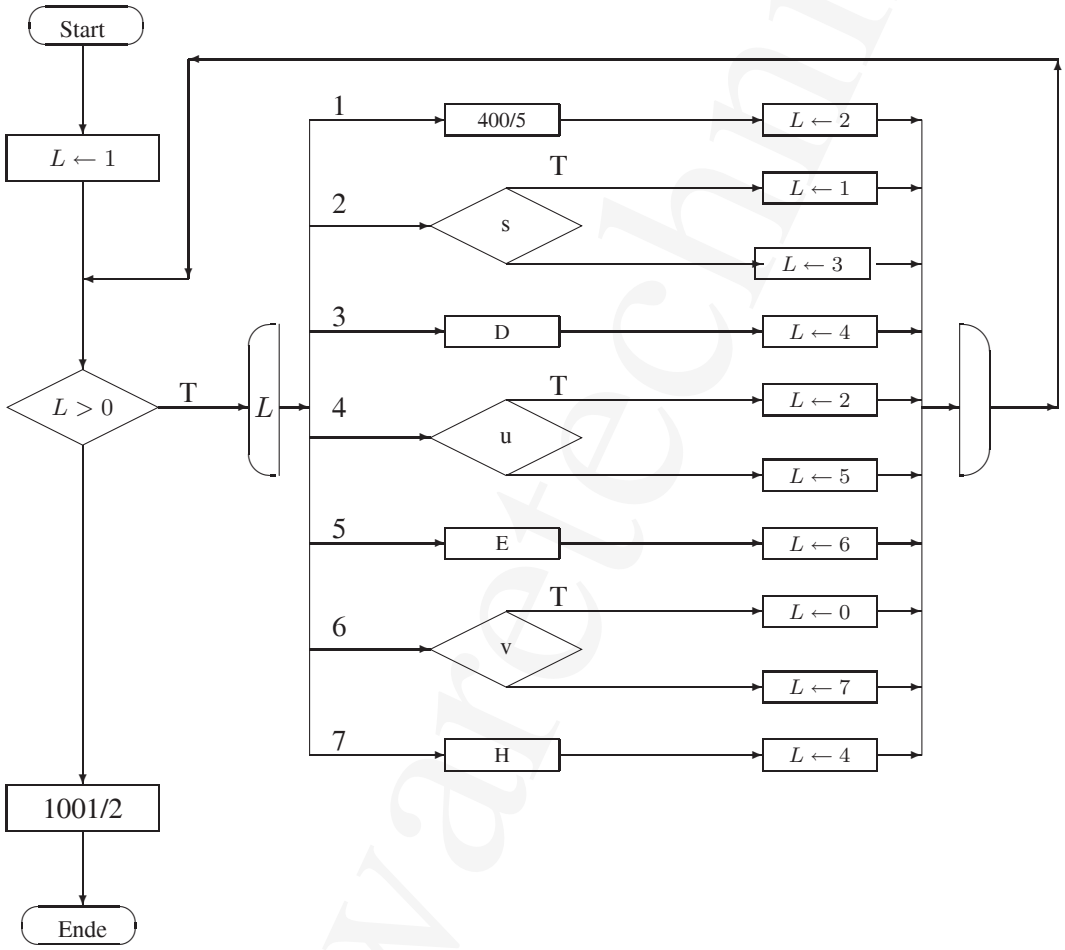
Hilfe der zusätzlichen Variable L (*label*) wird auf den Nachfolgeknoten verwiesen. Das Ende der nichtlineraren Kontrollstruktur bildet so der Knoten $101/2$, der als Ende den L-Wert $= 0$ bekommt. Abbildung F.20 Seite 316 zeigt das *while*-Konstrukt mit den L-Zuweisungen. L-Werte die nur einmal zugewiesen werden, können durch den Zweig des *case*-Konstruktes direkt ersetzt werden. Ausnahme bildet der Wert $L = 1$, weil es sich dabei um den Einstiegsknoten handelt. Abbildung F.21 Seite 317 zeigt das reduzierte *label structure program*. E.12.2:
Der PAP in Abbildung F.21 Seite 317 stellt eine lineare Kontrollstruktur dar und ist daher ohne Umkonstruktion in ein Struktogramm übertragbar. Abbildung F.22 Seite 318 zeigt die Lösung.

Lösung Aufgabe E.13:
E.13.1:
EAN enthält keine Rücksprüge (GOTO's). Die Kontrollstruktur ist da-



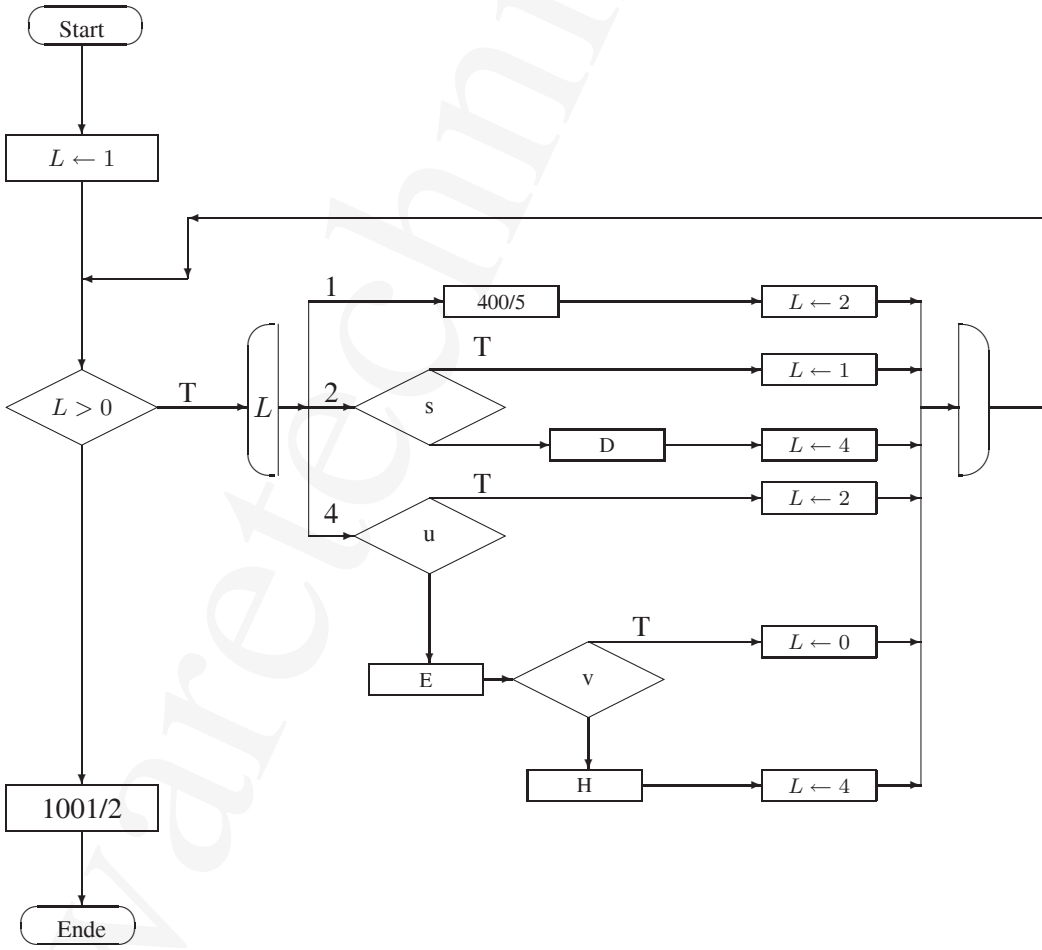
Legende:
Vgl. Abbildung E.8 Seite 270 und Abbildung F.18 Seite 314

Abbildung F.19: PAP: Einführung von Knotennummern



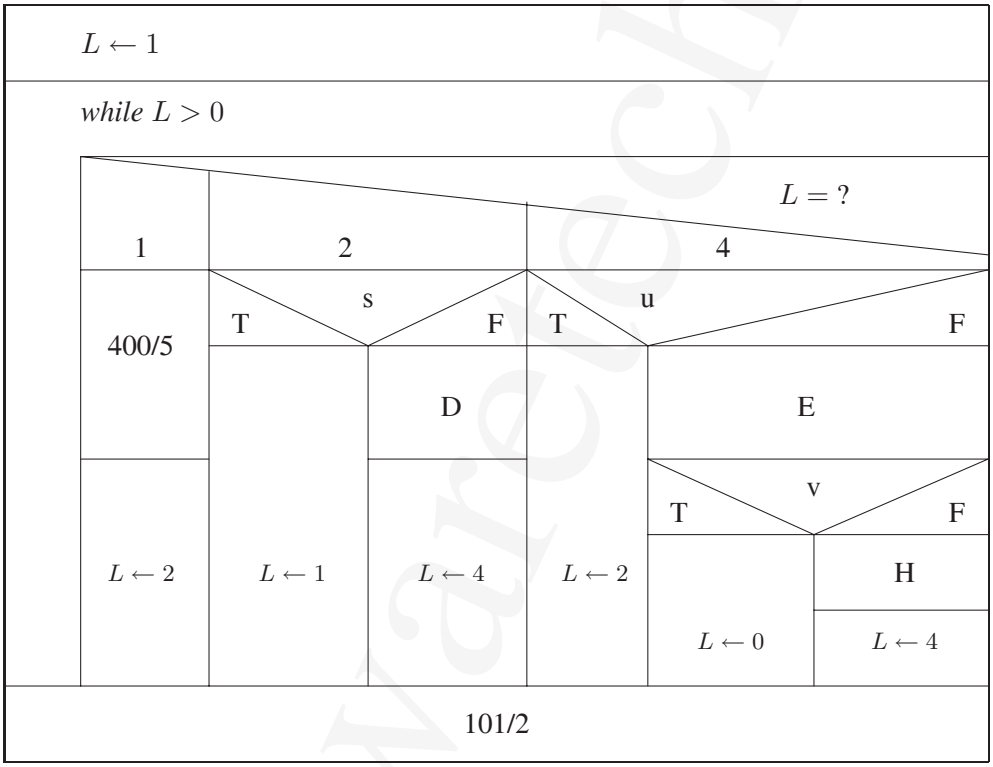
Legende:
Vgl. Abbildung E.8 Seite 270, Abbildung F.18 Seite 314 und Abbildung F.19 Seite 315

Abbildung F.20: PAP: Label Structure Program



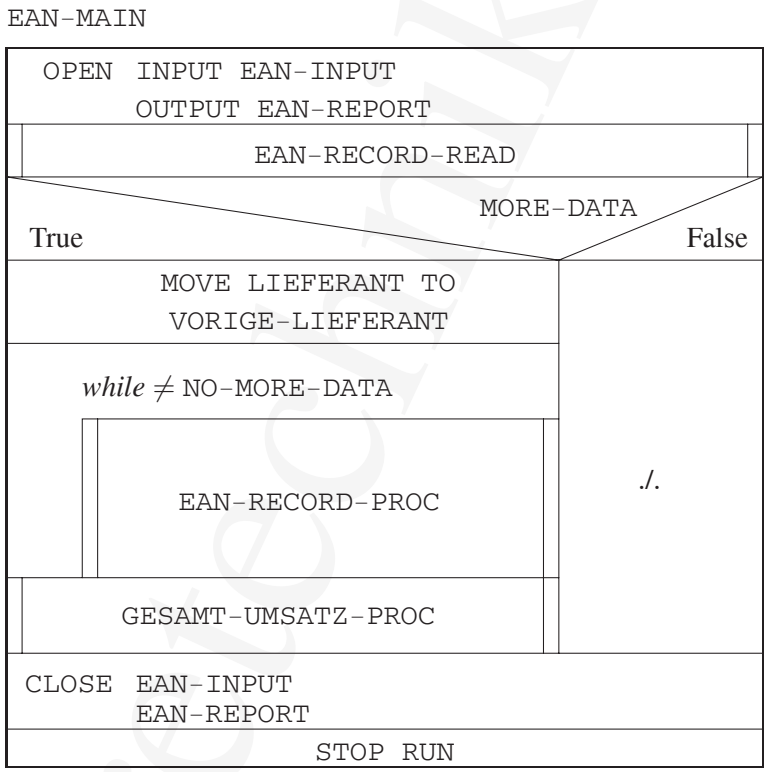
Legende:
Vgl. Abbildung F.20 Seite 316
Reduzierbar sind die L-Werte: 3,5,6,7

Abbildung F.21: PAP: Reduziertes *Label Structure Program*



Legende:
Vgl. Abbildung F.21 Seite 317

Abbildung F.22: PAP ⇒ Struktogramm



Legende:
Vgl. Aufgabe E.13.1 Seite 275.

Abbildung F.23: Kontrollstruktur: `EAN-MAIN`

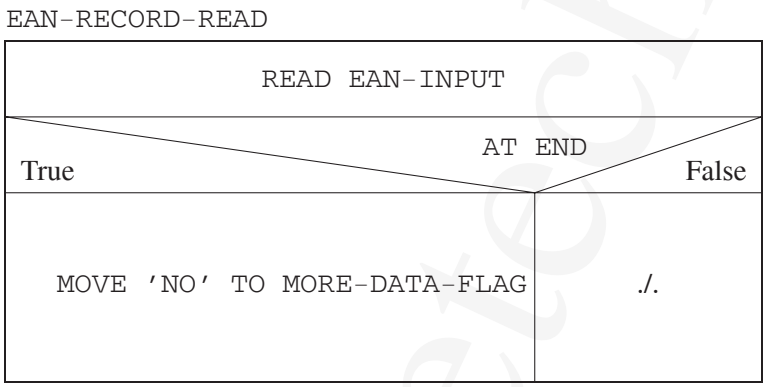
her als Nassi/Shneiderman-Diagramm in den Abbildungen F.23 . . . F.26 Seite 319 . . . 322 dargestellt. Programmbereiche, die nur aus leicht durchschaubaren Sequenzen bestehen, sind als Block ausgeiwesen – zum Beispiel „Kopf drucke“.

E.13.2:
Tabelle F.12 Seite 323 skizziert den Output von `EAN`.

E.13.3:
`LIEFERANT,EAN,BUCHUNGSZEITPUNKT`
E.13.4:

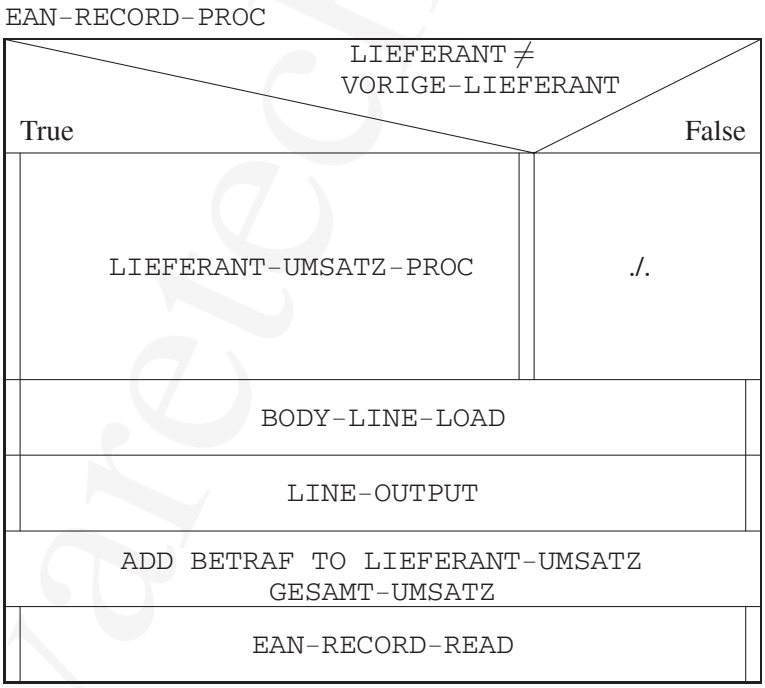
- Zum `EAN`-Ergebnis:

  1. Die Felder `LIEFERANT, EAN, BUCHUNGSZEITPUNKT` sind nur zu drucken, wenn ihr Wert sich gegenüber der vorherigen Zeile geändert hat. Bei Seitenwechsel ist die erste



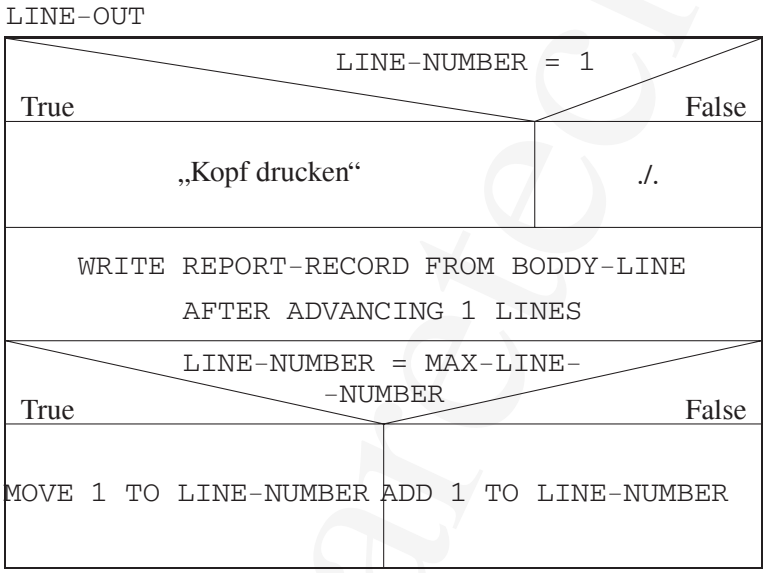
Legende:
Vgl. Abbildung F.23 Seite 319.

Abbildung F.24: Kontrollstruktur: `EAN-RECORD-READ`



Legende:
Vgl. Abbildung F.23 Seite 319.

Abbildung F.25: Kontrollstruktur: EAN-RECORD-PROC



Legende:
Vgl. Abbildung F.23 Seite 319.

Abbildung F.26: Kontrollstruktur: LINE-OUTPUT

```
Tag hhmmss       EAN       LF.MG.    Betrag   Umsatz Gesamtumsatz S.  1


111 015901 40 12345 12345 0 05 00     99.99
112 020310 40 11111 11111 0 05 00    199.99
                              ⋮
                                    insgesamt 57 Zeilen pro Seite
                              ⋮

Tag hhmmss       EAN       LF.MG.    Betrag   Umsatz Gesamtumsatz S.  3


115 050713 40 33333 33333 0 05 00      1.99
116 015901 40 12345 12345 0 05 00     99.99
                              05              18,946.81
117 050612 40 22222 22222 0 08 00    299.99
                              08                299.99
117 050713 40 33333 33333 0 09 00      1.99
                              09                  1.99
                                                      19,248.79
```

Tabelle F.12: EAN Output-Skizze

BODY-LINE stets zu drucken. Der Verzicht auf die laufende Ausweisung gleicher Daten erhöht die Lesbarkeit. Dazu sind in der WORKING-STORAGE SECTION unter AUXILIARY die entsprechenden Felder VORIGE-... zu definieren. In dem Paragraphen BODY-LINE-LOAD sind abhängig vom Wert LINE-NUMBER und den Werten VORIGE-... SPACES in die aktuellen Felder zu speichern. Das Laden der Felder VORIGE-... erfolgt an den Stellen bei denen VORIGE-LIEFERANT einen Wert erhält.

2. UMSATZ und GESAMTUMSATZ benötigen keine eigene Spalte, sondern sind als jeweils eigene Zeile mit entsprechenden, vorangestellten Textkonstanten auszuweisen. Dazu sind BODY-LINE und HEAD-LINE in der WORKING-STORAGE SECTION und die Paragraphen LIEFERANT-UMSATZ-PROC und GESAMT-UMSATZ-PROC anzupassen.

- Zum EAN-Quellcode:

1. Dokumentieren von konstruktionsbedingten Informationen mittels Affixe (Präfixe und Suffixe) – zum Beispiel Präfix

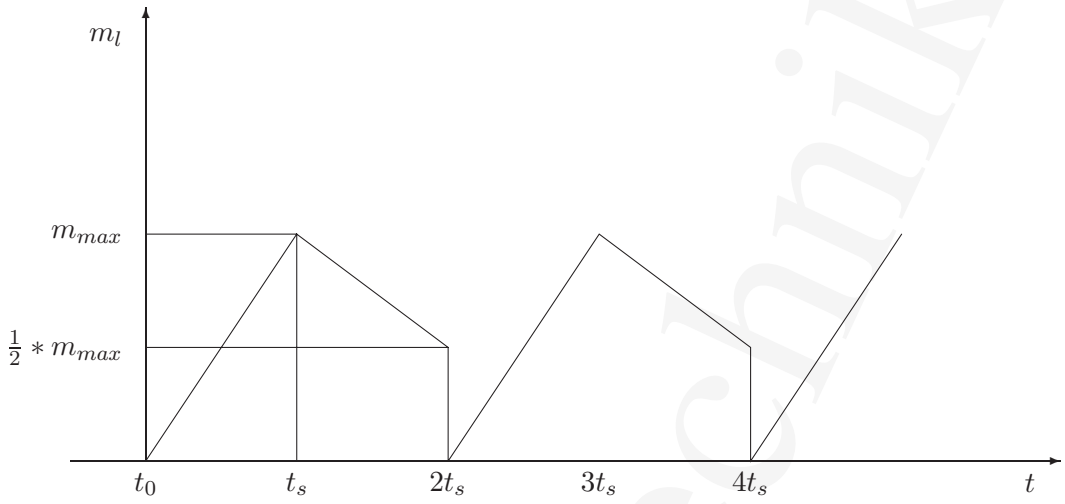


Abbildung F.27: Zeitlicher Verlauf der Lagermenge m_l

„I-“... für Felder in EAN-RECORD (Input) und „O-“... für EAN-REPORT.

2. Einführung von Endemarken (*scope delimiter*) statt Endepunkt – zum Beispiel END-IF.

Lösung Aufgabe E.14:

E.14.1:

Abbildung F.27 Seite 324 zeigt die Lagermenge m_l in Abhängigkeit von der Zeit t.

E.14.2:

Abbildung F.28 Seite 325 zeigt das Struktogramm der rekursiven Definition der Funktion $m_l(t)$:

$m_l(t)$

t /	$t < 0$	$0 \leq t < t_s$	$t_s \leq t < 2t_s$	$2t_s = t$	$2t_s < t$
	Nicht definiert!	$m_l \leftarrow \frac{m_{max}}{t_s} * t$	$m_l \leftarrow -\frac{m_{max}}{2t_s} * t + \frac{3}{2} * m_{max}$	$m_l \leftarrow 0$	$m_l \leftarrow m_l(t - 2t_s)$
		$return\ m_l$			

Abbildung F.28: Rekursives Struktogramm: Funktion $m_l(t)$

$$m_l(t) = \begin{cases} \text{Nicht definiert!} & \text{für } t < 0 \\ \frac{m_{max}}{t_s} * t & \text{für } 0 \leq t < t_s \\ -\frac{m_{max}}{2t_s} * t + \frac{3}{2} * m_{max} & \text{für } t_s \leq t < 2t_s \\ 0 & \text{für } 2t_s = t \\ m_l(t - 2t_s) & \text{für } 2t_s < t \end{cases} \quad \text{(F.1)}$$

Lösung Aufgabe E.15:

E.15.1:

Dokumentation AUFTRAG

Inhaltsverzeichnis

zu 0) Zusammenfassung
Das COBOL-Programm AUFTRAG erfaßt die Daten Kundenidentifizierung, Auftragsidentifizierung und Menge in Stück per Bildschirmmaske und ergänzt mit diesen Daten die Datei `auftrag.dat`.

zu 1) Ziel und Zweck von AUFTRAG
Das COBOL-Programm AUFTRAG dient zur Auftragserfassung im Dialogverfahren. Es ergänzt die Datei `auftrag.dat` (vgl. Abbildungen F.29 Seite 327 und F.30 Seite 327).
zu 2) Einsatz- und Entwicklungs-Umgebung
Das COBOL-Programm AUFTRAG ist ein „IBM AIX VS COBOL/6000“-Produkt. Es bedingt daher eine AIX-Umgebung auf einer IBM Workstation vom Typ RS/6000.

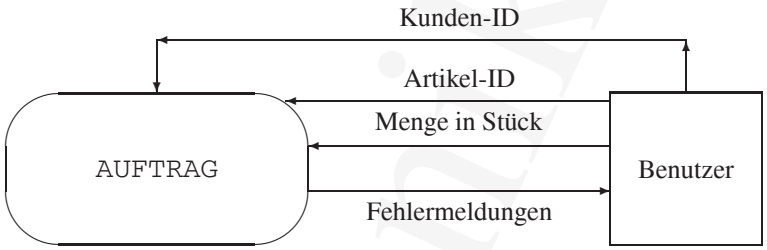
Standardmäßig befindet sich die Quellcodedatei `auftrag.cbl` im selben Verzeichnis wie die Datei der erfaßten Daten `auftrag.dat`.

zu 3) AUFTRAG-Leistungen
Benutzt wird hier die alphanumerische Identifizierung Lmn für die einzelnen Leistungen mit $mn = 01, \ldots, 99$.

$L01$ AUFTRAG schreibt mittels Bildschirmmaske (vgl. $L03$) den Auftragsdatensatz (vgl. $L02$) an das Ende der Datei `auftrag.dat`.

$L02$ Der Auftragsdatensatz hat folgende Struktur:

```
01  AUFTRAG-RECORD.
    05  KUNDE.
        10  KUNDE-ID                    PIC 9(5).
    05  ARTIKEL.
```



Legende:
Symbole gemäß *Structured Analysis and Design Specification*
Kunden-ID ≡ Kunden-Identifizierung
00000 ≡ Ende der Erfassung
Artikel-ID ≡ Artikel-Identifizierung

Abbildung F.29: Kontextdiagramm: AUFTRAG

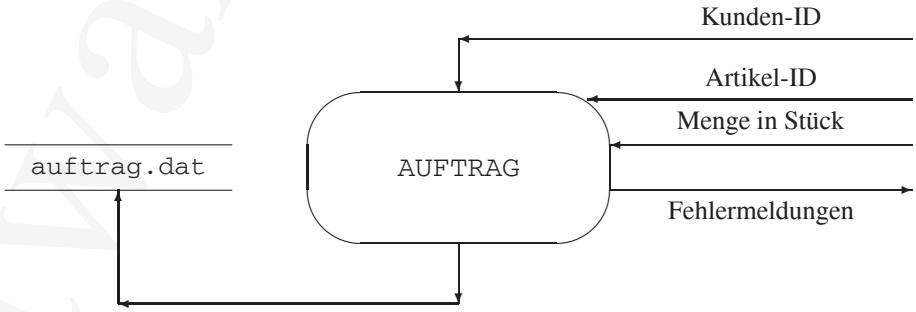


Abbildung F.30: $Level_0$-Diagramm: AUFTRAG

```
10  ARTIKEL-ID                  PIC 9(2).
10  MENGE                       PIC 9(4).
```

$L03$ Die Bildschirmmaske hat folgendes Format:

Beginnposition Text Zeile	Text Spalte	Daten Zeile	Daten Spalte	
8	10			AUFTRAGSERFASSUNG
11	10	11	28	Kunden-ID/00000: [ ]
13	10	13	31	Artikel-ID: [ ]
15	10	15	29	Menge in Stueck: [ ]

$L04$ Ist der Feldinhalt von `Kunden-ID` gleich `ZERO` (vgl. $L03$), dann schließt `AUFTRAG` die Datei `auftrag.dat` und endet.

zu 4) `AUFTRAG`-Kontrollstruktur
Abbildung F.31 Seite 329 zeigt die Kontrollstruktur von `AUFTRAG`.

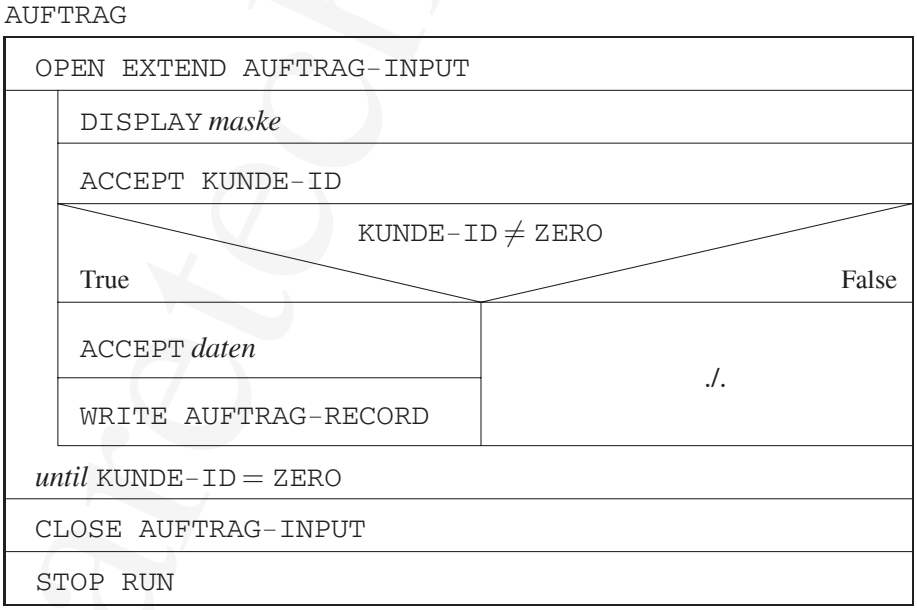
zu 5) Input/Output-Beschreibung
Input
vgl. $L03$
Bei nichtnumerischen Input-Daten wird eine Fehlermeldung ("That keystroke has no meaning here") vom `ACCEPT`-Statement auf dem Bildschirm ausgegeben.
Output
vgl. $L02$
Die sequentielle Datei `auftrag.dat` hat eine Satzlänge von 11 Byte.
Beispiel:

1	0	0	0	1	8	2	3	0	0	0

10001 $\equiv$ Kunden-Identifizierung
82 $\equiv$ Artikel-Identifizierung
3000 $\equiv$ Menge in Stück

E.15.2:

- Eine ET bildet die Iteration als „*while ... do ... od*"-Konstrukt ab. Die Kontrollstruktur von `AUFTRAG` enthält jedoch eine Iteration vom Typ „*do ... until ... od*" (`PERFORM WITH TEST AFTER UNTIL ...`).
- Die Sequenz zu Beginn des Struckturblocks `AUFTRAG` kann von einem ET-Verbundsystem nicht direkt abgebildet werden, es sei



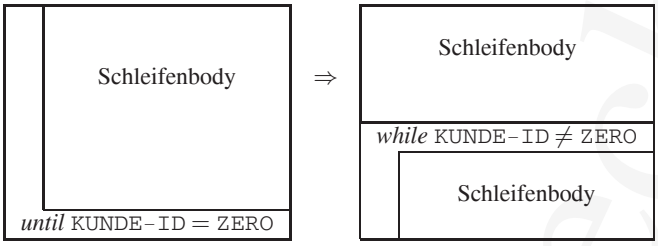
Legende:
Notation nach Nassi/Shneiderman.

Abbildung F.31: Kontrollstruktur: AUFTRAG

denn, man akzeptiert eine ET ohne Bedingungen — wie im folgenden angenommen.

Aufgrund dieser Fakten ist der Aussage zuzustimmen.

Zur Abbildung der Kontrollstruktur im ET-Verbundsystem wird die Iteration durch folgendes Äquivalent abgebildet:



ET-AUFTRAG		R1
A1	`OPEN EXTEND AUFTRAG-INPUT`	X
A2	*run* ET-Schleifenbody	X
A3	*run* ET-Iteration	X
A4	`CLOSE AUFTRAG-INPUT`	X
A5	`STOP RUN`	X

ET-Schleifenbody		R1
A1	`DISPLAY` *maske*	X
A2	`ACCEPT KUNDE-ID`	X
A3	*run* ET-Verarbeitung	X

ET-Verarbeitung		R1	R2
B1	`KUNDE-ID` $\neq$ `ZERO`	J	N
A1	`ACCEPT` *daten*	X	
A2	`WRITE AUFTRAG-RECORD`	X	
A3	*return*	X	X

ET-Iteration		R1	R2
B1	KUNDE-ID $\neq$ ZERO	J	N
A1	*run* ET-Schleifenbody	X	
A2	*return*	X	X

E.15.3:
$ login *user* ↩ # „Einloggen“
$ cd ... ↩ # Directory einstellen
$ auftrag ↩
10001 ↩ # Kunden-Identifizierung eingeben
38 ↩ # Artikel-Identifizierung eingeben
3000 ↩ # Menge in Stück eingeben
00000 ↩ # Programmende
$ cat auftrag.dat | more ↩
$ ⋮ # bisherige Datensätze
10001823000 # erfaßter Datensatz
$ logout ↩ # AIX-Systemeben verlassen

Hinweis: Wenn mehr als die definierte Anzahl von Ziffern in ein Feld eingegeben werden, dann gibt AUFTRAG eine Fehlermeldung („You are at the end of the field“) und einen Signalton aus. Wenn keine Datei auftrag.dat vorab besteht, dann meldet AUFTRAG einen Fehler („I/O-error“) und beendet.

E.15.4:

1. Abstellen der Vermischung zwischen Nutzdaten und Steuerungsdaten im Feld KUNDE-ID.
2. Plausibilitätskontrolle der Daten in den Feldern KUNDE-ID und ARTIKEL-ID gegen entsprechende Stammdateien.
3. Modifizierungsmöglichkeit schon erfaßter Datensätze.
4. Speicherung der Erfassungszeit und einer Identifizierung des Benutzers.
5. Präfix und Suffix bei Variablen einführen, um konstruktionsbedingte (COBOL-spezifische) Angaben direkt zu vermitteln.
6. Dateifehlerbehandlung zum Beispiel beim Fehlen von auftrag.dat.

Lösung Aufgabe E.16:

E.16.1:
In der ET-FOO sind die Regeln R5 und R13 redundant. R13 wird gestrichen.

E.16.2:
Die ET-FOO enthält keinen formalen Widerspruch.

E.16.3:
Es fehlt die Regel R13 mit der Bedingungsanzeigerfolge: N N J J.

E.16.4:
1. Konsolidierungsdurchlauf:
$R3 \widehat{+} R4 \Rightarrow R3'$ mit B4 = - .
$R5 \widehat{+} R6 \Rightarrow R5'$ mit B4 = - .
$R7 \widehat{+} R8 \Rightarrow R7'$ mit B4 = - .
$R9 \widehat{+} R10 \Rightarrow R9'$ mit B4 = - .
$R11 \widehat{+} R12 \Rightarrow R11'$ mit B4 = - .
$R14 \widehat{+} R16 \Rightarrow R14'$ mit B3 = - .

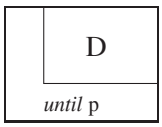
2. Konsolidierungsdurchlauf:
$R5' \widehat{+} R7' \Rightarrow R5''$ mit B3 = - .
$R9' \widehat{+} R11' \Rightarrow R9''$ mit B3 = - .
Tabelle F.13 Seite 338 zeigt die konsolidierte ET.

E.16.5:
Die ET-ALLES subsumiert unter der ELSE-Regel die folgenden Regeln der ET-FOO:
$R2, R3', R5'', R9''$, J-Zweig von $R14'$, $R15$
Diese Regeln haben jedoch bis auf R15 andere Aktionsanzeigerfolgen. Daher gilt: ET-Alles $\neq$ ET-FOO.

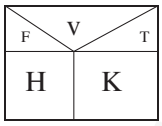
Lösung Aufgabe E.17:
E.17.1:
Die Abbildungen F.32 Seite 333, F.33 Seite 334, F.34 Seite 335 und F.35 Seite 336 zeigen die Schritte zur Linearisierung und Reduktion.

E.17.2:
Die Abbildung F.36 Seite 337 zeigt das Struktogramm.

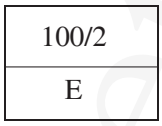
Lösung Aufgabe E.18:
E.18.1:



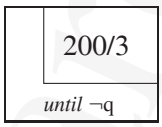
≡ 100/2 *do* D *until* p *od*



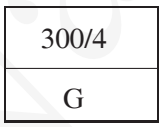
≡ 101/3 *if* v *then* K *else* H *fi*



≡ 200/3 100/2 E



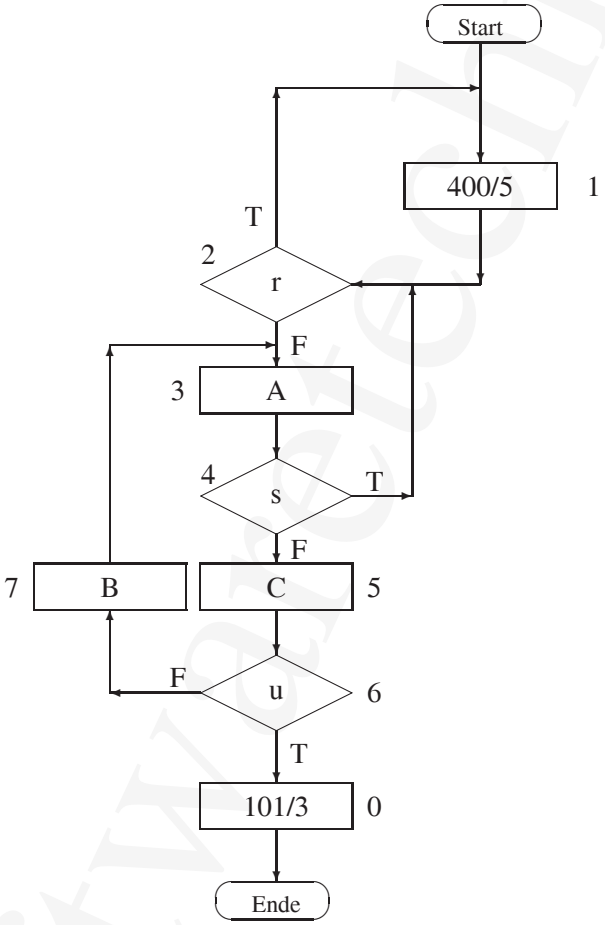
≡ 300/4 *do* 200/3 *until* ¬q *od*



≡ 400/5 300/4 G

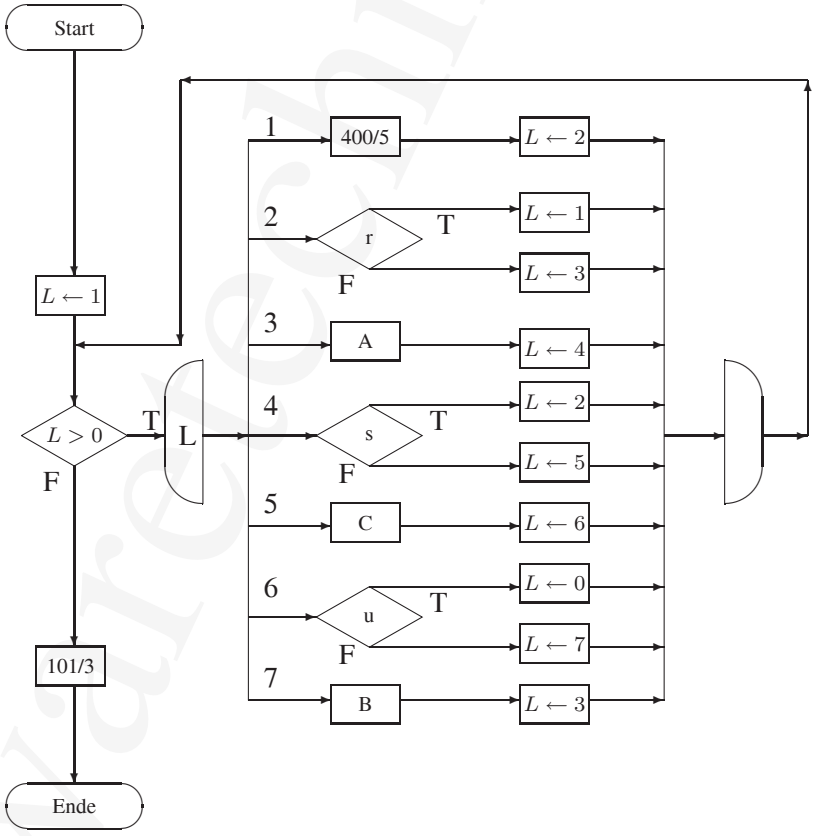
Legende:
Vgl. Abbildung E.9 Seite 281

Abbildung F.32: PAP: Zusammenfassung linearer Anteile



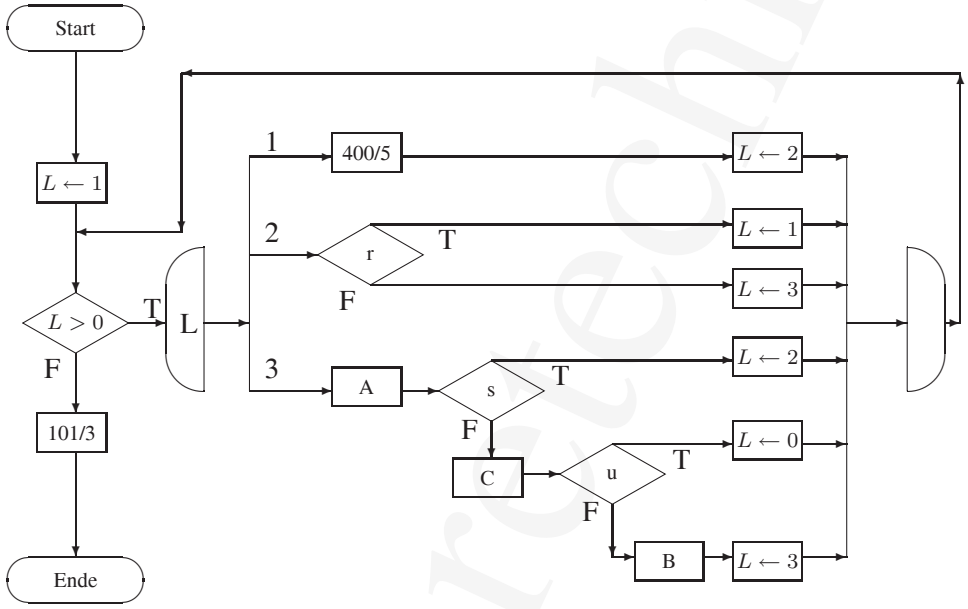
Legende:
Vgl. Abbildung E.9 Seite 281 und Abbildung F.32 Seite 333

Abbildung F.33: PAP: Einführung von Knotennummern



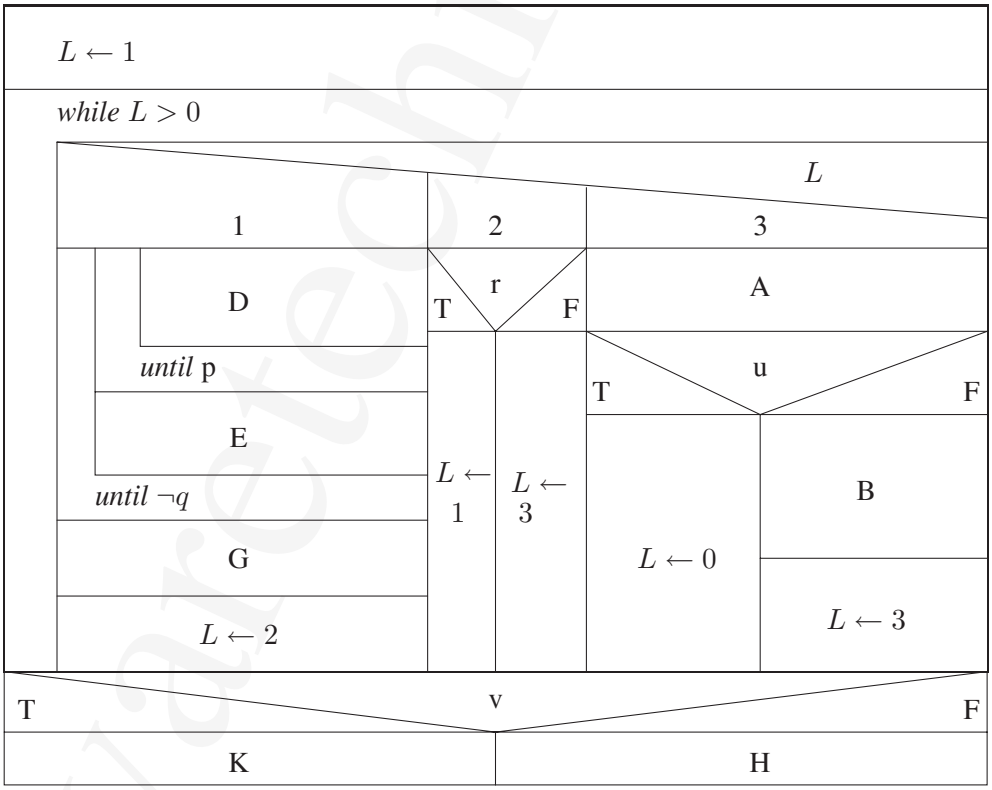
Legende:
Vgl. Abbildung E.9 Seite 281, Abbildung F.32 Seite 333 und Abbildung F.33 Seite 334

Abbildung F.34: PAP: *Label Structure Programm*



Legende:
Vgl. Abbildung F.34 Seite 335
Reduzierbar sind die L-Werte: 4,5,6,7

Abbildung F.35: PAP: Reduziertes *Label Structure Programm*



Legende:
Vgl. Abbildung F.35 Seite 336

Abbildung F.36: PAP ⇒ Struktogramm

ET-FOO		R1	R2	R3'	R5"	R9"	R14'	R15
B1	p ?	J	J	J	J	N	N	N
B2	q ?	J	J	J	N	J	N	N
B3	r ?	J	J	N	–	–	–	N
B4	s ?	J	N	–	–	–	N	J
A1	A						X	
A2	B	X					X	
A3	C		X	X	X	X		X
A4	D		X	X	X	X		X
A5	E	X	X	X	X	X		
A6	F	X						

Legende:
Vgl. Tabelle E.4 Seite 279.

Tabelle F.13: Konsolidierte ET

- Fall 1: $M_0 \equiv s_1 + s_5$
  Schaltfolge: t_5, t_6, t_1 (oder t_1, t_5, t_6 oder t_5, t_1, t_6) $\rightarrow$ Netz ist „tot“.

- Fall 2: $M_0 \equiv s_1 + s_4 + s_6$
  Schaltfolge: t_1, t_3, t_5, t_6, t_4, $t_2 \rightarrow M_0$

E.18.2:

- $M_0 \equiv s_2 + s_5 + s_6$
  Schaltfolge: t_5, t_6, t_3, t_4, t_2, $t_1 \rightarrow M_0$

Wenn die Schlinge bei der Transition t_3 aufgelöst wird, dann wird deutlich, daß t_3 bei obiger M_0 Schaltkonzession haben kann. Die Abbildung F.37 Seite 339 zeigt die Schlingenauflösung.

Lösung Aufgabe E.19:
E.19.1:
Die Abbildung F.38 Seite 340 zeigt das *bool* -Petri-Netz.

E.19.2:
Die Abbildung F.39 Seite 341 zeigt das Stelle/Transitions-Netz mit Kanalbeschriftung.

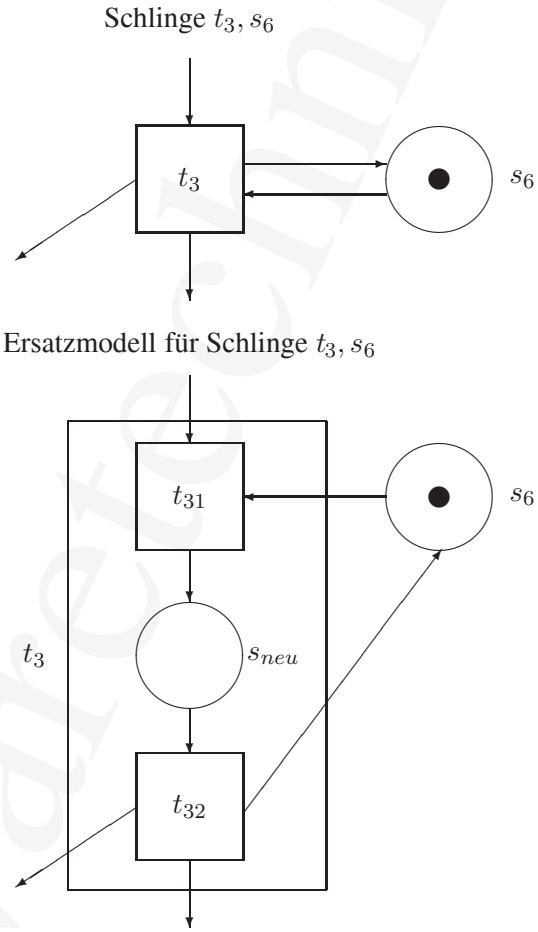
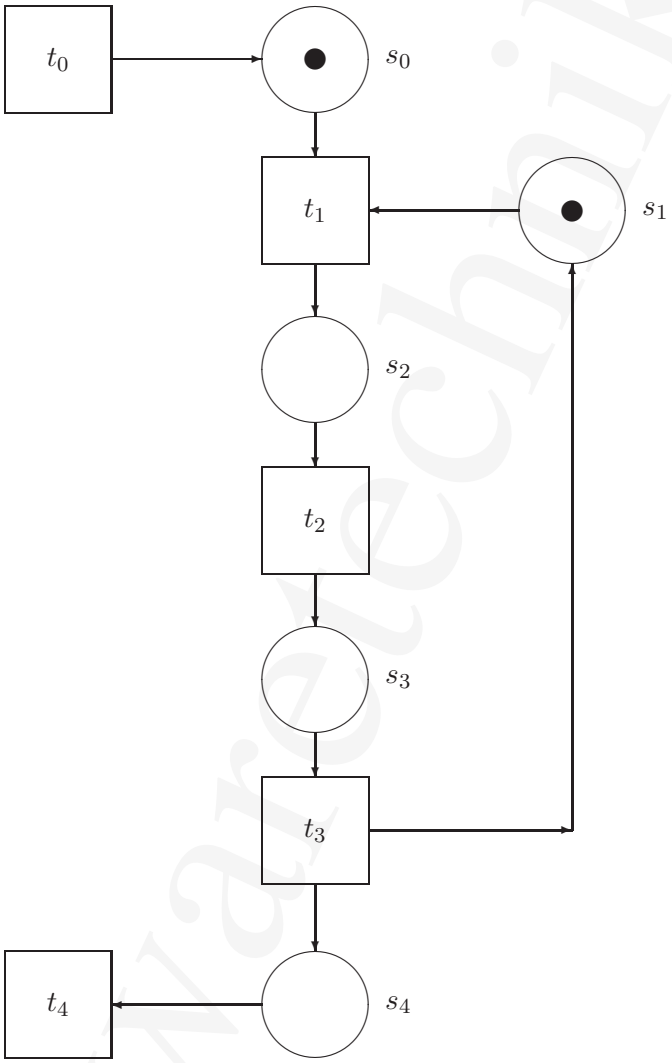


Abbildung F.37: Petrinetz: Schlingenauflösung



Legende:

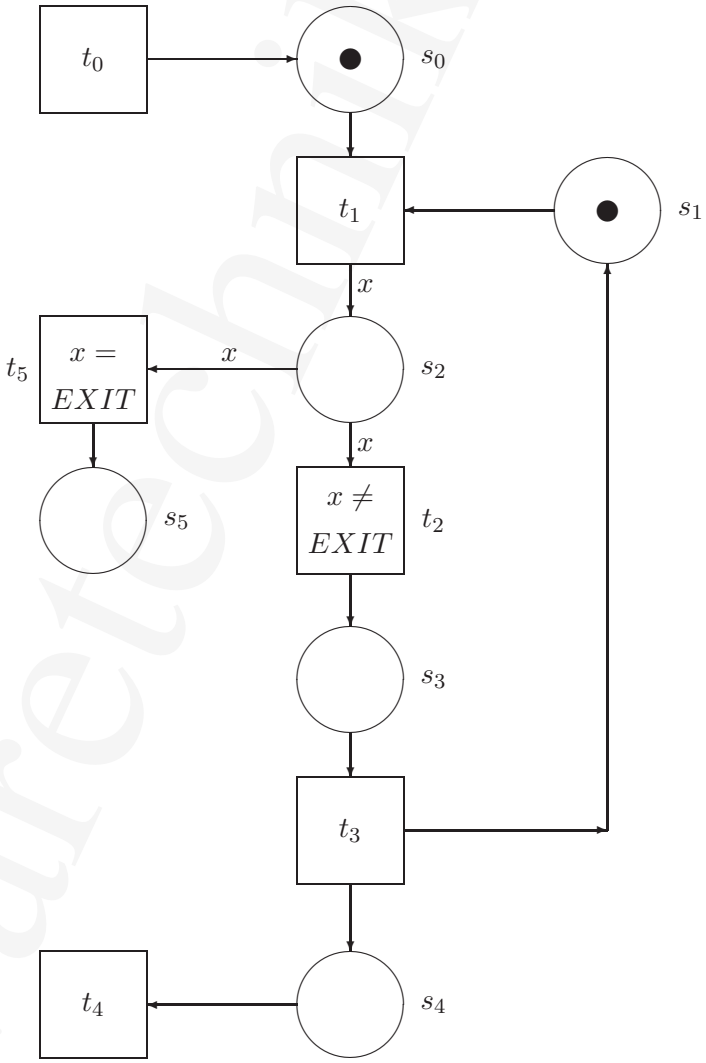
M_0 $\equiv s_0 + s_1$

t_4 $\equiv$ dient zum Abziehen der Marke, damit t_3 Schaltkonzession bekommen kann.

Schaltregel:

Die Transition t_i hat Schaltkonzession, wenn der Vorbereich $\triangleright t_i$ markiert und der Nachbereich $t_i \triangleleft$ nicht markiert ist.

Abbildung F.38: *bool* -Petri-Netz: READ-EVAL-PRINT-Zyklus



Legende:

M_0 $\equiv s_0 + s_1$

t_5 $\equiv$ READ-EVAL-PRINT-Zyklus verlassen

s_5 $\equiv$ READ-EVAL-PRINT-Zyklus ist beendet

Schaltregel:

Bedingungen der Abbildung F.38 Seite 340 und die Kantenbeschriftung müssen erfüllt sein, d. h. widerspruchsfreie Ersetzung der Variablen x durch individuelle Objekte.

Abbildung F.39: S/T-Petri-Netz: READ-EVAL-PRINT-Zyklus

Lösung Aufgabe E.20:

E.20.1:

Funktionale CF-Leistungen:

Benutzt wird hier die alphanumerische Identifizierung Lmn für die einzelnen Leistungen mit $mn = 01, \ldots, 99$.

$L01$ CF verwaltet folgende Daten:

- $L01.1$ Kundenidentifizierung, Adresse, Geschlecht, Bankabbuchungsgenehmigung
- $L01.2$ Gebührenforderungen und Zahlungen
- $L01.3$ IST-Kondition (Gewicht, Ruhepuls, Blutdruck)
- $L01.4$ mittel- und langfristige Trainingsziele
- $L01.5$ erzielte Leistungen pro Sportgerät (SG) und Datum

$L02$ CF erkennt den Kunden durch dessen Magnetkarte (MK).

$L03$ CF erstellt im Dialog mit dem Trainer für den Kunden dessen Trainingsprogramm (TP) und legt damit die benutzbaren SG fest.

$L04$ Das TP weist die SOLL-Werte pro SG und Datum aus.

$L05$ CF aktuallisiert das TP

- $L05.1$ wöchentlich anhand der erzielten Leistungen und/oder
- $L05.2$ nach Aufforderung durch den Kunden und/oder
- $L05.3$ aufgrund Erkenntnisse des Trainers.

$L06$ CF kontrolliert anhand der MK und des TP ob der Kunde das jeweilige SG benutzen darf.

$L07$ Das SG zeigt während der Übung auf einem Bildschirm an:

- $L07.1$ die bisher erzielten Leistungen,
- $L07.2$ die aktuellen Leistungen und
- $L05.3$ die SOLL-Leistungsdaten des Kunden.

$L08$ CF weist auf einem großen Bildschirm im Fitness-Raum eine monatliche Bestenlisten aus:

- $L08.1$ pro SG
- $L08.2$ pro Jahrgang und Geschlecht

$L09$ CF berechnet die Gebühren benutzungsabhängig,

$L09.1$ schreibt Rechnungen und

$L09.2$ überwacht die Forderungen (Bankeinzugsverfahren).

$L10$ CF druckt auf Anforderung des Kunden dessen sämtlichen Daten graphisch aufbereitet aus.

CF-Soft-/Hardware-Spezifikation:

$L11$ Jedes SG hat einen eigenen Rechner (SG-CPU) mit Bildschirm, Sensoren für die Leistungserfassung und MK-Leser (vgl. $L02$).

$L12$ CF fußt auf einer *client-server*-Architektur. Das verknüpft die SG-CPUs mit einem Studio-Rechner (CF-CPU).

$L13$ Für Bilanzierungs- und Wartungszwecke ist die CF-CPU über eine Wählmodemstrecke mit dem zentralen *Host* der Fitness-Studiokette verbunden.

E.20.2:
Aus Platzgründen hier nicht skizziert.

Anhang G

Index

Index