Spezifikationsmethoden für logische Abläufe in Echtzeit-Systemen

Autor(en): **Vogel, Eberhard W.**

Objekttyp: **Article**

Zeitschrift: **Technische Mitteilungen / Schweizerische Post-, Telefon- und Telegrafenbetriebe = Bulletin technique / Entreprise des postes, téléphones et télégraphes suisses = Bollettino tecnico / Azienda delle poste, dei telefoni e dei telegrafi svizzeri**

Band (Jahr): **61 (1983)**

Heft 3

PDF erstellt am: **11.07.2024**

Persistenter Link: https://doi.org/10.5169/seals-875695

Spezifikationsmethoden für logische Abläufe in Echtzeit-Systemen

Eberhard W. VOGEL, Bern

Zusammenfassung. *Spezifikationsmethoden für logische Abläufe in Echtzeitsystemen gewinnen zunehmend an Bedeutung, als Kommunikationsmittel, zur Dokumentation, um die internen Abläufe durch den Computer analysieren, simulieren, verifizieren zu können usw., sowie im Rahmen von Entwurfsmethoden. Der Autor versucht eine theoretische Klärung, worin die Spezifikation eigentlich besteht und welche Methoden für die formale (d. H. von Computer auswertbare) Spezifikation zur Verfügung stehen. Diese Methoden werden durch Beispiele aus der Literatur erläutert.*

Méthodes de spécification applicables aux opérations logiques des systèmes de traitement en temps réel

Résumé. *Les méthodes de spécification en question prennent de plus en plus d'importance, en tant que moyens de communication, pour la documentation, l'analyse, la simulation et la vérification par ordinateur d'opérations internes, etc., ainsi que dans le contexte de l'étude de projets. L'auteur tente d'élucider théoriquement en quoi consiste la spécification proprement dite et d'établir quelles méthodes (c'est-à-dire évaluables par ordinateur) peuvent être appliquées à la spécification formelle. Ces méthodes sont expliquées à l'aide d'exemples tirés d'ouvrages spécialisés.*

Metodi di specificazione per processi logici in sistemi a tempo reale

Riassunto. *I metodi di specificazione per processi logici in sistemi a tempo reale assumono sempre più importanza, come mezzo di comunicazione, per la documentazione, al fine di poter analizzare, simulare, verificare i processi interni mediante il calcolatore, come pure nel quadro di metodi di progettazione. L'autore tenta di spiegare teoricamente in che cosa consista la specificazione e quali siano i metodi a disposizione per la specificazione formale (cioè valutabile mediante calcolatore). I metodi vengono illustrati con esempi dalla relativa letteratura.*

1 Einleitung

Im folgenden soll ausschliesslich die Spezifikation *logischer Abläufe,* besonders in Echtzeitsystemen, behandelt werden. Diese Art Spezifikation kann man etwa als eine Beschreibung definieren, bei der unwichtige Einzelheiten weggelassen werden, und sie kann recht verschiedenartigen Zwecken dienen. Es ist ratsam, sich dies bei der Beurteilung von Spezifikationsmethoden oder -sprachen stets vor Augen zu halten.

In erster Linie ist jede Spezifikation *Kommunikationsmittel.* Sie wird beispielsweise gebraucht, damit der Auftraggeber eines Gerätes dem Hersteller seine Wünsche klarmachen kann (Definition der Anforderungen). Oder Standardisierungskomitees, wie CCITT und ISO, beschreiben, wie gewisse Prozesse (Signalisierung, Protokolle ...) nach ihrer Vorstellung auszusehen haben usw. Unter Umständen mag die Spezifikation auch der Kommunikation zwischen Mensch und Maschine dienen. Ein wichtiges Gebiet ist ferner die *Dokumentation.* Jedes Gerät oder System, das hergestellt wird, sollte ausreichend dokumentiert werden, damit man noch nach Jahren sehen kann, wie es aufgebaut ist und funktioniert (um es zum Beispiel warten zu können).

Auch heute noch werden selbst die grössten Systeme meist nur in «Prosa», d. h. in der Umgangssprache, beschrieben, wobei diese Beschreibung gewöhnlich durch Schemata, Tabellen, grafische Darstellungen und andere Illustrationen ergänzt wird. Ob eine solche Darstellung lesbar ist, ist ganz von den schriftstellerischen Talenten der Verfasser abhängig, und hier steht es nach aller Erfahrung nicht zum besten. Wichtige Information ist häufig über 10...20 Ordner verstreut, und es ist fast unmöglich, Widersprüche und Lücken in der Beschreibung festzustellen. Da es umständlich ist, verbale Beschreibungen, Illustrationen usw. zu ändern, gibt eine solche Dokumentation mit ziemlicher Sicherheit nicht den letzten Stand der Dinge wieder. In diesem Artikel sollen vorwiegend *formale* Methoden und Sprachen beschrieben werden, die durch eine präzise Definition charakterisiert sind, welche eine Auswertung durch den Computer ermöglicht. Sobald man zu formalen Spezifikationsmethoden übergeht, lässt sich hoffen, kompaktere Darstellungen zu erhalten, in denen alle gewünschten Informationen verhältnismässig leicht zu finden sind und bei welchen automatisierte Mittel benützt werden können, um Widersprüche und Lücken festzustellen sowie die Beschreibung auf dem letzten Stand zu halten. Stets tritt auch das Problem auf, dass sich Neulinge anhand der Dokumentation einarbeiten müssen. Dann ist es hilfreich, wenn Beschreibungen auf verschiedenen Abstraktionsebenen möglich sind. Wie erwähnt, ist eine wichtige Eigenschaft formaler Spezifikation ihre automatische Auswertbarkeit.

Dank der Fortschritte in der Elektronik wachsen die technischen Möglichkeiten unabsehbar. Damit werden allerdings auch die Systeme ständig komplexer, und die *Beherrschung dieser Komplexität* ist ein Problem, das ohne Mithilfe des Computers nicht mehr zu bewältigen ist. Neben der Prüfung der Anforderungen auf Vollständigkeit und Widerspruchsfreiheit handelt es sich beispielsweise darum, bestimmte Eigenschaften eines Prozesses nachzuweisen (wie die Freiheit von Selbstblokkierung oder dass der Prozess nicht in einer Schleife steckenbleibt), um Simulation, das Generieren von Test-Algorithmen usw., aber auch um viel trivialere Dinge, wie etwa die Beantwortung der Frage, welche Moduln auf eine gegebene Variable Zugriff haben.

Umfangreiche Anwendung finden Spezifikationsmethoden schliesslich beim *Entwurf von Systemen.* In der Softwarepraxis hat die Spezifikation hauptsächlich auf diesem Wege Einzug gehalten. Daher ist es zu verste-

hen, dass mancher in Zusammenhang mit Spezifikation zuerst oder gar ausschliesslich an Entwurfsmethodik denkt. Man muss aber beachten, dass beim Entwurf besondere Anforderungen an die Spezifikation gestellt werden. So sollte sie beispielsweise eine Entwicklung in der Art der sukzessiven Einführung weiterer Einzelheiten (Topdown-Manier) begünstigen oder mindestens ermöglichen. Der wichtigste Punkt ist jedoch, dass alles Entwerfen ein dynamischer Vorgang ist und man zu Beginn häufig noch gar nicht genau weiss, wie das Ganze schliesslich aussehen soll. Das ist eine völlig andere Situation, als wenn man etwa gewisse Eigenschaften eines gegebenen Systems mathematisch nachweisen will.

Diese Betrachtungen können vielleicht schon einen ersten Eindruck von der Vielfalt des Gebietes geben, das hier behandelt werden soll. Im folgenden wird versucht, einige wichtige Aspekte aufzuzeigen. Zuerst soll eine — notwendigerweise vorläufig — Antwort auf die Frage gegeben werden, wie sich die Aufgabe der Spezifikation aus theoretischer Sicht darstellt. Es folgen einige Betrachtungen, die der Klärung des Begriffs Echtzeitsystem bzw. -prozess dienen. Anschliessend wird ein Katalog der Anforderungen aufgestellt, die — je nach Anwendungszweck — an Spezifikationssprachen zu stellen sind. Im nächsten Abschnitt wird sodann versucht, einen Überblick über die bisher bekannten Methoden zur Spezifikation zu geben. Diese Methoden werden im letzten Abschnitt durch einige Beispiele aus der Literatur illustriert.

Zur Terminologie: Statt von logischen Abläufen wird im folgenden von (diskreten) *Prozessen* gesprochen, wobei wir einen Prozess mathematisch als eine Folge von Ereignissen definieren wollen (was immer das sein mag) oder besser als eine Menge solcher möglicher Folgen, da die Prozesse in der Praxis häufig ein Zufallselement enthalten. Dieses hat seine Ursache im menschlichen Verhalten oder generell in der Unberechenbarkeit des Verhaltens der Umwelt, im Auftreten von Störungen, dem Versagen von Komponenten usw. Ein *System* ist dann eine (z. B. digitale elektronische) Vorrichtung oder Anlage, die einen Prozess verwirklicht.

2 Aufgabe der Spezifikation

Nach unserer Definition ist jede Spezifikation eine Beschreibung eines Systems, Prozesses, Programms usw. Die wichtigste Feststellung in diesem Zusammenhang ist, dass man einerseits die zu erfüllende Aufgabe, anderseits aber auch Lösungen oder Lösungsansätze beschreiben kann, und dass mit einer Aufgabe noch keineswegs die Lösung gegeben ist.

Dazu ein Beispiel: Die Aufgabe sei, die Elemente der Menge

$$\{ n \in \{ 2,3 \} \mid (\exists\, x,y,z \in Z-\{0\})\ [x^n+y^n=z^n] \}$$

zu bestimmen, wobei Z die Menge der ganzen Zahlen bedeutet. Dies ist eine endliche Menge mit mindestens einem und höchstens zwei Elementen. Wir können diesen mathematischen Ausdruck ohne weiteres als eine formale Spezifikation der Aufgabe ansehen. Wenn *Fermats* Vermutung richtig ist, besteht die Menge allein aus dem Element 2, aber es handelt sich eben nur um eine Vermutung. Anderseits lässt sich die Menge nicht effektiv durch Ausprobieren bestimmen, weil man dann unendlich viele Fälle untersuchen müsste. Eine Lösung dieser Aufgabe ist also vorläufig nicht in Sicht und jedenfalls aus der Aufgabenstellung nicht abzuleiten.

Die Beschreibung der Aufgabe allein wird manchmal als Ideal der Spezifikation angesehen. Man sagt auch, es solle nur definiert werden, *was* zu tun ist und nicht das *Wie*. Das gilt besonders für Programme.

Beispiel: Schreibe ein Programm, das alle möglichen Aufstellungen von acht Damen auf einem Schachbrett bestimmt, bei denen keine eine andere schlagen kann. Mit dieser Aufgabenstellung ist noch kein Lösungsweg gegeben, aber das Programm ist auf höherer Abstraktionsebene eindeutig gekennzeichnet.

Handelt es sich jedoch um die Spezifikation eines Systems, so wird man im allgemeinen auch gewisse Lösungsansätze vorschreiben wollen. Nehmen wir beispielsweise an, es sei ein Digitalkonzentrator formal zu spezifizieren, und zwar so, dass die Spezifikation alles Wesentliche enthält. Die Spezifikation als eine einzige umfassend Aufgabe, ohne jedes Eingehen auf eine Gliederung in Teilaufgaben, wäre etwas total Unanschauliches, ungeheuer Komplexes, etwas dem menschlichen Geist nicht Angemessenes. Komplexe Systeme lassen sich nur als Komplex konzipieren und nicht als strukturlose Einheit.

In der Praxis besteht eine Spezifikation daher im allgemeinen aus der Definition von einerseits Aufgaben und anderseits Teillösungen. Dies sei durch das Schema der *Tabelle I* veranschaulicht. Es lassen sich dabei verschiedene *Abstraktionsebenen* unterscheiden: In der obersten existiert nur die Aufgabenstellung A. In der nächsten Ebene haben wir eine Teillösung B, durch die A in die Teilaufgaben A_1 und A_2 zerlegt wird; A ist damit gelöst und wird nicht mehr aufgeführt. Als einfaches Beispiel denke man die Berechnung des Ausdrucks

$$x \cdot (y+z)$$

wobei x, y, z gegeben sind. Eine Teillösung besteht darin, die Faktoren einzeln zu betrachten und insbesondere zuerst y + z zu berechnen. Die neuen Aufgaben sind nun die Berechnung von R := y + z und von x · R.

In den nächsten Abstraktionsebenen werden dann A_1 und A_2 zerlegt usw. In der tiefsten Ebene hat man nur noch Lösungen. Die ideale Spezifikation besteht nun darin, ein System auf einer *frei gewählten* Ebene zu beschreiben. Die Spezifikation auf der tiefsten Ebene wäre

Tabelle I. Schematische Darstellung verschiedener Abstraktionsebenen bei der Spezifikation der Aufgaben und ihrer Teillösungen

Abstraktionsebenen	Aufgaben	Lösungen
	A	——
	A_1, A_2	B
	$A_{11}, A_{12}, A_{13}, A_2$	B, B_1
	------	------
	——	$B, B_1, B_2, B_{11}, \ldots$

die vollständige Beschreibung einer Implementation des Systems.

In der Praxis gibt es meist gewisse Standardaufgaben, für die Lösungen bekannt sind und die man nicht genau zu definieren braucht. So genügt es beispielsweise, bei der Spezifikation einer Telefonzentrale informal anzugeben, dass sie das Signalisierungssystem Nr. 6 unterstützen soll, denn dieses ist anderswo definiert. Überhaupt zeigt sich, dass auf den höchsten Abstraktionsebenen informale Beschreibungen eher angemessen sind als formale, weil die präzise Definition der Aufgaben sinnlos komplex wäre. Damit soll nicht gesagt sein, dass man sich beim Entwurf eines Systems nicht gründlich überlegen sollte, welche Aufgaben zu erfüllen sind!

Dieses Schema gilt allerdings hauptsächlich für die Spezifikation eines bestehenden Systems. Beim Entwurf tritt das Problem auf, dass es sich um einen iterativen Vorgang handelt, bei dem vieles erst nach einigen Iterationen endgültig festgelegt werden kann. Es sollte daher die Möglichkeit bestehen, sich vage auszudrücken. Manche sehen darin sogar das Charakteristikum einer echten Spezifikationsmethode — im Gegensatz z. B. zu Programmiersprachen —, die alle Informationen vermitteln, die für eine vollständige Implementation nötig sind. Man muss sich aber darüber im klaren sein, dass automatisierte Methoden nur soweit angewendet werden können, wie formale Spezifikationen vorliegen. Vage Angaben in einer Spezifikation sind Lücken in der formalen Beschreibung.

Viel hängt davon ab, in welcher Form die Information dargeboten wird. Beispielsweise lassen sich auch Beziehungen zwischen Objekten (wie «Prozess A *benützt* Prozess B», «Prozess C *empfängt* Input D» usw.) formal behandeln, wie dies bei Datenbanken gemacht wird. [11], ohne dass man näher auf die Bedeutung der Beziehungen einzugehen braucht, siehe auch [2].

Schliesslich sei noch angemerkt, dass es Bestrebungen gibt, mit den Methoden der künstlichen Intelligenz für gegebene Aufgaben Lösungen automatisch zu gewinnen [21]. Wie das Beispiel am Beginn des Abschnittes zeigt, ist dies jedenfalls nicht immer möglich.

3 Echtzeitsysteme und -prozesse

In den Anfängen der Computerära spielte die *Zeit* nur insofern eine Rolle, als man lange auf die Ergebnisse warten musste. Davon abgesehen, erschien sie allenfalls als eine von vielen Variablen in Berechnungen physikalischen Charakters. Die wahre Zeit als Faktor, der das Ergebnis der Berechnung beeinflusst, fand ihren Weg in die Datenverarbeitung erst, als man anfing, Computer zur Prozesssteuerung einzusetzen.

Dies gibt Gelegenheit, auf den Unterschied zwischen *Berechnungen* und *Prozessen* einzugehen. Bei einer Berechnung hat man gewisse Daten als Argumente, auf die gewisse Operationen angewendet werden, und nach einiger Zeit erhält man neue Daten als Resultat. Die Charakterisierung eines Prozesses als «nicht endende Berechnung» ist mindestens schwach. So haben wir es in der Nachrichtentechnik beispielsweise mit kooperierenden, aber sonst relativ unabhängigen Prozessen zu tun, die Signale austauschen (packet switching usw.). Durch diese werden teilweise Aktionen ausgelöst, aber vielfach gewisse Operationen auch nur modifiziert; manchmal werden sie überhaupt ignoriert. Wir können einen Prozess also nur in Ausnahmefällen als eine Folge von durch Signale ausgelösten Berechnungen ansehen, wie etwa beim «transaction processing».

Im Softwarebereich bleiben die Überlegungen häufig auf eine Grosscomputeranlage beschränkt, innerhalb der Signale zuverlässig und ohne merkbare Verzögerung übertragen werden. In der Fernmeldetechnik dagegen werden die Signale von einem unzuverlässigen Übertragungsmedium übermittelt; sie können daher verlorengehen oder auch dupliziert werden, und meist erreichen sie den Empfänger nur mit einer Verzögerung, mit der man rechnen muss. Unter diesen Umständen ist es nicht verwunderlich, dass hinsichtlich der *Parallelität* von Prozessen (concurrency) unterschiedliche Vorstellungen herrschen.

In der allgemeinen Datenverarbeitung benützt man die Parallelität vorwiegend, um gewisse Operationen zu beschleunigen. Ferner entdeckte man, dass sich Aufgaben der Prozesssteuerung leichter programmieren lassen, wenn man verhältnismässig unabhängige «Prozesse» (oder tasks) einführt, die mit Signalen kommunizieren (siehe z. B. CHILL [38] bzw. ADA [9]). Besonders in Zusammenhang mit gemeinsamen Hilfsmitteln (shared resources) ergibt sich eine Vielzahl von Synchronisationsproblemen; typisch ist die Bewältigung durch die Rendez-vous-Technik von *Hoare* [14]. Das Problem der Zeit lässt sich meist auf das Problem «möglichst rasch» reduzieren.

Wenn man parallele Prozesse unter einem allgemeineren Gesichtspunkt betrachtet, stellt sich hauptsächlich das Problem, aus unzuverlässiger Information durch Signale usw. auf den Zustand der Umwelt zu schliessen, um das eigene Handeln anzupassen oder zweckentsprechend einzurichten. In vielen Fällen ist man nur an der Reihenfolge der Ereignisse interessiert, aber diese Reihenfolge hängt häufig von der Dauer gewisser Operationen oder der Verzögerung der Signale usw. ab. Das Problem ist also hier eher «nicht zu früh und nicht zu spät». Die Zeitabhängigkeit des Verlaufs eines Prozesses ist meistens unerwünscht, und man versucht, sie durch «Handshaking»-Mechanismen usw. zu vermeiden. Dies ist jedoch nicht immer möglich. Solche Methoden können höchst unökonomisch sein, die Umwelt hält sich selten an Protokolle, und wenn Signale verlorengehen können, muss man mindestens Zeitüberwachungen einführen.

Im Laufe der Zeit haben sich besondere Echtzeitbegriffe herausgebildet, Denkmodelle, die sich zum Verständnis der Abläufe als nützlich erwiesen haben. Beispiele: Ereignisse, Signale, Wartezustände, Warteschlangen und Puffer, Prioritäten, Timeouts, gegenseitiges Ausschliessen und andere mehr. Diese Konzepte sollten in einer Spezifikationssprache für Prozesse auf jeden Fall einigermassen einfach und sinnfällig ausdrückbar sein.

4 An Spezifikationsmethoden zu stellende Anforderungen

Im folgenden sollen Anforderungen an Spezifikationsmethoden bzw. wünschenswerte Eigenschaften zusammengestellt werden. Man beachte, dass sich diese An-

forderungen teilweise widersprechen. Je nach Anwendungszweck muss man dann die Betonung mehr auf das eine oder das andere legen. Auf jeden Fall ist nicht zu erwarten, dass eine einzige Spezifikationsmethode alle Wünsche erfüllt.

Wie schon mehrfach betont, sollte die Methode *formal* sein. Eine Spezifikationssprache muss in Syntax und Semantik präzis definiert sein; andernfalls ist eine automatisierte Auswertung nicht möglich. In einem gewissen Gegensatz dazu steht die Forderung, dass es möglich sein sollte, Teile einer Spezifikation absichtlich unbestimmt zu halten. Dies ist scharf zu unterscheiden vom möglichen Tatbestand, dass die Interpretation einer Sprache nicht eindeutig ist (ambiguity). Diese letztere Zweideutigkeit ist häufig nicht klar als solche erkennbar, so dass jeder geneigt ist, seine eigene Interpretation für die einzig richtige zu halten. Sobald eine Methode formal ist, lassen sich im allgemeinen Änderungen von Spezifikationen relativ leicht durchführen, etwa bei auf dem Bildschirm erzeugten Diagrammen usw.

Die Methode sollte es idealerweise gestatten, genau *die* Aufgaben und Lösungen zu definieren, die man definieren will, nicht mehr und nicht weniger (siehe Abschnitt 2). Es ist zum Beispiel nicht gut, wenn man stets einen bestimmten Signalmechanismus (beispielsweise mit FIFO-Warteschlange) spezifizieren *muss,* weil andere in der Sprache nicht darstellbar sind. Auch ist es nicht gut, wenn man die detaillierte Organisation einer Warteschlange vorschreiben muss, weil eine solche sonst überhaupt nicht dargestellt werden kann (Überspezifikation). Diese Forderung ist nach unserer Erfahrung ein Ideal, das praktisch nie erreicht wird, weil entweder die Ausdrucksmittel fehlen oder diese nicht genügend vielseitig sind. Das Mass der Ausdrucksfähigkeit einer Sprache könnte man vielleicht als ihre *Ausdruckskraft* (expressive power) bezeichnen. Daneben ist häufig die Rede von der *Modellkraft* (modeling power), die angibt, welche Prozesse durch eine Sprache überhaupt darstellbar sind, wenn auch allenfalls nur durch Überspezifikation. Diese Modellkraft stellt erfahrungsgemäss kein grosses Problem dar.

In Zusammenhang mit dem Entwurf ist es wichtig, dass eine Sprache oder Methode auch dann schon mit Nutzen angewandt werden kann, wenn die Aufgaben des zu spezifizierenden Systems noch nicht in allen Teilen festgelegt sind.

Die Methode sollte einfach sein oder auf möglichst wenigen und einfachen Grundsätzen beruhen, nicht nur damit sie leicht erlernbar ist, sondern auch damit sich bei der Auswertung gewisse Tatbestände (wie die Freiheit von Selbstblockierung) mathematisch ohne riesigen Aufwand beweisen lassen. Leider sind die besonders für die Verifikation geeigneten Methoden sehr abstrakt. Damit erfüllen sie eine weitere Forderung nicht, nämlich, dass eine Prozessbeschreibung nicht nur für den Computer, sondern auch für den Benützer leicht verständlich sein sollte. Man muss in diesem Zusammenhang bedenken, dass der Schritt von der intuitiven Vorstellung von einem Prozess zu seiner formalen Darstellung im Prinzip *nicht* durch exakte Methoden überprüft werden kann. Wenn also eine Beschreibung in einer Spezifikationssprache schwer verständlich ist, kann man nie ganz sicher sein, dass sie wirklich das beschreibt, was sie beschreiben soll.

5 Übersicht über die Methoden

In diesem Abschnitt soll versucht werden, die verfügbaren Methoden zu klassifizieren, soweit sie bisher bekannt geworden sind. Wir betrachten ausschliesslich *halbformale* und *formale* Methoden. Zu den ersteren rechnen wir solche, die nur gewisse Aspekte oder Relationen formalisieren, während eine vollständige formale Definition der Prozesse im allgemeinen nicht möglich ist; eine systematische Darstellung ist schwierig und soll hier nicht versucht werden. Dazu einige Beispiele:

Man kann zunächst eine Tabelle aufstellen, in der links gewisse Umstände und Ereignisse, rechts die entsprechenden Aktionen, mit denen reagiert werden soll, aufgeführt werden. Dies wird durch die sogenannten Entscheidungstabellen formalisiert [10]. Zu den halbformalen Methoden kann man auch Zustandsdiagramme [23] und Flussdiagramme [15] zählen oder eine Mischung beider (wie bei SDL [26]), wobei die Kästchen gewöhnlich verbalen Text oder «pictorial elements» enthalten. Eine weitere Möglichkeit besteht darin, Information über ein System in Listen zu sammeln, gegliedert nach *Objekten*, deren *Eigenschaften* und *Beziehungen* (vgl. z. B. [31]). Halbformale Methoden erlauben bereits eine gewisse automatisierte Auswertung. Sie können besonders beim Entwurf von Systemen mit Vorteil angewendet werden und sind auch geeignet, in Verbindung mit verbalen Beschreibungen gewisse Zusammenhänge zu illustrieren.

Interessanter ist die Frage, wie man ein System vollständig formal spezifizieren kann. Da lassen sich vor allem drei Wege unterscheiden (die man auch kombinieren kann). Die Eigenschaften eines Systems können — wenigstens im Prinzip — als Prädikate betrachtet werden. Man kann daher zunächst daran denken, die «Aufgaben» im Sinne von Abschnitt 2 mit Hilfe des Prädikatenkalküls zu definieren oder in einer Sprache, in der dieses Kalkül eingebaut ist (siehe z. B. [12, 18]). Sobald man die Welt der Prozesse durch ein bestimmtes Modell begrenzt hat, kann man den Prädikatenkalkül auch durch andere Formalismen ersetzen; ein Beispiel dazu sind die «synchronisation trees» in [22]. In diesem Zusammenhang wird ferner von manchen Autoren temporale Logik verwendet [7, 20].

Eine grössere Bedeutung haben *axiomatische Methoden,* wie sie unter dem Stichwort «abstract data types» bekannt geworden sind [19]. Analog zur axiomatischen Methode der Mathematik werden dabei Objekte mit den zugehörigen Attributen und Operationen durch Axiome definiert, die die Objekte, Attribute und Operationen zueinander in Beziehung setzen (siehe z. B. die *Peano*-Axiome für die natürlichen Zahlen [6]). Dies sei durch das berühmte Standardbeispiel eines «stack» illustriert: Wir haben es mit zwei Objekttypen zu tun, genannt STACK und ITEM. Es gibt eine Konstante NEW vom Typ STACK (leerer stack). Ausserdem werden drei Operationen (partielle Funktionen) definiert:

- PUSH: STACK × ITEM → STACK
- POP: STACK → STACK
- TOP: STACK → ITEM

(d. h., die Funktion PUSH hat zwei Argumente vom Typ STACK bzw. ITEM und ergibt ein Objekt vom Typ STACK usw.). Dazu genügen vier Axiome:

- TOP (PUSH) (S,I)) = I
- POP (PUSH) (S,I)) = S
- TOP (NEW) = error
- POP (NEW) = error

Sie gestatten es insbesondere, jedes Objekt vom Typ STACK durch wiederholte Anwendung der Operation PUSH auf NEW darzustellen. Bei der axiomatischen Methode gibt es verschiedene Varianten (siehe z. B. [5] und [13]); im Beispiel wurde die sogenannte algebraische Methode benützt. Ursprünglich zum Einführen neuer Datenobjekte in Programmiersprachen gedacht (siehe das Klassenkonzept in SIMULA [27]), eignen sich diese Methoden jedoch auch zur Definition von Prozessen; dabei können sowohl Aufgaben als auch Lösungen spezifiziert werden. Derartige Spezifikationen sind sehr abstrakt und für Dokumentationszwecke kaum geeignet, dagegen ideal, wenn es sich darum handelt, Zusammenhänge mathematisch zu beweisen.

Als dritte lässt sich eine Methode identifizieren, die wir die der *mathematischen Modelle* nennen wollen. Bei dieser wird mit mathematischen Mitteln eine abstrakte Maschine definiert, die einen Prozess realisiert oder genauer eine Klasse solcher Maschinen. Eine Spezifikationssprache besteht dann aus einem Symbolismus, der es gestattet, verhältnismässig einfach und sinnfällig eine bestimmte Maschine aus der Klasse zu bezeichnen. Natürlich kann man immer mit mathematischen Mitteln beliebige Prozesse definieren, aber eine solche Darstellung wäre im allgemeinen weder einfach noch sinnfällig oder leicht zu verstehen. Einfache Beispiele für die dritte Methode sind der endliche Automat [23] und die Petri-Netze [25].

Um auf das Schema von Tabelle I zurückzukommen: Was bei dieser Methode spezifiziert wird, sind im Grunde Lösungen, allerdings meist Lösungen auf höherer Abstraktionsebene. Beispielsweise verkörpern die «Transitionen» in Petri-Netzen Aufgaben, die nicht näher beschrieben werden oder die im Rahmen des Symbolismus allenfalls wieder durch Teillösungen spezifiziert werden können, vgl. auch die «allgemeine Netztheorie» [8]. Man kann das sogar als Vorteil ansehen, weil dadurch ähnliche Lösungen bei verschiedenartigen Anwendungen klarer hervortreten und die Bildung allgemeiner Theorien erleichtert wird.

Wie schon erwähnt, haben sich in Zusammenhang mit Echtzeitsystemen gewisse spezifische Begriffe herausgebildet, und beim Konzipieren komplexer Systeme pflegt man grossenteils in derartigen Begriffen zu denken. Es liegt daher nahe, solche Begriffe als Grundelemente für das mathematische Modell zu verwenden. Dabei wird gleichzeitig die bei dieser Methode bestehende Gefahr der Überspezifizierung minimiert. Damit sich die Spezifikation auf höherer Ebene bewegt und nicht die Verwirklichung in allen Einzelheiten vorschreibt, besteht üblicherweise eine stillschweigende Übereinkunft darüber, die Definitionen der Grundelemente als «sample definition» aufzufassen, d. h. als Beispiele dafür, wie sie implementiert werden könnten, und im übrigen jede andere Implementierung zu erlauben, die dasselbe äussere Verhalten gewährleistet. Im Prinzip könnte man die Grundelemente auch axiomatisch definieren. Der grosse Vorteil der Methode der mathematischen Modelle liegt darin, Beschreibungen zu liefern, die verhältnismässig leicht verständlich sind.

6 Beispiele

In diesem Abschnitt sollen einige Beispiele kommentiert werden. Es ist nicht möglich, im Rahmen dieses Berichtes die Methoden im einzelnen zu beschreiben, und es muss daher auf die angegebene Bibliographie verwiesen werden. Zunächst eine Bemerkung über gewisse verbreitete formale Hilfsmittel, nämlich über Programmiersprachen, Hardware-Beschreibungssprachen [35], Schaltschemata und dergleichen. Diese Mittel dienen der Beschreibung von Implementationen; auch sie können als Spezifikationssprachen angesehen werden, aber sie beziehen sich im allgemeinen auf ein festes, meist tiefes Abstraktionsniveau.

Weitverbreitete Hilfsmittel für die Spezifikation von Prozessen sind ferner Entscheidungstabellen [10], Zustandsdiagramme [23] und Petri-Netze [25]. Entscheidungstabellen können zu den halbformalen Methoden gezählt werden. Zustandsdiagramme und Petri-Netze beruhen auf mathematischen Modellen. Besonders das den Petri-Netzen zugrunde liegende Modell zeichnet sich nicht nur durch Einfachheit, sondern auch durch grosse Vielseitigkeit aus. Erhebliche Anstrengungen wurden unternommen, Algorithmen zu entwickeln, mit denen gewisse Eigenschaften der Netze nachgeprüft werden können (z. B. ob sie «live» sind). In dieser Hinsicht wurde Arbeit von fundamentaler Bedeutung geleistet [8]. Es hat sich aber leider herausgestellt, dass alle diese Algorithmen so komplex sind, dass sie ausser in einfachsten Fällen praktisch nicht angewendet werden können. Der erhoffte Ausweg ist die «allgemeine Netztheorie». Es lässt sich nachweisen, dass man bei Verwendung der sogenannten schwachen Transitionsregel jeden Prozess von praktischer Bedeutung modellieren kann, sobald man zusätzlich Prioritäten einführt; dabei ist angenommen, dass man weder Wahrscheinlichkeiten noch Zeiten spezifizieren will. In den meisten Fällen wäre eine solche Darstellung allerdings überspezifiziert oder völlig abstrakt, d. h. bezüglich Ausdruckskraft lassen Petri-Netze sehr zu wünschen übrig. Es verwundert deshalb nicht, dass fast jeder, der Petri-Netze praktisch anwenden will, irgendwelche Erweiterungen einführt, um die Ausdruckskraft zu verbessern. Auch in der Literatur finden sich zahlreiche derartige Erweiterungen oder Methoden, die auf Petri-Netzen basieren (siehe z. B. [24, 30]).

Es gibt eine grosse Zahl von Methoden, die als Hilfe bei der Entwicklung komplexer Systeme gedacht sind und mehr oder weniger formale Mittel als Bestandteil enthalten [16]. Sie berühren unser Thema nur am Rande, besonders da sie meist höhere Abstraktionsebenen anvisieren; der Vollständigkeit halber sollen sie aber erwähnt werden. Als Beispiele seien genannt: PSL/PSA, SADT und SREM. Bei PSL/PSA [31] werden die Systeme allein durch ein Schema von Objekten (wie Prozesse, Input, Output, Ereignisse usw.) und deren Eigenschaften und gegenseitigen Beziehungen beschrieben; eine gewisse automatisierte Auswertung ist möglich (Prüfen der Beschreibung auf Vollständigkeit und Widerspruchsfreiheit usw.). Bei SADT [28, 29] ist das Vorgehen ähnlich, durch das Benützen von Diagrammen anschaulicher, aber weniger formalisiert. Auch SREM [1] verfolgt eine ähnliche Linie wie die vorgenannten Methoden, ist jedoch mehr auf Echtzeitanwendungen aus-

gerichtet. Es handelt sich um ein Konglomerat heterogener Mittel und Wege, gewissermassen um den heroischen Versuch, trotz fehlenden theoretischen Hintergrundes ein Maximum an Computerunterstützung zu gewinnen. Der rechnermässige Aufwand ist hoch. Zur Methode gehört auch eine Beschreibungssprache RSL, die auf einem ziemlich einfachen Modell beruht.

Seit einigen Jahren wurde die Frage der formalen Spezifikation von Standardisierungskomitees aufgegriffen. So entwickelte die International Electrotechnical Commission (IEC) sogenannte «function charts» für Kontrollsysteme [36]. Diese Methode hat eine gewisse Verwandtschaft mit den Petri-Netzen; der Anwendungsbereich erscheint sehr beschränkt, und die Definition der Methode ist dürftig. Beim CCITT wurde eine grafische «Functional Specification and Description Language» (SDL) [26, 37] entwickelt, besonders im Blick auf die Darstellung von Vermittlungsvorgängen in Telefonzentralen usw. Diese Diagramme sind eine Mischung von Zustands- und Flussdiagrammen mit dahinterstehendem, gut definiertem mathematischen Modell. Die Methode ist insofern halbformal, als der Text in den Kästchen beliebig gewählt werden kann; es können auch bildliche Elemente (pictorial elements) verwendet werden. Sobald man den Text formalisiert, z. B. indem man Befehle einer Programmiersprache benützt, erhält man eine wirklich formale Methode. Gewisse Mängel (fehlende Struktur, fehlendes Datenkonzept usw.) sind bekannt, und man ist gegenwärtig bemüht, die Sprache zu verbessern und zu erweitern. Auch bei der ISO werden Spezifikationsmethoden entwickelt, aber es liegen noch keine endgültigen Ergebnisse vor.

Als Beispiel für die axiomatische Methode sei ein vom CNET[1] entwickeltes Werkzeug zur Spezifikation, genannt OASIS [3], erwähnt. Dieses basiert auf der algebraischen Variante. Hier war das Hauptziel, mathematische Beweise über Systemeigenschaften durchzuführen oder nachweisen zu können, ob eine Implementation einer gegebenen Spezifikation entspricht usw. Der komplexeste Prozess, der bisher spezifiziert wurde, ist das sogenannte «alternating bit protocol» [4].

Bei den Schweizerischen PTT-Betrieben wurde eine Sprache, genannt SYM [32, 33, 34], entwickelt mit dem ursprünglichen Zweck, Echtzeitsysteme rein formal zu beschreiben. Diese Sprache beruht auf einem sehr allgemeinen Prozessmodell, das besonders die Zeit einschliesst. Diesbezüglich unterscheidet sich SYM von den meisten anderen Spezifikationssprachen. Das Problem der Überspezifikation wurde in der Weise gelöst oder mindestens verringert, dass die Sprache durch benützerdefinierte Makros erweitert werden kann; deren Definition ist dann als «sample definition» zu verstehen. Es wurde mit der Implementation eines SYM-Simulators begonnen, so dass SYM in Zukunft auch als Simulationssprache für Echtzeitprozesse verwendet werden kann. Ferner wurde eine auf SYM abgestimmte Diagrammtechnik entwickelt, die eine gewisse Ähnlichkeit mit SDL hat. Sie ist fast ebenso einfach, enthält jedoch die Zeit, Strukturierungsmöglichkeiten und mehr Optionen bezüglich des Signalmechanismus. Die Diagrammtechnik kann auch mit SYM kombiniert werden.

[1] CNET = Centre National d'Etudes des Télécommunications

Schliesslich seien noch zwei mehr theoretisch orientierte Methoden erwähnt. In beiden Fällen gibt es ein zugrunde liegendes einfaches Prozessmodell, und gemäss der ersten in Abschnitt 5 aufgeführten formalen Vorgehensweise besteht die Spezifikation in der formalen Definition einer Aufgabe. *R. Milner* entwickelte mit seinem Calculus of Communicating Systems [22] eine Art Algebra zur Beschreibung von Prozessen. Ein Prozess ist dabei durch die Folge der jeweils akzeptablen «inputs» gekennzeichnet, und es wird eine «Beobachtungsäquivalenz» definiert. Die Betonung liegt auf der Verifikation, dem mathematischen Nachweis von Eigenschaften usw. Das Modell im Hintergrund ist im wesentlichen der endliche Automat, der in Rendez-vous-Technik mit anderen Automaten kommuniziert. Die Systembeschreibungen sind recht abstrakt und nicht leicht verständlich.

Der Spezifikationssprache COSY [17] liegt ein Modell zugrunde, bei dem ein Prozess als eine im allgemeinen nichtdeterministische Folge von (nicht näher spezifizierten) Operationen aufgefasst wird. Besonders wurde dabei an die Verwaltung gemeinsamer Hilfsmittel (shared resources) gedacht. Prozesse werden dadurch spezifiziert, dass durch sogenannte «path expressions» eine Teilordnung der Operationen definiert wird, d. h. es wird spezifiziert, welche Operation auf welche unmittelbar folgen darf. Der Hauptzweck ist offensichtlich wiederum das Beweisen von Eigenschaften und Beziehungen. Leider ist es nicht einfach, aus den «path expressions» usw. ein klares Bild zu gewinnen, wie der definierte Prozess tatsächlich abläuft.

Es ist anzunehmen, dass diese Methoden in Zukunft weniger für Dokumentationszwecke als bei der automatisierten Verifikation usw. eine Rolle spielen werden.

7 Schlussbetrachtungen

Der Leser, der beabsichtigt selbst formale Spezifikationsmethoden anzuwenden, wird sich nun vielleicht fragen, welche Methode er anwenden soll. Dazu muss er sich erst klar werden, welchen Zweck er mit einer solchen Methode hauptsächlich verfolgen will. Ist es eine rechnergestützte Entwicklung (CAD), eine bessere Dokumentation, will er verifizieren und die Eigenschaften eines Systems nachweisen, will er durch Simulation Daten über die Leistungsfähigkeit gewinnen ...? Der Autor hofft, gezeigt zu haben, dass man nicht alles auf einmal haben kann, wenigens sind die Methoden heute noch nicht soweit fortgeschritten.

Es gibt die Regel, «man nehme das Beste, was verfügbar ist». Es ist jedenfalls nicht ratsam, auf eine künftige ideale Methode zu warten, ebensowenig wie bei den Programmiersprachen. Aber was ist verfügbar? Wenn man hauptsächlich an den Entwurf denkt, kann man auf Methoden wie PSL/PSA, SADT und SREM zurückgreifen, besonders wenn man alles auf höherer Abstraktionsebene behandeln will. Wegen des mangelnden theoretischen Hintergrundes ist allerdings die Anwendbarkeit (Simulation, Verifikation usw.) begrenzt, und man darf bezweifeln, dass diese Methoden — soweit heute verfügbar — eine grosse Zukunft haben. Anderseits ist es auch nicht unbedingt eine ideale Strategie, sich an die von grossen Firmen oder Standardisierungskomitees unterstützten Methoden zu halten. Solche Or-

ganisationen haben zwar die Macht, das Überleben ihrer Methoden zu sichern, auch wenn sie alles andere als ideal sind. Aber vielleicht werden sich auf lange Sicht doch einfach die besseren Methoden durchsetzen, denn mit ihnen lässt sich auch besser arbeiten. Standardisierte Methoden kann man nicht ignorieren, aber man braucht nicht unbedingt seine Arbeit darauf aufzubauen. Schliesslich kann sich die Benützung einer Spezifikationsmethode mit beschränkten Ausdrucksmöglichkeiten nachteilig auswirken. Es besteht die Gefahr, dass sich damit sozusagen auch das Weltbild verengt, dass man sich bei Implementationen an die beschränkten Möglichkeiten anpasst und vorteilhaftere Lösungen nicht wählt, nur weil die Dokumentation schwierig ist.

Allgemein anerkannte und gleichzeitig wirklich befriedigende und voll implementierte Methoden sind gegenwärtig nicht in Sicht. Anderseits sollte nach unseren Ausführungen auch klar sein, dass die eigene Entwicklung einer Spezifikationsmethode ein schwieriges und langwieriges Unterfangen ist, durchaus mit der Entwicklung einer Programmiersprache vergleichbar. Es muss dem Leser überlassen bleiben, aus diesen Überlegungen seine eigenen Schlüsse zu ziehen.

Bibliographie

[1] *Alford M. W.* A Requirements Engineering Methodology for Real-Time Processing Requirements. IEEE Transactions on Software Engineering. New York, SE-3, (1977) 1, S. 60.

[2] *Balzer R. et al.* Informality in Program Specifications. IEEE Transactions on Software Engineering. New York, SE-4, (1978) 2, S. 94.

[3] *Barberye G. et al.* OASIS: Un Outil d'Aide à la Spécification Interactive et Structurée. Proc. Journées BIRGE, Grenoble, 1982, S. 199.

[4] *Bartlett K. A. et al.* A Note on Reliable Full-Duplex Transmission over Half-Duplex Links. Communications of the ACM, 12 (1969) 5, S. 260.

[5] *Bartussek W.* and *Parnas D. C.* Using Assertions about Traces to Write Abstract Specifications for Software Modules. In: Proc. Information Systems Methodology, Venice (1978). Lecture Notes in Computer Science, Bd. 65. Berlin, (1978), S. 211.

[6] *Behnke H. et al.* (ed.). Grundzüge der Mathematik, Bd. 1. Göttingen, 1962.

[7] *Bochmann G. V.* Hardware Specification with Temporal Logic: An Example. IEEE Transactions on Computers, C-31 (1982) 3, S. 223.

[8] *Brauer W.* (ed.) Net Theory and Applications. Lecture Notes in Computer Science, Bd. 84. Berlin, 1980.

[9] *Brender R. F.* and *Nassi I. R.* What is ADA? Computer Long Beach, Cal., 14 (1981) 6, S. 17.

[10] *Büchi R.* Entscheidungstabellen, Giessen, 1976.

[11] *Chamberlin D. D.* Relational Data-Base Management Systems. Baltimore, Computing Surveys, 8 (1976), S. 43.

[12] *Greif I.* A Language for Formal Problem Specification. Communications of the ACM, 20 (1977) 12, S. 931.

[13] *Guttag J. V.* and *Horning J. J.* The Algebraic Specification of Abstract Data Types. Berlin, Acta Informatica, 10 (1978), S. 27.

[14] *Hoare C. A. R.* Communicating Sequential Processes. Communications of the ACM, 21, (1978) 8, S. 666.

[15] *Koch G.* und *Rembold U.* Einführung in die Informatik, Teil 1. München, 1977.

[16] *Koch G. et al.* Einführung in die Informatik, Teil 2. München, 1980.

[17] *Lauer P. E. et al.* COSY — A System Specification Language Based on Paths and Processes. Acta Informatica. Berlin, 12 (1979), S. 109.

[18] *Leung W. H.* and *Ramamoorthy C. V.* An approach to Formal Specification of Control Modules. IEEE Transactions on Software Engineering. New York, SE-6 (1980) 5, S. 485.

[19] *Liskov B. H.* and *Zilles S. N.* Specification Techniques for Data Abstractions. IEEE Transactions on Software Engineering. New York, SE-1, (1975) 1, S. 7.

[20] *Manna Z.* and *Pnueli A.* Verification of Concurrent Programs: The Temporal Framework. In: Boyer, R. S., Moore, J. S. (ed.). The Correctness Problem in Computer Science. International Lecture Series in Computer Science. London, 1981, S. 1.

[21] *Manna Z.* and *Waldinger R.* Synthesis: Dreams => Programs. IEEE Transactions on Software Engineering, SE-5, (1979) 4, S. 294.

[22] *Milner R.* A Calculus of Communicating Systems. Lecture Notes in Computer Science. Berlin, (1980) 92.

[23] *Minsky M.* Computation: Finite and Infinite Machines. Englewood Cliffs, 1972.

[24] *Noe J. D.* and *Nutt G. J.* Macro E-Nets for Representation of Parallel Systems. IEEE Transactions on Computers, C-22, (1973) 8, S. 718.

[25] *Peterson J. L.* Petri-Nets. Computing Surveys. Baltimore, 9 (1977) 3, S. 223.

[26] *Rockström A.* and *Saracco R.* SDL-CCITT Specification and Description Language. New York, IEEE Transactions on Communications, COM-30, (1982) 6, S. 1310.

[27] *Rohlfing H.* SIMULA — Eine Einführung. BI Hochschultaschenbuch 747, Mannheim, 1973.

[28] *Ross D. T.* Structured Analysis (SA): A Language for Communicating Ideas. New York, IEEE Transactions on Software Engineering, SE-3, (1977) 1, S. 16.

[29] *Ross D. T., Schomann, Jr., K. E.* Structured Analysis for Requirements Definition. New York, IEEE Transactions on Software Engineering, SE-3, (1977) 1, S. 6.

[30] *Symons F. J. W.* Modelling and Analysis of Communication Protocols Using Numerical Petri Nets. PH. D. Thesis, University of Essex, Mai 1978.

[31] *Teichroew E. A.* and *Hershey E. A.* PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. New York, IEEE Transactions on Software Engineering, SE-3, (1977) 1, S. 41.

[32] *Vogel E.* A Model Approach to the Description of Hardware Systems. In: 1975 International Symposium on Computer Hardware Description. Languages and their Applications. New York, IEEE (1975) S. 32.

[33] *Vogel E.* Zur formellen Beschreibung von Echtzeitsystemen. Bern, Techn. Mitt. PTT 56 (1978) 1, S. 22.

[34] *Vogel E.* SYM: Eine formale Sprache zur Beschreibung von Echtzeit-Systemen und -Prozessen. In: Programmiersprachen für die Nachrichtentechnik. Bern, Nachrichtentechnisches Kolloquium 1980/81.

[35] Hardware Description Languages. Encino, Cal., Computer, 7 (1974) Dezember (Special Issue).

[36] International Electrotechnical Commission (IEC), Technical Committee No 3, Subcommittee 3B: Draft: Preparation of Function Charts for Control Systems. Document 3B (Sekretariat) 30. Januar 1982.

[37] Functional Specification and Description Language (SDL). Rec. Z. 101...104, CCITT Yellow Book, vol. VI. 7, Geneva, 1980.

[38] CCITT: Introduction to CHILL. Geneva, 1980.