Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association suisse des électriciens, de l'Association des entreprises électriques suisses

Herausgeber: Schweizerischer Elektrotechnischer Verein ; Verband Schweizerischer Elektrizitätsunternehmen

Band: 73 (1982)

Heft: 2

Artikel: APL : ein unabhängiges und leistungsfähiges Werkzeug zur Beschleunigung der Entwicklung von dialogorientierten Systemen

Autor: Hartmann, U.

DOI: https://doi.org/10.5169/seals-904915

und von Kundeninformationen usw. im geographischen Kontext wirtschaftlich sind. Das Vorhandensein der Kundeninformationen, der Informationen über die Betriebsmittel des Niederspannungs- und Mittelspannungsnetzes sowie der Informationen über die geographische Lage dieser Betriebsmittel und Kunden fordert geradezu auf, die Daten zu verknüpfen. So ist es durchaus denkbar, dass in späterer Zukunft von dem Hausanschluss des einzelnen Kunden bis zur entsprechenden Ortsnetzstation und von dort über das Mittelspannungsnetz bis zum Umspannwerk Verknüpfungen geschaffen und ausgewertet werden können.

Adresse der Autoren

W. Ferenz, Dipl.-Ing., und *M. K. Hoffmann*, Dipl.-Ing., Abteilung Datenverarbeitung der Schleswag Aktiengesellschaft, D-2370 Rendsburg.

APL – ein unabhängiges und leistungsfähiges Werkzeug zur Beschleunigung der Entwicklung von dialogorientierten Systemen

Von U. Hartmann

Die heutige Situation in der EDV ist einerseits durch ein laufend verbessertes Preis/Leistungs-Verhältnis charakterisiert. Dadurch werden immer mehr neue EDV-Projekte wirtschaftlich interessant. Andererseits verzögert der Mangel an Programmierern die Realisierung weiterer EDV-Projekte. Aus dieser Diskrepanz ergibt sich ein sogenannter Anwendungsstau. APL (A Programming Language) passt ideal in diese Situation, indem diese Sprache gegenüber den konventionellen höheren Programmiersprachen etwas mehr Rechenzeit konsumiert, hingegen die Programmierproduktivität erheblich steigert. Ein breites Feld von Anwendungen in Elektrizitätsunternehmen kann mit APL gelöst werden. Eine Auswahl von Beispielen aus verschiedensten Gebieten wie Administration, Planung, Technik, Graphik und Transaktionen zeigt die universellen Möglichkeiten von APL.

La situation actuelle en informatique est caractérisée d'une part par l'amélioration constante de la relation prix/performances, qui fait apparaître de plus en plus de projets informatiques comme économiquement intéressants. Par ailleurs, la pénurie de programmeurs empêche de réaliser de nouveaux projets. Cette situation a pour conséquence un «bouchon» dans le domaine des applications. APL (A Programming Language) s'adapte parfaitement à cette situation car ce langage consomme un peu plus de temps de calcul que les langages conventionnels supérieurs de programmation, mais augmente considérablement la productivité de la programmation. APL permet de résoudre une vaste gamme d'applications dans les entreprises d'énergie électrique. Les possibilités universelles d'APL sont présentées à la lumière d'exemples sélectionnés de différents domaines comme l'administration, la planification, la technique, la graphique et les transactions.

1. Einleitung

Die kontinuierliche Verbesserung des Preis/Leistungs-Verhältnisses auf dem Computer-Hardware-Sektor lässt immer neue Anwendungen für eine EDV-Realisierung interessant erscheinen. Die Zahl möglicher Anwendungen nimmt somit dauernd zu. Andererseits ist seit längerer Zeit generell festzustellen, dass für die Software-Entwicklung nur ungenügend Programmierkapazität verfügbar ist. Dadurch entsteht ein Anwendungsstau, d.h., die Zahl der zu realisierenden Anwendungen nimmt laufend zu und kann kaum abgebaut werden.

Die am weitesten verbreiteten höheren Programmiersprachen FORTRAN, COBOL und PL/1 waren seinerzeit ein wichtiger Schritt zur Verbesserung der Programmierproduktivität gegenüber dem Programmieren in Assembler. Neben den erwähnten Sprachen kann heute für die verschiedensten speziellen Anwendungsbereiche auf entsprechend zugeschnittene, anwendungsorientierte Sprachen zugegriffen werden. Es wird deshalb kurz begründet, weshalb APL neben all den bestehenden Programmiersprachen als universelle Sprache vorgestellt und deren weitere Verbreitung unterstützt werden soll.

Untersuchungen [1], [2] haben ergeben, dass rund 50% aller Programme nur ca. 2% der Systemressourcen benötigen. So ist es naheliegend, nach Möglichkeiten zu suchen, diesen grossen Anteil seltener benutzter Programme derart zu realisieren, dass die Programmerstellung möglichst effizient vorgenommen werden kann. Dagegen wird in Kauf genommen, dass die Ausführung dieser Programme unter Umständen rechenzeitintensiver als in einer anderen Sprache wird.

APL bietet mit seiner sehr prägnanten und mächtigen Symbolik genau diese gewünschten Möglichkeiten. So lässt sich die Produktivität in der Phase der Programmerstellung gegenüber konventionellen höheren Programmiersprachen etwa im Verhältnis 6:1 verbessern, und die Schreibarbeit nimmt ebenfalls im gleichen Verhältnis ab [1], [7]. APL wird nicht übersetzt, sondern mit Hilfe eines Interpreters direkt ab Quellencode verarbeitet, wodurch die Programme rechenzeitintensiver werden können.

2. Entstehung und Verbreitung

APL heisst «A Programming Language» und wurde von K.E. Iverson 1962 mit der Absicht veröffentlicht [3], eine systematische Notation für die Vektor-Algebra zu definieren, die gleichzeitig gestattet, Algorithmen einzuführen. Dieses formale System in Form einer Operatorensprache ist als universelle Programmiersprache für Applikationen aus den verschiedensten Gebieten geeignet.

Die Erweiterung von Operatoren auf Felder beliebigen Ranges sowie die Verallgemeinerung von Vektor- und Matrix-Operationen lehnen sich an entsprechende bekannte mathematische Operationen an. Diese Tatsache muss betont werden, damit klar wird, dass die Arbeit mit APL wesentlich auf einem mathematischen Vorstellungsvermögen aufbaut, wie dies für Vektor- und Matrizenrechnung üblich ist. Mit dieser Aussage wird die universelle Einsatzmöglichkeit von APL keineswegs in Abrede gestellt.

Nach FORTRAN, COBOL und PL/1 ist APL in den USA und Kanada eine der am weitesten verbreiteten Programmiersprachen. Die meisten der bekannten Computerhersteller bieten heute APL-Übersetzer auf ihren Systemen an. Grosse Timesharing-Netze in den USA offerieren ebenfalls die Möglichkeiten der Benutzung von APL. APL eignet sich aber nicht nur als Sprache auf Großsystemen (wie z.B. Univac 1100 oder IBM/370), sondern ist zum Teil auch auf Tischrechnern wie

IBM 5110 [1] und dem Prozessrechner IBM/7 [10] installiert worden. Eine der ersten Implementationen von APL wurde bereits 1968 auf dem System IBM 1130 als IBM-Produkt zur Verfügung gestellt [16].

3. Eigenschaften von APL

Es besteht nicht die Absicht, eine Einführung in die Benutzung von APL zu geben; dazu existiert genügend Literatur, z.B. [4]. Hingegen sollen die Grundprinzipien von APL soweit erklärt werden, dass die Auswirkungen auf das Programmieren und die Unterschiede zu anderen Sprachen diskutiert werden können. Einige Schwächen und deren Gewichtung im Rahmen der Programmierung in APL werden ebenfalls gestreift.

3.1 Zeichensatz

Neben den in höheren Programmiersprachen üblichen Zeichen benötigt APL vor allem über 60 Operatoren als spezielle Symbole. Einige dieser Operatoren werden dabei als sogenannte Overstruck Character aus zwei Spezialzeichen gebildet. Auf der Schreibmaschine geschieht dies durch Schreiben des ersten Zeichens, anschliessend Backspace und Schreiben des zweiten Zeichens. Zum Beispiel setzt sich der Operator ⍋ (sortieren) aus den Zeichen △ und | zusammen. Diese Darstellungsweise kann in gewissen Fällen zu Schwierigkeiten führen, z.B. wenn die Eingabe eines Backspace als Löschung des vorangehenden Zeichens interpretiert wird.

Der besondere Zeichensatz von APL bietet aber grundsätzlich keine Probleme bei der Implementierung von APL. Die üblicherweise als schreibende Terminals eingesetzten Kugelkopfschreibmaschinen können mit dem entsprechenden APL-Kugelkopf ausgerüstet werden. Moderne Bildschirmterminals werden einfach mit einem zweiten Character-Generator ausgerüstet und sind dann zwischen ASCII- und APL-Zeichensatz umschaltbar, z.B. Tektronix 4013 oder Silent TI-700.

In gewissen Implementationen von APL wird die Notwendigkeit des besonderen APL-Zeichensatzes dadurch umgangen, dass ein sogenannter Digraph-Mode [17] definiert wird. Dabei wird ein besonderes Umschaltzeichen (z.B. $) festgelegt, und alle speziellen APL-Operatoren werden durch diese Umschaltzeichen, gefolgt von einer mehr oder weniger aussagefähigen Buchstabenkombination, dargestellt. Diese Variante stellt aber nur ein Notbehelf dar, denn durch eine solche Schreibweise wird die Lesbarkeit und Übersichtlichkeit von APL-Ausdrücken sehr eingeschränkt.

3.2 Daten

In APL werden nur zwei Typen von Daten, nämlich Zahlen oder Zeichenketten, unterschieden. Im Gegensatz zu den bekannten höheren Programmiersprachen wird vom Benützer keine Unterscheidung zwischen Integer-, Real- und Boolschen Variablen verlangt. Diese früher aus Hardwaregründen notwendige Differenzierung ist heute nicht mehr von derselben Bedeutung. In APL wird die Konversion von einer Zahlendarstellung in die andere während der Programmausführung automatisch vorgenommen.

Texte sind immer Vektoren oder Matrizen, wobei jeweils ein Zeichen einer Vektor- oder Matrixkomponente entspricht und individuell ansprechbar ist.

Als Datenstrukturen sind in APL Skalare, Vektoren, Matrizen und mehrdimensionale Matrizen vorgesehen. Eine Variable muss nie im voraus deklariert werden; Typ oder Struktur ergeben sich bei der Zuweisung von Werten. Eine Variable besteht aus einem Dimensionsvektor zur Beschreibung der Datenstruktur sowie dem eigentlichen Datenvektor mit den Werten. Die Tatsache, dass die Struktur von Daten dynamisch verändert werden kann und auch nicht deklariert werden muss, ist als besondere Stärke der Dialogsprache APL zu werten.

Möglichkeiten zum Ansprechen von Teilen einer bestimmten Datenstruktur werden später aufgezeigt. Wie auch in höheren Programmiersprachen üblich, können Komponenten von Vektoren und Matrizen durch Indizieren von Variablen angesprochen werden.

Höhere Datenstrukturen wie z.B. Listenstrukturen sind in APL nicht explizit vorgesehen, können aber auf einfache Art selbst programmiert werden und sind auch in gewissen APL-Implementationen verfügbar. Rekursive Strukturen sind in [13] beschrieben. Die Verknüpfung von Abfragesprachen mit APL erlaubt den generellen Zugriff auf Datenbanken [12].

3.3 Operationen

3.3.1 Elementaroperationen und Ausdrücke

Ausdrücke dienen dazu, Operanden zu transformieren und miteinander zu verknüpfen. Zwei Unterschiede gegenüber anderen Programmiersprachen sind offensichtlich. So erfolgt die Interpretation eines Ausdruckes von rechts nach links, was eine gewisse Angewöhnung erfordert. Ebenfalls im Unterschied zu den üblicherweise gültigen Vorrangregeln zwischen verschiedenen Operatoren gibt es in APL keine Vorrangregeln, d.h., sämtliche Operatoren werden mit gleicher Priorität abgearbeitet. Auch diese Regel ist ungewohnt. Eine andere Prioritätenordnung der Operatoren ist aber kaum sinnvoll, da entsprechende Regeln auch für den Programmierer nur mühsam erlernt und damit richtig angewendet würden.

Die Verarbeitung von Ausdrücken bezieht sich ganz generell auf jede Struktur, wenn gewisse Konformitätsbedingungen erfüllt sind. So wird im Beispiel eines Ausdruckes

```
A + B
```

nicht nur eine skalare Addition von zwei Werten ausgeführt, sondern abhängig von der Datenstruktur von A und B, kann es sich auch um die Addition von Vektoren, Matrizen oder mehrdimensionalen Matrizen handeln.

3.3.2 Strukturoperationen

Das Grundkonzept der dynamischen Definition von Daten durch Struktur und zugehörige Werte berechtigt nicht nur die Werte, sondern auch die Struktur von Daten nach Belieben zu ändern. Eine Fülle von entsprechenden Strukturoperationen gestatten, die Reduktion, das innere und äussere Produkt, Neustrukturierung, Aneinanderreihen, Komprimieren und Expandieren, die Bildung von Indexmengen, die Zugehörigkeit zu Mengen, Sortieren, Entschlüsseln, Verschlüsseln, Formatieren von Daten vorzunehmen. Diese Strukturoperatoren stellen eine wesentliche Erweiterung gegenüber den bekannten Programmiersprachen dar und erlauben in sehr vielen Fällen die Verarbeitung von Feldern anstelle des Schreibens von programmierten Schleifen zur Ausübung einer bestimmten Operation auf die einzelnen Elemente einer Variablen.

3.3.3 Beispiele

Zwei ganz einfache Beispiele sollen die Möglichkeiten dieses Konzeptes erläutern.

In einer Matrix U von 3 Zeilen und 4 Kolonnen sind die Energieumsätze pro Spannungsebene (3 Spannungsebenen) und Quartal (4 Quartale) gespeichert.

```
      ⎕ 10  8  7 11 ⎕
U ← ⎕ 40 55 35 30 ⎕
      ⎕ 88 70 70 87 ⎕
```

Der einfache Befehl einer Reduktion entlang der ersten Koordinate in der Form

```
      +/[1]U
138 133 112 128
```

liefert die totalen Umsätze pro Quartal.

Der Ausdruck

```
      ⌈/U
11 55 88
```

liefert die maximalen Umsätze pro Spannungsebene. Die Einfachheit dieser Operation zeigt klar die Unterschiede zu den konventionellen Programmiersprachen.

Ein weiteres Beispiel bezieht sich auf die Anwendung des inneren Produktes. Die Definition des inneren Produktes ist von der Matrizenrechnung der Mathematik abgeleitet worden, indem Zeilen und Spalten komponentenweise durch Multiplikationen und anschliessende Addition miteinander verknüpft werden. Die Darstellung der Matrix-Multiplikation für zwei Matrizen A und B lautet deshalb in APL

```
A+.×B
```

Anstelle der Operatoren $+ \cdot \times$ kann im allgemeinen Fall eines inneren Produktes eine beliebige andere Kombination von Elementaroperatoren stehen. Im folgenden Beispiel wird das innere Produkt dazu verwendet, die Existenz und Position eines Namens in einer gegebenen Namenliste zu ermitteln. Mit TX als Namenliste kann über das innere Produkt (hier $\wedge \cdot =$) und den Indexgenerator diese Aufgabe auf einfache Weise gelöst werden.

```
      ⎕ ALPHA ⎕
TX ← ⎕ BETA  ⎕
      ⎕ GAMMA ⎕
      ⎕ DELTA ⎕

      (↑TX∧.=5↑'GAMMA')⍳1
```

Als Resultat dieses Ausdruckes erhält man 3, da GAMMA auf der 3. Zeile von TX zu finden ist.

3.4 Programmstrukturen

3.4.1 Anweisungen

Neben der Wertzuweisung (←), die mehrfach pro Zeile vorkommen kann, und der Sprunganweisung (→) sind Eingabe und Ausgabe durch das Zeichen Quad (⎕) definiert. ⎕ steht dabei als dummy-Variable an einer beliebigen Stelle eines Ausdruckes. Bei der Interpretation eines solchen Ausdruckes wird beim Antreffen von ⎕ auf Input gewartet. Alle eingegebenen Werte werden als Daten dieses Ausdruckes weiter verwendet. Das Symbol ⎕ für die Ausgabe kann auch mehrfach innerhalb einer Zeile stehen, beispielsweise um Zwischenresultate eines Ausdruckes zu erhalten.

3.4.2 Prozeduren

APL kennt das Konzept von Prozeduren, die als externe Prozeduren zu verstehen sind, mit lokalen und globalen Variablen arbeiten können und nicht mehr als zwei explizite Parameter aufweisen dürfen. Diese Prozeduren heissen in APL Funktionen und können voll rekursiv verwendet werden. Ausserdem kann eine Funktion gleichwertig zu den bekannten APL-Operatoren in Ausdrücken an beliebiger Stelle stehen. Die Verwendung von Prozeduren gestattet eine klare, modulare Programmierung, sofern die Verwendung von globalen Variablen restriktiv gehandhabt wird.

3.4.3 Strukturierte Programmierung

Mit der Methode der strukturierten Programmierung versucht man, die Idee, das Verhalten und die Dynamik eines Programmes sichtbar zu machen [4]. APL kommt diesen Bestrebungen entgegen, da sehr prägnante Ausdrücke überall dort möglich sind, wo Operatoren auf Datenstrukturen und nicht nur auf Skalare angewendet werden. Obwohl die üblichen Elemente wie «repeat», «while do» oder «if else» fehlen, lassen sich strukturierte Abläufe mit den vorhandenen Möglichkeiten programmieren.

3.5 Dialogfähigkeit

APL kann durch Umschalten zwischen Rechnermodus und Funktionsdefinition jederzeit für die direkte Ausführung der Ausdrücke oder die Definition neuer Programme verwendet werden. Der Umstand, dass APL über einen Interpreter ausgeführt wird, bedingt keine Programmumwandlung während der Programmentwicklung und gibt eine volle Transparenz mit exakter Fehleranzeige während der Programmausführung.

Ein Editor ist in APL fest implementiert. Funktionen können so während der Ausführung verändert werden, und auch Daten können bei Programm-Stops nach Belieben abgerufen und ebenfalls verändert werden.

Die Verbindung von APL zum Betriebssystem ist mittels vorgegebenen Systemkommandos und Systemfunktionen realisiert. Über das Konzept der Arbeitsbereiche (workspace) wird in APL die gesamte Programm- und Datenverwaltung unter Berücksichtigung der Probleme des Datenschutzes und der Datensicherheit gelöst.

All diese Eigenschaften tragen dazu bei, dass die volle Ausrichtung auf Dialogapplikationen in APL sehr gut unterstützt wird und dass durch die gleichzeitige Definition von Editor und Systembefehlen innerhalb APL eine Portabilität angeboten werden kann, die dem Programmierer die Arbeit auf verschiedenen Systemen vereinheitlicht und damit wesentlich vereinfacht.

4. Anwendungen

Das universelle Konzept von APL gestattet den Einsatz dieser Sprache in den verschiedensten Anwendungsgebieten. Die hier angegebenen Beispiele lassen sich natürlich immer auch in einer anderen Programmiersprache lösen.

4.1 Administration

Die Planung über mehrere Jahre, wie z. B. die Investitionsplanung [6], ist ein dynamischer Prozess, der laufenden Änderungen unterworfen ist. Für die Ausführung solcher Planungsrechnungen eignet sich APL ausgezeichnet [7].

Die Budgetierung gilt als typisches Beispiel einer Anwendung, die relativ selten benötigt wird, dann aber rasche Reaktion und eine grosse Flexibilität erfordert. Eine Lösung in APL kann diese Gesichtspunkte berücksichtigen. Neben den laufenden Eingaben von Budgetzahlen im Dialog können jederzeit Teile des Budgets und besondere Summierungen verlangt werden. Spezielle Auswertungen sind so einfach zu programmieren, dass auch die Quartalergebnisse über dieses Budget-Programmsystem in APL verarbeitet werden.

Wird die Budgetrechnung mit der Ergebnisrechnung gekoppelt, so können zusätzlich Umsatz, Absatz und Kosten gezielt analysiert werden [7].

4.2 Message-System

Oft ist es schwierig, dem Benutzer den Ablauf einer transaktionsorientierten Applikation genügend klar veranschaulichen zu können. Hier bietet sich APL an, um mit Prototypen von Applikationen den Transaktionsaufbau und -ablauf experimentell vorzuführen. Damit ist es möglich, Fehler und Schwächen eines Transaktionssystems bereits in der Designphase zu eliminieren [2].

Der Verkehr zwischen Rechenzentrum und Bildschirmbenützern kann durch die Verbreitung von wichtigen Meldungen über das tägliche Betriebsgeschehen verbessert werden. Sind die entsprechenden Informationen in APL gespeichert, so kann über die einfache Funktion DI jede Information von jedem Terminal aus abgerufen werden.

```
    ∇ DI N
[1]   (N=+\T[;1]='$')/[1]T
    ∇
```

4.3 Technische Applikationen

Ein bekanntes Beispiel stellt die Extrapolation von Zeitreihen mit Hilfe von APL dar. Verschiedene Modelle können über entsprechende numerische Methoden der Kurvenapproximation angepasst, statistische Grössen können ermittelt werden, und damit kann eine Extrapolation vorgenommen werden.

Derartige Rechnungen verlangen neben der Möglichkeit, Alternativen schnell und ohne grossen Arbeitsaufwand durchzuführen, auch leicht änderbare und erweiterungsfähige Programme, was von APL in ausgezeichneter Weise unterstützt wird [11].

Die Methode von Box-Jenkins wird in [8] als vollständiger Satz von APL-Programmen beschrieben. Diese wurden am Beispiel des Elektrizitätsverbrauches in Finnland überprüft.

Die Klassifizierung von Daten in wenige repräsentative Klassen wird durch die Methoden der Cluster-Analyse abgedeckt. Solche Verfahren arbeiten mit Matrix-Operationen und Mengenbildungen und eignen sich z.B. für die Berechnung von charakteristischen Lastkurven.

Andere Applikationen aus dem technischen Sektor eignen sich in ähnlicher Weise für eine Realisierung mittels APL. Unter Umständen kann auch nur die Entwicklung eines Algorithmus in APL vorgenommen werden, um anschliessend das ausgetestete Verfahren z.B. in FORTRAN umzusetzen (State estimation).

4.4 Graphische Anwendungen

Der Operator ⍞ ermöglicht den Dialog im reinen ASCII-Zeichensatz zwischen System und Terminal. Damit erhält man auch in APL die vollständige Transparenz auf der Ebene des Kontrollzeichensatzes eines bestimmten Terminals. Das folgende Beispiel zeigt die entsprechende Koordinatentransformation, um für ein graphisches Terminal (Tektronix 4013) eine graphische Ausgabe zu erhalten. Das Koordinatenpaar (x, y) wird mit der terminalspezifischen Matrix M multipliziert.

```
      | 32   0   0   0  1 |
  M ← | 96   0   1  0¯32  |
      | 32   0   0   1  0 |
      | 64   1  0¯32  0   |

    ∇ R←X KOTRAM Y
[1]   R←,M+.×1,X,Y,[1.5](X,Y)+32
    ∇
```

Spezielle graphische Software ist kaum notwendig und kann in komplexeren Anwendungen als Package individuell entwickelt werden [9]. Ein Anwendungsbeispiel stellt die graphische Darstellung der Transaktionsrate über einen Tag dar. Mit Hilfe von Strukturoperatoren werden Mittelwerte, Minima und Maxima gebildet, die als Verlauf dargestellt werden.

In analoger Weise kann die Verwendung von ⍞ als Input-Operator die Koordinaten des Crosshair Cursor liefern. Anwendungsgebiete sind z.B. der Aufbau von Plänen für elektrische Netzschemas oder die Netzplantechnik durch interaktive Definitionen von Knotenpunkten und Verbindungen am graphischen Terminal. Gleichzeitig werden Texte und Daten eingegeben, um etwa mit dem CPM-Algorithmus Projekte durchzurechnen [6].

4.5 Instruktion

Ausbildung und Instruktion via Terminal haben den Vorteil, dass jedermann an seinem Arbeitsplatz und auch zu frei wählbaren Zeiten einzelne Ausbildungsschritte durcharbeiten kann. Für den Instruktor ergeben sich bei Änderungen keine Probleme für die Verteilung von neuen Informationen, und Instruktionen sind dauernd auf dem aktuellen Stand. Neben besonderen Softwaresystemen (ASET bei UNIVAC) eignet sich auch APL für diese Aufgaben [15]. In [6] wird ein Trainingssystem zur Vertiefung der APL-Kenntnisse beschrieben.

4.6 Beweise

Die Korrektheitsbeweise von Programmen sind heute noch sehr umfangreich. Zwei Eigenschaften von APL, nämlich die Reduktion von Operatoren auf einige wenige sowie die Beschreibung von Definitions- und Wertebereich in APL, erlauben grundsätzlich Korrektheitsbeweise von APL-Programmen mittels APL-Beweissequenzen durchzuführen [4], [14].

5. Zukunft

Die Basispublikation zu APL von Iverson erschien 1962 [3], also vor bald 20 Jahren. Die heutige Situation von APL kann im Hinblick auf zukünftige Entwicklungen wie folgt beurteilt werden:

– APL ist bezüglich Hardware auf besondere Ein- und Ausgabemöglichkeiten angewiesen, und die interpretative Verarbeitung ergibt eine relativ starke Prozessor- und Arbeitsspeicherbelastung durch APL-Anwendungen.

– Einige Eigenschaften von APL wie die Rechts-nach-links-Interpretation von Ausdrücken und der stark mathematische Hintergrund bedingen ein Umdenken des Programmierers. Als Basis muss eine grosse Vertrautheit in der Arbeit mit Matrizen und ein entsprechendes Vorstellungsvermögen vorausgesetzt werden.

– Für neue sprachorientierte Rechnerarchitekturen ist das Konzept der APL-Sprache besonders geeignet. Mit der Einführung spezieller Prozessoren werden heute gültige Einschränkungen in der Anwendung von APL in Zukunft wegfallen [5].

– Die grossen Leistungssteigerungen bei den Kleinrechnern werden auch dort in Zukunft verstärkt den Einsatz von APL möglich machen.

– Die spezielle, mathematisch orientierte Denkweise in APL wird voraussichtlich dazu führen, dass auch in Zukunft nicht jedermann in APL programmieren wird. Die Dialogmöglichkeiten von APL lassen jedoch eine zunehmende Ver-

wendung von APL-Software erwarten. Diese muss aber von professionellen APL-Programmierern erstellt werden und kann dann dem Systembenutzer zur Verfügung gestellt werden.

In diesem Rahmen ist in Zukunft mit einer zunehmenden Verbreitung von APL als universelle Programmiersprache zu rechnen.

Literatur

[1] APL, ein Instrument zur Erhöhung der Produktivität, IBM-Bulletin Nr. 113, April 1979.
[2] Modelle und Prototypen für Anwendungsprogramme, IBM-Bulletin Nr. 120, Juli 1980.
[3] *Iverson, K.E.:* A Programming Language, New York, 1962.
[4] *Giloi, W.K.:* Programmieren in APL, Berlin, 1977.
[5] *Giloi, W.K.:* Rechnerarchitektur, Berlin, 1981.
[6] *Nass, K.W.:* APL/1100 – eine erstaunliche Programmiersprache und ihre universellen Anwendungen, Datascope Heft 29, 1979.
[7] *Govil, C.P.:* Kaufmännische Anwendungen mit APL, Datascope Heft 35, 1980.
[8] *Kaltio, S.:* A system for time series analysis, ITS-Meeting March 26–30, 1979, Nottingham University.
[9] *Niehoff, W.H., Jones, A.L.:* An APL approach to presentation graphics, IBM Systems Journal, No. 3, 1980.
[10] *Alfonseca, M.* et al: An APL interpreter and system for a small computer, IBM Systems Journal, No. 1, 1977.
[11] *Fricke, W.:* Bevölkerungs- und Arbeitsplatzprojektion mit APL, IBM-Nachrichten Nr. 248, Februar 1980.
[12] *Blaser, A., Lehmann, H.:* Abfragesprachen in Datenbanken, IBM-Nachrichten Nr. 251, Oktober 1980.
[13] *Gull, W.E., Jenkins, M.A.:* Recursive Data Structures in APL, CACM Vol 22/1, Januar 1979.
[14] *Iverson, K.E.:* Notation as a tool of thought, CACM Vol 23/8, August 1980.
[15] *Spoelstra, J.:* APL in Computer assisted instruction, a selecting mechanism, APL 80, Amsterdam, 1980.
[16] *Berry, P.:* APL/1130 Primer, White Plains, 1968.
[17] APL System, Programmer Reference, UNIVAC UP-8139.

Adresse des Autors

U. Hartmann, dipl. Math. ETH, Bernische Kraftwerke AG, 3000 Bern 25.

Création du réseau numérique ENEL pour la transmission de données

Par D. Bisci, A. Schiavi et O. Venturini

Es wird auf die grundlegenden Konzeptionskriterien hingewiesen, die bei der Schaffung des Datennetzes der Ente Nazionale per l'Energia Elettrica (ENEL) zur Anwendung gelangten, ferner auf die den Computeranwendern gebotenen Dienste sowie die globale Architektur des Netzes. Spezieller Nachdruck wird auf die Beurteilung der Leistungen und auf die seit der Inbetriebnahme des Netzes gewonnene Erfahrung gelegt. Schliesslich wird kurz auf die Erweiterungspläne des ENEL-Netzes eingegangen.

Cet article indique les critères de conception fondamentaux qui ont été retenus pour la création du réseau de données de l'Ente Nazionale per l'Energia Elettrica (ENEL), les services fournis aux utilisateurs d'ordinateurs ainsi que l'architecture globale du réseau. L'évaluation des performances et l'expérience acquise depuis la mise en exploitation du réseau sont particulièrement soulignés. Enfin, le plan d'expansion future du réseau ENEL est brièvement évoqué.

1. Introduction

Du fait de l'augmentation considérable du nombre des applications de traitement à distance et de l'augmentation du nombre des installations de terminaux de données qui en est résultée ces dernières années, ENEL a reconsidéré l'ensemble de la structure de son système de transmission de données à l'échelle nationale.

Il fallait répondre à trois exigences fondamentales:

– l'accès des terminaux au centre informatique ENEL implanté à Milan, où deux systèmes principaux sont consacrés au calcul technique et scientifique,

– l'interconnexion des ordinateurs régionaux ENEL pour assurer un échange des données entre différents secteurs régionaux,

– la commutation de messages entre le siège social de ENEL implanté à Rome et les huit départements régionaux, y compris la transmission numérique en facsimile à grande vitesse.

Dès la fin de 1979, il apparut clairement que le simple système de multiplexage temporel n'était plus suffisant pour faire face à la demande croissante de liaisons de transmission de données, surtout pour ce qui concerne le premier point mentionné ci-dessus. La manque de souplesse du réseau et le coût élevé des liaisons de transmission de données, chacune étant affectée à une application spécifique, en étaient les principaux inconvénients.

C'est pourquoi la décision fut prise de construire un réseau de données intégré selon le principe d'un partage des ressources de communication entre différents utilisateurs, applications et terminaux.

Le plan de mise au point de ce réseau à l'échelle nationale visait deux objectifs principaux: premièrement, la possibilité de sélectionner le service de calcul désiré à partir du terminal utilisateur ou à partir de l'ordinateur principal demandeur en cas de communications d'ordinateur principal à ordinateur principal; deuxièmement, la réalisation de voies de communications se prêtant à un trafic intensif, de façon à optimiser le rapport coût-bénéfice de la transmission de données.

2. Critères de conception de base

Les contraintes les plus importantes dont il a fallu tenir compte dans la conception du réseau furent:

- la nécessité de desservir un large éventail de terminaux utilisateurs et d'ordinateurs principaux, d'où un large éventail de protocoles de bout en bout,
- la possibilité limitée de prévoir la demande à long terme de voies de transmission de données dans de nombreuses applications nouvelles.

  Il a donc fallu aborder les travaux d'études de façon ouverte pour s'assurer que le réseau pourrait être étendu d'une façon commode, tout en commençant par une absolue transparence à l'égard des utilisateurs, tout au moins au cours des phases initiales.

  Les spécifications du réseau de données ENEL furent donc fondées sur les principes suivants:
- une structure de base capable de prendre en charge le protocole CCITT X.25 dans les stades de son évolution future,
- la possibilité d'effectuer une simple commutation de circuits «transparente» tant en mode synchrone qu'en mode asynchrone,
- la possibilité d'effectuer un contrôle d'erreurs et une concentration statistique pour les utilisateurs en mode asynchrone.