Teil C: Vergleich mit Programmiersprachen, die vor bzw. parallel zu PEARL entwickelt wurden

Von Prof. Dr. **H. Rzehak**, München

1. Erste Konzepte für problemorientierte Programmiersprachen

Die ersten Konzepte für problemorientierte Programmiersprachen sind zunächst nur eine konsequente Weiterentwicklung der Assembler-Programmierung. Am Beispiel von FORTRAN IV ist dies z.B. noch deutlich sichtbar in folgenden Punkten:

- Die Bezeichner von Variablen sind eher eine Benennung von Speicherbereichen wie in der Assembler-Programmierung als Bezeichner von arithmetischen Objekten (gleiche Deklarations-Attribute für unterschiedliche Objekte, EQUIVALENCE-Anweisung, COMMON-Vereinbarung)

- Die Kontrollstrukturen und die Unterprogramm-Anschlußtechnik sind noch sehr stark an der einfachen Umsetzung in Maschinenbefehle orientiert (verschiedene Formen der GOTO-Anweisung, IF-Anweisung, Wiederholungsanweisung, COMMON-Bereiche)

Die stark im Vordergrund stehenden numerischen Anwendungen haben jedoch bewirkt, daß man bereits in den ersten Konzepten für problemorientierte Programmiersprachen arithmetische Ausdrücke in starker Anlehnung an die gewohnte Form niederschreiben konnte. Die E/A - Anweisungen sind stark an den Erfordernissen zum Ausdrucken von Tabellen orientiert. Angaben zur Transformation der Grössen (im wesentlichen Zahlen) zwischen der internen Darstellung und einer gewünschten externen Darstellung können gemacht werden (FORMAT-Anweisungen).

Mit der Entwicklung von ALGOL 60 werden weitere grundlegende Konzepte eingeführt. Dies ist vor allem die Blockstruktur und der damit zusammenhängende Gültigkeitsbereich von Objekten, wobei eine dynamische Speicherverwaltung zur Laufzeit der Programme erforderlich wird. Dies ist ein deutlicher Qualitätssprung. Der Hauptspeicher wird erheblich effektiver genutzt, die Programme werden übersichtlicher. In ALGOL 60 sind auch deutlich verbesserte Kontrollstrukturen eingeführt worden (IF...THEN...ELSE; SWITCH). Der Kontrollfluß kann damit wesentlich problemnäher dargestellt werden. Diese wenigen Beispiele zeigen, daß in ALGOL 60 Sprachelemente eingeführt wurden, die gegenüber der Assembler-Programmierung eine erheblich problemnähere Formulierung von Algorithmen erlauben.

In diesen ersten Programmiersprachen gibt es keine für den Benutzer sichtbare Schnittstelle zwischen dem Programmiersystem und dem Betriebssystem. Man kann also auf Steuermechanismen und Dienste des Betriebssystem nicht zugreifen. Möglichkeiten zur Programmierung von parallelen Abläufen gibt es nicht.

2. Neue Konzepte in PEARL

PEARL enthält gegenüber den ersten problemorientierten Programmiersprachen eine Reihe von neuen Konzepten. Sieht man zunächst einmal von den speziellen Erfordernissen einer Sprache für Realzeit-Aufgaben ab, so sind dies:

- Strengere Typenbindung, insbesondere in algorithmischen Operationen. Jeder Operator verlangt Operanden eines bestimmten Typs und für jeden arithmetischen Ausdruck ist ein Typ wohldefiniert. Automatische Typwandlungen werden von wenigen Ausnahmen bei Zuweisungen abgesehen nicht durchgeführt.

- Einführung von Zeigern auf Objekte (Referenzen) als besondere Datentypen.

- Neue Standard-Datentypen (Zeichenreihen, Strukturen, Zeit) und benutzerdefinierte Datentypen.

- Beschränken von Zugriffsrechten (z.B. Konstanten-Bezeichner) und Daten-Initialisierung.

- Verbesserung der Kontrollstrukturen (z.B. REPEAT, CASE).

- Strukturierung des Programmes in getrennt übersetzbare Einheiten (Moduln).

Noch augenfälliger ist die Innovation durch die Sprachelemente für Realzeit-Aufgaben:

- Parallel ausführbare Programme (TASK's) und Anweisungen zur Koordinierung und zeitlichen Steuerung von TASK's.

- Synchronisationsvariable (SEMA, BOLT).

- Elemente zur Behandlung von asynchronen, externen und internen Ereignissen (INTERRUPT, SIGNAL).

- Beschreibung der Peripherie, die durch ein Programm angesprochen wird (SYSTEM-Division).

- Einheitliches E/A-Konzept für Standardperipherie und Prozessperipherie.

- Transformation des Verhaltens von Peripheriegeräten auf ein Verhalten, das vom Benutzer spezifiziert ist (INTERFACE).

Zwei Punkte sollen dabei noch besonders hervorgehoben werden. PEARL wurde in der Absicht konzipiert, eine möglichst gute Portabilität der Programme zu erreichen. Im allgemeinen sind Programmsysteme für Realzeitaufgaben nicht unabhängig vom unterliegenden Betriebssystem. Die Portabilität der Programme wäre demnach nur bei gleichen Eigenschaften des Betriebssystems gegeben. Da eine Normung von Schnittstellen zum Betriebssystem nicht durchsetzbar ist, hat man zur Herstellung der Portabilität in PEARL eine Reihe von Sprachelementen eingeführt, die Dienst des Betriebssystems auf Sprachebene beschreiben. Dadurch wird indirekt eine Schnittstelle zum Betriebssystem definiert, die durch ein entsprechende Laufzeitsystem an das reale Betriebssystem angepaßt werden muß.

Sollen Programme für Prozessrechner auf andere Rechensysteme übertragen werden, so müssen sie an die neue Umgebung angepaßt werden. Da hier wegen der Vielfalt der Anwendungsbereiche und der verwendeten Gerätetechnik eine Vereinheitlichung unmöglich ist, wurde in PEARL versucht, die gerätetechnischen Abhängigkeiten an bestimmten Stellen des Programmes zu konzentrieren, um sie leicht austauschen zu können. Dies führte zur Beschreibung der Gerätekonfiguration in einem installationsabhängigen Programmteil (SYSTEM-Division), in der auch die im algorithmischen Teil des Programmes verwendeten Bezeichner für Peripherie-Endstellen eingeführt werden. Die Zuordnung der im Programm verwendeten Gerätebezeichner zu verfügbaren Geräten oder Dateien wird gewöhnlich in sehr vereinfachter Form mittels der Kommandosprache beschrieben und durch das Betriebssystem vorgenommen. Oft scheitert das Übertragen eines Programmes auf andere Rechensysteme auch daran, daß das Verhalten der Geräte unterschiedlich ist, so daß die Anpassung durch eine geänderte Konfigurationsbeschreibung alleine nicht möglich ist. Dafür bietet PEARL ein besonderes Sprachelement (INTERFACE) an, mit dessen Hilfe die Transformation auf ein anderes symbolisches Gerät (benutzerdefinierte DATION) durchgeführt werden kann.

3. PEARL im Vergleich zu Real-time FORTRAN und CORAL 66

Die Verwendung von FORTRAN oder CORAL 66 in der Realzeit-Programmierung entsprang dem Wunsch, eine - zumindest regional - weitverbreitete Programmiersprache zu verwenden. Beide Sprachen wurden jedoch zunächst nicht für diesen Bereich konzipiert. Um die Formulierung von Realzeit-Programmen in FORTRAN zu ermöglichen, wurde unter der Bezeichnung "Real-time FORTRAN" (RT-FORTRAN) ein Satz von Unterprogrammen (SUBROUTINE's) vorgeschlagen, mit deren Hilfe die nötigen Dienste des Betriebssystemes angesprochen oder andere in FORTRAN schwer formulierbare Aufträge (z.B. Zeichenreihen-Verarbeitung) ausgeführt werden können. Diese Unterprogramme sind in Assembler geschrieben und werden beim Binden hinzugefügt. Streng genommen handelt es sich hierbei um keine eigene Sprache sondern um ein Paket von anwendungsspezifischen Unterprogrammen. Änderungen am FORTRAN-Kompilierer sind nicht erforderlich.

Die Einführung von RT-FORTRAN wurde vielerorts nur als eine Übergangslösung bis zur Verfügbarkeit einer guten Realzeit-Programmiersprache angesehen. Leider konnte man sich bis heute nicht auf einen einheitlichen Vorschlag einigen, so daß es drei verschiedene Versionen gibt von

- VDI/VDE
- Purdue-Europe
- ANSI

RT-Fortran leidet an allen schlechten Eigenschaften von FORTRAN:

- schlechte Lesbarkeit der Programme; nicht selbstdokumentierend,
- Kontrollstrukturen unterstützen nicht das strukturierte Programmieren
- Fehleranfällige Programmierung

Neben den drei verschiedenen Versionen von RT-FORTRAN sind die Programme nicht portabel, weil die Werte von verschiedenen Parametern betriebssystemabhängig sind. Die Verwendung von RT-FORTRAN kann aus heutiger Sicht nicht mehr empfohlen werden. Sie ist vielleicht unvermeidbar, wenn bereits bestehende Programmkomponenten mit verwendet werden sollen oder keine anderen geeigneten Kompilierer verfügbar sind.

CORAL 66 war ursprünglich für die Systemprogrammierung entworfen worden. Hierfür hat es eine Reihe von bemerkenswerten Eigenschaften:

- Starke Anlehnung an ALGOL 60, insbesondere Übernahme der Blockstruktur

- Einteilung in getrennt übersetzbare Segmente; globaler Datenbereich ähnlich COMMON in FORTRAN

- Zugriff auf Adressen von Variablen beliebigen Typs und auf absolute Adressen; Adressrechnungen sind möglich

- Tabelle von Maschinenworten als "Datentyp"

- Einfügen von Assembler-Kode ist möglich.

Die letzten drei Möglichkeiten sind manchmal ganz nützlich, machen das Programmieren jedoch fehleranfällig und erschweren die Programmpflege. Ferner sind keine E/A-Anweisungen in der Sprache definiert.

CORAL 66 enthält keine speziellen Elemente für Realzeit-Anwendungen. Segmente können allerdings parallel ausgeführt werden. Durch Assembler-Einschübe hat man Zugang zu allen Betriebssystem-Funktionen. Realzeit-Programmierung mit CORAL 66 besteht mehr darin, daß man gewisse algorithmische Teile (z.B. die Arithmetik) problemorientiert formuliert, während man für die speziellen Realzeitfunktionen auf Assembler-Programmierung zurückgreifen muß. Programme in CORAL 66 für Realzeitanwendungen sind vollständig maschinenabhängig.

Bei einem Vergleich mit PEARL ergibt sich, daß gegenüber RT-FORTRAN die Programme einen größeren Dokumentationswert besitzen, leichter wartbar sind und mit weniger Aufwand erstellt werden können. Ferner ist die Programmierung in PEARL weniger fehleranfällig. Gegenüber CORAL 66 ergibt sich, daß PEARL-Programme besser auf andere Systeme übertragbar, d.h. weniger maschinenabhängig sind. Sieht man von der Notwendigkeit der Assembler-Programmierung ab, so besitzt CORAL 66 einige durchaus bemerkenswerte Eigenschaften. Der Vorteil von PEARL wäre noch größer, wenn die PEARL-Kompilierer Referenzen und Typ-Deklarationen zuließen. Leider sind beide Sprachelemente nicht in Basic-PEARL enthalten, sodaß diese wichtigen Elemente einer modernen Programmiersprache nicht in allen PEARL-Kompilierern realisiert sind.

4. Die Kleinrechner-Programmierung: BASIC für Prozeßrechner

BASIC kann trotz seiner Verbreitung nicht als eine geeignete Sprache zur Programmierung größerer Programmsysteme angesehen werden, die über Jahre hinaus gewartet werden müssen. Dies liegt in erster Linie an

- der fehlenden Programmstruktur (z.B. nur globale Objekte)

- dem fehlenden Prozedurkonzept für algorithmische Abstraktion (z.B. keine Parameter und keine lokalen Objekte für Unterprogramme)

- den schlechten Kontrollstrukturen

BASIC kann daher nur einen vernünftigen Einsetzbereich für schnell zu erstellende Programme geringer Lebensdauer und ohne Qualitätsansprüche haben. Zudem gibt es sehr viele BASIC-Dialekte, die keineswegs kompatibel sind. Programme sind daher kaum übertragbar.

Für Realzeitprogramme gibt es ebenfalls eine barocke Vielfalt von Erweiterungen, teilweise mit neuen Sprachelementen, teilweise ähnlich wie bei RT-FORTRAN durch Aufruf von Assembler-Unterprogrammen und teilweise durch Zugriff auf Daten des Betriebssystems. BASIC ist durch die Hinzunahme von Elementen für Realzeit-Programme sicherlich überfrachtet. Hinzu kommt, daß viele BASIC-Systeme interpretierend arbeiten, was für Realzeit-Anwendungen hinderlich ist.

5. CORAL für Minis: RTL/2

RTL/2 ist als Teil 2 eines Normentwurfes "Programming Languages for General Applications in Real-Time Computing Systems" des BSI veröffentlicht worden. Da als Teil 1 dieses Normentwurfes CORAL 66 veröffentlicht wurde, ist gelegentlich gefolgert worden, daß RTL/2 mehr für die Anwendung in kleineren Rechensystemen gedacht ist. Tatsächlich wird jedoch in dem zitierten Entwurf keine Abgrenzung der Anwendungsbereiche vorgenommen. RTL/2 ist weitgehend parallel neben CORAL 66 entwickelt worden. Beide Entwicklungen haben sich wenig beeinflußt.

Gemäß dem BSI-Standard soll der volle Sprachumfang der Systemprogrammierung vorbehalten sein, da er unsichere Sprachelemente enthält, auf die man aber aus Effizienzgründen glaubt, nicht verzichten zu können. Für Anwendungsprogramme sollte nur eine "sichere Teilmenge" der Sprache verwendet werden. Der volle Sprachumfang kann durch folgende Stichworte gekennzeichnet werden:

- Ein Programm besteht aus getrennt übersetzbaren Moduln; Bausteine eines Moduls sind:
  - Umgebungsbeschreibung
  - Typdefinitionen
  - Zuweisung zu Modulkonstanten
  - Deklaration von Programmbausteinen ("brick")
- Die Umgebungsbeschreibung spezifiziert die Programmbausteine anderer Moduln, die angesprochen werden sollen, und zulässige Betriebssystem-Aufrufe.
- Programmbausteine ("bricks") sind:
- Prozeduren
- Stacks
- Datenbereiche

Die Daten-"bricks" dienen zur Aufnahme globaler Daten (benannter COMMON), während mit Stack-"bricks" lokale Datenbereiche für eingriffsinvariante Prozeduren realisiert werden können. Programmbausteine, die in anderen Moduln ansprechbar sein sollen, müssen gekennzeichnet werden.

- Blockstruktur
- Referenzen auf alle Datentypen
- Assembler-Einfügungen
- Ein/Ausgabe ist in der Sprache nicht definiert; in einem Anhang ist jedoch ein knapp formulierter Vorschlag für Ein/Ausgabe-Prozeduren enthalten.

Für Realzeitanwendungen können Prozeduren im Sinne paralleler Rechenprozesse ausgeführt werden. Die nötige Ablauforganisation muß durch den Programmierer erfolgen (z.B. Bereitstellen von lokalem Speicherbereich als Stack-brick). Steueranweisungen für die Rechenprozesse und die Synchronisation sind nicht Bestandteil der Sprache. Hierzu müssen die in der Umgebungsbeschreibung aufgeführten Betriebssystem-Aufrufe verwendet werden.

Zusammengefaßt ergibt sich, daß eine Realzeit-Programmierung mit der für Anwendungen vorgesehenen "sicheren Teilmenge" nicht möglich ist. Es wird ferner ein spezielles Betriebssystem unterstellt; die Programme sind generell nicht portabel.

6. Sprachentwicklungen, in denen Überlegungen des PEARL-Entwurfes bereits sichtbar sind: LTR

Die in Frankreich entwickelte Sprache LTR (Language Temps Réel) enthält Elemente zur Steuerung paralleler Rechenprozesse und erlaubt eine Programmierung von Realzeit-Aufgaben ohne Assembler-Einfügungen. Ein LTR-Programm hat die nachfolgende Struktur:

Umgebungsbeschreibung
Globale Daten
Globale Prozeduren

Lokale Daten Lokale Prozeduren Startmechanismus	} Task, mehrere Tasks in einem Programm möglich
Startmechanismus für das Programmsystem	

Die Umgebungsbeschreibung hat ähnliche Aufgaben wie die SYSTEM-Division in PEARL. Im Startmechanismus wird zwischen zwei Arten von Tasks unterschieden: Eine "Software-Task" stellt einen Rechenprozess bezüglich des unterliegenden Betriebssystem dar. Der Startmechanismus bewirkt einen entsprechenden Zustandsübergang der Task. Eine "Hardware-Task" dagegen hat den Charakter eines Unterbrechungsprogramms, das durch die eintreffende Unterbrechung unmittelbar gestartet wird. Der Startmechanismus beschreibt die Zuordnung zu dem entsprechenden Unterbrechungssignal (INTERRUPT). Einzelne Tasks sind getrennt

übersetzbar. Globale Prozeduren, die nicht in der Übersetzungseinheit enthalten sind, müssen als EXTERNAL spezifiziert werden.

Die Sprache hat eine Blockstruktur und bietet die gebräuchlichen Datentypen an. Besondere Typen für Uhrzeiten und Zeitdauern sind nicht vorgesehen. Referenzen, auch nicht typgebundene, können deklariert werden. Benutzerdefinierte Datenstrukturen (VIRTUAL DATA) können verarbeitet werden. Mit Hilfe von Referenz-Operatoren können Suchvorgänge auf Mengen (Datentyp SET) unmittelbar formuliert werden. Spezielle Realzeitelemente sind für die Zustandsübergänge der Tasks vorgesehen. Synchronisation ist mit Hilfe von Ereignis-Variablen möglich. Zur Verwaltung von Betriebsmitteln steht ein Mechanismus zur Verfügung der ähnlich den mehrstufigen SEMAS arbeitet, aber zusätzlich eine Abfrage der Belegung der SEMA-Variablen erlaubt. Bei einer LTR-Implementierung kann zwischen preemptiver und nicht preemptiver Rechnerkern-Vergabe gewählt werden. Es wird zwischen der Ein/Ausgabe unter Kontrolle des Betriebssystem und direkter Ein/Ausgabe unterschieden.

Bemerkenswert ist ferner, daß LTR eine Makro-Definition kennt, und daß Testhilfen (Definition von Kontrollereignissen, Trace) in der Sprache festgelegt sind. Zulässig sind ferner noch Assembler-Einschübe; es wird allerdings darauf hingewiesen, daß diese möglichst nicht benutzt werden sollen.

7. Was bietet PEARL ?

Die bisherigen Ausführungen kann man in der Feststellung zusammenfassen, daß von den aufgeführten Sprachen höchstens LTR ein ernsthafter Konkurrent zu PEARL sein kann. Ein grundsätzlicher Unterschied sind die in LTR vorgesehenen Unterbrechungsprogramme (Hardware Tasks), die in PEARL bewußt nicht aufgenommen wurden. Da bekannt ist, daß derartige Programme besonderen Einschränkungen genügen müssen, die statisch, d.h. zur Übersetzungszeit, kaum prüfbar sind, hat man in PEARL auf eine mögliche schnellere Reaktion des Rechensystems zu Gunsten einer sicheren Programmierung verzichtet. Diese Entscheidung ist sicherlich durch die billiger und schneller werdende Hardware gerechtfertigt. Sehr zu begrüßen ist bei LTR die Definition von Testhilfen in der Sprache. In PEARL dürfte das Ein/Ausgabe-Konzept allgemeiner und mächtiger sein.

Man kann feststellen, daß PEARL die Konkurrenz von LTR nicht zu fürchten braucht, wenn man mit PERL Full PEARL meint. Leider enthält Basic PEARL eine Reihe von unbequemen Einschränkungen wie:

- keine Initialisierung von Feldern und Strukturen
- keine Konstanten-Bezeichner für die Indexgrenzen zulässig
- Prozeduren als Prozedur-Parameter nicht zulässig
- keine Prozedurvereinbarungen auf Task-Ebene möglich
- Fehlen einer Funktionsprozedur NOW und DATE
- umständliche String-Verarbeitung wegen fehlender String-Selektoren

Leider fehlen in Basic PEARL auch einige mächtige Sprachelemente wie

- Referenzen und
- benutzerdefinierte Datentypen,

die eigentlich in einer modernen Programmiersprache enthalten sein sollen. Es wäre zu überlegen, ob hier nicht der Sprachumfang von Basic PEARL als minimale Untermenge von PEARL erweitert werden soll. Glücklicherweise gehen bezüglich dieser Punkte eine Reihe von PEARL-Kompilierer über Basic PEARL hinaus. Es wäre zu wünschen, daß sich die übrigen Anbieter von PEARL-Kompilierern hier anschließen, um die Verwendung von PEARL noch attraktiver zu machen.

SCHRIFTTUM

DIN 66 253 Teil 2 (Entwurf Nov. 1980)
Programmiersprache PEARL
Full PEARL

ANS/ISA S61-1; S61-2; S61-3
Industrial Computer System FORTRAN
Procedures for Executive Funktions,
Process, Input/Output, and Bit Manipulation

VDI/VDE 3556
Prozess FORTRAN 75, an extension of FORTRAN, for Process Control Computer Applications

IPW/EWICS TC1, 2/80
Industrial Real-time FORTRAN; Definition of Procedures for Application of FORTRAN for the Control of Industrial Processes

BSI Doc 76/63605
Draft Standard Specification for Programming Languages for General Applications in Real-time Computing Systems
Part 1: Specification for CORAL 66

EWICS TC2
Report Real-time Extensions and Implementations of Real-time BASIC; Internal workin document of EWICS TC2, 1975

EWICS TC2
Industrial Real-Time BASIC: Level 1
IRTB-F 78/14

BSI Doc 76/64581
Draft Standard Specification for Programming Languages for General Applications in Real-time Computing Systems
Part 2: Specification for RTL/2

AFNOR NF Z 65-350
Real Time Language LTR