Sprachkonzepte für die Parallelprogrammierung in Ada und PEARL

B. P. Eichenauer, München, vorgetragen auf der PEARL-Tagung 1981

Zusammenfassung

Die Sprachkonzepte für die Parallelprogrammierung in Ada und PEARL werden miteinander verglichen, um das Einsatzgebiet der beiden Programmiersprachen abzustecken. Aufgrund der weitgesteckten Ziele von Ada werden mit Ada mehr die Bedürfnisse des Systemprogrammierers abgedeckt. PEARL bietet dem Automatisierungsingenieur Ausdrucksmittel für die problemangemessene Darstellung der Parallelverarbeitung in Prozeßautomatisierungsprogrammen.

Abstract

The language concepts for parallel programming in Ada and PEARL are compared to find the applications area for both programming languages. As Ada has longrange goals Ada better satisfies the necessities of systems programming. PEARL offers language elements for the elegant formulation of parallel processing in process automation programs.

In jeder technischen Disziplin gibt es Methoden und Verfahren, die zur Lösung von bestimmten Aufgabenstellungen besonders geeignet sind. Wenn man den Aussagen zahlreicher Kollegen Glauben schenken dürfte, so würde eine technische Disziplin hiervon allerdings eine Ausnahme machen, nämlich die Programmierung von Digitalrechnern. Nach PL1 und ALGOL 68 kommt nun Ada. Alle diese Programmiersprachen erheben den Anspruch auf universelle Einsetzbarkeit entweder im kommerziell /technisch wissenschaftlichen Bereich oder im Bereich der Realzeit-Anwendungen. Als Beispiel seien hier die Ausführungen von I.C. Pyle / 1 / über das Einsatzspektrum von Ada zitiert:

"Ada is for programming embedded computer systems... ."

Embedded computer systems range from intelligent terminals and smart instrumentation to air traffic control or factory automation, via laboratory data monitoring, numerically controlled machine tools, navigation and guidance systems, stored program controlled telephone exchanges, batch and continuous production control, environmental monitoring, and future domestic products containing microcomputers. The computer involved may be large or small, single or a collection of many processors, or part of a computer network."

Außer diesem Anspruch auf Universalität findet man in Sprachvergleichen zwischen Ada und PEARL u.a. folgende Ansichten über PEARL (Sandmayr / 2 /):

"It is even doubtful whether features are to be included in a language or whether they belong to the problem set and should be realized by simpler tools provided in the language"

oder

"Furthermore, its design is not very consistent, e.g. interrupts exist besides the very elaborated timing specification possibilities, ..."

Angesichts solcher Aussagen muß man sich die Fragen stellen,

- ob und in welchen Teilbereichen PEARL weiterhin Vorteile gegenüber universellen Realzeitprogrammiersprachen wie z.B. Ada bietet und
- ob denn die Ziele, die mit der Entwicklung von PEARL verfolgt wurden, allgemein richtig verstanden worden sind?

Zur Beantwortung dieser Fragen werden im folgenden die Voraussetzungen und Konzepte geschildert, von denen bei der Entwicklung der Sprachelemente für die Parallelprogrammierung in den beiden Realzeitprogrammiersprachen Ada und PEARL ausgegangen wurde. Es kommt uns dabei nicht auf eine lehrbuchhafte Schilderung aller gebotenen Möglichkeiten für die Parallelprogrammierung an. Wir wollen vielmehr plausibel machen, daß PEARL nach wie vor für den Anwenderkreis, für den die Sprache eigentlich geschaffen wurde, nämlich für Automatisierungsingenieure und Experimentatoren, attraktiv ist.

Die Abstraktionsebenen von PEARL und Ada

PEARL / 3 , 4 / ist eine Programmiersprache, die speziell für die Automatisierung von technischen Prozessen und von wissenschaftlichen Experimenten entworfen wurde. Die Abstraktionsebene bzw. das Vokabular von PEARL wurde so gewählt, daß die ingenieurmäßige Darstellung der Ergebnisse des Systementwurfs und der Systemdefinition von Prozeßautomatisierungssystemen möglichst unmittelbar in ein Automatisierungsprogramm umgesetzt werden kann. PEARL soll es auch dem nicht ständig programmierenden Automatisierungsingenieur ermöglichen, Automatisierungsprogramme zu entwerfen oder doch wenigstens ohne großen Aufwand nachzuvollziehen.

Ada / 5 / wurde wie PEARL für die Programmierung von Prozeßrechensystemen (sog. embedded computer systems) entworfen, soll aber das gesamte Spektrum aller denkbaren Anwendungen, d.h. von der konventionellen Prozeßrechner-Programmierung bis hin zur Entwicklung von Spezialprogrammen mit minimaler Betriebssystem-Unterstützung abdecken. Um dieses Ziel zu erreichen, wurden in Ada zwar gewaltige Sprachmittel für die Vereinbarung von Daten, für den Datenschutz und für die Formulierung von Rechenalgorithmen bereitgestellt; bei den Sprachelementen für die Parallelprogrammierung und die Ein/Ausgabe hat man sich jedoch auf das Allernotwendigste zu beschränken versucht (im Fall der Ein/Ausgabe gibt es nur einen Vorschlag für ein Unterprogrammpaket). Ada geht davon aus, daß die Programme für die zeitliche und logische Synchronisation des Programm- und Prozeßgeschehens und für die Prozeß-Ein/Ausgabe von Programmierern im Rahmen des Automatisierungsprogramms geschaffen werden. Der Automatisierungsingenieur kann Ada nur benutzen, wenn er fundierte Kenntnisse in der Systemprogrammierung besitzt.

Vereinbarung von Tasks

Prozeßautomatisierungsprogramme unterscheiden sich von kommerziellen Programmen im wesentlichen dadurch, daß sie synchron zueinander und mit Vorgängen außerhalb des Prozeßrechensystems ablaufen müssen. Während in kommerziellen Programmen normalerweise nur der durchzuführende Rechenalgorithmus beschrieben werden muß und es unerheblich ist, wann das Programm abläuft, sind in einem Prozeßautomatisierungsprogramm im Rahmen der sog. Automatisierungsfunktion / 6 / zu jedem Rechenalgorithmus auch die Bedingungen für seinen Ablauf und die Anforderungen an die Ein/Ausgabe anzugeben.

In der Regel werden mit einem Prozeßrechensystem mehrere parallel zueinander ablaufende Vorgänge in einem technischen Prozeß (sog. Teilprozesse) bearbeitet. Deshalb sind innerhalb eines Automatisierungsprogramms auch Vorkehrungen für die parallele bzw. quasiparallele Durchführung der zugeordneten Automatisierungsfunktionen zu treffen. Den verschiedenen parallel zueinander ablaufenden sequentiellen Teilprozessen in einem technischen Prozeß werden dabei im Automatisierungsprogramm sog. Programme oder Tasks zugeordnet, welche die Automatisierungsfunktionen für die Teilprozesse realisieren.

Betrachtet man die gängigen Automatisierungsvorhaben, so zeigt sich, daß sich die durchzuführenden Aufgaben mit einer festen Anzahl von Tasks realisieren lassen, die den Aufgaben statisch zugeordnet werden. Ein derartiger Entwurf eines Automatisierungsprogramms ist wünschenswert, weil dadurch u.a. die Betriebsmittelplanung und insbesondere auch der Sicherheitsnachweis erheblich erleichtert wird. Auch die Betriebssysteme der gängigen Prozeßrechner setzen voraus, daß eine feste Anzahl von Programmen bzw. Tasks in unmittelbar startbarer Form vorliegen.

In PEARL wurde deshalb nur die statische Definition von Tasks vorgesehen, denen bei ihrer Vereinbarung eine Priorität zugeordnet werden kann. PEARL-Tasks liegen beim Start eines PEARL-Programmsystems in startbarer Form vor und können bei den meisten Prozeßrechner-Betriebssystemen unter weitgehender Verwendung der vorhandenen Programmorganisation implementiert werden.

Die ursprünglich in PEARL vorgesehenen dynamisch entstehenden und vergehenden Tasks (sog. Subtasks) wurden 1980 / 7 / wieder aus dem PEARL-Sprachentwurf entfernt, weil ihre Implementierung bei den heutigen Prozeßrechnern erhebliche Probleme bereitet und darüber hinaus einen enormen Laufzeit- und VerwaltungsOverhead erfordert (u.a. Warten am Blockende auf die Beendigung aller Subtasks; Zugriff auf gemeinsame Datenbereiche; mehrfache Inkarnation einer Subtask, usw.).

Während bei der Konzipierung des PEARL-Taskings anwendungsorientierte Gesichtspunkte den Umfang der Sprachmittel bestimmen, soll das Ada-Tasking universell einsetzbar sein. Es soll z.B. gleichermaßen für die Entwicklung von Prozeßautomatisierungsprogrammen wie für die Entwicklung von Betriebssystemen geeignet sein.

Um dieses Einsatzspektrum zu befriedigen, wird in Ada von einem sehr allgemeinen dynamischen Taskmodell ausgegangen, das bezüglich der Vereinbarung von Tasks große Ähnlichkeiten mit dem ehemaligen PEARL-Subtaskkonzept auf-

weist. Ada- Tasks werden im Vereinbarungsteil einer Ada-Programmeinheit (Paket, Unterprogramm, Block oder Task) vereinbart; die Vereinbarung von Tasktypen und von sogenannten Taskfamilien (Felder von Tasks) ist möglich.

Hinsichtlich der Implementierung des Ada-Taskings gelten die oben bei der Erwähnung des PEARL-Subtaskkonzepts angeführten Bemerkungen. Insbesondere dürften unter Verwendung der derzeit im Einsatz befindlichen Prozeßrechner-Betriebssysteme dynamische Tasks und die damit verbundene komplexe Speicherorganisation kaum effizient realisierbar sein.

Die Autoren von Ada waren sich dieser Tatsache wohl bewußt. Sie gehen davon aus, daß für Ada ein relativ kleiner maschinenorientiert zu schreibender Kern erstellt wird, auf dem das ggf. anwendungsspezifisch in Ada zu schreibende Betriebssystem aufsitzen soll. Es bleibt abzuwarten, ob sich für heutige Prozeßrechnerstrukturen die Parallelläufigkeiten in Betriebssystemen hinreichend effizient mittels des Ada-Taskingkonzepts darstellen lassen.

Sowohl in PEARL als auch in Ada werden die relativen Prioritäten von Tasks untereinander als wichtige Voraussetzung für den ordnungsgemäßen Ablauf eines Automatisierungsprogramms angesehen. Da sich die Wichtigkeit einer Task während des Ablaufs eines PEARL-Programmsystems in Abhängigkeit von äußeren Ereignissen ändern kann, ermöglicht PEARL im Gegensatz zu Ada auch Prioritätsänderungen während des Ablaufs einer Task. Für PEARL ist die Prioritätsangabe i.A. eine Anweisung an die Laufzeitorganisation zur prioritätsgerechten Bereitstellung von Betriebsmitteln, während Ada die Prioritätsangabe als Organisationsanweisung an den Compiler ansieht.

Explizite Steuerung von Tasks

Für die Ablaufsteuerung von Tasks gemäß den Angaben in einer Automatisierungsfunktion sind in PEARL Steueranweisungen vorgesehen, mit denen Tasks gestartet (ACTIVATE), zurückgestellt (SUSPEND), fortgesetzt (CONTINUE) und beendet (TERMINATE) werden können. Beim Aktivieren und Fortsetzen einer Task kann die Priorität einer Task geändert werden.

In Ada sind keine Sprachelemente für die explizite Steuerung von Task vorgesehen. Alle Tasks, die in einem Vereinbarungsteil einer Ada-Programmeinheit vereinbart sind, werden kreiert und gestartet, sobald das Programm in die Programmeinheit eintritt. Unterbrechungen des Ablaufs einer Task werden nur implizite über Synchronisationsanweisungen herbeigeführt.

Synchronisation

Bei der Synchronisation von Tasks untereinander und mit zugeordneten Teilprozessen in einem technischen Prozeß unterscheiden wir zwischen logischer und zeitlicher Synchronisierung.

Logische Synchronisierung

Die logische Synchronisierung von Tasks untereinander erfolgt in PEARL über sog. Synchronisationsvariable. Vorgesehen sind sog. BOLT-Variable zur Organisation des synchronisierten Zugriffs auf Betriebsmittel (z.B. Speicherplatz) und SEMAphores zur Organisation des synchronisierten Ablaufs von Tasks.

In Ada wird die Synchronisierung von Tasks über das sog. Rendevouz-Konzept herbeigeführt.

Ada geht davon aus, daß die Synchronisierung von Tasks normalerweise erforderlich ist, weil eine Task eine Service-Leistung von einer anderen Task benötigt. Die Task, welche die Service-Leistung anfordert, begibt sich, falls die Leistung noch nicht erbracht worden ist, in den Wartezustand. Falls die Leistung schon vor ihrer Anforderung erbracht wurde, wartet die Task, welche die Service-Leistung erbringt, auf die Task, welche die Service-Leistung benötigt. Zu dem Zeitpunkt, zu dem die Leistung sowohl gefordert wird als auch erbracht ist, kann zwischen den beiden Tasks Information ausgetauscht werden; danach laufen beide Tasks wieder unabhängig voneinander weiter.

In dem folgenden PEARL-Beispiel werden zwei Tasks P und Q synchronisiert, wobei Q am Synchronisationspunkt Information an P übergibt:

```
DCL  S  SEMA ;
DCL  MESSAGE  FIXED ;
       .
       .
       .

P :  TASK ;
       .
       .
       .

     REQUEST S;
       .
       .

     (Verwendung von MESSAGE)
       .
       .

     END;
```

```
Q : TASK
      .
      .
      .

    MESSAGE : = .....
    RELEASE S;
      .
      .
      .
    END;
```

Dasselbe Beispiel läßt sich in Ada z.B. gemäß:

```
task P is
    entry A;
end P;
task body P is
        .
        .
        .
    accept A (MESSAGE : in INFO) do
        .
        .
        .
    end A;
        .
        .
        .
end P;
task Q;
task body Q is
  MESS : INFO;
        .
        .
        .
    P.A (MESS);
        .
        .
        .
end Q;
```

darstellen. Wie daraus ersichtlich ist, läßt sich in Ada die Synchronisation von Tasks unter Vermeidung von Programmgrößen aus der Umgebung der Tasks formulieren. Insbesondere wird der bei vielen Aufgabenstellungen notwendige Informationsaustausch zwischen Tasks (z.B. Auftraggeberidentifikation) in adäquater Weise ermöglicht.

Für die Erstellung zuverlässiger Programme ist auch selektive Anforderung oder Bereitschaft zu einem Rendezvous wichtig. Hierzu zwei Beispiele:

Soll in dem vorangegangenen Beispiel dafür gesorgt werden, daß spätestens nach einer Sekunde eine Maßnahme getroffen wird, falls P nicht zu einem Rendezvous bereit ist, so kann dies in Ada gemäß:

```
select P.A (MESS)
or delay 1.0;
      .
      .
      .
    (Maßnahme)
end select;
```

erfolgen.
Auch die das Rendezvous anfordernde Task kann Ersatzmaßnahmen angeben, falls das Rendezvous nicht innerhalb einer bestimmten Zeitspanne zustandekommt:

```
select accept A (MESSAGE : in INFO) do
      .
      .
      .
    end A;
or delay 1.0;
else begin
      .
      .
      .
    end;
end select;
```

Das in Ada vorgesehene Rendezvous-Konzept ermöglicht i.A. eine sehr elegante und durchsichtige Formulierung der Synchronisierung des Ablaufs von Tasks. Da es der einzige Mechanismus zur Synchronisierung in Ada ist, muß es aber auch zur Organisation des synchronisierten Zugriffs auf gemeinsame Daten mehrerer Tasks eingesetzt werden und kann hier zu umständlichen und aufwendigen Konstruktionen führen.

Als einfaches Beispiel sei hier die gemeinsame aber sich gegenseitig ausschließende Benutzung eines Datenfiles FILE betrachtet. In PEARL kann dieses Problem relativ einfach und durchsichtig durch Zuordnung einer Semaphore-Variablen erledigt werden:

```
DCL S SEMA PRESET 1
      .
      .
      .
P : TASK;
    REQUEST S;
      .
      .
      .
  (Verwendung von FILE)
      .
      .
    RELEASE S;
    END;
```

```
Q : TASK;
    REQUEST S;
      .
      .
      .
  (Verwendung von FILE)
      .
      .
      .
    RELEASE S;
    END;
      .
      .
      .
```

Da in Ada nur die Ablaufsynchronisierung über das Rendezvous-Konzept möglich ist, muß das Datenfile über eine eigens zu diesem Zweck einzuführende Task verwaltet werden:

```
task ORG is
    entry GET;
    entry DONE;
end ORG;
task body ORG is
    FILE : DATA;
      begin loop
        accept GET (F : out DATA) do
          F : = FILE;
        end GET;
        accept DONE (F : in DATA) do
          FILE : = F;
        end DONE;
        end loop;
        end;
end ORG;
task P;
task body P is
    FILE : DATA;
    begin
          ORG.GET (FILE);
            .
            .
            .
            .
          ORG.DONE (FILE);
    end P;
```

Wie auch bei dem früheren Beispiel ist auch hier ersichtlich, daß die vielleicht größere Zuverlässigkeit bei der Programmierung durch erhebliche Speicher- und Laufzeit-Overhead erkauft wird.

Die logische Synchronisation von Tasks mit parallelen Abläufen im technischen Prozeß erfolgt normalerweise über Interrupts. Im Systemteil eines PEARL-Moduls kann das Prozeßrechensystem und insbesondere auch das Interruptwerk des eingesetzten Prozeßrechners beschrieben werden. Dabei können an die verschiedenen im Benutzerhandbuch ausgewiesenen Interruptklemmen Namen vergeben werden.

Beispiel: INTR : KLEMME (7)⟶;

Der Interrupteingang, der im Benutzerhandbuch mit KLEMME (7) bezeichnet wird, erhält den Namen INTR.

Im Problemteil eines PEARL-Moduls wird nun zunächst die Verwendung der Unterbrechungen, die über INTR erzeugt werden, festgelegt. Dabei wird zwischen Unterbrechungen, die das PEARL-Programmsystem insgesamt (sog. INTERRUPTs) und Unterbrechungen, welche von einer der gerade ablaufenden Tasks erzeugt wurden und nur diese betreffen (sog. SIGNALS) unterschieden.

Nachdem nun INTR zu Beginn des PEARL-Problemteils gemäß:

```
SPECIFY INTR INTERRUPT;
```

als Unterbrechung, auf welche das gesamte Programmsystem zu reagieren hat, ausgewiesen wurde, kann die Synchronisierung zwischen Task und Teilprozess formuliert werden. Dazu kann einer der früher erwähnten Anweisungen zur expliziten Steuerung von Tasks eine WHEN-Bedingung (WHEN-Schedule) vorangestellt werden, die angibt, wann die Steueranweisung ausgeführt werden soll:

```
WHEN INTR ACTIVATE T;
```

Außerdem kann mit einer RESUME-Anweisung der Ablauf einer Task unterbrochen werden, bis bestimmte Interrupts eintreffen:

```
WHEN INTR RESUME;
```

Die logische Synchronisation zwischen Tasks und Abläufen außerhalb des Rechensystems wird in Ada auf das früher skizzierte Rendezvous-Konzept zurückgeführt, in dem einem Eingang in die Task eine Interruptklemme zugeordnet wird:

```
      .
      .
      .
INTR : integer : = 8#706#;
```

```
.
.
.
task INTR_HANDLER is
  entry HANDL;
  for HANDL use at INTR;
end INTR_HANDLER;
```

Im Unterschied zu PEARL, wo die Zuordnung zwischen Interrupt und Task beim Starten einer Task erfolgt, wird in Ada die Verbindung zwischen Interrupt und Task bei der Taskvereinbarung vorgenommen. Ein Austausch des Interruptantwortprogramms zur Laufzeit ist deshalb nicht möglich.

Zeitliche Synchronisierung

Ebenso wichtig wie die logische Synchronisation ist für das Prozeßrechnen die Ausführung von Tasks unter vorgegebenen Zeitbedingungen. Um die Häufigkeit, mit der Tasks ausgeführt werden sollen bzw. die Ablaufgeschwindigkeit von Tasks an die Geschwindigkeit der Abläufe außerhalb des Rechensystems anpassen zu können, müssen die bei der Systemanalyse ermittelten Daten über das zeitliche Prozeßverhalten - und ggf. auch das zeitliche Verhalten der Automatisierungsmittel selbst -bei der Erstellung eines Automatisierungsprogramms berücksichtigt werden.

Da, wie eingangs erwähnt, die ingenieurmäßige Beschreibung einer Automatisierungsaufgabe möglichst unverändert in ein PEARL-Programm umgesetzt werden soll, wurden in PEARL neue Datentypen und zugeordnete Rechenoperatoren vorgesehen, die eine ingenieurmäßige Behandlung von relativen und absoluten Zeitangaben (DURATION und CLOCK) ermöglichen.

Unter Verwendung von Zeitangaben können die Bedingungen für den zeitbedingten Start einer Task oder zur zeitweisen Unterbrechung einer Task problemorientiert formuliert werden:

```
ALL 5 SEC ACTIVATE MESSWERT_ERFASSUNG;
AT 12:0:0 ACTIVATE ZAEHLER_LESEN;
AFTER 10 MIN RESUME;
```

Die Möglichkeiten zur Behandlung von Interrupts und zur Angabe von Zeitbedingungen stellt keineswegs, wie u.a. Sandmayr / 2 / meint, eine Inkonsistenz in PEARL dar. Der PEARL-Anwender muß die Möglichkeit haben, auf Interrupts aus dem technischen Prozeß spezifisch zu reagieren; er soll aber möglichst weitgehend von der Entwicklung von Systemprogrammen, die nichts mit seinem Automatisierungsproblem zu tun haben (z.B. Ableitung der Uhrzeit aus Zeitunterbrechungen, E/A-Treiber, usw.) befreit werden.

In Ada ist für die zeitliche Synchronisation von Task mit Vorgängen außerhalb des Rechensystems nur eine Anweisung zur zeitweisen Unterbrechung des Taskablaufs vorgesehen:

```
DELAY zeitangabe;
```

die der PEARL-Anweisung:

```
AFTER durationangabe RESUME;
```

entspricht. Alle übrigen zeitsteuernden Bedingungen, wie z.B. das ALL- oder AT-Schedule, müssen in Ada im Rahmen des Anwenderprogramms ausgehend von Unterbrechungssignalen eines Zeitgebers realisiert werden.

Fazit

Wie die Gegenüberstellung der Konzepte für die Parallel-Programmierung in Ada und PEARL zeigt, bieten beide Sprachen ein hinreichendes Vokabular zur Erstellung von Prozeßautomatisierungsprogrammen.

PEARL ist mehr auf die Bedürfnisse des automatisierenden Ingenieurs und auf die Leistungsfähigkeit heutiger Prozeßrechensysteme zugeschnitten und nimmt dafür ein reduziertes Einsatzspektrum in Kauf.

Die Parallelprogrammierung in Ada orientiert sich an den Bedürfnissen des System-Programmierers um möglichst alle denkbaren Einsatzfälle abzudecken, macht aber die Entwicklung von speziellen Betriebssystemen und von Anwenderpaketen erforderlich, in denen die regelmäßig für die Automatisierung technischer Prozesse nötigen Dienste angeboten werden. Da solche Dienste nicht durch die Sprache vorgeschrieben werden, ist bei Ada die gleiche Situation wie bei Realzeit-FORTRAN zu erwarten. Automatisierungsprogramme werden unter Voraussetzung unterschiedlicher Laufzeitsysteme erstellt und sind damit nur bedingt austauschbar.

Schrifttum

/ 1 / Pyle, J.C:
The Ada Programming Language;
A. Guide for Programmers;
Prentice Hall International; 1981

/ 2 / H. Sandmayr:
Sprachen für die Echtzeitprogrammierung;
Fachtagung Prozessrechner 1981 München
abgedruckt in: PEARL Rundschau, II, Nr. 2,
(Juni 1981)

/ 3 / DIN 66253:
Teil 1 - Basic PEARL (Vornorm)

/ 4 / Teil 2 - Full PEARL (Normentwurf)
Beuth-Verlag Berlin/Köln; 1980

/ 5 / United States Department of Defense:
Reference Manual for the Ada
Programming Language;
proposed standard document; July 1980

/ 6 / R. Lauber:
Prozeßautomatisierung I
Aufbau und Programmierung von
Prozeßrechensystemen;
Springer-Verlag Berlin/Heidelberg/New York; 1976.

/ 7 / PEARL Arbeitskreis:
Bericht zur 38. PAK-Sitzung
am 14.7.1980 (PAK 4-80)