PEARL-PC: Ein Echtzeit-Programmiersystem für IBM PC's

Erwin Kneuer, Werum GmbH
Glogauer Straße 2 A, 2120 Lüneburg

Zusammenfassung

Im gleichen Maße wie der Einsatz von Personal Computern für Echtzeitanwendungen zunimmt, steigt auch der Bedarf an geeigneten Entwicklungsumgebungen. Da sich auch in Zukunft die Preisschere Hardwarekosten zu Softwareentwicklungs- und Wartungskosten weiter öffnen wird, sich der Lebenszyklus vieler Anwendungsprogramme über mehrere Hardwareentwicklungen erstreckt und sehr häufig die gleiche Anwendungssoftware unter verschiedenen Systemumgebungen ablaufen soll, bekommt der Aspekt der Software-Portabilität immer größere Bedeutung.
PEARL-PC ist hierfür ein geeignetes Werkzeug. Es unterstützt die Entwicklung von portabler Echtzeitsoftware durch Bereitstellung der Echtzeitsprache PEARL mit einem Debugger auf Sprachebene.
Nach einer Charakterisierung der Echtzeitsprache PEARL wird in diesem Beitrag auf die Handhabung und Implementierung von PEARL-PC näher eingegangen.

1. Die Echtzeitsprache PEARL

Die Sprachdefinition PEARL (Process and Experiment Automation Realtime Language) wurde 1982 vom DIN genormt /2/. Auf Prozeßrechnern hat sie sich bereits in mehr als 200 Projekten bewährt /3/.

Die hier nur kurz charakterisierten Eigenschaften von PEARL sind detailliert in /4/ beschrieben. Für diesen Sprachumfang hat die Firma Werum GmbH, Lüneburg, ein portables PEARL-Programmiersystem entwickelt, das außer auf IBM PC XT und AT auch auf mehr als 10 unterschiedlichen Rechnerarchitekturen, die von Mini/Micro- bis zu Großcomputern reichen, verfügbar ist.

Modularität

Ein PEARL-Programm wird aus getrennt übersetzbaren Moduln mit genau definierten Schnittstellen aufgebaut, was die Programmentwicklung wesentlich erleichtert und für große Anwendungssysteme letztlich unentbehrlich ist.

Portabilität

Ein PEARL-Modul besteht aus einem Systemteil und/oder einem Problemteil.
Im Systemteil ordnet der Programmierer den rechnerabhängigen Objekten (Interrupts und E/A-Geräten) beliebige Namen zu, unter denen sie im Problemteil rechnerunabhängig angesprochen werden. Da PEARL zudem rechnerunabhängige, vom DIN genormte Datenstrukturen und Anweisungen auch für Multitasking und Synchronisierung bietet, kann der Problemteil eines PEARL-Moduls rechnerunabhängig erstellt werden. Diese Eigenschaft ist nicht nur für die Portierung von PEARL-Programmen wichtig: Sie ermöglicht insbesondere die immer bedeutsamer werdende Cross-Entwicklung von Echtzeit-Programmen für kleine Zielsysteme auf komfortablen Host-Rechnern, wobei hier unter Entwicklung nicht nur Übersetzung sondern vor allem Ausführung und Off-line-Test gemeint sind.

Strukturiertes Programmieren

PEARL enthält die wichtigsten Konzepte moderner höherer Programmiersprachen, wie PASCAL oder Ada. So gliedert sich der Problemteil eines PEARL-Moduls in Tasks (parallele Aktivitäten), Prozeduren und Blöcke, wobei Prozeduren und Blöcke geschachtelt werden dürfen. Die Kontrollstrukturen haben folgende Form:

- IF ... THEN ... ELSE ... FIN
- FOR ... FROM ... BY ... TO ... REPEAT ... END

WHILE ... REPEAT ... END

CASE ... ALT ... OUT ... FIN

Sie können geschachtelt werden.

Problemorientierte Datenstrukturen auch für Echtzeit-Aufgaben

Standardmäßig stehen in PEARL folgende Datentypen zur Verfügung:

FIXED und **FLOAT** mit angebbarer Genauigkeit
BIT und **CHARACTER** - Strings mit angebbarer Länge
CLOCK und **DURATION** für Uhrzeiten und Zeitdauern
Referenzen (**REF**) für indirekte Adressierung (Pointer)
Geräte und Files (**DATION**) für Standard- und Prozeß-Ein/Ausgabe
Interrupts (**INTERRUPT**) für externe Unterbrechungen
Semaphoren (**SEMA**) und Boltvariablen (**BOLT**) zur Koordination des Zugriffs von Tasks auf gemeinsam benutzte Objekte.

Der Benutzer kann daraus selbst komplexere Datenstrukturen wie Felder, hierarchische Strukturen (**STRUCT**) und Listen aufbauen; mittels der **TYPE**-Definition können diese auch als neue, problemorientierte Datentypen vereinbart werden. Außerdem erlaubt PEARL die Einführung neuer, problemorientierter Operatoren für beliebige Datenstrukturen mittels der **OPERATOR**-Definition.

Multitasking, Scheduling

Zur Beschreibung zeitlich paralleler Abläufe besitzt PEARL transparente, leicht erlernbare, rechnerunabhängige Echtzeit-Sprachelemente:

ACTIVATE Steuerung
AT Zyklus_Startzeit **ALL** Frequenz **DURING** Dauer **ACTIVATE** Erfassung
WHEN Bunker leer **ACTIVATE** Nachschub
TERMINATE
SUSPEND
CONTINUE Protokoll
AFTER 5 SEC **RESUME**
PREVENT Reaktion
REQUEST Foerderstrecke (i)
RELEASE Kommunikationspuffer

Rechnerunabhängige E/A-Anweisungen

PEARL verfügt nicht nur über rechnerunabhängige Anweisungen für

- formatierte E/A : **PUT** 'ARTNR:', Artikel.Nr **TO** Drucker **BY** LIST

und

- binäre E/A : **READ** Stammdatensatz **FROM** Stammdatei **BY** POS (i, j)

sondern auch über rechnerunabhängige Anweisungen für

- Prozeß-E/A : **TAKE** Druck **FROM** Druckgeber
  SEND Auf **TO** Ventil

Beispiel Geräteüberwachung

Die Task Erfassung soll zyklisch alle 5 Sekunden je einen Meßwert (16 Bit) von 10 Geräten erfassen, ihn mit der aktuellen Uhrzeit und der Gerätenummer zu einer "Messung" vervollständigen, diese Messung jeweils an die Prozedur Regelung übergeben und anschließend alle 10 Messungen auf ein Logbuch schreiben.
Die Prozedur Regelung soll u.a. feststellen, ob der aktuelle Meßwert an den Positionen j bis j+3 den Bitstring '1011' enthält.

```
MODULE ( Geraete_Ueberwachung ) ;

SYSTEM ;
Geraet (1:10): DIGIN (1:10) * (0:15) ;
Logbuch: DISK ;

PROBLEM ;
SPECIFY Geraet ( ) DATION IN BASIC ,
        Logbuch   DATION OUT MESSUNG ;

TYPE MESSUNG STRUCT
    [Wert BIT (16) ,
     Zeit CLOCK ,
     Geraet FIXED (15) ] ;

Start:  TASK ;
        OPEN Logbuch ;
        ALL 5 SEC ACTIVATE Erfassung ;
END ;
```

```
Erfassung:   TASK ;
    DECLARE Mess (10) MESSUNG ;
    FOR i FROM 1 BY 1 TO 10 REPEAT
        TAKE Mess(i).Wert FROM Geraet (i) ;
        Mess(i).Zeit := NOW ;
        Mess(i).Geraet := i ;
        CALL Regelung ( Mess(i) ) ;
    END ;
    WRITE Mess TO Logbuch ;

    Regelung: PROC ( Mess MESSUNG IDENT ) ;
        ...
        IF Mess.Wert.BIT (j: j+3) == '1011'B
            THEN... ELSE...
        FIN ;
    END ;
END ;

MODEND ;
```

2. Implementierung

2.1 Hardware- und Software-Voraussetzungen zur Programmentwicklung

Zur Compilierung von PEARL-Moduln auf IBM PC werden mindestens die folgenden Betriebsmittel benötigt:

- 150 KB Laufbereich
- 10 MB Festplatte
- Coprozessor 8087 auf IBM PC XT und 80287 auf IBM PC AT
- Betriebssystem MS-DOS mit Universal Development Interface RTCS/UDI
  oder das
  Echtzeit-Betriebssystem PC/RTX auf XT und AT/RTX auf AT
- 1 Terminal.

Voraussetzungen für die Ausführung von PEARL-Programmen auf IBM PC sind:

- Coprozessor 8087 auf XT und 80287 auf AT
- Echtzeit-Betriebssystem PC/RTX auf XT und AT/RTX auf AT.

Die RTCS/UDI-Schnittstelle und die Echtzeit-Betriebssysteme PC/RTX und AT/RTX sind Produkte der Firma RTCS, USA, und realisieren die UDI- und iRMX86-Schnittstellen von Intel auf den IBM PC - Rechenanlagen.

Auf Basis von PC/RTX und AT/RTX hat Werum die virtuelle, durch DIN 66253 definierte, einheitliche PEARL-Betriebssystem-Schnittstelle auf dem IBM PC realisiert, so daß PEARL-Programme normgemäß auf dem IBM PC ausgeführt werden können. Die Größe dieser PEARL-Betriebssystem-Schnittstelle (des PEARL-Ablaufsystems) beträgt maximal 50 KB.

Die einzelnen Schritte der Programm-Entwicklung lassen sich auf IBM PC XT oder AT wie folgt unter den verschiedenen Betriebssystemen vornehmen:

	MS-DOS	MS-DOS mit UDI	PC/RTX / AT/RTX
Editieren	o	o	o
Übersetzen	-	o	o
Binden	-	o	o
Ausführen	-	-	o
Testen	-	-	o

Unter MS-DOS erstellte Dateien können nach PC/RTX und AT/RTX übernommen und von PEARL-Programmen benutzt werden und umgekehrt.

In Assembler ASM86, PL/M86 oder PASCAL86 erstellte Prozeduren können von PEARL-Programmen aus aufgerufen werden.

2.2 Eigenschaften von PEARL-PC

Compiler

Der Compiler übersetzt einen PEARL-Modul in einen ASM86-Assembler-Modul. Dabei können die zu übersetzenden Moduln praktisch beliebig groß sein. Wenn die Modulgröße es erfordert, wird der gesamte freie Hauptspeicher genutzt und, falls dieser nicht ausreicht, automatisch auf die Festplatte ausgewichen.
Die Übersetzungsleistung beträgt bei mittlerer Modulgröße (etwa 500 Quellzeilen) ca. 10 Quellzeilen je Sekunde auf IBM PC AT.

Mittels Compiler-Parameter können u.a. Listings der Quelle und des erzeugten ASM86-Moduls verlangt werden, wobei im Listing des ASM86-Moduls Hinweise auf die Ursprung-

zeilen im Quellprogramm eingefügt sind, wodurch auch eine eventuelle Emulation erleichtert wird.
Im Quell-Listing wird die Blockschachtelungstiefe protokolliert.
Außerdem erstellt der Compiler auf Wunsch eine Cross-Referenzliste aller benutzten Objekte mit Angaben der Quellzeilen-Nummern für ihre Definition und Benutzung.
Zudem kann die Laufzeit-Überwachung von Feldgrenzen und Referenzen durch Compiler-Parameter ein- oder ausgeschaltet werden.

Der Compiler analysiert Programme umfangreich und recht genau. Die Fehlermeldungen erfolgen im Klartext mit Angabe der Quellzeilen-Nummer.

Ein Preprozessor gestattet das Einfügen von Programmtexten aus Files (%INCLUDE) und bedingte Compilierung (%IF). Damit können zum Beispiel in mehreren Moduln verwendete Schnittstellen aus einer getrennten Textdatei zur Übersetzungszeit textuell einkopiert werden, so daß die Einhaltung der Modulschnittstellen implizit gewährleistet wird.
Die Portierung von PEARL-Programmen von anderen Rechnern zum IBM PC oder umgekehrt wird zusätzlich dadurch erleichtert, daß der auf IBM PC eingesetzte portable PEARL-Compiler auf vielen anderen Rechnern zur Verfügung steht: HP 1000 F, HP 3000, IBM 4341, Intel 8086, LSI 11, ND 100, PEARL Engine 68.000 (Motorola 68.000), Siemens R/M-Serie, Siemens 7.000-Serie, VAX 11/750, Zilog Z 8000.

Laufzeitsystem

Um ein ablauffähiges PEARL-Programm zu erhalten, müssen die nach ASM86 übersetzten PEARL-Moduln noch assembliert und zusammen mit einer PEARL-spezifischen Library sowie einigen System-Libraries mit dem LINK86 gebunden werden. Das Programm wird dann durch Eingabe seines Namens mit Hilfe des Human Interface gestartet.

Das PEARL-Laufzeitsystem benutzt den RTX-Nukleus für das Tasking und das UDI-Interface für alle E/A-Operationen auf Nicht-Prozeßgeräten.
Mit geringem Aufwand sind Adaptionen möglich, wie z.B.

E/A nur mit BASIC-I/O
oder
Konfigurierung von PEARL-Anwendungen als First-Level oder I/O-Jobs.

Die Stackgröße einer Task kann durch einfache Definition einer PEARL-Variablen festgelegt werden, ansonsten wird ein Default-Wert angenommen. Stack-Overflow wird vom Laufzeitsystem erkannt und gemeldet. Außerdem hat der Anwender die Möglichkeit, gezielt auf bestimmte Fehlermeldungen zu reagieren.

Offene Treiberschnittstelle

Die Systemteil-Auswertung wird durch eine sogenannte Konfigurationsliste gesteuert, die alle Standard-Konfigurationsmöglichkeiten des IBM PC beschreibt. Sollen diese Möglichkeiten erweitert werden, z.B. beim Anschluß eines standardmäßig nicht vorgesehenen Gerätes, so kann der Anwender selbst mit einfachen Mitteln diese Konfigurationsliste anpassen und dem Compiler (sogar dynamisch) bekanntmachen.

PEARL Debugger

Für den interaktiven Test von PEARL-Programmen auf Sprachebene hat Werum einen portablen, in PEARL programmierten Debugger entwickelt /1/ , der auch auf IBM PC verfügbar ist.
Dieser Debugger bietet für den Einzeltest von Programmkomponenten komfortable Display-, Trace- und Unterbrechungsmöglichkeiten (Breakpoints):

- **DISPLAY** Mess.Wert
- Mess-Wert := '1101'B
- **LINE TRACE FROM** 55 **TO** 75 **ON**
- **CALL TRACE IN** Erfassung **OFF**
- **BREAK ON** 457 ; **HALT** ; **END**
- **BREAK ON ENTRY** Regelung ; **DISPLAY** Mess ; **END**

Das letzte Beispiel zeigt einen Breakpoint auf den Eingang der Prozedur Regelung mit einem Schnappschuß der Variablen Mess, ohne daß der Debugger in den Dialogbetrieb übergeht, wie es beim vorletzten Beispiel durch das HALT-Kommando verlangt wird.

Über diese "sequentiellen" Möglichkeiten hinaus unterstützt der Debugger auch den Integrations- und Gesamttest durch die Möglichkeiten der Diagnose und Veränderung der Zustände von Tasks und Synchronisationsvariablen sowie durch Schnittstellen zur Simulation der Prozeßperipherie und durch Simulation der Zeitachse:

- **STATUS TASK**
- **STATUS TASK** Erfassung
- **ACTIVATE** Erfassung
- **STATUS SEMA** Puffer_Sema
- **RELEASE** Puffer_Sema

Durch die STATUS-Kommandos kann der Anwender gewissermaßen auf PEARL-Sprachebene in das PEARL-Betriebssystem hineinschauen. Z.B. erhält er als Antwort auf das

Kommando STATUS TASK Erfassung folgende Informationen im Klartext ausgegeben:

- Aktueller Zustand der Task Erfassung
  (nicht aktiv, aktiv, laufend, blockiert, suspendiert, wartend auf I/O)
- Gegebenenfalls Zeitpunkt der Aktivierung, Blockierung, Suspendierung
- Verursacher der Aktivierung, Blockierung, Suspendierung
- Zur Ausführung anstehende Anweisung mit Rückverfolgung bei (geschachtelten) Prozeduraufrufen.

Der Anwender kann nun den Zustand einer Task interaktiv mit den PEARL-Tasking- und Synchronisier-Anweisungen ändern (ACTIVATE, TERMINATE, SUSPEND, CONTINUE, PREVENT, REQUEST, RELEASE).

Literatur

/1/ Brunner, P.J.; Meffert, K.; Windauer, H.:
PEARL-Testsystem. PEARL Rundschau, Band 2, Nr. 6, Dezember 1981, S. 8 - 13.

/2/ DIN 66253, Teil 2:
Programmiersprache PEARL. Full PEARL. Beuth Verlag, Berlin 1982.

/3/ PEARL-Referenzliste. Erhältlich beim PEARL-Verein, Geschäftsstelle München.

/4/ Werum, W.; Windauer, H.:
Introduction to PEARL. Vieweg 1985, 3. Auflage.