Petsy: Ein PEARL Test System zum Austesten von physikalisch verteilten PEARL-Programmen unter Realzeit-Bedingungen

P. Heine / Karlsruhe, F. v. Stülpnagel / München

AUSZUG

Im Rahmen eines Projektes mit dem Ziel, die Echtzeit-Sprache PEARL in ein verteiltes Rechnersystem zu implementieren, wurde in Form eines PEARL-Programm-Paketes das PEarl-Test-SYstem PETSY entworfen und teilweise implementiert. Dieses ermöglicht bei Echtzeit-Bedingungen physikalisch verteilte PEARL-Programme unter seiner Kontrolle ablaufen zu lassen. Es ist bei einem vorhandenen PEARL- und Kommunikations-Betriebssystem weitgehend portabel.

Schlüsselwörter: PEARL, Test-System, verteilt, Echtzeit, Portabelität.

ABSTRACT

Within a project which had the aim to implement the high level process control language PEARL on a distributed system we designed and partly implemented the PEarl Test SYstem PETSY in form of a program package.

PETSY allows to run in realtime conditions physically distributed PEARL-programs under its control. With an existing PEARL- and communication-operating-system it is widely portable.

Keywords: PEARL, Test-System, distributed, realtime, portability.

1. EINLEITUNG

Einen wesentlichen Bestandteil eines Systems von Programmproduktionsmitteln stellt ein Testsystem dar, das die Entwicklung komplexerer Programme unterstützt und damit erheblich zur Zeit- und Kostenersparnis bei der Automatisierung technischer Prozesse beiträgt. Dies gilt insbesondere, wenn für die Realisierung logisch und physikalisch verteilte Programm-Moduln eingesetzt werden, deren Ablauf hochgradig nebenläufig ist und die unter Echtzeit-Bedingungen ausgetestet werden sollen.

Bei dem Entwurf des PEarl-Test-SYstems PETSY konnte auf einer Studie aufgebaut werden, die anläßlich eines Referates für den PDV-Arbeitskreis 'Testen und Verifizieren von Prozeßrechnersoftware' erstellt wurde [1] und in der Anforderungen an Testwerkzeuge formuliert sind.

Die Implementierung erfolgte auf der vorhandenen Hardwarekonfiguration eines verteilten, fehlertoleranten Mehrrechnersystems. Dieses ist ausreichend veröffentlicht [2], so daß hier anhand der Abb. 1 nur noch kurz darauf eingegangen werden soll:

Das Mehrrechnersystem, Really Distributed Computer Control System (RDC-System) genannt, besteht aus RDC-Stationen und einer Zentralwarte, die ringförmig über eine Sammelleitung gekoppelt sind.

Eine RDC-Station besteht ihrerseits aus einem steuernden und einem Prozeß-Mikrorechner, auf dem Assembler- sowie PEARL-Programme ebenso wie auf den Siemens 310 Rechnern der Zentralwarte ablaufen können [3, 4, 9] die RDC-Stationen sind also mit den Siemens 310-Rechnern software-verträglich. Der Benutzer hat Zugriff zu den an den Ring gekoppelten Rechnern über einen Siemens 310-Rechner der Zentralwarte, an welchen neben der üblichen Standardperipherie (Terminal, Bildschirm, Platte) noch ein sogenanntes Ein-Ausgabe-Farbbildschirm-

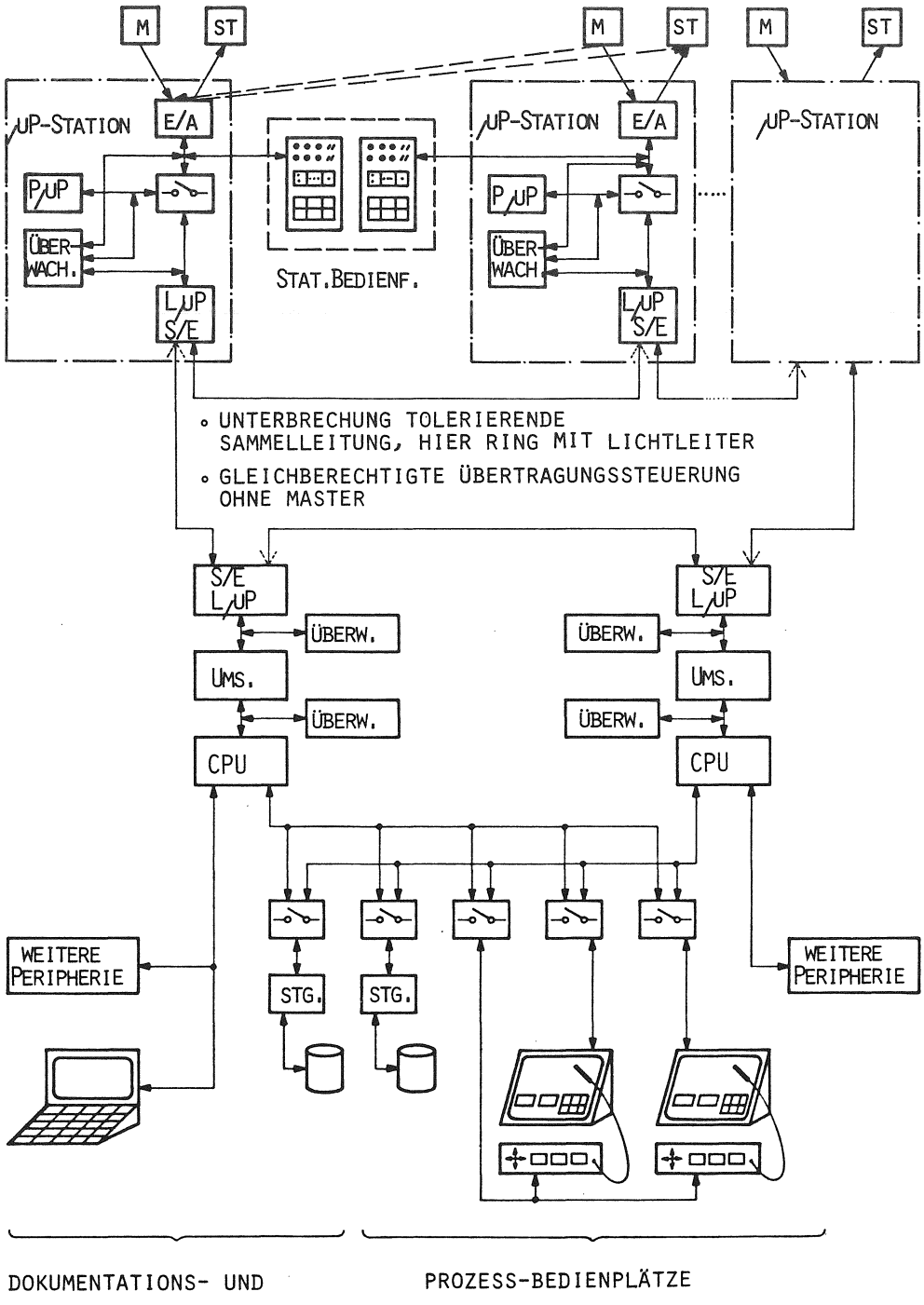

Abb. 1 : Konfiguration des RDC-Systems

System (EAF-System [5]) angeschlossen ist. Dieses dient zur Prozeßsteuerung über den Bildschirm mit Hilfe eines Lichtgriffels.

2. AN PETSY GESTELLTE ANFORDERUNGEN

Mit PETSY sollte ein portables Programmpaket erstellt werden, welches ermöglicht, physikalisch verteilte PEARL-Moduln unter Realzeit-Bedingungen auszutesten. Eine weitere Forderung dabei war, daß für den Testling zum Ablauf unter PETSY kein spezieller (instrumentierter) Code erzeugt werden muß.

Ferner sollte es möglich sein, den programmierten Ablauf der PEARL-Tasks (deren Zustände) beeinflussen zu können durch eng an die PEARL-Syntax und -Semantik angelehnte Anweisungen der PETSY Kommando-Sprache; ein

Beispiel hierfür ist das 'Aktivieren' und 'Suspendieren' von PEARL-Tasks.

Als Grundfunktion des PETSY wurde gefordert, eine praktisch unbeschränkte Menge Haltepunkte an frei wählbaren Stellen des verteilten urgeladenen Testlings setzen zu können, um dann beim Auflaufen auf einen Haltepunkt jeweils, über das Sammelleitungssystem, lesend und schreibend auf die vorhandenen PEARL-Objekte zugreifen zu können.

Die Bedienung des PETSY durch seinen Monitor sollte von jedem beliebigen Teilnehmer des zugrundeliegenden homogenen verteilten Systems durch Laden des PETSY-Monitor-Moduls in diesen Teilnehmer möglich sein.

3. KONFIGURATION DES PETSY

Die Abb. 2 zeigt die Lade-Konfiguration des PETSY. Ein wesentliches Merkmal ist, daß alle Teilnehmer-Rechner für die Kommunikation mit dem Monitor des PETSY jeweils den gleichen Kommunikations-Modul enthalten und daß eine strenge Trennung zwischen dem Monitor-Modul und den Kommunikations-Moduln besteht.

Der Monitor-Modul ist nur in dem Teilnehmer-Rechner vorhanden, von dem aus der Dialog mit dem Gesamtsystem geführt wird.

Der Einrechnerbetrieb erfordert keine spezielle Behandlung insofern der Rechner zwei Kommunikations-Moduln enthält, nämlich den dem Monitor wie auch den dem Testling zugehörigen.

Um kurze Antwortzeiten, sowie eine gute Modularisierung zu erhalten, wurde die Gesamtaufgabe auf einzelne konkurrierende Tasks verteilt.

So enthält der Monitor-Modul eine Task für den Dialog mit dem Benutzer, eine Task für die Ausgabe auf Drucker und eine Task für den Empfang einer Nachricht, falls der verteilte Testling einen Haltepunkt erreicht hat.
Die Kommunikationsmoduln enthalten je eine Task zum Senden und Empfangen von Nachrich-

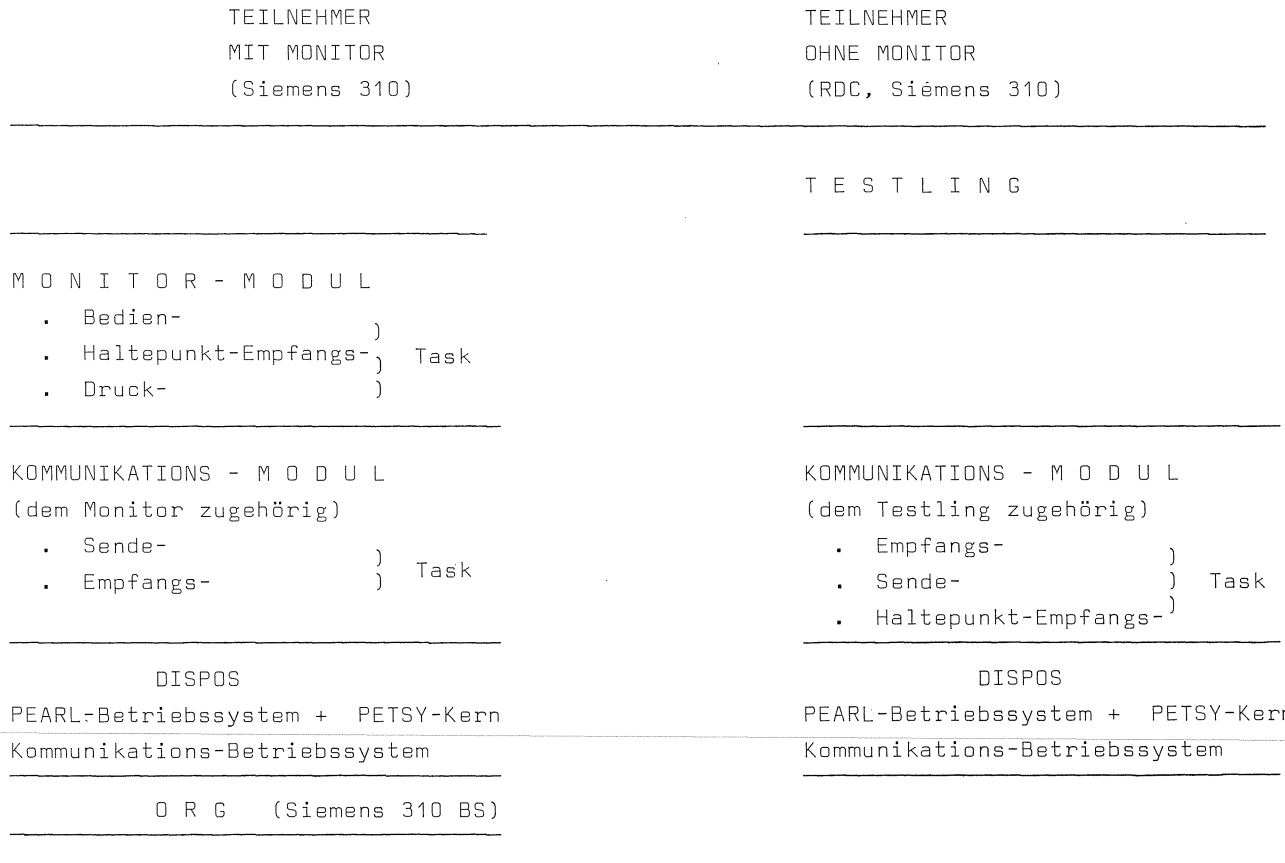

Abb. 2 : Ladekonfiguration des PETSY

ten, sowie eine Task für die asynchrone Unterbrechung beim Auflaufen des Testlings auf einen Haltepunkt innerhalb des eigenen Teilnehmer-Rechners.

Ein PEARL-Betriebssystem und ein Kommunikations-Betriebssystem unter stützen auf unterster Stufe den konkurrierenden Ablauf der PEARL-Task bzw. den Nachrichtenverkehr.

Die Tasks innerhalb eines Moduls synchronisieren sich untereinander über Synchronisationsvariablen (Semaphoren); Tasks verschiedener Module synchronisieren sich über den Austausch von Nachrichten mit time-out.

4. SCHNITTSTELLEN

4.1 Anwender-Schnittstelle (Kommandosprache)

Die der Entwicklung des PEARL-Testsystems zugrundegelegte Anforderungsdefinition enthält einen Funktionsumfang, der für einen effektiven Test von Programmen, die in einer Echtzeitsprache formuliert sind, als unentbehrlich angesehen wird und der auf eine, dem Niveau der höheren Programmiersprache angemessene, leichte Handhabbarkeit abgestimmt ist. Dabei wird zunächst auf komplexere Funktionen und über das Notwendigste hinausgehenden Komfort verzichtet.

Die Funktionen lassen sich grob in drei Gruppen gliedern. Zu einer ersten Gruppe lassen sich Funktionen zusammenfassen, die im wesentlichen für den zeitunkritischen Test eines Programmes geeignet sind und auch in den meisten anderen Test- und Debugprogrammen vorkommen. Dazu gehören Funktionen zur Bestimmung des Testobjektes (TEST) und dessen Lokalisation, Funktionen zur Behandlung von Testpunkten (BREAK, INSERT, CANCEL), Funktionen zum Anzeigen und Verändern von Speicherinhalten (DISPLAY, SET), sowie Funktionen zum Starten und Anhalten des Testlings (START, GO, STOP, KILL).

Eine zweite Gruppe enthält Funktionen, die den Bediener des Testsystems unterstützen. Dazu gehören Hilfs- und Auskunftsfunktionen wie LIST und HELP.

In einer dritten Funktionsgruppe lassen sich die Funktionen zusammenfassen, welche die Echtzeit-Besonderheiten von PEARL berücksichtigen. Mit diesen Testfunktionen kann unmittelbar in die Steuerung der konkurrierenden Prozesse des zu testenden Programmes eingegriffen werden, wobei die Zeit als Einflußgröße eine wesentliche Bedeutung hat. Bei diesen zeitkritischen Testanweisungen sind nicht die Objekte des Prozesses, sondern die des Betriebssystems Gegenstand der Betrachtung bzw. der Manipulation.

Hierher gehören Funktionen zur Prozeßverfolgung (PATHTRACE, VALUETRACE), Prozeß-Kontrollanweisungen, die syntaktisch wie semantisch eng an PEARL angelehnt sind (ACTIVATE, CONTINUE, PREVENT, RESUME, SUSPEND, TERMINATE, TRIGGER, ENABLE, DISABLE), ferner Systemübersichtsfunktionen zur Beeinflussung des Gesamtsystems (LOCK, UNLOCK). Die A b b . 3 enthält alle in der Anforderungsdefinition festgelegten Funktionen zusammen mit den entsprechenden Kommandos.

4.2 Compiler-Schnittstelle

Die Schnittstelle des Compilers zum PETSY besteht pro Modul aus zwei L i s t e n , welche Informationen über die Bezeichner und ihre Attribute enthalten, sowie über Adressen und Zeilennummern. Dabei ist die erste Liste ein sogenanntes Vormerkbuch, bestehend aus:

- Bezeichner-Name
- Speicherklasse (global, lokal)
- TYP
- entsprechend dem TYP:
  Länge oder Genauigkeit oder Unterscheidungskennung
- Relativadresse zum Modulumfang

und die zweite Liste enthält:

- Quellzeile
- Relativadresse zum Modulumfang

Es bleibt noch zu bemerken, daß die Adressierung relativ zum Modulumfang zu Beginn des Monitor-Laufes dadurch absolutiert wird, daß der Monitor vom Betriebssystem eine Liste der Modul-Ladeadressen erhält.

Funktion	Kommando	
Default Testobjekt	TEST	I
Testpunkt setzen	BREAK	
Testpunkt löschen	DELETE	
Testpunkt aktivieren	INSERT	
Testpunkt deaktivieren	CANCEL	
Starten des Testobjektes	START	
Weiterstarten	GO	
Test beenden	STOP	
Genereller Stop	KILL	
Anzeigen Speicherinhalt	DISPLAY	
Verändern Speicherinhalt	SET...=...	
Auflisten Testpunkte	LIST	II
Hilfsfunktion	HELP	
Anzeigen Semavariable	DISPLAY	III
Verändern Semavariable	REQUEST	
Anzeigen Boltvariable	DISPLAY	
Verändern Boltvariable	ENTER	
	LEAVE	
	RESERVE	
	FREE	
Anzeigen Taskzustand	DISPLAY	
Verändern Taskzustand	ACTIVATE	
	CONTINUE	
	PREVENT	
	RESUME	
	SUSPEND	
	TERMINATE	
Task blockieren	LOCK	
Task entblockieren	UNLOCK	
Anzeigen von Interrupts	DISPLAY	
Interruptoperationen	TRIGGER	
	ENABLE	
	DISABLE	
Verfolgen Ablauf	PATH-TRACE	
Verlogen Variable	VALUE-TRACE	

Abb. 3 : PETSY-Kommandos

4.3 Betriebssystem-Schnittstelle

Auf unterster Ebene liegt der 'Kern' des PETSY, welcher eine Erweiterung des bestehenden PEARL-Betriebssystems bezügl. PETSY-Funktionen ist (A b b . 2). Die wesentlichste Funktion des 'Kerns' ist, beim Auflaufen des Testlings auf einen Haltepunkt innerhalb des eigenen Teilnehmer-Rechners, dem PETSY-Monitor darüber eine Nachricht zukommen zu lassen. Hierzu wird das Fehlerverhalten der Teilnehmer-Rechner bei einem nicht-interpretierbaren Befehl ausgenutzt.

Deshalb werden Haltepunkte dadurch gesetzt, daß der Inhalt der Adresse, an der sich der Haltepunkt befinden soll, durch einen nicht-interpretierbaren Befehl ersetzt wird, der einen Zustandswechsel in einen 'Fehlerzustand' bewirkt. In diesem 'Fehlerzustand' wird eine dafür vorgesehene 'Haltepunkt-Empfangs-Task' aktiviert und der 'Fehlerzustand' wird wieder verlassen.

Die 'Haltepunkt-Empfangs-Task' aktiviert ihrerseits die 'Sende-Task', die dann, unterstützt vom Kommunikations-Betriebssystem, dem Monitor die Haltepunkt-Adresse und die Kennung des Teilnehmer-Rechners übermittelt.

Diese Methode der Haltepunkt-Behandlung hat gegenüber anderen Methoden zwei wesentliche Vorteile:

Erstens verändert sie den Code des Testlings und somit auch sein Realzeit-Verhalten nur minimal und nur an der Stelle des Haltepunktes, und zweitens benötigt der Testling für den Ablauf unter der Kontrolle des Testsystems keine spezielle Behandlung durch den Compiler, im Gegensatz zu der Methode, bei welcher der Compiler zum Austesten des Testlings speziellen (instrumentierten) Code erzeugt.

5. IMPLEMENTATION

Der im IITB der Fraunhofer-Gesellschaft, Karlsruhe, installierte PEARL-Compiler wurde von dem Softwarehaus W. Werum, Lüneburg, erstellt. Er läuft im IITB auf einer Siemens R30 /Siemens 310 und erzeugt wahlweise Code für den Ablauf auf Siemens R30- oder Siemens 310/RDC-Maschinen. Der Sprachumfang des Compilers liegt nahezu bei FULL-PEARL; sein genauer Umfang kann aus [7],[8] entnommen werden.

Das PEARL- und Kommunikations-Betriebssystem 'DISPOS' für die RDC-Rechner wurde im IITB entwickelt und im Assembler codiert [4], [9]. Dies gilt auch für die Haltepunkt-Behandlung des PETSY auf unterster Stufe.

Der Monitor-Modul und die Kommunikations-Moduln des PETSY sind in PEARL codiert. Daraus folgt, daß PETSY bei einem vorhandenen PEARL- und Kommunikations-Betriebssystem weitgehend portabel ist.

PETSY wurde bisher nur in den Funktionsstufen I und II (vgl. A b b . 3) realisiert, noch nicht in den Funktionen, welche die vom PEARL-Betriebssystem verwendeten Objekte betreffen (Stufe III).

Der beschränkte Speicherausbau der Siemens 310 von maximal 128 KByte erschwert die vollständige Implementation des PETSY.

Overlay-Technik wird vom Siemens 310-Betriebssystem nicht unterstützt. Deshalb ist die vollständige Implementation auf der Siemens R30 geplant.

Dabei kann von der Portabilität des PETSY Gebrauch gemacht werden, die darauf beruht, daß das Testsystem weitgehend selbst in PEARL geschrieben ist.

Neu zu erstellen (wenn auch nicht neu zu definieren) wäre in diesem Fall die unter Abschnitt 4.3 aufgeführte Betriebssystem-Schnittstelle.

6. Zusammenfassung

Wir haben das PEARL-Testsystem PETSY vorgestellt, das im IITB Karlsruhe entwickelt wurde. Als Zusammenfassung sollen abschließend charakteristische Merkmale gesammelt aufgeführt werden:

- PETSY erlaubt es, PEARL-Programme auf der (PEARL-) Sprachebene auszutesten, unter weitgehender Verwendung der PEARL-Schlüsselwörter in der PETSY-Kommandosprache;
- Durch die Methode des dynamischen Einsetzens der Testpunkte wird das Realzeit-Verhalten des Testlings nur geringfügig geändert, und dieser benötigt keine spezielle Behandlung durch den Compiler, um im Test ablauffähig zu sein.
- PETSY ist modular derart konfiguriert, daß ohne Änderung der PETSY-Moduln ein Testling wahlweise entweder im Einrechner-Betrieb auf demselben Rechner wie der PETSY-Monitor oder auf einem oder mehreren Teilnehmer-Rechnern ablaufen kann.
- Der modulare Aufbau von PETSY erlaubt ferner leicht Erweiterungen hinzufügen.
- PETSY besitzt dadurch, daß es weitgehend in PEARL für PEARL-Programme geschrieben ist, den hohen Grad an Portabilität, wie die Sprachimplementation selbst.

Literaturverzeichnis

[1] I. Hertlin, M. Mackert: Beitrag zu "Testen und Verifizieren von Prozeßrechner-Software", PDV-Bericht KfK-PDV 179. Dez. 79, des Kernforschungszentrums Karlsruhe.

[2] Echtzeitrechnersystem mit verteilten Mikroprozessoren, BMFT-Forschungsbericht DV 79-01.

[3] I. Hertlin, L. Lorenz: PEARL-Programmierung auf der Siemens 310: Übersetzungssystem, Beitrag zur SAK-Tagung 12.-14. Mai 1980, KfK Jülich.

[4] G. Bonn, P. Heine: PEARL-Programmierung auf der Siemens 310: ein PEARL-Betriebssystem; Beitrag zur SAK-Tagung 12.-14. Mai 1980, KfK Jülich.

[5] Bildprogrammierbares Ein-/Ausgabe-Farbbildschirmsystem (EAF) als Warte; PDV-Bericht 137, 1978, des Kernforschungszentrums Karlsruhe.

[6] Full PEARL Language Description, PDV-Bericht 130, 1977, des Kernforschungszentrums Karlsruhe.

[7] W. Werum, H. Windauer: PEARL, Vieweg-Verlag 1978.

[8] H.U. Steusloff: MEHRRECHNER-PEARL, PEARL-Rundschau, Band 1, No. 4, Dez. 80.

[9] D. Heger: Systemergänzungen und Pilot-erprobung eines fehlertoleranten Echtzeitsystems mit verteilten Mikroprozessoren (RDC-System), BMFT-Forschungsbericht DV, Mai 1981.