Echtzeit

Nr. 2, Januar 2015

Mitteilungen
des GI/GMA/ITG-Fachausschusses
Echtzeitsysteme

GESELLSCHAFT FÜR INFORMATIK E.V.

VDI VDE VDI/VDE-Gesellschaft
Mess- und Automatisierungstechnik

ITG INFORMATIONSTECHNISCHE
GESELLSCHAFT IM VDE

Impressum

Herausgeber	GI/GMA/ITG-Fachausschuss Echtzeitsysteme http://www.real-time.de
Sprecher	Prof. Dr. Dr. Wolfgang A. Halang FernUniversität in Hagen Lehrstuhl für Informationstechnik 58084 Hagen wolfgang.halang@fernuni-hagen.de
Stellvertreter	Prof. Dr. Dieter Zöbel Universität Koblenz-Landau Institut für Softwaretechnik 56016 Koblenz zoebel@uni-koblenz.de
Chefredaktion	Prof. Dr.-Ing. habil. Herwig Unger FernUniversität in Hagen Lehrstuhl für Kommunikationsnetze 58084 Hagen herwig.unger@fernuni-hagen.de
Redaktion, Layout	Dipl.-Ing. Jutta Düring FernUniversität in Hagen Lehrstuhl für Informationstechnik 58084 Hagen jutta.duering@fernuni-hagen.de
ISSN	2199-9244

Redaktionell abgeschlossen am 21. Januar 2015

Einreichung von Beiträgen:
Alle Leserinnen und Leser sind aufgerufen, das Mitteilungsblatt auch zukünftig durch Beiträge mit zu gestalten, um den Informations- und Meinungsaustausch zwischen allen an den Fragen der Echtzeitprogrammierung Interessierten zu fördern.

In dieser Ausgabe:

1 Tagung Echtzeit 2015: Call for Paper

Jutta Düring, Fachausschuss Echtzeitsysteme

Die Tagung „Echtzeit“ findet in diesem Jahr am 12. und 13. November 2015 wie gewohnt in Boppard am Rhein statt. Neu ist die Durchführung zusammen mit der GI-Fachgruppe Betriebssysteme (www.betriebssysteme.org).

Konsequenterweise heißt unser Leitthema „**Betriebssysteme und Echtzeit**“. Zu folgenden und benachbarten Themen werden Vorträge über Methoden, praktischen Einsatz, Erfahrungen und Ausblicke erbeten. Exponate sind immer willkommen.

- Entwurf, Schnittstellen und Architekturen
  - Mikrokernbasierte Betriebssysteme
  - Anwendungsspezifische Maßschneiderung
  - Bewertung und systematischer Entwurf von Systemschnittstellen
- Echtzeitkommunikation
- Modellierung und Simulation
- Plattformen
  - Einsatz von Graphik-Coprozessoren
  - Echtzeitanwendungen auf Multi/Manycore-Systemen
  - Ausnutzung rekonfigurierbarer Hardware
- Virtualisierung
- Energieeffizienz, Sicherheit und Fehlertoleranz
- Programmiersprachen
- Bewertung von Echtzeiteigenschaften
- Aktuelle Anwendungen
- Ausbildung

Stichtag für die Vortragsanmeldung ist **Montag, der 20. April 2015**. Nähere Informationen finden Sie unter www.real-time.de/CfP.html.

2 Kommunikation mit Peripheriegeräten unter PEARL2020

Christina K. Houben, FernUniversität in Hagen
christina.houben@fernuni-hagen.de

Am Lehrstuhl für Informationstechnik der FernUniversität in Hagen unter Leitung von Prof. Dr. Dr. Halang und in Zusammenarbeit mit Partnern aus Forschung und Industrie wird die Erstellung einer neuen Normvorlage für PEARL geplant, und zwar unter dem vorläufigen Akronym PEARL2020. Das Vorhaben hat zum Ziel, die Sprache für sicherheitskritische Echtzeit-Systeme zu wappnen, indem sie die Einschränkungen in der Programmierung für die Sicherheitsintegritätslevel nach IEC 61508-3 unterstützt, inhärent sichere Synchronisationsmechanismen bereitstellt und Prioritäten durch anwendungsorientierte Antwortzeiten ersetzt etc. Ein weiterer Punkt ist, die Ein- und Ausgabe zu vereinfachen und an den Stand der Technik anzupassen. Dazu möchten wir im Folgenden unsere Überlegungen zur Kommunikation mit Peripheriegeräten

darstellen. Gleichzeitig geben wir die neuen Syntaxregeln für PEARL in erweiterter Backus-Naur-Form wie in Tabelle 1 beschrieben an und erläutern ihre Semantik. Gerne nehmen wir Kritik und weitere Anregungen zu unseren Vorschlägen entgegen.

Tabelle 1: Syntax der erweiterten Backus-Naur-Form

`<...>`	Nichtterminal: Symbol muss weiter ersetzt werden
`FETT`	Schlüsselwort: Symbol nicht weiter ersetzbar
`"..."`	Terminal: Symbol nicht weiter ersetzbar
`::=`	Zuweisungszeichen in einer Produktionsregel
`...|...`	Alternative: entweder das eine oder das andere Symbol wird verwendet
`{...}`	Block: Zusammenfassung mehrerer Symbole zu einem
`... ...`	Konkatenation (Leerzeichen): Symbole werden aneinandergehangen
`[...]`	Option: Symbol wird keinmal oder einmal verwendet
`[...]*`	Wiederholung beliebig oft: Symbol keinmal, einmal oder mehrmals verwenden
`[...]+`	Wiederholung mindestens einmal: Symbol wird einmal oder mehrmals verwendet

Dies führt uns zur ersten Syntaxdefinition in Abbildung 1, die zeigt, wie ein PEARL-Programm aus Modulen aufgebaut ist.

```
<Programm-Def>   ::= [<Modul-Def>]+
<Modul-Def>      ::= MODULE [<Modulname>] [<SIL-Definition>] ";"
                        [<Schedulingteil>]
                        [<Exportteil>] [<Importteil>]
                        [<Systemteil>] [<Problemteil>]+
                     MODEND ";"
<Modulname>      ::= "(" <Bezeichner> ")"
<Bezeichner>     ::= <Buchstabe> [<Buchstabe> | <Ziffer> | "_"]*
<Buchstabe>      ::= "A" | "B" | ... | "Z" | "a" | "b" | ... | "z"
<Ziffer>         ::= "0" | "1" | "2" | ... | "7" | "8" | "9"
```

Abbildung 1: Syntax für ein PEARL-Programm und -Modul

2.1 Aufbau des Systemteils

In PEARL2020 wird wie bisher auch eine Trennung in System- und Problemteil vorgenommen, um die Portabilität der erstellten Programme zu ermöglichen. Der Systemteil dient ausschließlich dazu, die eingesetzte Hardware-Konfiguration festzulegen, indem die Systemnamen und damit maschinenabhängigen Namen von Peripheriegeräten auf benutzerdefinierte bzw. logische Namen abgebildet werden, welche problemorientiert sein sollen. Wird die erstellte Software auf ein anderes System portiert, müssen i.d.R. nur die Systemnamen angepasst werden, der ganze Problemteil bleibt unberührt, was zur Portabilität beiträgt. Ein Beispiel aus [1] für eine Hardware-Konfiguration und den zugehörigen Systemteil ist in Abbildung 2 angegeben.

In dem Beispiel werden nicht nur die direkt angeschlossenen Peripheriegeräte definiert, sondern auch Zwischenstationen wie der Multiplexer `MUX32` und der A/D-Wandler `ADC50`. Der Vorteil davon ist, dass der Aufbau der Hardware-Konfiguration dokumentiert wird, sofern es nicht schon woanders in graphischer Form getan wurde. Die bildliche Dokumentation an anderer Stelle sollte im Sinne der funktionalen Sicherheit unbedingt erfolgen.

```
MODULE;
SYSTEM;
   ADC50->CPU200*4;
   MUX32->ADC50;
   THERMELEM(1:20):->MUX32*1,20;
   DEHNELEM(1:12):->MUX32*21,12;
PROBLEM;
   (*  ...  *)
MODEND;
```

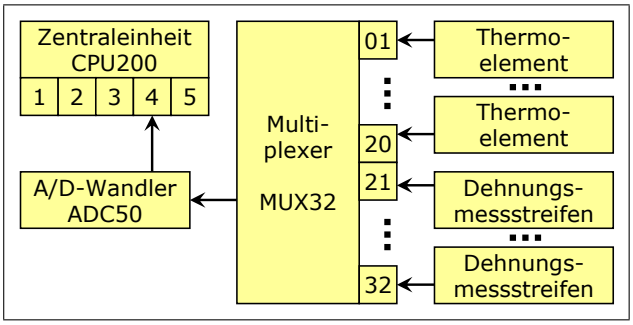


Abbildung 2: Systemteil und Hardware-Konfiguration in PEARL90

In der folgenden Liste sind drei verschiedene Möglichkeiten angegeben, wie man bisher unter PEARL90 Peripheriegeräte bzw. die Ports, unter denen sie angeschlossen sind, vereinbaren konnte. Alle drei Möglichkeiten sollen unter PEARL2020 zur Verfügung stehen, woraus sich die in Abbildung 3 angegebene Syntax für den Systemteil ergibt.

(1.) Übersetzerseitig definierte Bezeichner für Standard-Peripherie: Hierzu zählen bspw. die Bezeichner `A1`, `B1` und `C1` für die Konsole unter WinSTon [2], und `/XC` als Systemdatenstation, um Befehle an den Bedieninterpreter von RTOS-UH zu senden [3].

(2.) Angabe von Ports, an denen Peripheriegeräte angeschlossen sind: Ein Beispiel hierfür ist der Systemteil aus Abbildung 2 nach [1]. Darin werden unter dem Namen `CPU200` mehrere Anschlusspunkte (`*3`) eines systemspezifischen Digitaleingangs zusammengefasst. Gibt es mehrere gleichartige Geräte, so werden diese durch eine Ziffer in Klammern angesprochen wie in `DIGOUT(1)*3`. Geräteklassen, Gerätebezeichner und Anschlusspunkte müssen im Handbuch des jeweiligen Mikroprozessors nachgeschlagen werden. Zu dem Zeitpunkt, zu dem PEARL als Programmiersprache für SPSen zur Verfügung steht, können als Eingabeports `E0.0`, `E0.4`, `E1.2` etc. angesprochen werden, als Ausgabeports `A15.0` etc. und als Merker `M0.0` etc., siehe [4].

(3.) Absolute oder relative Pfade, über die Peripheriegeräte gemountet sind: Dateien werden in PEARL als Datenstationen gehandhabt. Nur für Dateien konnten in Basic PEARL nach [1] Pfade angegeben werden. In PEARL unter RTOS-UH gab es jedoch schon die Möglichkeit, Datenstationen unter absoluten Pfaden oder Pfaden relativ zum Arbeitsverzeichnis anzugeben, z.B. `XYZ: LD/5.3/abcd/efg`, siehe [3]. In Betriebssystemen wie Linux oder Windows ist dies sogar üblich, ein Beispiel ist `/dev/ttyS1` für eine serielle Schnittstelle unter Linux.

```
<Systemteil>     ::= SYSTEM ";"
                       [<Verdrahtung>]*
                       [<Geraetezuord.>]*
                       [<Geraetespez.>]*
<Richtung>       ::= "->" | "<-" | "<->"
<Verdrahtung>    ::= <Systemname> [<Richtung> <Systemname>]+ ";"
<Geraetezuord.>  ::= <Benutzername> ":" <Richtung> <Systemname> ";"
<Geraetespez.>   ::= {SPC | SPECIFY} <Benutzername> ":"
                       <Geraeteklasse> USING {In | Out | InOut} ";"
<Systemname>     ::= <Bezeichner> ["(" <Nummer> [, <Nummer>] ")"]
                                  ["*" <Nummer> [, <Nummer>] ]
<Benutzername>   ::= <Bezeichner> |
                       <Bezeichner> ["(" <Nummer> [, <Nummer>] ")"]
```

Abbildung 3: Syntax für den Systemteil von PEARL2020

Der Systemteil von PEARL2020 wird in Zukunft aus drei Teilen bestehen, nämlich einer Übersicht über die Verdrahtung der Peripheriegeräte, der Zuordnung von Benutzer- zu Systemdatenstationen, und der Spezifikation der Geräteklassen. Letztgenanntes wurde bei PEARL90 erst im Problemteil angegeben, in Zukunft möchten wir jedoch alle drei Teile im Systemteil zusammenfassen, um alle Informationen über angeschlossene Peripheriegeräte an einem Ort übersichtlich darzustellen. Außerdem wird so das in [5] genannte Problem vermieden, bei dem die Typkonsistenz bei getrennter Übersetzung von System- und Problemteil nicht überprüfbar ist. Verdrahtung und Gerätezuordnung werden wie in PEARL90 auch codiert. Zusätzlich gibt es die Möglichkeit, einen ganzen Verdrahtungsweg vom am weitesten entfernten Peripheriegerät bis zur Hauptplatine innerhalb einer einzigen Textzeile anzugeben. Weitreichender geändert wurden nur die Gerätespezifikationen, womit sich der nächste Abschnitt beschäftigt.

2.2 Bisherige Vereinbarung von Datenstationen

Im Problemteil werden die Eigenschaften der im Systemteil angegebenen Datenstationen spezifiziert. Durch diese Eigenschaften wird ein spezieller Typ von Datenstation definiert, wodurch die Vorteile der Typisierung auf die Kommunikation mit Peripheriegeräten übertragen werden. Zu diesen Vorteilen gehört, dass durch die Überprüfung der Typkorrektheit bestimmte Fehler bereits zur Übersetzungszeit erkannt werden, wodurch die Anzahl der Laufzeitfehler reduziert wird [6]. Beispielsweise kann der Übersetzer die korrekte Adressierung bestimmter Speicherbereiche mit den Befehlen `SKIP`, `ADV` oder `POS` gewährleisten. Bisher konnten in PEARL90 folgende Datenstations-Eigenschaften festgelegt werden [1]:

→ Art der Datenstation: `ALL`, `ALPHIC`, `BASIC`
→ Richtung der Datenübertragung: `IN`, `OUT`, `INOUT`
→ Anzahl der Dimensionen (bei Dateien und Terminals): keine, beliebig viele oder eine bestimmte Anzahl an Spalten bzw. Kolonnen, Zeilen und Seiten
→ aus Gründen der Rückwärtskompatibilität zu noch älteren PEARL-Versionen als PEARL90 konnte für `ALPHIC`-Stationen das Token `CONTROL(ALL)` angegeben werden, um anzuzeigen, dass Formatlisten zur Textausrichtung im Tabellenmuster benutzt werden dürfen

Tabelle 2: Arten von Datenstationen in PEARL90

Datenstation	`ALL`	`ALPHIC`	`BASIC`
Kommandos	`READ` und `WRITE`	`GET` und `PUT`	`TAKE` und `SEND`
Inhalt	Bitketten wie im RAM	Unicode-Schriftzeichen	digitale Prozessdaten
Beispiel	Magnetplatte	Drucker	Messwertgeber
Warteschlange	ja: Speicherhierarchie	ja: Aufträge spoolen	nein: meist direkte E/A
Formatierung	nein	ja: `A`, `SKIP`, `POS`, `ADV`, ...	nein: nur Einzelwerte
Dimensionen	ja: Datensätze	ja: Seiten, Zeilen, ...	nein: nur Einzelwerte

Es gibt also insgesamt vier verschiedene Haupteinteilungen für Datenstationen: Art, Richtung, Dimensionen und Textausrichtung. Die Unterteilung und Bedeutung der Arten ist in Tabelle 2 ausführlich angegeben. Werden für die Datenstationen Dimensionen spezifiert, so müssen noch weitere Eigenschaften genauer vereinbart werden [1]:

→ ob nur ganze Zeilen übertragen werden dürfen (`TFU`) oder ob auch Teilstücke übertragen werden dürfen (`TFU MAX`)

→ wie entlang der Dimensionen positioniert werden darf: `DIRECT` für absolute Positionierung (`COL`, `LINE`, `POS`, `SOP`), `FORWARD` für relative Positionierung nur in Vorwärts-Richtung, `FORBACK` für relative Positionierung in beide Richtungen (`X`, `SKIP`, `PAGE`, `ADV`)

→ ob bei Erreichen des Endes der Datenstation von vorn begonnen werden darf (`CYCLIC`) oder nicht (`NOCYCL`)

→ ob eine Fehlermeldung ausgegeben werden soll, wenn über das Zeilenende hinaus gelesen oder geschrieben wird (`NOSTREAM`) oder einfach in die oder aus der nächsten Zeile weitergeschrieben oder gelesen wird (`STREAM`)

Einige Betriebssysteme stellen darüber hinaus noch weitere Eigenschaften zur Typisierung einer Datenstation bereit, in RTOS-UH sind dies die in Tabelle 3 angegebenen Parameter für Geräte, die nicht notwendiger Weise in Dateisysteme gegliedert sind. Diese Parameter können in RTOS-UH mit dem `DD`-Befehl (engl.: display device parameters) abgefragt werden [3]. Damit ist unsere Übersicht über die aktuelle Typisierung von Datenstationen abgeschlossen, womit wir als nächstes zu den Neuerungen in PEARL2020 übergehen.

Tabelle 3: Geräteparameter ohne notwendiges Dateisystem in RTOS-UH

Geräteparameter	PEARL90	PEARL2020
Eingabe möglich	`IN`	✓
Ausgabe möglich	`OUT`	✓
Ein- und Ausgabe möglich	`INOUT`	✓
Zeilenvorschub (`LF`) mit Wagenrücklauf (`CR`)	✗	✗
Dialogfähig	✗	✗
Rückspulbar	`CYCL`, `FORBACK`	✗
Erlaubt Formatierbefehle	`ALPHIC`	✗
Abbruch der Eingabe bei `LF`, `CR`, `EOF`	`TFU MAX`	✗
Fehlererkennung bei Timeout	✗	✓
Braucht `OPEN` bzw. `CLOSE`	✗	✗

2.3 Neue Spezifikationen von Geräten

Seit dem Erscheinen des PEARL90-Sprachreports hat sich bei der Hardware und in Betriebssystemen einiges geändert. Speicherplatz kostet wenig und ist daher ausreichend vorhanden, so dass Befehle zum Komprimieren von Daten unnötig sind. Betriebssysteme stellen standardmäßig Dienste bereit, welche Verwaltungsaufgaben wie Zwischenspeichern von Druckaufträgen in einer Warteschlange oder die direkte Weiterleitung von Steuervariablen an D/A-Wandler vor den Anwendungsprogrammen verbergen und in Treiber bzw. I/O-Dämonen auslagern. Dieses Verhalten ist auch in Standards wie z.B. POSIX vereinheitlicht. Dadurch braucht sich ein Programmierer um die Art und Weise, wie Peripheriegeräte im Einzelnen angesprochen werden, nicht zu kümmern. Deshalb lässt sich die benötigte Syntax für die Ein- und Ausgabe im Sprachdesign stark vereinfachen.

Darüber hinaus kommt man auch von einem anderen Standpunkt aus zu diesen Vereinfachungen der Sprache. Aus Sicht der objektbasierten Programmierung können die verschiedenen Eigenschaften der Datenstationen in Klassen gekapselt werden. Diese Klassen sind Strukturen, die zusätzlich Prozeduren beinhalten können (und nicht nur Zeiger auf Prozeduren wie bei `STRUCT`s

in PEARL90). An objektbasierten Eigenschaften erlaubt PEARL2020 Objekte, Klassen, Kapselung, Ad-hoc- und parametrische Polymorphie mit heterogener Übersetzung, nicht jedoch Prototypen, Vererbung, Subtyp-Polymorphie, homogene Übersetzung parametrisierter Klassen oder dynamisches Binden [7]. Daraus ergeben sich für die Kommunikation mit Peripheriegeräten folgende Möglichkeiten:

→ Die Richtung der Datenübertragung wird weiterhin mit `In`, `Out` und `InOut` angegeben, allerdings sind dies die Namen der verschiedenen `INTERFACE`s, die durch eine Datenstations-Klasse veröffentlicht werden. Bei der Instanziierung einer Klasse muss sich der Programmierer mittels `USING` für eines der Interfaces entscheiden, siehe [8].
→ Die Zugehörigkeit von Prozeduren zu Klassen verhindert, dass bspw. fälschlicherweise eine Prozedur zur absoluten Cursorpositionierung aus Treiber A für eine Datenstation zur relativen Positionierung aus Treiber B verwendet wird.
→ Die Schlüsselwörter `ALL`, `ALPHIC` und `BASIC` für die Angabe der Art einer Datenstation entfallen, weil diese Eigenschaften im Treiber bzw. durch das Betriebssystem gekapselt sind.
→ Dies erlaubt die drei verschiedenen Schreib- und Lesekommandos `WRITE` und `READ`, `PUT` und `GET`, `SEND` und `TAKE` einzusparen. Sie werden durch treiberspezifische Prozeduren ersetzt. Deren Prozedursignaturen können wie bisher mit `#INCLUDE` [3] durch den Präprozessor des PEARL-Übersetzers eingelesen und spezifiziert werden. Die üblichen Kommandos können dennoch weiterhin über die Treiber zur Verfügung gestellt werden, wie z.B. `Open`, `Close`, `Take` und `Send` für den Schrittmotor aus Abbildung 4.
→ Aus dem gleichen Grund entfallen die Schlüsselwörter, die mit den Dimensionen einer Datenstation einhergehen, nämlich `DIM` für die Dimensionsanzahl, `TFU` und `TFU MAX` für den Umgang mit zu kurzen Zeilen und `STREAM` und `NOSTREAM` für den Umgang mit zu langen Zeilen, als auch `DIRECT`, `FORWARD` und `FORBACK`, sowie `CYCL` und `NOCYCL` für die Art der Cursorpositionierung. Die korrekte Art und Weise des Zugriffs wird nicht mehr über Schlüsselworte der Sprache gehandhabt, sondern durch Syntaxprüfung gegen die eingebundenen Prozedursignaturen.

Dazu geben wir ein Beispiel für eine Datenstation in Abbildung 4 an, mit der ein Treiber für einen Schrittmotor bereitgestellt wird, der das RS-232-Protokoll benutzt. Treiber sind zwar Klassen, werden jedoch in PEARL2020 wie auch in ObjectPEARL mit dem Schlüsselwort `PERIPHERAL` definiert, welches im Gegensatz zum Schlüsselwort `CLASS` folgende Besonderheiten aufweist:

- Es werden spezielle Interrupt-Routinen bereitgestellt, die sich auf Hardware-Ebene von klassischen Methoden unterscheiden [11].
- Geräteklassen dürfen nur im Systemteil zur Spezifikation verwendet werden, nicht jedoch für beliebige Objekte im Problemteil.
- Nur solche Klassen dürfen Konstrukte zur Ausnahmebehandlung einsetzen. Dadurch wird das in PEARL90 bekannte Schlüsselwort `RST` ersetzt. Außerhalb von `PERIPHERAL`-Klassen sind Methoden der Ausnahmebehandlung nicht zugelassen, sondern sollen durch strenge Typprüfungen nach dem Vorbild der Sprache NewSpeak vermieden werden.
- Alle Methoden einer `PERIPHERAL`-Klasse werden so übersetzt, dass sie nur in einem Stück ausgeführt werden können, um Betriebsmittel nicht unnötig lang zu blockieren.
- Die Kommunikation mit Peripheriegeräten darf nur synchron erfolgen.
- Im Zusammenhang mit Peripheriegeräten werden Unterbrechungen eher erlaubt, obwohl Unterbrechungen nach IEC 61508-3 im Allgemeinen verboten sind.

Die bisherigen und auch die folgenden Sprachdefinitionen für die Kommunikation mit Peripheriegeräten sind aus den folgenden Gründen trotz der Neuerungen in Richtung Objektbasierung

```
/* Beschreibung:
 *   Treiber fuer einen Schrittmotor, der ueber die serielle
 *   Schnittstelle mit dem RS-232-Protokoll angesprochen wird.
 * Methoden:
 *   Open / Close -- ersetzen OPEN und CLOSE aus PEARL90
 *   Take / Send -- ersetzen TAKE und SEND fuer BASIC-Dations
 */
MODULE(RS232SchrittmotorTreiber);
TYPE RS232Schrittmotor PUBLIC PERIPHERAL [
  PUBLIC;
    INTERFACE(Std);
      Open(Nr FIXED): PROC;
      Close: PROC;
    INTERFACE(In) EXTENDS(Std);
      Take: PROC RETURNS(FIXED(8));
    INTERFACE(Out) EXTENDS(Std);
      Send: PROC(Schritte FIXED(8));
    INTERFACE(InOut) EXTENDS(In, Out);
  PRIVATE;
    #IFDEF RTOSUH;
      DCL PortName CHAR(6);
    #ELSE #IFDEF LINUX;
      DCL PortName CHAR(10);
    #ELSE #IFDEF WINDOWS;
      DCL PortName CHAR(4);
      DCL Handle FIXED;
    #FIN;
];
RS232Schrittmotor.Open: PROC(Nr FIXED);
  ! Portbezeichnung in Abh. der Laufwerksnummer
  ! Einstellungen der Schnittstelle zuruecksetzen
  ! Baudrate auf 1000 Symbole pro Sekunde setzen
  #IFDEF RTOSUH;
    PortName := 'LD/' CAT TOCHAR(PortNr + 48) CAT '.0';
    EXEC('SB ' CAT PortName CAT ' 1000');
  #ELSE; #IFDEF LINUX;
    PortName := '/dev/ttyS' CAT TOCHAR(PortNr + 48);
    EXEC('stty sane ' CAT PortName);
    EXEC('sttc ispeed 1000 ospead 1000 ' CAT Portname);
  #ELSE; #IFDEF WINDOWS;
    PortName := 'COM' CAT TOCHAR(PortNr + 48);
    SetBaudRate(PortName, '1000,n,7,1');
    Handle := CreateFile(PortName, GENERIC_READ OR
      GENERIC_WRITE, 0, NIL, OPEN_EXISTING, 0, 0);
  #FIN;
END;
RS232Schrittmotor.Close: PROC; ... END;
RS232Schrittmotor.Take: PROC RETURNS(FIXED); ... END;
RS232Schrittmotor.Send: PROC(Schritte FIXED);
  #IFDEF RTOSUH;
    EXEC('O ' CAT PortName CAT '; ECHO ' CAT TOSTRING(Schritte);
  #ELSE; #IFDEF LINUX;
    EXEC('ECHO ' CAT TOSTRING(Schritte) CAT ' ' CAT PortName);
  #ELSE; #IFDEF WINDOWS;
    WriteFile(Handle, Schritte BY TYPE FIXED(8), 8, NIL, NIL);
  #FIN;
END;
```

Abbildung 4: Treiber für einen Schrittmotor in PEARL2020

für die höchsten Sicherheitsintegritätslevel SIL 4 und SIL 3 nach IEC 61508-3 geeignet. Treiber wie in Abbildung 4 können als zum Übersetzer gehörig betrachtet werden. Innerhalb der nach außen gelegten Methoden `Open`, `Close`, `Take` und `Send` besteht die Möglichkeit Betriebssystemfunktionen bzw. Dienste aufzurufen, die zu einer Schale bzw. Scheibe des Echtzeitsystems gehören, welche ohne Verzögerung eine Aus- oder Eingabe bewirken. Dafür kann der Code aus den Treiber-Klassen vom Übersetzer direkt in die aufrufende Stelle kopiert werden, ähnlich `INLINE` in C. Dies bewirkt, dass gegenüber dem `TAKE` und `SEND` der `BASIC`-Datenstationen aus PEARL90 weiterhin keine Zeitverzögerung auftritt. Für andere Datenstationen, die eine Warteschlange zum Sammeln von Aufträgen brauchen, können Betriebssystemfunktionen äußerer Schalen bzw. Scheiben verwendet werden. Außerdem besteht die neue Ein-/ Ausgabesyntax ausschließlich aus Prozeduraufrufen, und diese Prozeduren bestehen i.d.R. nur aus einem einzigen Aufruf einer betriebsbewährten oder vom TÜV geprüften Betriebssystemfunktion. Für höchste SIL werden darüber hinaus nur die drei sehr einfachen Geräteklassen `AnaloguePin` für analoge Signale, `DigitalPin` für digitale Signale und `PWMPin` für pulsweitenmodulierte Signale erlaubt. Komplexere und daher fehleranfälligere Geräteklassen wie der RS-232-Treiber werden nur in Anwendungen, die einem geringen oder gar keinem SIL unterliegen, erlaubt.

2.4 Kommunikation in Tasks und Prozeduren

Bisher haben wir uns mit der Hardware-Konfiguration im Systemteil und der Spezifikation der logischen Geräte im Problemteil beschäftigt. Nun kommen wir zur Verwendung der Datenstationen innerhalb von Tasks und Prozeduren. Dafür gibt es in PEARL90 folgende Sprachmittel:

→ Manche Betriebsmittel müssen mit `OPEN` initialisiert werden, was beinhaltet, dass sie reserviert und in einen definierten Anfangszustand versetzt werden.
→ Im Sprachreport [2] können mit `OPEN BY` weitere Eigenschaften einer Datenstation festgelegt werden, allerdings belaufen sich diese Eigenschaften nur auf `IDF` zur Angabe eines Dateinamens und `NEW`, `OLD`, `ANY`, falls eine Datei neu erstellt, eine alte weitergeschrieben werden soll oder beides.
→ Analog verhält es sich im Sprachreport mit dem Schließen von Datenstationen mittels `CLOSE` oder `CLOSE BY` für das Schließen mit (`PRM`) oder ohne (`CAN`) Löschen der Datei.
→ Zusätzlich steht unter dem PEARL-System von RTOS-UH auch das Schlüsselwort `EXCLUSIVE` beim Öffnen einer Datenstation zur Verfügung, was anzeigt, dass das Betriebsmittel nur von der öffnenden Task benutzt wird [3].
→ In [1] wird darüber hinaus auf einen A/D-Wandler von Krupp-Elektronik verwiesen, dessen Integrationszeit mittels `OPEN BY` eingestellt werden konnte. Der Name der Eigenschaft wird leider nicht genannt und konnte nachträglich nicht ermittelt werden. Es ist aber davon auszugehen, dass es sich um eine Schlüsselwort-ähnliche Prozedur handelte.
→ Zum Abfangen von Fehlern, die sowohl beim Öffnen, Schließen, als auch bei formatierten Ein-/ und Ausgaben auftreten, wird in allen Literaturangaben das Schlüsselwort `RST` genannt, welches einen `FIXED` als Referenzparameter entgegennimmt. Tritt ein Fehler auf, steht in dem `FIXED` anschließend eine Fehlernummer, mit der sich die Art des Fehlers identifizieren lässt.

Die eben genannten Sprachmittel verändern wir so, dass sie in das bisher ausgearbeitete Konzept der Objektbasierung passen, es werden nämlich die Schlüsselwörter `OPEN`, `CLOSE`, `READ`, `WRITE` etc. durch Methoden ersetzt. Eigenschaften wie `OLD`, `CAN` und `IDF` werden durch Übergabevariablen an die zuerst genannten Methoden realisiert. Dadurch können auch Integrationszeiten wie im Beispiel des Krupp-Elektronik-Wandlers einfach angegeben werden. Das Wort `EXCLUSIVE` aus

RTOS-UH wird durch noch zu entwickelnde einfach zu benutzende und strukturierte Synchronisationsmechanismen abgelöst werden. Die Fehlererkennung mittels `RST` wird in PEARL2020 durch ein Ausnahmekonzept behandelt werden, welches u.A. Fehler durch Typprüfung zur Übersetzungszeit vermeidet.

Somit verbleiben für den Aufruf von Ein-/ und Ausgabefunktionen noch zwei Änderungen. Die erste Änderung gab es in RTOS-UH schon, ist aber bisher noch nicht in PEARL realisiert, nämlich Timeouts bei Lese- und Schreiboperationen, siehe Tabelle 3. Diese fördern das Echtzeitverhalten, indem sie Verklemmungen auflösen. Hierauf sowie auch auf Synchronisation und Fehlerbehandlung werden wir in einem anderen Bericht näher eingehen.

Die zweite Änderung betrifft die Angabe von Zeitschranken für E/A-Operationen. PEARL2020 wird die Angabe von Zeitschranken anstelle von Prioritäten unterstützen, und dies nicht nur auf die E/A beschränkt. Dies ist nicht nur anwendungsorientierter, sondern ermöglicht die Kombination der Sprache mit zeitgerechten Zuteilungsstrategien wie dem Antwortzeitalgorithmus [9]. In ObjectPEARL wird diese Eigenschaft mit dem Schlüsselwort `AT` für `TIMESTAMPED`-Methoden realisiert [10]. Ein Beispiel hierfür aus [10] ist in Abbildung 5 gezeigt.

```
DAConverter PERIPHERAL [
  PROC Init RETURNS(FIXED);
  PROC Done RETURNS(FIXED);
  PROC SetValue(V FIXED, Time CLOCK);
  PROPERTY Value FIXED WRITE SetValue TIMESTAMPED;
];

DCL DA1 AT ADDR 123;
IF DA1.Init /= 0 THEN (* Fehler *);
DA1.Value := 4 AT 12:00:00;
```

Abbildung 5: Methodenaufruf mit Zeitschranken in ObjectPEARL

Die Programmiersprache ObjectPEARL wurde in [11] entwickelt. Danach ist das Schlüsselwort `PERIPHERAL` eine Erweiterung von `CLASS` und stellt Peripheriegeräte dar. Es unterscheidet sich von `CLASS` dadurch, dass spezielle Interrupt-Routinen bereitgestellt werden, die sich auf Hardware-Ebene von klassischen Methoden unterscheiden. Diese Sprachsyntax ermöglicht es, abstrakte Klassen zu definieren, um Fehlertoleranz-Mechanismen umzusetzen, z.B. durch redundante Sensoren, die durch eine einzige Klasse gekapselt werden [11].

2.5 Dateiverarbeitung

Bisher musste man in PEARL90 für den Zugriff auf eine einzige Datei gleich zwei Datenstationen definieren, siehe Abbildung 6 aus dem Sprachreport. Darin wird zuerst eine Datenstation mit `SPC` spezifiziert, die eine über den Port `DIGE(1)*0*1,16` angeschlossene Festplatte ist. Danach wird mit `DCL` eine zweite Datenstation deklariert, die eine Datei auf einer mit `CREATED` angegebenen Datenstation entspricht. In PEARL90 muss also um zwei Ecken programmiert werden. Das letzte bisher noch nicht erwähnte Schlüsselwort in der E/A ist `ASSIGN`, mit dem einer deklarierten Datenstation ein neuer Dateiname auch zur Laufzeit zugewiesen werden kann.

In PEARL90 wie auch in vielen anderen Programmiersprachen gibt es wenigstens drei verschiedene Arten von Dateien, nämlich maschinenlesbare Dateien in Interndarstellung, menschenlesbare mit Schriftzeichen in ASCII oder Unicode und Verzeichnisse. Erstgenannte werden in PEARL90

```
MODULE(RegalFoerderZeug);
SYSTEM;
  RFZ: DIGE(1)*0*1,16;
  ...
PROBLEM;
  SPC RFZ DATION IN BIT(16);
  DCL Rfz DATION IN BIT(16) CREATED(RFZ);
  ...
MODEND;
```

Abbildung 6: Dateizugriff für ein Regalförderzeug in PEARL90

als `ALL`-Datenstationen vereinbart, letztgenannte als `ALPHIC`. Diese Unterteilung und Beispieldatenformate sind in Abbildung 7 angegeben.

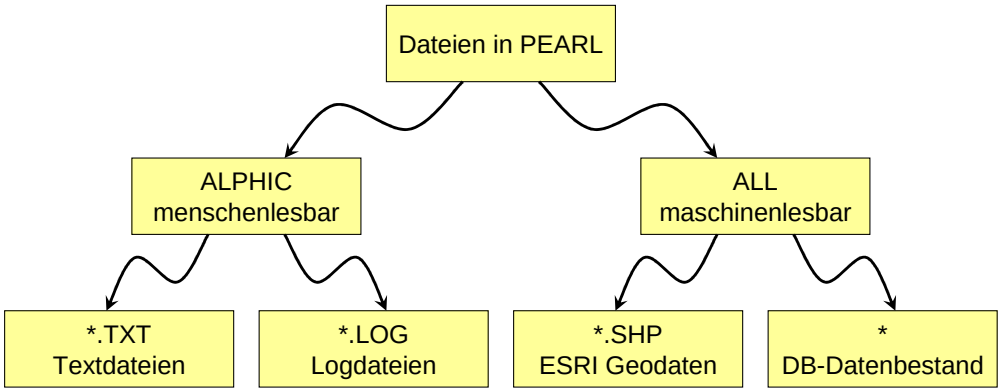


Abbildung 7: Arten von Dateien in PEARL90

Unter RTOS-UH kann man auch für Dateien weitere Eigenschaften festlegen, analog denen in Tabelle 3. Dazu zählen, ob Verzeichnisse mit `DIR` angelegt werden können, oder ob eine direkte Cursorpositionierung mit `SEEK` möglich ist.

In PEARL2020 wird nur noch die Spezifikation *einer* Datenstation für eine Datei nötig sein. Die verschiedenen Eigenschaften wie Dateiname, Datenformat (`FIXED`, `FLOAT`, etc. oder Schriftzeichen) werden wie bei den anderen Datenstationen auch über die vom Übersetzer zur Verfügung gestellten Treiber umgesetzt, wie auch das Schlüsselwort `ASSIGN` als Methode.

2.6 Zusammenfassung

Wir haben dargestellt, wie die Kommunikation mit Peripheriegeräten bisher in PEARL90 und seinen Derivaten unter RTOS-UH umgesetzt wurde. Daraus haben wir Verbesserungen hinsichtlich einfacher Handhabung abgeleitet und das E/A-Konzept in ein objektbasiertes Paradigma eingebunden. Gleichzeitig wird funktionale Sicherheit nach IEC 61508-3 erreicht.

Als Themen, die unabhängig von der Ein- und Ausgabe auch in anderen Bereichen der Programmierung auftauchen, jedoch die E/A beeinflussen, verbleiben für weitere Untersuchungen: Synchronisation, Verklemmungen, Fehlerbehandlung, Fehlertoleranz und Antwortzeiten statt Prioritäten.

2.7 Danksagung

Besonderer Dank gilt Prof. Dr. R. Müller, der Diskussionspunkte zur (a)synchronen Kommunikation und Fehlerbehandlung ausführlich erörterte. Trotz seines Einsatzes beinhaltet diese Darstellung Klassen und Methoden statt Schlüsselwörter. Die Autorin möchte jedoch versichern, dass dennoch zur Übersetzungszeit strenge Typprüfungen der verwendeten E/A-Klassen möglich sind.

Weiterer Dank gilt dem Bundesministerium für Wirtschaft und Energie für die Mitfinanzierung des Forschungsprojekts „Normung der Echtzeitprogrammiersprache PEARL hinsichtlich funktionaler Sicherheit“ (01FS14002).

Literatur

[1] L. Frevert: Echtzeitpraxis mit PEARL, Vol. 2, Springer, 1987

[2] GI-Fachgruppe 4.4.2: PEARL 90 Sprachreport, GI Gesellschaft für Informatik e.V., 1995

[3] W. Gerth: RTOS-UH, Version 5.4, 2006, Handbuch

[4] H. Tapken: SPS – Theorie und Praxis, Vol. 3, Europa-Lehrmittel, 2013

[5] H. Kölle: Konzeption und prototypische Umsetzung des E/A-Systems für einen PEARL-Compiler, in: *Informatik aktuell, Industrie 4.0 und Echtzeit*, Springer Vieweg, 2014

[6] A. Poetzsch-Heffter: Konzepte Objektorientierter Programmierung, Springer, 2000

[7] C.K. Houben: Fostering Functional Safety of Real-time Systems with Concepts of Object Orientation, In: *Proc. of 19th Intl. Conference on Methods and Models in Automation and Robotics (MMAR)*, 2014

[8] A.H. Frigeri, C.E. Pereira, W.A. Halang: An Object-Oriented Extension to PEARL90, In: *Proc. of Int. Symp. on Object-Oriented Real-time Distributed Computing*, S. 265–274, 1998

[9] W.A. Halang, R.M. Konakovsky: Sicherheitsgerichtete Echtzeitsysteme, Springer, 2013

[10] M. Colnaric, D. Verber, W.A. Halang: Improving Integrity of Embedded Computers in Control, In: *Annual Reviews in Control*, Volume 27, Issue 1, S. 47–54, 2003

[11] D. Verber: Object Orientation in Hard Real-Time System Development, Dissertation, Universität zu Maribor, Slovenia, 1999

3 Vorwort des Tagungsbandes Echtzeit 2014

„Industrie 4.0“ ist in aller Munde und wird heftig diskutiert. Deshalb hat das Programmkomitee die diesjährige Fachtagung Echtzeit unter das Leitmotiv „Industrie 4.0 und Echtzeit“ gestellt. Aber was bedeutet dieser schillernde Begriff überhaupt? Der Lenkungskreis Plattform Industrie 4.0 gibt sich vollmundig:

> „Der Begriff Industrie 4.0 steht für die vierte industrielle Revolution.“

Ein solcher Anspruch ist sicher vermessen und unseriös, denn „bemerkenswert ist die Tatsache, dass erstmalig eine industrielle Revolution ausgerufen wird, noch bevor sie stattgefunden hat“ [1]. Das emotional aufgeladene Konzept Industrie 4.0 wird mit Heilserwartungen verknüpft und in vielfältigster Weise interpretiert, weil es an einer wissenschaftlich exakten Definition mangelt. Das Bundesministerium für Bildung und Forschung (BMBF) umreißt grob, worum es gehen soll[2]:

> „Die realen Abläufe und ihre Steuerung und Optimierung durch virtuelle IT-gestützte Prozesse werden derzeit durch ein technisches Bindeglied verkoppelt: Der Einbau vernetzter, leistungsfähiger eingebetteter Systeme – so genannter Cyber-Physical Systems – in viele Alltagsgegenstände stellt die direkte Verbindung von realer Welt mit intelligenten Steuerungsprozessen im so genannten Internet der Dinge und Dienste her. . . . In der Industrie ermöglicht diese verteilte, aber vernetzte Intelligenz bessere Monitoring- und autonome Entscheidungsprozesse, um Unternehmen und ganze Wertschöpfungsnetzwerke in *nahezu Echtzeit* steuern und optimieren zu können. Möglich werden damit individualisierte Produkte zu den Bedingungen einer hoch flexiblen Großserienproduktion.“

Die in diesem Zitat enthaltenen und für den GI/GMA/ITG-Fachausschuss Echtzeitsysteme relevanten Termini sind Echtzeit sowie eingebettete und cyber-physikalische Systeme. Der letztgenannte Begriff ist zwar recht neu, damit bezeichnete Systeme sind aber in der Automatisierungstechnik seit rund vier Jahrzehnten gang und gäbe, und im Echtzeitbetrieb arbeitende, in Messanlagen „eingebettete“ digitale Datenverarbeitungssysteme gibt es sogar bereits seit 70 Jahren[3]. Während das Hauptaugenmerk des Fachausschusses auf Systeme mit harten Echtzeitbedingungen, die darüber hinaus oft auch noch sicherheitskritisch sind, gerichtet ist, stellen im Rahmen von Industrie 4.0 zu entwickelnde Systeme nach der Aussage des BMBF nur weiche Echtzeitanforderungen.

Somit stehen auch aus Sicht der Echtzeitsysteme alle Informations- und Kommunikationstechniken zur Verfügung, die für Industrie 4.0 gebraucht werden. „. . . Jetzt gilt es (nur), diese Technologien für die Industrie zu erschließen“ [1].

Dass es sich aus der Echtzeitperspektive bei Industrie 4.0 nicht um die vierte industrielle Revolution, sondern einen „alten Hut“ handelt, mag auch erklären, warum nur recht wenige Beiträge zu diesem Leitmotiv eingereicht wurden. Diese beschäftigen sich mit der semantischen Integration von Feldgerätedaten im Rahmen durchgängiger Fabrikvernetzung, der Organisation kollaborativer Fertigung eines Produkts mittels eines Multiagentensystems zur Vernetzung der durchaus unterschiedlichen Steuerungssysteme der beteiligten Maschinen mit dem Ziel, die Losgröße eins zu erreichen, sowie der Genauigkeit der bei der automatischen Umkonfiguration flexibler Produktionssysteme erforderlichen zeitlichen Synchronisation verteilter Steuerungen. Die im Zuge von Industrie 4.0 geplante vollständige Vernetzung wird zu enormen informationellen Sicherheitsproblemen führen. Deshalb ist ein Beitrag ihrer Lösung hinsichtlich Gewährleistung der Übertragungssicherheit in der mobilen Steuerungs- und Überwachungstechnik unter Echtzeitbedingungen und ein anderer dem Schutzaspekt durch Authentisierung und Autorisierung in der Logistik und im Gesundheitswesen gewidmet.

---

[1] R. Drath: Industrie 4.0 – eine Einführung. *open automation*, 3/14, 16–21, 2014

[2] `http://www.bmbf.de/de/9072.php`

[3] K. Zuse: Vorwort zu *Constructing Predictable Real Time Systems* von W.A. Halang und A.D. Stoyenko, Boston-Dordrecht-London: Kluwer Academic Publishers 1991

Als aktuelle Anwendungen werden eine als Knoten in Sensornetzen verwendbare Komponente für selbständig ausgeführte Fluoreszenzmessungen sowie eine auf Tablet-PCs laufende Geschwindigkeitsregelung beschrieben, mit der das Echtzeitverhalten des Betriebssystems Android in Verbindung mit Drahtlosübertragung für Einsatzmöglichkeiten in Automobilen untersucht wird.

Eine modulare und erweiterbare Plattform erlaubt echtzeitfähige Simulation verschiedenster Sensoren mit unterschiedlichen digitalen Schnittstellen in Test- und Entwicklungssystemen für Steuergeräte. Um in solchen Umgebungen gemessene Werte verschiedener Signalquellen zeitlich zu synchronisieren, werden neue Ansätze vorgestellt. Der Simulation unmittelbar zugänglich werden auch im Echtzeitbereich immer häufiger zur Software-Spezifikation eingesetzte UML-Beschreibungen durch automatische Übersetzung in Simulink-Modelle.

Die Programmierumgebung OpenPEARL90 ist gedacht, PEARL insbesondere für Ausbildungszwecke unter Linux zur freien Verfügung zu stellen. Sie nimmt eine Zwischenübersetzung nach C++ vor und ihr Laufzeitsystem bietet umfassende Ein-/Ausgabemöglichkeiten sowie eine offene Treiberschnittstelle. Um die Erstellung hoch verläßlicher und verifizierbarer Software für sicherheitskritische Anwendungen zu unterstützen, wurde eine erweiterte Teilmenge von PEARL definiert, die den Sicherheitsanforderungen der Stufe SIL 3 gemäß IEC 61508 genügt und die in die nächste Version der PEARL-Norm eingehen wird.

Frau Dipl.-Ing. Jutta Düring sei ganz herzlich für die überaus sorgfältige Überarbeitung der eingegangenen Texte sowie die schöne Gestaltung des Tagungsbandes und dem Springer-Verlag für die verbesserten Konditionen zu seiner Publikation gedankt. Ganz besonderer Dank gebührt in diesem Jubiläumsjahr Konrad Zuse, der vor 70 Jahren en passant die Echtzeitsysteme erfunden hat.

Hagen, im August 2014

Wolfgang A. Halang
Herwig Unger

4 Best Paper Award und Graduiertenwettbewerb 2014

Jutta Düring, Fachausschuss Echtzeitsysteme

Als besten Tagungsbeitrag der „Echtzeit 2014“ hat das Programmkomitee

„Plug and Work für verteilte Echtzeitsysteme mit Zeitsynchronisation“

von Sebastian Schriegel, Jürgen Jasperneite und Oliver Niggemann prämiert.
(http://dx.doi.org/10.1007/978-3-662-45109-0_2)

Die Beiträge der Gewinner des Graduiertenwettbewerbs 2014 lauten

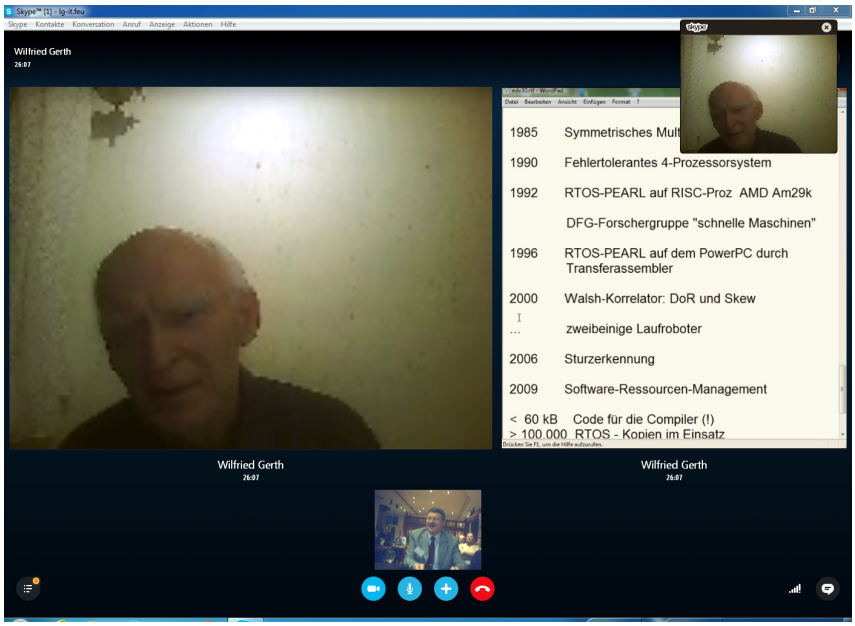
- Jürgen Hillebrand: Eine sicherheitsgerichtete Echtzeitprogrammiersprache für die Sicherheitsstufe SIL 3 gemäß DIN EN 61508 (http://dx.doi.org/10.1007/978-3-662-45109-0_3)
- Holger Kölle: Konzeption und prototypische Umsetzung des E/A-Systems für einen PEARL-Compiler (http://dx.doi.org/10.1007/978-3-662-45109-0_5)
- Hermann Ludwig Lorenz: Der Raspberry Pi als Plattform für Fluoreszenzmessungen unter Echtzeitbedingungen (http://dx.doi.org/10.1007/978-3-662-45109-0_8)

Prof. Dr. Juliane Benra gratulierte den Preisträgern und überreichte die Urkunden.
v. l.: J. Hillebrand, H. Kölle, H. L. Lorenz, J. Benra, S. Schriegel

5 Ehrenmitgliedschaft für Prof. Dr.-Ing. Wilfried Gerth

Jutta Düring, Fachausschuss Echtzeitsysteme

In Anerkennung seiner Leistungen auf dem Gebiet der Echtzeitsysteme und der Entwicklung der Programmiersprache PEARL hat der Fachausschuss im November 2014 Prof. Dr.-Ing. Wilfried Gerth zu seinem Ehrenmitglied ernannt.



Professor Gerth konnte zur Ehrung auf der Tagung Echtzeit 2014 leider nicht persönlich erscheinen; er wurde per Videokonferenz hinzu geschaltet. In seinem Vortrag verknüpfte er wichtige Stationen seines 70-jährigen Lebens mit wichtigen Ereignissen aus 70 Jahren Echtzeit. Die Konferenzteilnehmer würdigten dies mit einem überwältigenden Applaus.

6 GI-Konferenz Autonome Systeme

Jutta Düring, Fachausschuss Echtzeitsysteme

Der Arbeitskreis Echtzeitkommunikation veranstaltet einmal jährlich im Oktober für einen eingeladenen Kreis die Konferenz „Autonome Systeme“ auf Mallorca. In 2008 als Doktorandenseminar gestartet entwickelte es sich zu einer Konferenz mit 40 bis 50 Teilnehmern aus ganz Europa.

Der Konferenzband 2014 beinhaltet folgende Beiträge:

Session 1: Hardware and Embedded Systems

- D. Verber: High-performance Computing for Embedded Control Systems
- S. Widmann: An Inherently Safe Microprocessor Architecture
- M. Schaible: A Consensus-oriented Crowd-verifiable Microprocessor Architecture
- M. Kirchhoff and W. Fengler: Realization of an Embedded Hard Realtime Softcore Processor
- Th. Hempen and Th. Tempelmeier: The Test Concept for a Communication Link to an Unmanned Aerial Vehicle
- C. K. Houben: Selected Aspects of Functional Programming Fostering Technical Safety within PEARL2020

Session 2: Systems, Models and Design

- M. Müller, T. Machleidt and W. Fengler: SoC Design for Complex Standalone Optical Measurement Devices
- I. Kaiser, W. Fengler and T. Fröhlich: Development of a Processing System with the 3W-Model
- B. Däne: Tool-supported Design of an Application-specific Soft Microprocessor
- Z. Szeifert: Deflection Queuing: Examining the Efficiency of a New Switching Approach
- G. Zhang, Z. Li, B. Zhang and W. Halang: Power Electronics Converters: An Overview

Session 3: Theoretic Approaches and Algorithms

- F. Blaauw, L. Bazylevska and M. Aiello: Gamfication in a Consulting Company
- H. Lefmann and D. Winkler: The Bahncard Problem and Upgrades
- S. Boonkrong: Multi-factor Authentication for Internet Banking Login Process
- A. Bukowiec, T. Gidlewicz and J. Tkacz: Coloring of Interpreted Petri Nets
- J. Ch. Chedjou et al.: On Transforming Graph-theoretical Problems into Ordinary Differential Equations
- M. Ali et al.: A Review of Object Classification for Video Surveillance Systems
- A. Haj Mosa, M. Ali, F. Al Machot and K. Kyamakya: A Computerized Method to Diagnose Strabismus

Session 4: Content, Network and Services

- M. M. Kubek: DocAnalyser – Searching with Web Documents
- P. Krause: A New Approach for Peer-to-Peer Information Retrieval Systems
- R. Leonhard-Pfleger: A Categorization Scheme for Feedback in Networks
- S. Vongsingthong, H. Unger and Ph. Meesad: Identifying Clusters within Facebook's User Behavior
- A. Lapin and E. Schiller: Integrated Cloud-based Computational Services
- E. Canzani, H.-C. Heldt, S. Meyer and U. Lechner: Towards an Understanding of the IT Security Information Ecosystem

Das Programm beinhaltet auch Kurzvorträge ohne Veröffentlichung im Konferenzband. Außerhalb der Veranstaltungen bleibt ausreichend Zeit für Diskussionen bei einer Tasse Kaffee oder einem Spaziergang am Strand.

Termin 2015

In diesem Jahr findet die Konferenz vom **25. bis 30. Oktober 2015** in Cala Millor auf Mallorca statt. Nähere Informationen finden Sie unter http://www.fernuni-hagen.de/kn/autsys/.

Auch wenn die Teilnahme an der Konferenz ausschließlich auf Einladung basiert, können Sie sich bei Interesse gerne an Professor Unger unter herwig.unger@fernuni-hagen.de wenden.

7 Umzug von Webseiten

Jutta Düring, Fachausschuss Echtzeitsysteme

Die Webseiten des Instituts für Regelungstechnik der Universität Hannover zum Echtzeitbetriebssystem RTOS-UH und zur Echtzeitprogrammiersprache PEARL sind umgezogen. Die neuen Adressen lauten:

http://rtos.iep.de/ bzw. http://www.pearl90.de/

Informationen zu PEARL finden Sie ebenfalls auf den Webseiten des Fachausschusses unter den Menüpunkten „Service“ und „Archiv“ (http://www.real-time.de/).