Die Programmierumgebung OpenPEARL90

Marcel Schaible[1] Rainer Müller[2]

[1]Lehrstuhl für Informationstechnik, insb. Realzeitsysteme
FernUniversität, 58084 Hagen
Marcel.Schaible@FernUni-Hagen.de

[2]Hochschule Furtwangen, Fakultät CEE, Robert-Gerwig-Platz 1, 79120 Furtwangen
mueller@hs-furtwangen.de

Workshop Echtzeit 2014 in Boppard

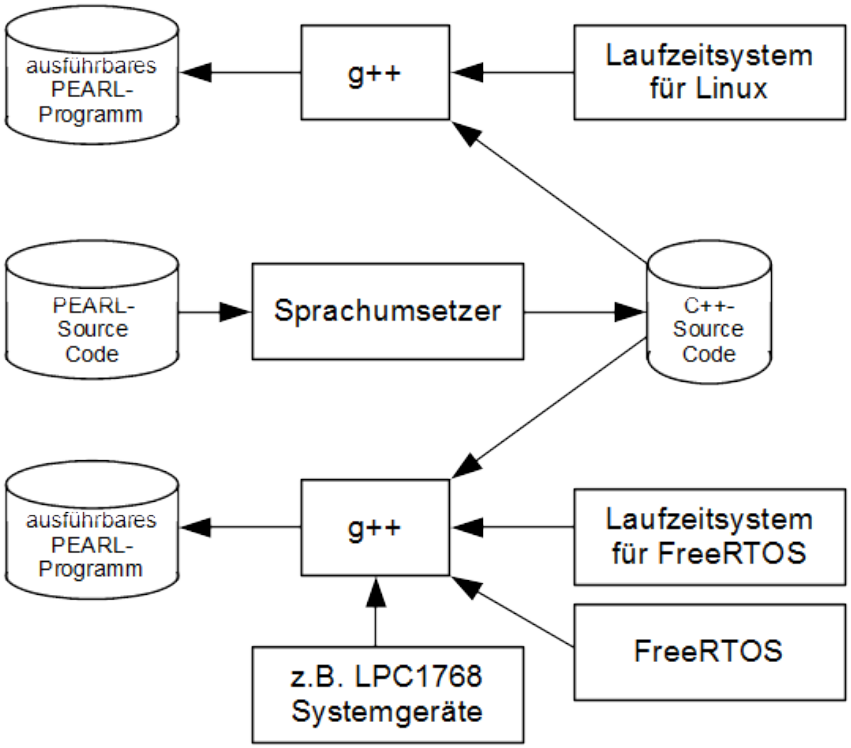
Agenda

1. Motivation
2. Architekturüberblick
3. Sprachumsetzer PEARL $\rightarrow$ C++
4. Codegenenerierung
5. Entwicklungsstand des Übersetzers
6. Laufzeitsystem
7. Zusammenfassung und Ausblick

Motivation

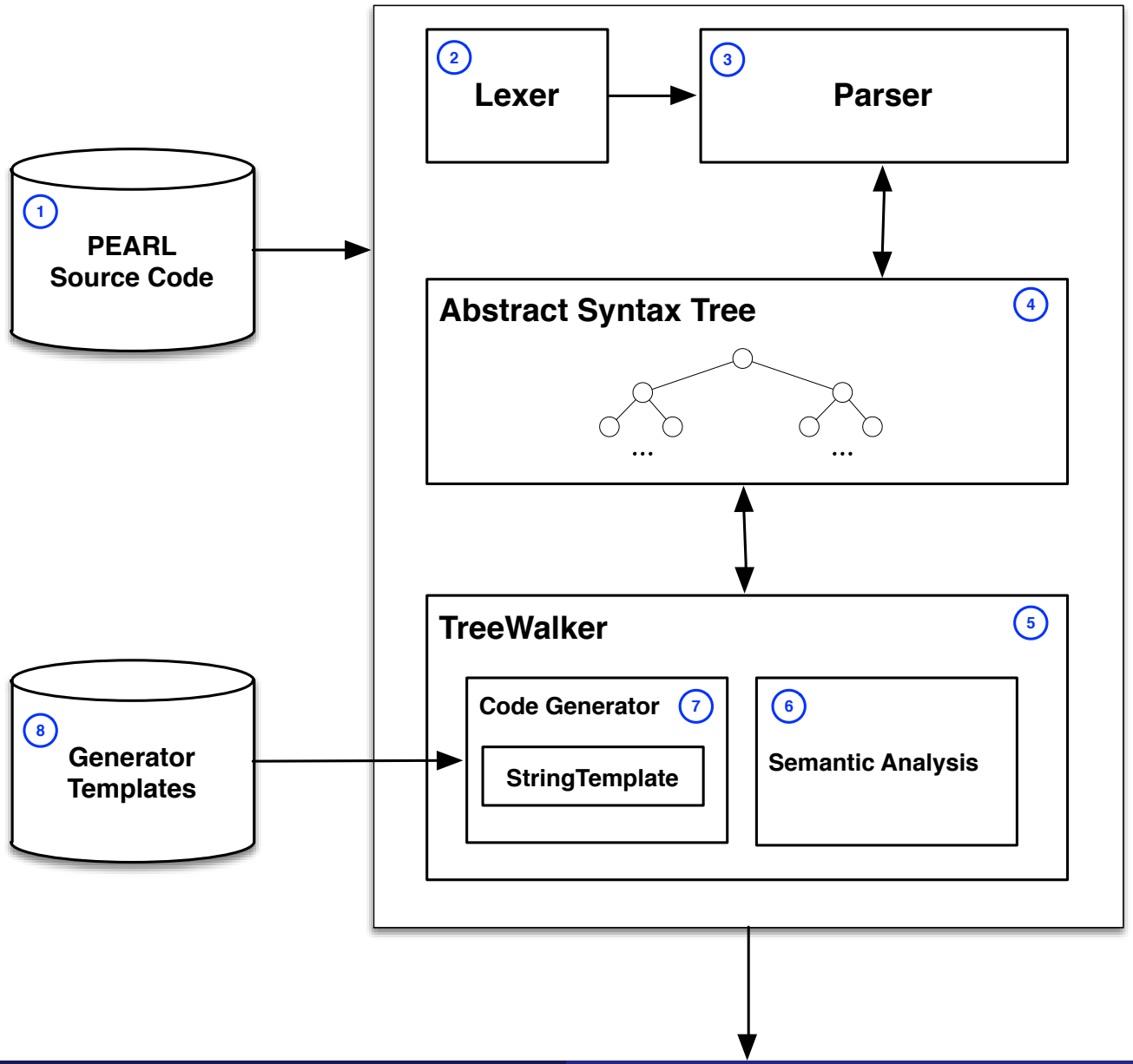
- Ideale Programmiersprache für Ausbildungszwecke um nebenläufige Abläufe zu beschreiben
- restriktive Programmiersprache
- Unterstützung des Sprachstandards PEARL90
- Zielplattform: primär Linux ohne Modifikationen mit C++-Übersetzer
- Lizenzmodell: OpenSource und weitgehend konform mit der BSD Lizenz
- Systematische Trennung von Sprachumsetzer und Laufzeitsystem
- Quelltexte und Dokumentaion frei über die Projekt-Webseite verfügbar

Struktur

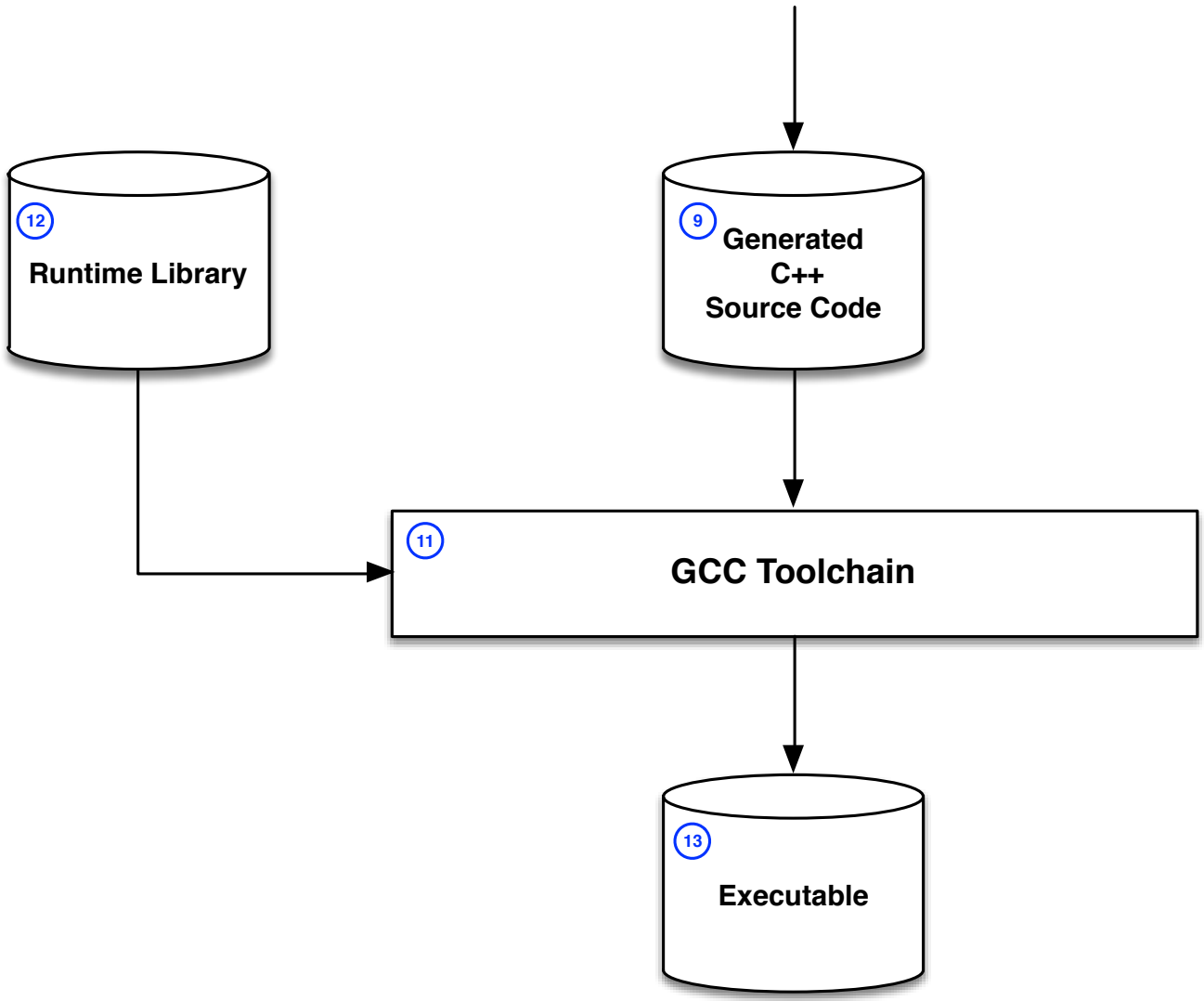


- Sprachumsetzer von *PEARL* auf C++
- Targetabhängiges Laufzeitsystem
  - Linux
  - FreeRTOS auf Mikrocontroller

Architekturüberblick I



Architekturüberblick II



Sprachumsetzer PEARL → C++

- Übersetzerskelett wird mit ANTLR aus einer EBNF-ähnlichen Syntaxbeschreibung von PEARL erzeugt.
- **AN**other **T**ool for **L**anguage **R**ecognition: objektorientierter Lexer- und Parsergenerator mit der Eigenschaft LL(k).
- Parser erzeugt während der Quelltextanalyse einen abstrakten Syntaxbaum und eine dazugehörige Besucherklasse.
- Algorithmus für den Baumdurchlauf wird mitgeneriert
- Mittels __cpp__ kann C++-Code direkt in das Kompilat eingefügt werden.

Besucherklasse

- Besucherklasse enthält für alle Sprachkonstrukte eine Methode mit ihrem Kontext.
- Der Algorithmus füe dem AST-Durchlauf ruft entsprechend des gerade bearbeiteten Sprachkonstruktes die passende Methode auf.
- Sichtbarkeitsregeln werden unter Zuhilfenahme einer globalen Symboltabelle verwaltet.

```
public class MyVisitor<T> extends TreeVisitor<T>
                           implements Visitor<T>
  @Override public T visitModule(ModuleContext ctx) { ... }
  @Override public T visitTask_decl(Task_declContext ctx) { ... }
  @Override public T visitProblem(Problem_Context ctx) { ... }
  @Override public T visitSystem(System_Context ctx) { ... }
```

Codegenerierung

- Strikte Trennung von Syntax- und Semantikanalyse und Codegenerierung
- StringTemplate: Spezifikation in einer EBNF-ähnlichen Sprache

```
module(src,name,system,taskspecifiers,tasks,cpp_inlines) ::= <<
  <if(system)> <system> <endif>
  <if(taskspecifiers)> <taskSpecifierList(taskspecifiers)> <endif>
  <if(tasks)> <tasks> <endif>
  ...
>>

taskspecifier(taskname) ::= <<
"SPCTASK(<taskname>);"
>>
```

Übersetzer: Entwicklungsstand

- Sprachumfang

  Ausdrücke sind bis auf Typprüfung fertig.

  Tasks sind weitgehend fertig.

  Systemteil fehlt.

  PEARL-spezifische Funktionen sind teilweise implementiert.

  Prozeduren, Schleifenkonstrukte sind in Arbeit.

- Semantische Analyse

  Symboltabellenverwaltung Grundfunktionalität implementiert.

  Typüberprüfungen sind für einfache Datentypen teilweise vorhanden.

- Fehlerbehandlung

  Syntaxfehler werden entsprechend der ANTLR Spezifikation erkannt.

  Semantikfehler teilweise vorhanden bzw. in Arbeit.

- Codegenerierung

  StringTemplates für unterstützte Konstrukte fertig

Laufzeitsystem

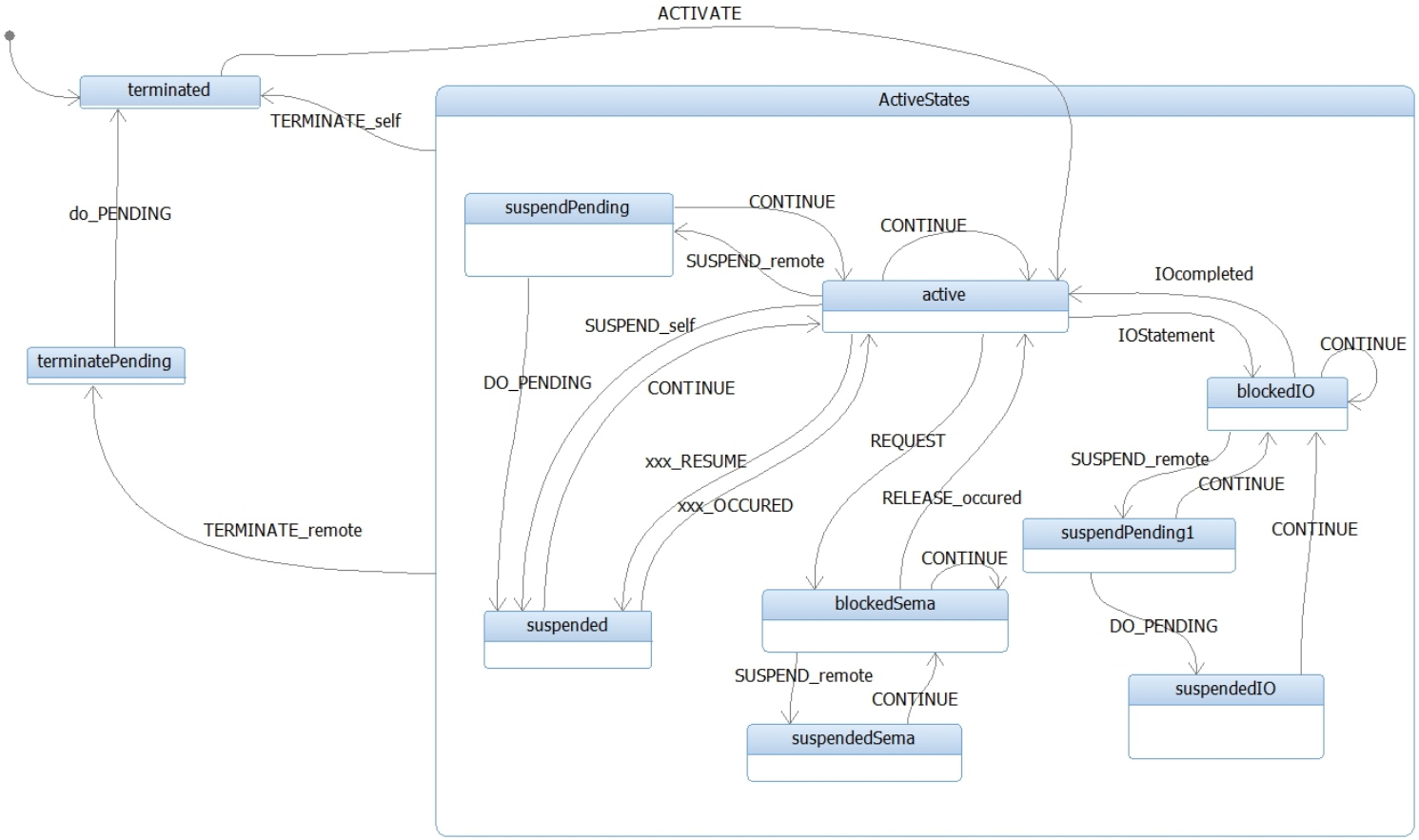
- Klasse `Task` stellt die Taskingfunktionalität bereit
- weitere C++ Klassen stellen die Funkionalitäten der Datentypen bereit
- Datentypen FIXED, BIT, FLOAT, SEMAPHORE, CHARACTER, REF CHARACTER und DATION stehen bereit
- bei Ausführungsproblemen wird eine C++ Exception augelöst und eine Log-Meldung erstellt
- es gibt eine Abbildung der *PEARL* SIGNAL-Semantik auf den C++-Exception Mechanismus
- PUT/GET unterstützt noch nicht alle Formate
- BOLT und INTERRUPT werden noch nicht unterstützt

TASK unter Linux I

- Abbildung auf pthread-Library
- SUSPEND wird von pthread unter Linux nicht unterstützt
- Abbildung von SUSPEND auf blockierenden Systemaufruf führt zu Problemen bei TERMINATE

TASK unter Linux II

- → modifiziertes Taskzustandsdiagramm



- Semantik von nicht eingetragenen Übergängen ist noch offen

Execution Timing

Task ACTIVATE auf nur einem Core

	ACTIVATE
INTEL XEON e3 / 3,5GHz / Debian 7 / 64 Bit	0.7 μs
INTEL i3 / 2.2GHz VM / Debian 7 / 32 Bit	11 μs
Raspberry Pi / ARM11 / 700MHz / Raspbian / 32 Bit	19 μs

Standard Peripherie

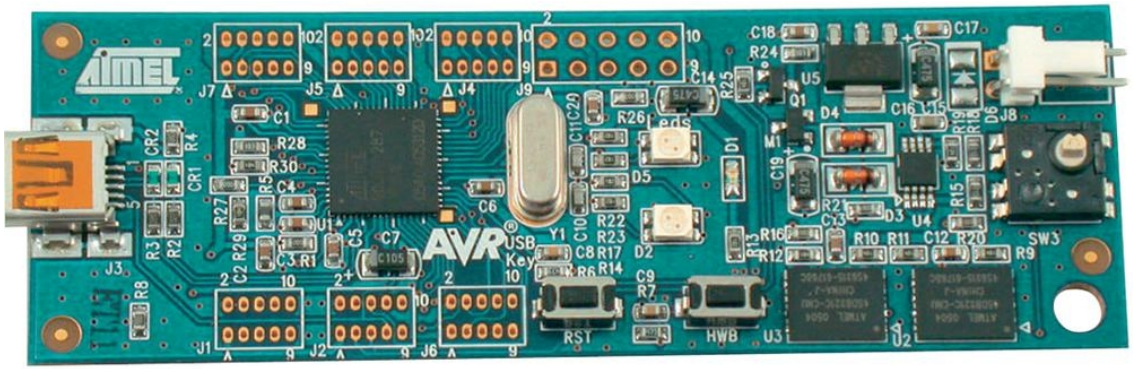
Disc('/tmp/workdir' ,10) erlaubt Dateien auf diesem Verzeichnis (mit Anzahl der Deskriptoren)

StdStream(...) ermöglicht Zugriff auf stdin, stdout und stderr

Pipe(..) erlaubt über *named pipes* die Kommunikation mit anderen Prozessen

Prozeßperipherie

- Digitale Ein- und Ausgänge über USB
  - Octopus-Board von Embedded Projects ist abgekündigt
  - Ersatz AT90USBKEY2 von ATMEL



  - Interfaceplatine mit Ein- und Ausgangebeschaltung entsteht als Semesterprojekt an der HS Furtwangen
- offene Treiberschnittstelle ist dokumentiert

Bachelor- und Masterarbeiten I

Aktuelle Arbeiten:

- FernUni Hagen:
  Semantische Analyse einer Echtzeitprogrammiersprache mittels Verhaltensmustern
- FernUni Hagen:
  SmallPEARL-Treiber für die Peripherieinfrastruktur des Raspberry Pi
- HS Furtwangen:
  Test der Programmierumgebung OpenPEARL
- FernUni Hagen:
  Portierung der experimentellen SmallPEARL-Laufzeitumgebung auf ein Echtzeitbetriebssystem

Bachelor- und Masterarbeiten II

Offene Themen:

- FernUni Hagen:
  Codegenerierung für eine Echtzeitprogrammiersprache mittels generischer Schablonen

Abgeschlossene Arbeiten:

- HS Furtwangen:
  Sprachmapping von PEARL auf Linux
- HS Furtwangen:
  Konzeption und prototypische Umsetzung eines I/O-Systems für einen PEARL-Compiler

Probleme mit der PEARL Sprachnorm I

Die Sprachnorm ist leider nicht exakt und widerspruchsfrei.
Beispiele:

Signal Handler wird mit einem impliziten RETURN (bzw. TERMINATE) abgeschlossen

- eine Prozedur mit Rückgabewert hat dann einen undefinierten Rückgabewert

$\rightarrow$ ein Signalhandler *muß* mit GOTO, RETURN oder TERMINATE abgeschlossen sein

Signals sind nicht zwingend gefordertes

- ein Nebensatz erwähnt, dass Fixed Overflow ein abfangbares SIGNAL sei
- beim Datentyp ist dies nicht explizit erwähnt

$\rightarrow$ bei allen Ausführungsproblemen werden SIGNALS erzeugt

Probleme mit der PEARL Sprachnorm II

Tasking ist nicht exakt beschrieben

- es ist festgelegt daß ACTIVATE auf eine laufende Task zu einem Fehler führt
- es ist nicht festgelegt, was bei einem TERMINATE auf eine ruhende Task passiert

$\rightarrow$ die Änderungen der Taskzustände werden komplett per State-Event-Matrix beschrieben

Probleme mit der PEARL Sprachnorm III

User Dations für Prozeß-Ein-/Ausgaben

- es ist gefordert, dass alle Ein-Ausgabe über Userdations abläuft
- Codebeispiele im Sprachreport machen dies bei Prozeß-Ein-/Ausgabe *nicht*

→ TAKE und SEND gehen direkt über das Systemgerät vom Typ `DATION ... BASIC`

DATION ... BASIC haben keinen Transfertyp

- keine Typprüfung bei TAKE und SEND möglich

→ Diskussion: Anpassung der Sprachdefinition auf einfache Datentypen und ALL

Mitarbeit erwünscht!

Offene Themen:

- Komplettierung der PEARL spezifischen Sprachkonstrukte
- Testentwicklung
- Entwicklungsumgebung
- Codeerzeugung für Mikrokontroller
- Dokumentation

Projekt-Webseite	http://smallpearl.sourceforge.net
Arbeitskreis	http://www.real-time.de/ak-compiler.html

Ansprechpartner:

AK und Übersetzer	Marcel Schaible	marcel.schaible@fernuni-hagen.de
Laufzeitsystem	Rainer Müller	mueller@hs-furtwangen.de