PEARL-News

Ausgabe 2 November 2002

Mitteilungen der GI-Fachgruppe 4.4.2 Echtzeitprogrammierung und PEARL

ISSN 1437-5966

Impressum

Herausgeber	GI-Fachgruppe 4.4.2 Echtzeitprogrammierung und PEARL URL: http://www.real-time.de
Sprecherin	Prof. Dr.-Ing. B. Vogel-Heuser Bergische Universität, Fachbereich 13 Lehrstuhl Automatisierungstechnik / Prozeßinformatik Fuhlrottstraße 10, D-42097 Wuppertal Telefon: 0202/439-3848 Telefax: 0202/429-2944 E-Mail: bvogel@uni-wuppertal.de
Stellvertreter	Dr. P. Holleczek Universität Erlangen-Nürnberg, Regionales Rechenzentrum Martensstraße 1, D-91058 Erlangen Telefon: 09131/85-27817 Telefax: 09131/30 29 41 E-Mail: holleczek@rrze.uni-erlangen.de
Redaktion	Prof. Dr. R. Müller HTWK Leipzig, Fachbereich Elektrotechnik und Informationstechnik Wächterstraße 13, D-04107 Leipzig Telefon: 0341/307-61153 Telefax: 0341/307-61159 E-Mail: mueller@fbeit.htwk-leipzig.de
	Prof. Dr. R. Müller FH Furtwangen, Fachbereich Computer- & Electrical Engineering Robert-Gerwig-Platz 1, 78120 Furtwangen Telefon: 07723/920-416 Telefax: 07723/920-610 E-Mail: mueller@fh-furtwangen.de
ISSN	1437-5966

Redaktionell abgeschlossen am 20. November 2002

Einreichung von Beiträgen
Diese Zeitschrift soll nicht nur Mitteilungsblatt sein, sondern auch eine Plattform für den Informations- und Meinungsaustausch zwischen allen an den Fragen der Echtzeitprogrammierung Interessierten bilden. Diskussionsstoff bzw. offene Fragen gibt es auf unserem Gebiet reichlich.

Wir möchten Sie, liebe Leserinnen und Leser, daher ausdrücklich ermuntern, auch in Zukunft die PEARL-News durch Ihre Beiträge mit zu gestalten.

Rainer Müller (Furtwangen)
Rolf Müller (Leipzig)

Inhalt

1 Fachgruppe 4.4.2 mit neuem Namen

Wie bereits in der Juni-Ausgabe der PEARL-News berichtet, hatte die Fachgruppenleitung auf ihrer Sitzung im Mai dieses Jahres beschlossen, den Namen der Fachgruppe zu ändern. Anlass dafür war der Umstand, dass die alte Bezeichnung eine einseitige Ausrichtung auf die Echtzeitprogrammiersprache PEARL suggerierte, die Arbeit der Fachgruppe aber das gesamte Gebiet der Echtzeitprogrammierung umfasst. Die Leitung des FA 4 der GI hat dem Änderungswunsch zugestimmt. Die Fachgruppe trägt daher seit einigen Wochen den Namen „Echtzeitprogrammierung und PEARL“. Die Überarbeitung des Logos der Fachgruppe ist noch nicht abgeschlossen, d.h. Vorschläge dafür sind weiterhin willkommen. Das neue Logo soll den Tätigkeitsbereich „Echtzeitprogrammierung und PEARL“ eindrucksvoll widerspiegeln.

2 Burns/ Wellings: “Real-Time Systems and Programming Languages“

Anlässlich des Erscheinens der dritten Auflage des bekannten Buches von Burns und Wellings schickte der Leiter des AK5, Professor Thiele, an Professor Burns eine E-Mail, die im Folgenden zusammen mit der Antwort wiedergegeben ist.

Dear Prof. Burns,

I would like to congratulate to the 3rd edition of your book "Real-Time Systems and Programming Languages"which is the mostly recommended book on the subject for my students in the lecture "Real-Time Software Design 1&2". One of the most attractive features of your and Andy Wellings book is its excellent presentation of high-level subjects in an outstanding conceptual form.
The Real-Time Language PEARL plays in my lectures the role of a conceptual basis with respect to the control-engineering abstraction level and I am very pleased with the inclusion of PEARL in your book.

With respect to future editions of your book, I would only like to recommend to reference, besides the book of Werum and Windauer, 1985, the actual German Standard PEARL 90 (DIN 66253-2, April 1998) which is available now also as a Language Report in English for international reference via the Website www.real-time.de.
Furthermore, the official writing of the abbreviation "PEARL"(from: Process and Experiment Automation Realtime Language) is in capital letters and would be preferred by us instead of Pearl.

Thanks very much in advance, yours sincerely,
Georg Thiele

Many thanks for your email and comments.
We will be more careful about PEARL in the next edition

Thanks again
Alan

– Professor Alan Burns, Head of Department
Department of Computer Science
University of York
Heslington, York, UK
YO10 5DD

email: `alan.burns@cs.york.ac.uk`
web: `http://www.cs.york.ac.uk/~burns/`

Auf die nächste Auflage dieses Buches darf man also gespannt sein; ebenso wie auf die Fertigstellung des von der Arbeitsgruppe 4.4.2 geplanten „PEARL-Buches“.

3 Realzeitprogrammierung an der FH Furtwangen

Seit dem Wintersemester 1995/96 findet ca. alle zwei Semester die Lehrveranstaltung „Realzeitprogrammierung“ an der FH Furtwangen statt. Die Veranstaltung ist in einen theoretischen Teil und einen praktischen Teil gegliedert.

Die Studierenden kommen aus den elektrotechnischen Studiengängen ($\approx 80\%$), sowie den Informatikstudiengängen. Bei stark wechselnder Belegung und beschränkter Anzahl von Praktikumsplätzen (maximal 16 Teilnehmer) haben durchschnittlich 12 Studierende die Veranstaltung belegt.

Vorkenntnisse in Systemprogrammierung oder Multitaskingprogrammierung können nicht erwartet werden. Inhalt des Theorieteils der Veranstaltung sind daher:

- Grundlagen des Taskschedulings und Auswirkungen des Schedulers auf die Deterministik einer Anwendung
- Grundlagen der Interprozesskommunikation
- Beispiele der Interprozesskommunikation in verschiedenen Programmiersprachen
- Beispiele von Programmkonstrukten zur Einhaltung von Zeitvorgaben

Die Laborausstattung umfasst 5 Modelle (Aufzug, Kugelsortieranlage, Bandwickler, Roboter und Nadeldrucker) mit Gleichstrommotoren, Schrittmotoren und Magnetaktuatoren. Allen Modellen ist gemeinsam, dass die Ansteuerelektronik die Signale des Steuerungsprogramms überwacht und bei einer zeitlichen Fehlansteuerung einen Hardwareausfall simuliert. Auf diese Weise wird die Sensibilität für die Gefahr, die von einem fehlerhaften Programm ausgehen kann, gefördert. Für alle Modelle genügen weiche Realzeitanforderungen.

Die Rechnerausstattung ist gezielt inhomogen gehalten, sodass die unterschiedlichen Modelle mit verschiedenen Systemen angesteuert werden können. Die Palette der Betriebssysteme und Programmiersprachen ist in nachstehender Tabelle dargestellt:

Betriebssystem	Programmiersprachen
Linux	PEARL, C, C++
RT-Kernel 4.5 unter MS-DOS 5.0	C, C++
QNX 4.0	C, C++

Die Erfahrung der vergangenen Semester zeigt:

- Die den Studierenden aus dem Studienalltag unbekannte Programmiersprache PEARL wird zunächst sehr misstrauisch betrachtet. Nach der Gewohnung an die strenge Typprüfung und die ungewohnte Ein-/Ausgabe sind die Studierende von den Diensten der Sprache sehr angetan.
- Das den PEARL-Kennern bekannte Problem von PEARL unter Linux[1] bereitet keine nennenswerten Schwierigkeiten.
- Die Hauptprobleme der Studierenden liegen bei der Umsetzung der Aufgabe auf parallele Prozesse und deren Koordinierung. Der Gedanke, einen Prozess gezielt zu blockieren, ist stets die größte Hürde.

Seit diesem Semester findet die Veranstaltung im Umfang von 4 SWS (früher 2 SWS) statt, sodass mehr Zeit für den praktischen Teil bleibt. Eine weitere Vertiefung dieser Ausbildungskomponenten ist derzeit aus Deputatsgründen leider nicht möglich.

Rainer Müller
mueller@fh-furtwangen.de
www.foo.fh-furtwangen.de/~mueller

[1] Preemption funktioniert nur, wenn die niederpriore Task dem PEARL-Scheduler die Kontrolle übergibt. Ein `CONTINUE;` genügt hierzu.

4 Specification PEARL Constructs for Embedded Real-Time Systems Co-Design

Abstract. *In the article a HW/SW co-design methodology is presented, which enables early reasoning about system integration as well as verification of the designs. Specification PEARL methodology is based on a specification language with the same name, whose origins are in the standard Multiprocessor PEARL language. It has been enhanced by additional components for asymmetrical multiprocessor systems design as well as by additional parameters for RTOS parameterisation and feasibility analysis. Timed State Transition Diagrams have been introduced for program/task modelling, supporting the PEARL process model. The resulting task models are easily translated to PEARL task prototypes. The methodology and its specification language components are being presented.*

Keywords: real-time systems, co-design, modelling.

4.1 Introduction

Early research in hardware and software co-design dealt more with the optimisation of the processor load by distributing it among hardware and software or the minimisation of hardware cost (e.g. COSYMA (Henkel et al., 1994), VULCAN (Gupta, 1995)). In order to reach higher quality of the designs and easier maintainability of the produced systems it was discovered that co-design methodologies should enable systematic design and some form of verification and validation thereof. Often a known hardware specification language (e.g.: VHDL, HDL) is combined with a high-level programming language (e.g.: DFL, C) for multiprocessor programming (e.g.: VULCAN, COWARE (De Man et al., 1996)). Unified programming languages were also defined to specify behaviour and structure, describing processors, peripheral devices and communication interfaces (e.g.: CHINOOK (Borriello et al., 1996)).

To enable verification, formal languages and/or mathematical notations are used, for which subsequently a proof can be worked out. Dedicated state transition automata like CRSM (Shaw, 1992) are often used as the basic internal computation model (e.g.: POLIS (Balarin et al., 1997)). On the other hand for pragmatical reasons often simulation is used to check the correctness of the designed system or parts thereof. Co-designing systems with time limitations also led to the introduction of real-time scheduling algorithms into their co-design and simulation (e.g. (Mooney, 1998)).

A good example for the wide variety of verification methods, which can be used for embedded real-time systems, is the VHDL language, for which verification methods have been devised, ranging from formal methods to co-simulation and combinations thereof (e.g.: (Khalil et al., 1998, Oppenheimer et al., 1999)).

Specification PEARL co-design methodology (Gumzej, 1999) uses unified specification and implementation languages and Timed State Transition Diagrams for program modelling. The HW/SW components' timing information is used during the co-simulation with EDF scheduling, which is meant for the verification of the produced designs.

4.2 Origins and principles of Specification PEARL

Specification PEARL is a HW/SW architecture specification and description language. It originates from Multiprocessor PEARL (DIN 66253: Part 3), which was extended in the manner, foreseen by the standard. Some constructs have been added to enable the construction of asymmetrical multiprocessor architectures. Additional parameters to the constructs have been introduced in order to enable RTOS parameterisation and support feasibility analysis of the designs.

Multiprocessor PEARL was devised as an extension of the programming language PEARL (DIN 66253: Parts 1,2) in order to explicitly support the programming of distributed systems and applications. The principles of Multiprocessor PEARL, which have been adopted in Specification PEARL, are:

- constructs for the description of hardware configurations,
- constructs for the description of software configurations,
- constructs for the specification of communication and its characteristics (peripheral and process connections, physical and logical connections, transmission protocols), as well as

- constructs for specifying both conditions and the method of carrying out dynamic reconfigurations in cases of failure.

In addition to these the proposed Specification PEARL methodology has the following characteristics, usually required for specification languages:

- abstraction (implementation details are suppressed),
- easy readability, but nevertheless precise notation,
- provision for unambiguous and complete descriptions of requirements and design,
- support for effective communication between clients, designers and users about the system to be developed,
- the possibility of extending specifications into executable prototypes easily,
- systematic integration of the specification method into the whole development process, and
- intermediate design checking as well as design verification before implementation.

In the sequel the design cycle of Specification PEARL designs is presented followed by a discussion on the chosen components and their properties.

4.3 Specification PEARL Methodology

The designer may start his/her work from either the software or hardware points of view, whichever is clearer, and join these parts later on in the design process.

HW part design usually represents an extension of the environment, being controlled, whereby first the system architecture is layed out followed by the definition of processing nodes components properties (e.g.: processor speed, memory amount, interface type and speed etc.).

SW part design is started, when the functionality of the designed system had been defined. The SW architecture is set-up. If the HW architecture is known, a SW to HW architecture mapping is usually established at this phase. Functionally complete "collections"of programs are identified for each of the different working conditions of the system. Their internal structure depends entirely on the application.

After the design is complete, it may be verified for coherency and feasibility. The program part prototype may be compiled for the specified HW configuration and evaluated by schedulability analysis. A more precise timing estimate may herewith be gained for each activity. When we are satisfied with the design, the appropriate HW components may be chosen and program prototypes enhanced to their full functionality.

As in standard PEARL a database with actual components' properties is foreseen to be used during the design process. Existing database entries may be used as well as new ones created for not yet listed components with the full set of parameters needed.

4.3.1 Hardware architecture components

The HW architecture design is done in two layers. The top layer contains components, representing the processing nodes of a multiprocessor and/or distributed system. They represent the foundation of the hardware system architecture. Because these processing nodes are named STATIONS in Specification PEARL terminology, the top layer is also named "Station layer". The components of the processing nodes (e.g.: processors, memories, interfaces, etc.) are defined on a more detailed layer, also carrying the name "Component layer".

There are four kinds of STATIONS. The BASIC station is used to represent a general purpose processing node. It may run user programs as well as the operating system (RTOS). In asymmetrical multiprocessor designs separate STATIONs may be used for the RTOS (KERNEL) and user programs (TASK). A station may also be composed of substations (multiprocessor processing nodes) by defining a COMPOSITE station - hierarchical architectures.

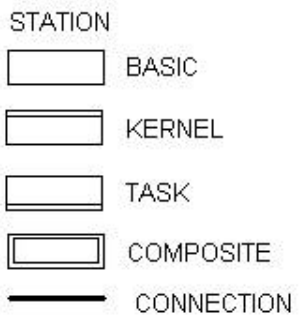

Figure 1: Station layer HW constructs

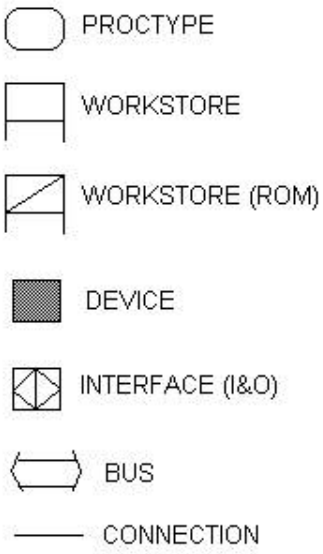

Figure 2: Component layer HW constructs

Stations communicate among each other through the established connections, which are defined on the component layer by lines between their INTERFACES. The different types of stations, which can be used on the Station layer, are listed in Figure 1.

After the top-level HW architecture design is finished, the STATION components are defined. They are chosen from Figure 2, inter-connected and described by their properties (Table 2).

4.3.2 Software architecture components

The SW architecture design is done in four layers: "Collection layer", "Module layer", "Task layer"and "TSTD layer". First COLLECTIONS, which correspond to individual STATIONs' operating states with their interconnections, are identified. TASKs, performing the COLLECTION's activities, are designed on a separate more detailed layer. They are assigned deadlines for their execution as well as trigger conditions. They are modelled by Timed State Transition Diagrams (TSTD) (Gumzej, 1999), which enable the specification of pre-, post- conditions as well as time frames of their activities.

The SW architecture of a system is composed of the components from Figure 3 with their respective properties (Table 3). The top level SW architecture consists of COLLECTIONS, with interconnections, which match those from the Station layer HW architecture. On the next sub-ordinate layer the collections' modules with their interfaces are defined. TASKS are designed on the "Task layer", being the sub-layer

STATION (general properties): Name, type (BASIC, COMPOSITE or specific), super station - name (unless the station is COMPOSITE), states of operation (NORMAL, EXCEPTION, CRITICAL,...).
KERNEL: Real-time clock resolution, scheduling strategy, MAXTASK, MAXSEMA, MAXEVENT, MAXQEVENT, MAXSCHEDEVENT.
TASK: Supervisor name (KERNEL).

Table 1: Station layer constructs' properties

PROCTYPE: Processor ID, processor speed.	**WORKSTORE:** Memory area size, access type (READ/WRITE, READ), number of wait cycles to access the area.	**BUS** Bandwidth.
DEVICE		
INTERFACE: Interface ID, driver ID or start address of the driver, single transfer unit (smallest item being transferred in a packet) or DMA, synchronisation mechanism, transfer speed.		

Table 2: Component layer constructs' properties

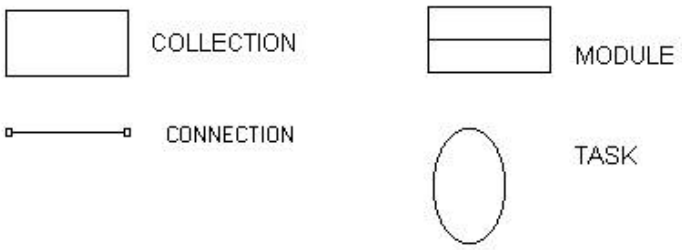

Figure 3: SW architecture constructs

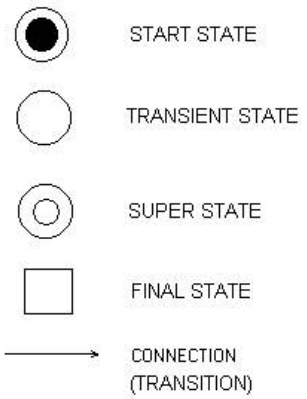

Figure 4: TSTD layer constructs (COLLECTION, MODULE and TASK layers)

COLLECTION: Station ID (residence), State ID, PORTs: Port ID, line attributes (Station layer), protocol.
MODULE: Module ID, Import and export references.
TASK: Task ID, trigger condition (on demand, timer, interrupt, signal), deadline, maximum execution time.

Table 3: Software configuration layer constructs' properties

of the corresponding "Module layer".

The interconnections between collections represent logical references to physical inter-STATION connections (through PORTS) as well as internal communication and synchronisation channels. The synchronisation and communication constructs are referenced from the TASK states through PEARL system calls instead of introducing new Task layer constructs. TASKS are represented by Timed State Transition Diagrams on their respective "TSTD (sub-)layers".

Timed STDs are based on the idea of CRSM (Shaw, 1992). They have the form of timed hierarchical state charts and consist of the following state types and properties (see Figure 4):

- start states (task trigger conditions - each start state represents one),
- transient states (continuation preconditions, timeout condition, on-timeout action, actions to be performed within the current state in the form of mini-specifications and PEARL system calls),
- super states (representing sub-charts) and
- final states (finalisation actions, return to start state-option).

4.4 Conclusion

The method for real-time systems co-design and specification Specification PEARL enables parallel design of the hardware and software parts and offers textual as well as graphical notations. For PEARL programmers it offers a good way to extend the SYSTEM part of their program for a distributed system. On the other hand it is not strictly bound to PEARL. Its output can be used as input of a configuration manager program, which loads and reloads the SW components according to the state changes of the processing nodes, being parts of the HW configuration.

Specification PEARL is more than just a specification language - it represents a methodology, composed of co-design and verification methods for design and prototyping of feasible real-time systems designs.

4.5 References

1. F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki and B. Tabarra, Hardware-Software Co-Design of Embedded Systems: The POLIS Approach, Kluwer Academic Publishers, 1997.
2. G. Borriello, P. Chou and Ross B. Ortega, "Embedded System Co-Design: Towards Portability and Rapid Integration", Hardware/Software Co-Design, Kluwer Academic Publishers, 1996, pp. 243-264.
3. R. Gumzej, Embedded System Architecture Co-Design and its Validation, Doctoral thesis, University of Maribor, Slovenia, 1999 (in Slovene; extended abstract in English).
4. R. Gupta, Co-Synthesis of Hardware and Software for Digital Embedded Systems, Kluwer Academic Publishers, 1995.
5. J. Henkel, Th. Benner, R. Ernst, W. Ye, N. Serafimov, and G. Glawe, "COSYMA: A Software-Oriented Approach to Hardware/Software Codesign", The Journal of Computer and Software Engineering, 2(3):293-314, 1994.

6. M. Khalil, Y. Le Traon and C. Robach, "Control-flow System Diagnosis: An Evolutive Method", In Proceedings of the 24th EUROMICRO Conference. Västeras, Sweden, August 1998.

7. H. De Man, I. Bolsens, B. Lin, K. Van Rompaey, S. Vercauteren and D. Verkest, "Co-Design of DSP Systems", Hardware/Software Co-Design, Kluwer Academic Publishers, 1996, pp. 75-104.

8. V. J. Mooney III, Hardware/Software Co-Design of Run-Time Sytems, Ph.D. thesis, Stanford University, USA, 1998.

9. Oppenheimer F., Schumacher G., Nebel W, "OOCOSIM - objektorientierte Spezifikation und Simulation eingebetteter Realzeitsysteme". In it+ig, Schwerpunktthema: Entwurfsmethoden für eingebettete Systeme, Vol. 2, 1999.

10. A.C. Shaw, "Communicating real-time state machines", IEEE Trans. Software Engineering, Vol. 18, No. 9, pp. 805-816, 1992.

Roman Gumzej, Matjaž Colnarič
University of Maribor
Faculty of Electrical Eng. and Comp. Sci.
Smetanova 17, 2000 Maribor, Slovenia
tel.: +386-2-220-7432 fax: +386-2-251-11-78
roman.gumzej@uni-mb.si