PEARL Rundschau

Contents

April 1982 Band 3 Nr. 1

PEARL Rundschau

April 1982
Band 3
Nr. 1

Der PEARL-Verein e.V. (PEARL-Association) hat das Ziel, die Verbreitung der Realzeitprogrammiersprache PEARL (Process and Experiment Automation Realtime Language) und ihre Anwendung sowie die Einheitlichkeit von PEARL-Programmiersystemen zu fördern.
Sitz des Vereins ist Düsseldorf. Seine Geschäftsstelle befindet sich in 7000 Stuttgart 1, Seidenstraße 36

Herstellung durch die VDI-Verlag GmbH, Düsseldorf

Die PEARL-Rundschau ist das Mitteilungsblatt des PEARL-Vereins e.V. Neben Vereinsangelegenheiten werden für alle PEARL-Interessenten Informationen über Erfahrungen, Anwendungsmöglichkeiten und Produkte gegeben.
Preis des Einzelheftes für Nichtmitglieder: DM 15,–, für Studenten DM 5,–. Jahresabonnement DM 75,–.
Zur Veröffentlichung werden sowohl fachliche Abhandlungen über Realzeit-Anwendungen, als auch Kurznachrichten zu Themen des Prozessrechnereinsatzes und der Verwendung höherer Sprachen angenommen, soweit sie für PEARL-Interessenten von Bedeutung sein können.

Beiträge können jederzeit, auch unaufgefordert, an ein Mitglied des Redaktionskollegiums geschickt werden. Autoren werden gebeten, bei der Schriftleitung ein Info-Blatt zur Manuskriptgestaltung anzufordern.

From the Editor:

This English edition of the "PEARL-Rundschau" is another first in the series of interesting developments which could be experienced by the readers of this journal, which is devoted to the programming language PEARL, its development, standardization and use. It is the attempt to answer a need which has long been expressed by our readers and the nembers of the PEARL-Association, i.e. the request for more detailed information on PEARL in English language.

Not much need to be said about the development of PEARL (=Process and Experiment Automation Realtime Language), the application oriented programming language for digital computers in measurement an control. This purpose is exactly described by the name. The language was developed in the seventies in close cooperation between industry, research laboratories and universities in Germany and has since then been inplemented and used on a broad basis. A brief description thereof has been compiled by the PEARL-Association in a information folder which is available in the following languages:

Bulgarian
Chinese
English
French
German
Italian
Japanese
Portuguese
Spanish.

While the name of the language has been derived from the English description of its purpose, the name of the journal is kind of "typically German". Literally translated could be called "PEARL-survey", but "Rundschau" has a somewhat broader meaning than that. It is to indicate that one is willing to look over the own fence, i.e. to take aspects into account which are a little bit outside of the "inner circle" of detailed knowledge about one's own area of work. Well, one should not raise to high expectations and it shall therefore be admitted that this first issue in English language does not show very much of this broad outlook. But there will be others.

For the beginning we felt that the broad availability and support of the language should be demonstrated. Therefore this issue mainly consists of translations of descriptions of PEARL translation systems. And, after all, what is a programming language without compilers? To demonstrate the power of PEARL as an application oriented development tool, one article describes some interesting applications of PEARL: a distributed computer system in a steel plant, the application of PEARL in power distribution and an industrial data communication network.

For the reader, who got interested in PEARL after having read this issue and wants more information, we have provided another little service: a list of publications on PEARL with short comments on the respective contents.

Let me close by commenting to a point of dicussion which is often raised these days. The reader may forgive that for this purpose I samewhat misuse some of the most famous words ever written:
"PEARL, or Ada, that ist NOT the question"!

PEARL is a language for the application engineer, Ada is a language for the language designer, as Grace Hopper, one of the best known pioneers of data processing, called it. And there are always many priest who make a good living out of the words of one prophet.

Last but not least I want to thank the American students who untertook the effort to translate the slang of us technicians into English and all those colleagues who did the proofreading.

I would be glad if this issue of our "Rundschau" would raise your interest in PEARL to a degree that you might want to try it.

Peter F. ELZER

Press release, concerning the foundation of the PEARL association. This public notice was sent out in February 1980 from the press office of the VDI = Verein Deutscher Ingenieure (= Association of German Engineers) to a list of more than 300 adresses.

PEARL Support Organization founded

There is now a support organization for the realtime programming language PEARL, the PEARL Association (in German: "PEARL Verein e.V.") The association will, as a guest organization of the Association of German Engineers (VDI), have its business adress in the VDI building in Düsseldorf.

The name 'PEARL' stands for 'Process and Experiment Automation Realtime Language'. It is a programming language for the application of computers and microprocessors to automation purposes.

This language was developed over the last ten years in a close cooperation of process computer manufacturers, users, and research institutes in the Federal Republic of Germany. Its main purpose ist to transform the programming of computers for automation applications from a kind of 'black magic', practiced by only a few experts, into a clear and easy-to-use technique for engineers and other interested users. The development of PEARL was therefore supported from the beginning by the German 'Bundesministerium für Bildung und Wissenschaft' (BMBW) and the 'Bundesministerium für Forschung und Technologie'(BMFT). Also worthy of special mention in this conncetion is the 'Project for Process data Processing' (PDV = Projekt Prozessdatenverarbeitung) at the Center for Nuclear Research in Karlsruhe, which has been trusted since 1972 with the management of the support of process control computer development and applications in the Federal Republic of Germany by the German 'Bundesministerium für Forschung und Technologie'. The 'Verein Deutscher Ingenieure' (VDI) has participated in the development of PEARL since 1972 through a committee of the 'VDI/VDE-Gesellschaft Meß- und Regelungstechnik'. A subset of PEARL ('Basic PEARL') was accepted by the DIN (Deutsches Institut für Normung = German Standards Institute) as a standard (DIN 66253) and has also been submitted to the International Standards Organization (ISO) for international standardization.

In order to be able to perform the necessary advisory and informative activities - especially as far as users are concerned - after the introduction of such a programming language into common use, a support organization in the legal form of a registered association ('eingetragener Verein') has been formed. Its main purposes are:

1. To further the distribution of PEARL through:
   - Response to questions about PEARL.
   - Publication of the conceptional and technical advantages of PEARL in literature and presentations at conferences.
2. To promote the use of PEARL through:
   - Collection, evaluation and distribution of user experience.
   - Encouragement of the development of programming aids.
   - Organization of user groups in cooperation with technical and scientific organizations.
   - Support of training courses.
   - Mediation of exchange of user program packages.
3. To promote the uniformity of PEARL programming systems through:
   - Support of standardization activities.
   - Distribution of test program systems.
   - Support of the cooperation at the panels and committees associated with PEARL
   - Appropriate protection of the name 'PEARL'.

The foundation of the association was actively supported by such users as e.g. energy supply companies. The 'Fraunhofer-Gesellschaft' played a prominent role in the preparatory work necessary to set up the organization.
The initial session took place at the University of Karlsruhe on Dec. 18th 1979. The participants came from many different sections of the computer manufacturing and user industry. Amongst others, the following firms were represented: AEG-Telefunken, Badenwerk, BBC-Mannheim, DEC, DORNIER-System, EWAG-Nürnberg, GEI, GPP, KRUPP-Atlas-Elektronik, MBP, MODCOMP, OBAG, Pfalzwerke, PSI, SEL, SIEMENS, Programmierbüro WERUM.

The following people were elected to the first Board of Directors of the organization:

- Prof. Dr.-Ing. R. Lauber, Chairman
  Institute for Control Engineering and
  Process Automation
  University Stuttgart
- Dipl.-Ing. G. Müller, Vice Chairman
  Brown Boveri & Cie. AG, Mannheim
  Fachbereich Netzleittechnik

- Dr.-Ing. P. Elzer
  DORNIER System GmbH
  Friedrichshafen

According to the bylaws new elections were held in December 1981 and the Board of Directors now consists of:

- Dipl.-Ing. D. Eberitzsch
  Krupp-Atlas Elektronik
  Bremen
- Prof. Dr.-Ing. L. Frevert
  Bad Salzuflen
- Prof. Dr.-Ing. R. Lauber
  Institute for Control Engineering
  and Process Automation
  University Stuttgart
- Dr.-Ing. K. Marenbach
  AEG-Telefunken
  Frankfurt
- Dr. rer. nat. H. Steusloff
  IITB
  Karlsruhe

Until the final establishment of a headquarter, information about the association, its goals, charter, organization, possibilities of participation etc. can be received either directly from the members of the Board of Directors, or through

PEARL-Verein
Geschäftsstelle Stuttgart
Seidenstr. 36
D-7000 Stuttgart 1

The PEARL Implementation of AEG-Telefunken and ATM

S. Eichentopf, Konstanz

1. Introduction

From the beginning - that is since 1969 - AEG-Telefunken has assisted with the language definition work of PEARL. As the language definition stabilized in the mid-70's after much revision, AEG-Telefunken took up implementation of the new language with increased capacity on the basis of its previous work. Changes concerning PEARL definition could easily be dealt with by chosen semiautomatic implementation process. A PEARL implementation has thus come about whose language scope is on the newest level of the official PEARL language definition /1, 2/.

In April 1980, AEG-Telefunken left its development and manufacture of process control computer activities along with the corresponding basic software to the newly-founded daughter company ATM computer GmbH. The PEARL implementation by AEG-Telefunken is therefore distributed and serviced by ATM under the name ATM 80 PEARL.

Both companies - AEG-Telefunken and ATM - are members of the PEARL Association.

2. Language Scope

The language scope of ATM PEARL was established very early in the implementation, a short time before the final boundaries of Basic PEARL /1/ were laid. In making this choice, the performance capabilities of the language and, with that, the benefits for the user stood in the foreground. The only language elements of Full PEARL /2/ that where not included were those whose implementation was possible only with disproportionately high cost, or those that could not be realized in a sufficiently efficient way with the operating system at hand.
ATM PEARL has only the following limitations with respect to Full PEARL:

- There is no operator declaration, however, the large, complete set of implemented standard operations is available.
- There is no interface declaration, though there are numerous interfaces supplied for various peripheral devices and data management.
- There are no synchronization objects of the sort BOLT, only of the sort SEMA.
- No lists of semaphores in semaphore assignments.
- No (dynamic) changes of task priorities.
- Certain limitations on array and structure displays.
- Only constants may be used as indexes of bit strings.

ATM 80 PEARL greatly exceeds Basic PEARL. The following fundamental elements of Full PEARL are contained in ATM 80 PEARL but not in Basic PEARL:

- all types of declarations on any level of nested blocks, particularly, procedures in tasks,
- arbitrary sequences of declarations,
- reference objects,
- arrays and structures as structure elements, and with that, arbitrary structure nesting,
- arbitrary and dynamic lower and upper array index bounds,
- dynamic character string length,

- type specification for the introduction of freely chooseable structure designations,
- conditional expressions,
- assignment of arrays and structures as wholes,
- assignment as the right side of an assignment (multiple assignment) and as an expression in other expressions,
- sections of arrays that contain more than only one array element,
- arbitrary sections of character and bit strings,
- arbitrary expressions in initial attributes in declarations,
- identity specifications (SPC ... IDENT ...) for the assignment of (more) access operations or designators to existing objects (not only to procedure parameters),
- arrays and structures as procedure parameters with both parameter passing mechanisms (value through INITIAL and reference through IDENT) as well as procedure results,
- procedures as procedure parameters,
- array and structure displays for direct output of array and structure values,
- all possible ways of writing BIT and CHARACTER constant notations,
- standard operators SQRT, EXP, LN, SIN, COS, TAN, ARCTAN, TANH, LWB, UPB, DUR FLOAT, FLOAT DUR, DUR/-FLOAT, DUR/DUR, REM,
- lists of schedule elements for all task operations (not only one element per task operation),
- suspension and delay statements (SUSPEND and RESUME) also for "unusual" tasks,
- interrupt statement TRIGGER for simulation of appearance of interrupts,
- standard function ORIGIN for identification of events that disable task starts,
- standard function NOW for determination of the actual time of day,
- ONEOF attribute with the transfer element type from data stations and, with that, files with data of different types,
- arbitrary expressions as arguments of input-output formats,
- system divisions in more than only one program module,
- no limitations with the syntax of the system component.

Many of these language elements not only raise the programming comfort and contribute to ease of reading and to better self-documentation of the program, but also can significantly improve program efficiency. A few examples of this:

- memory optimization through arrays whose index bounds are dynamically set according to input values with minimal increase in run-time with respect to array declaration,
- reduction of the number of additional tasks through the possibility of schedule lists at activation together with the standard operation ORIGIN for determination of which schedule element led to activation of the task,
- assignment of structures and arrays as wholes instead of element by element:

```
DCL (AR1, AR2)    (50) FIXED;
DCL (ST1, ST2)
    STRUCT [ E1 TYP1, E2 TYP2, E3 TYP3];
```

/ FULL PEARL /	/ BASIC PEARL /
AR1 : = AR2 ;	FOR I = 1 TO 50 REPEAT AR1 (I): = AR2 (I); END
ST1: = ST2;	ST1.E1 : = ST2.E1; ST1.E2 : = ST2.E2; ST1.E3 : = ST2.E3;

- avoidance of index transformations through arbitrary lower and upper bounds that can be negative,
- reduction of the number of address

calculations with help of identity specifications (SPC ... IDENT....), for example with arrays of structures:

```
TYPE STR  STRUCT [E1 TYP1, E2 TYP2,
E3 TYP3] ;
DCL A (12,10) STR;
DCL   (I,J) FIXED;

/FULL PEARL/               /BASIC PEARL/
BEGIN
SPC AIJ STR
    IDENT (A(I,J));
AIJ. E1 : = ...            A(I,J). E1 : = ...
... AIJ. E2 ...            ... A(I,J). E2 ...
AIJ. E3 : = ...            A(I,J). E3 : = ...
END;
```

3. COMPILER

3.1 Procedure, construction

PEARL lends itself to the formulation of the compiler because it has, with its manifold·data types and modern data structures, an "algorithmic language nucleus" that has good performance capabilities. The compiler is thus written in the language it translates - a procedure that is frequently used when the language to be implemented is suitable for it. With a so-called bootstrap system, the compiler is brought into the computer on which it will run, and on which it can translate itself. Only a few small input/output routines, which must first have been written into the assembler for this procedure, remain as assembler routines in the same way as basic start-up and management programs.

Besides the management program, the compiler consists of altogether nine PEARL programs that each contain several tasks and that, controlled by the management program, are executed one after the other, whereby they communicate with one another through auxiliary memory files.

The first six of these programs make up the "general" - or machine independent part of the compiler, the other three make up the code generator. The interface between all programs of the general section of the compiler is an internal intermediate language that consists of, besides a series of lists, a sequential flow that is a simplified and more strongly modified copy of the source program to be translated.

At the interface between the compiler general section and the code generator, the copy of the source program is represented in reverse polish notation, in which array indexing, structure element selection, procedure calls, ect., are handled as special operations. This intermediate language is relatively close to machine language, but is still, for practical purposes, computer- independent.

Five of the programs in the general portion of the compiler are compiler passes in the sense that they analyze the current program in the internal intermediate language from different viewpoints, and modify it appropriately for the succeeding passes. For analysis, bottom-up parsers, or, more exactly, bounded context parsers that were optimized through measures including skillful class construction are used. Essential components of these parsers are parser tables that are automatically produced from given (context free) grammars, and with whose help analysis is carried out.

The rules of the given grammars can contain instructions for actions that are taken into the parser tables and are executed during the analysis for the manipulation of lists and for the modification of the sequential flow of the intermediate language.

The first PEARL program of the general portion of the compiler processes the global program structure, processes the block nesting of the program to be translated, and collects complete definitions of designators in a block dependent address book. At the same time, the lexical analysis runs parallel in time to the parser task, acting like a finite automata and arranging designators and simple constants into appropriate reference lists and replacing key words and special character chains by corresponding simpler symbols of the internal intermediate language.

The second program of the upper section of the compiler processes the system component.

The third program processes complete declarations and specifications with exception of expressions contained in them and completes the address book correspondingly with instructions about properties of the data objects. In addition to that, all existing indicators are, in place of their "applied appearances" and "defined appearances", replaced by appropriate references to the address book.

The fourth program of the general section of the compiler is not a pass; it tests complete type attributes for correctness and identifies type attributes that represent the same type. Also, if desired, a reference list of all PEARL data objects in the program to be translated is produced as well as output with all locations of data objects and points at which the objects are called in the source program (rows, columns).

The fifth program processes expressions and statements with the exception of input/output statements which are processed in the sixth program.

The first program of the code generator prepares for the code generation in that it assigns temporary memory space for PEARL data objects. The second program does the actual code generation, while the last program brings the code produced into linkage modules in which the still open forward address references are inserted. The code generator, then, produces linkage modules directly, without producing assembler programs first.

Besides the linkage modules of the object program, the compiler or code generator produces files that contain information about the translated program; in fact, they contain information about its structure, in particular about associations between the source program and its data objects on one hand, and between corresponding code and data addresses of the generated object program on the other. These associations are necessary for program testing at source level.

3.2 Compiler data

The compiler runs on the computers ATM 80-20/4 and ATM 80-20/5 with at least 96 K byte, but better with 128 K byte main memory capacity, and on the more advanced computer ATM-80-30, the main memory capacity of which lies between 128 K byte and 1 M byte. In each case, an auxiliary memory with freely choosable devices (magnetic discs, floppy discs) is required:

The compiler program and the compiler tables occupy about 300 K byte statically in the auxiliary memory. Dynamically, at compilation of a program, an additional 300-500 K byte of auxiliary memory is required, depending upon the program to be compiled. The maximum required amount of main memory at compilation is 70 K byte with 96 K byte computers, and, with computers with (at least) 128 K byte of main memory capacity is 83 K byte, of which 14 K byte is needed by the so-called communication sector and 69 K byte by the task running sector.

The largest PEARL source program modules capable of being translated in one piece can, with the amount of main memory given, be 2000 to 3000 lines long, depending on the source program line structure (essentially one elementary instruction per line) and other program characteristics.

If the source program is stored in a disc file, and the linkage modules generated are stored in disc files, which is normally the case, about 250 source program lines per minute are compiled, including linkage module generation but not including protocol time.

The object program generated can be executed on the computers ATM 80-20/4 and ATM 80-20/5 as well as on the ATM 80-10 and ATM 80-05/HD after loading and linking.

3.3 Compiler Restrictions

The compiler is embedded in the ATM 80 programming system and uses its file management. It is started with a simple command. The source programs to be translated can be stored in source files in the auxiliary memory where they can be processed with a text editor and management programs of the file management system in dialogue with the computer, or they can be read in from the compiler directly through punch cards.

The linkage modules generated are stored in auxiliary memory files.

At compilation, the compiler generates a protocol whose scope, disregarding an unconditionally required minimum, can be controlled through a parameter in the compiler start command. The following parts belong to the maximum protocol scope:

- format-true source listing with line numbers,
- listing of all PEARL data objects in the program with statement of the location of definition and of the complete call locations in the source program (exact location statement with lines and branches) as well as with the state of the memory of the object program generated,
- code listing in a form similar to the assembler with references to the corresponding source lines and branches,
- index of the generated linkage modules with static length and storage files,
- degree of packing or length of important compiler lists.

In addition, error messages are produced as the need arises with the exact sort (from over 200) and location of the error. Generation of error messages does not, with the exception of a few serious cases, lead to interruption of the compilation procedure, so that with only one compilation many independent errors in the source program can be identified.

4. Object Programs

4.1 Run-time Packet

The language elements of PEARL that cannot be realized simply through inline sequences of machine instructions or directly through appropriate operating system functions are realized with the help of a runtime packet. These furnish the required capabilities either alone, or by applying a suitable set of operating system functions. In particular, tasks of the PEARL program are in this way built up 1:1 from "programs" managed by the realtime operating system MARTOS-K or activities of the tasks are built up from processes through these programs.

The run-time packet is strongly modularized and the modules are reentrant. At loading and linking of a specific PEARL object program, only the modules of the run-time packet required for the program are loaded and linked, and that is done automatically from the appropriately library.
From the special requirements of the various PEARL user programs results, because of the direct memory requirements of these programs for their code and data, a memory requirement of modules of the run-time packet of ca:

- 4 K byte resident in the so-called communication sector,
- from at least 3.5 K byte to at most 19 K byte can lie in either the communication sector or the run sector for the user tasks, as well as
- at least 1 K byte to at most 3.5 K byte must remain ready for use in another run sector exclusively for the run-time packet.

The run-time packet also contains the interfaces for the peripheral devices as well as for the file management of the operating system.

4.2 Loading and Linking

The linkage modules of the object program produced by the compiler can be loaded and at the same time linked with the linkage editor-loader of the programming system, which operates in dialogue. Substantially more comfortable is, however, the use of the loader of the so-called programming system generation. It is operated with a sequence of commands that can be given in arbitrary ways, particulary in auxiliary memory files. With that, a loading/linking procedure requires between one minute and several minutes, depending upon the scope of the program (system) to be loaded.

Modules that do not consist of PEARL sources and that can be called from PEARL programs as global procedures can also be loaded and linked. In these procedures, besides the (actual) procedure parameters, complete sets of data objects of the PEARL program introduced as GLOBAL can be referred to.

4.3 Program Test

With the maintenance version MV 500 for ATM 80-30 compiled PEARL programs can be tested in the source language. The 'source relative PEARL test system' QPTS is a component of the run-time packet of the ATM 80 PEARL programming system.

The essential part of this test system is that it does not use special test variants for the program to be tested; the object programs produced by the compiler can be executed unchanged with or without the test. This was achieved through two measures: the computer produces, as mentioned above, information files on the object program separate from the linkage modules of the object program, and the test system makes modifications on the program to be tested where required. These are undone when they are no longer needed.

For operation of the test system, there are simple commands and dialogues. Both can be input with any type of device. Commands can also be supplied through command files that are made up in advance. Protocol output of the test system goes to any arbitrary protocol device.

The commands can be supplied with simple conditions on which their execution is dependent.

Specifications in the commands and dialogues that refer to the program to be tested consist of the numbers of the lines in the source listing as well as of the identifiers of PEARL data objects and tasks in the source program. In a second section, array indexing and structure element selection is also possible.

Commands can be made dependent or independent on a reached test interrupt point (breakpoint).

The essential commands and dialogues serve as aids for the insertion and deletion of test interrupt points, for the resumption of execution after a test interrupt point has been reached, and to supply information about task states, semaphores, schedules of tasks, and data values. In addition to that, complete instructions for tasks, semaphores, interrupts and signals are available as commands.

5. Documentation, Training

To the product scope of ATM 80 PEARL belongs the following documentation:

- A PEARL language description /3/ that is meant in the first place to be a supplement to the ATM 80 PEARL users handbook and is, therefore, more oriented in its presentation to the systematic presentation of PEARL than on a didatic viewpoint. This language discription is separately obtainable.
- A user's handbook in which implementation dependencies, installation and use of the compiler, treatment of the generated object programs, and the addition of non-PEARL programs is described; this user's handbook is only obtainable with delivery of the compiler;
- Documentation on the run-time packet as far as it is required by or profitable for the user.

ATM offers two-week long courses that are supplemented through practical exarcises on the computer as an introduction to PEARL.

6. Applications

ATM 80 PEARL has been delivered to many University institutes, sections of AEG Telefunken, and other firms.

ATM 80 PEARL has been installed for software production in almost ten larger projects to date. Three of these projects lie in the television and radio sector, two are in the sector of radar data processing, and the rest lie in the military sector. Further projects are under consideration.

One application of ATM 80 PEARL is, as mentioned, the ATM 80 PEARL compiler itself.

For the first time with the development version then available, a model of a flexible manufacturing system controlled by PEARL programs was implemented for INTERKAMA '77 by coworkers of Dr. Jünemann (University of Dortmund). It actually dealt with only a model, but with a non-trivial one (for example, 80 digital inputs and outputs, 40 tasks). An article over this model project is contained in /4/.

References to literature

/1/ Preliminary Standard DIN 66 253
Part 1
Information Processing
Programming Language PEARL
Basic PEARL
Beuth Verlag Berlin, Köln 1981

/2/ Draft Standard DIN 66 253
Part 2
Information Processing
Programming Language PEARL
Full PEARL
Beuth Verlag Berlin, Köln 1980

/3/ PEARL for ATM 80 computer systems
Language Description
ATM Computer GmbH Konstanz, München 1980

/4/ Periodical "Datenverarbeitung AEG TELEFUNKEN"
Nr. 2/3, 1977

Author's Address

S. Eichentopf
ATM Computer GmbH
Bücklestr. 1-5
775o Konstanz

The BBC PEARL Subset PAS2

Dr.-Ing. B. Krüger, Dipl.-Ing. Müller

1. Scope of Language

The BBC PEARL subset PAS2 is a comprehensive subset of PEARL 73. The selection of language elements was substantially influenced by process automation projects that reach from the smallest applications to complex multi-computer systems on one hand and from process-oriented scientific and technical installations to management applications on the other.

2. Hardware Requirements

The compilation of the source programs can be done either on a host computer with a PL/I compiler (for example IBM 370), or on a target machine of the type PDP 11/34 to PDP 11/70. In the latter case, 64k words and an external memory (for example RK 05, RL 01, RK 06, RK 07) are required. (A floating point processor is not necessary.)

3. PEARL Software System

We regarded the language as an important component, but not as a component capable of standing alone in a user oriented programming system for process-control applications.

3.1 Compiler

The compiler itself is written in the high-order programming language PL/1. As shown in figure 1, it is divided into several self-contained phases.

During test and compilation of the source program about 500 different errrors can be recognized and identified in more detail. (e.g. by statement number, object name, as well as detailled supplementary reports).

During compilation the following lists are generated:

- A source program listing with supplementary references to statement numbers, block levels, levels nesting, relative addresses, as well as additional information about e.g. size of module, date of generation, and compilation time.
- A cross-reference list in which the location of declaration, the complete set of attributes, and the occurrence of each individual object in the statements are listed.
- A list of all global objects including all attributes.

Special compiler directives allow to generate optimized code either with respect to space or to time.

3.2 Linkers

Special emphasis is also laid upon early error recognition during link-time. Means to this end are e.g.:

- Test of global variables with respect to attribute equivalence and resolved references.
- Protocolling of date of generation compilation time, and name of each module to be linked.
- Sum Checks of the object code of each module.

3.3 BBC PEARL Operating System

The BBC PEARL Operating System (POS) was developed especially to support the capabilities of PEARL. The operating system including the drivers for standard and process peripherals was written in an optimized form with the assembler code of the object machine. Thus, the PEARL application programs achieve run times and reaction times that have proved very satisfactory in a multitude of time-critical projects.

For a variety of errors (ca. 240) which are detected at run time, the operating system provides messages that contain the following information:

- Task in which the error occured,
- Time at which the error occured and (as far as relevant)
- Backtrace addresses (module name and relative address) with respect to all active block levels.
- Device involved
- Further supplementary information.

The occurrence of an error - either results in a predefined action by the operating system or can be wandled by the user (ON condition). For this purpose the user is supplied with additional information at the time of the occurrence of the error, which allows him to react in a defined way in his application program.

3.4 Test system and Handling

The support system performs the following tasks:

- Setting of and inquiry after date and system time
- System initialization
- Data type specific response and modification of appropriate declared variables and arrays or array components in the source program.
- Complete reproduction of all tasking commands including the schedules of each task in the application program.
- Breakpoints for all executable statements that are either resident in main memory, or

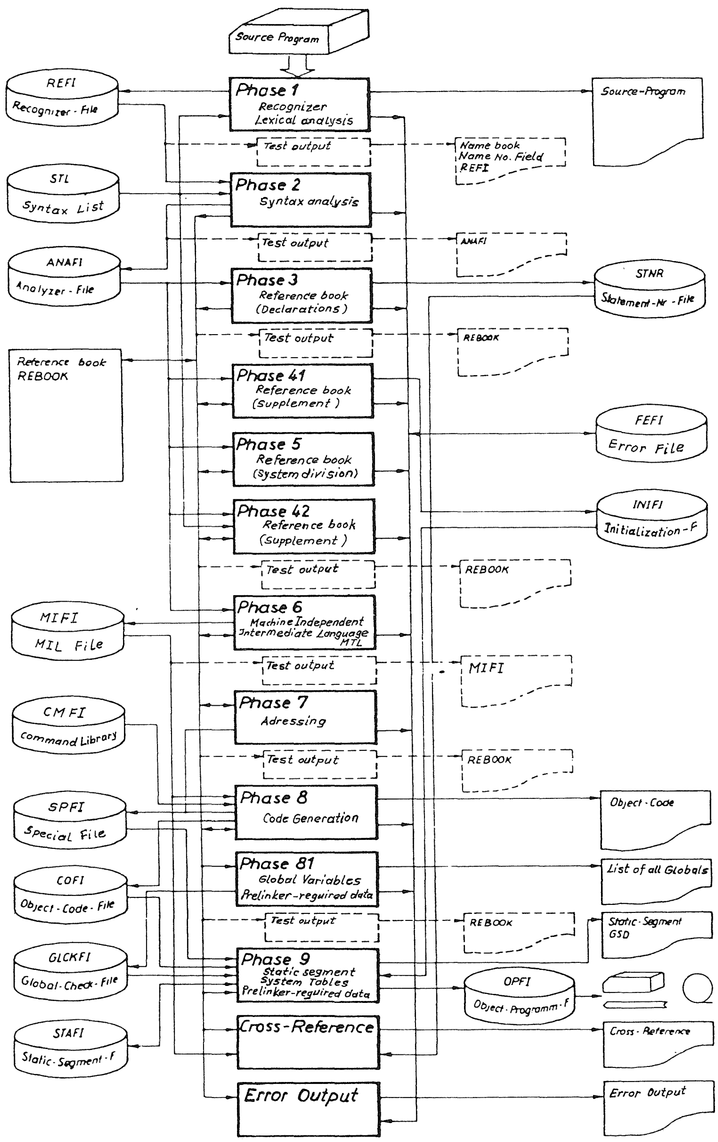


Fig. 1 Compiler Structure

loaded from external memory as 'overlay procedures'.

In addition, it can be stated whether the breakpoint shall be effective when the marked statement

- is executed for the first time or the n^{th} time
- is executed by a specific task T_m.

Moreover, at breakpoints the user can chose between a shutdown of the entire system or suspension of the task causing the breakpoint (less influence on the real time environment).

4. PEARL Standard Packages

Developed by means of the BBC PEARL programming system, a number of standard packages have been developed and implemented:

- EOS/KYBODAT (Event oriented notation for control purposes),
- IMAGODAT (Table oriented notation of partially graphic color image systems),
- SES (Standard EVU System) for control of distributed energy supply nets.

A PEARL Softwaresystem for Multi-Processor Systems

Dr. P. Elzer[1]**, Dr. H.-J. Schneider,** Friedrichshafen

Most of today's and all future systems will be processor based. There is a trend to multi-processor-systems. This ist true for all types of systems, not excluding airborne ones. Up to now the majority of these systems is programmed in assembly language, a very awkward and expensive job.

Seeing the difficulties arising from low level coding, Dornier System implemented a High-Order-Language-System based on PEARL to program Multi-Processor-Systems in an airborne or similar environment. From this environment certain conditions for the implementation resulted. It was necessary to minimize the overhead produced by the operating system. The generated code was optimized to a very high efficiency with respect to time and memory.

Originally the aim of PEARL was process-control. Due to the application area here, subsetting of PEARL was possible. This was done with high efficiency of code and a smaller modular operating system in mind.

On the other hand extensions to allow distributed processing were implemented.

The systems consists of

- Language (Subset of BASIC-PEARL)
- Compiler
- Assembler
- Linker/Loader
- Testing aids
- Special hardware for testing

It exists on a host-computer and is written in FORTRAN for portability. The target processors as implemented up to now are DORNIER DP 432, AEG 80-20 and DORNIER DP 426, which is based on an INTEL 8026.

The system was successfully used in several applications.

1)Currently with Brown Boverie & Cie., Ladenburg

1. Introduction

It is a well known fact that High-Order Languages (HOL's) are one of the most successful means to improve the productivity of programmers as well as the quality of programs. For several years, however, there was a heated discussion among experts as to whether or not this was also true for real-time and other time-critical applications, like e.g. avionics or guidance and control applications. But mostly this discussion was not very well supported by quantitative data, and it was therefore felt necessary to conduct a study (1) on the applicability of High-Order Languages to guidance and control. The task was also, to find out, which special aspects had to be taken into consideration in this - admittedly difficult - application area. The study concentrated on the Language PEARL (= Process and Experiment Realtime Automation Language), because it was the most promising candidate language in the defense environment.

The results were very encourageing. It turned out that all of the relevant problems could be formulated in the language. It was not even necessary to exploit its full descriptive power. There was one exception, however: PEARL did not contain yet all the elements necessary for the programming of distributed systems and had therefore to be slightly expanded for this purpose.

Another important result was that the efficiency of the compiler and the size of the underlying operating system were of crucial importance for the usefulness of a HOL in guidance and control applications. The reasons for this are that, in this class of applications memory, however cheap, still is subject to severe limitations like physical size, energy consumption, or weight. Dynamic efficiency of the programs is of importance, too, because guidance and control processes tend to be extremely time-critical.

It also turned out that translators for HOL's in guidance and control had to provide very elaborate test and integration aids because of the intrinsic difficulties in testing and integrating embedded computer systems.

It was therefore decided that Dornier System should develop a PEARL translation system under contract with the German MOD (BMVg) which fulfilled the following requirements:

- Extreme Efficiency of the compiled code
- Elimination of Operating System Overhead as far as possible
- Possibility to program distributed systems
- Possibility to separate code-elements in RAM from those in PROM-type memory Optional support for system integration
- Adaptability to various target processors
- Easy transportability between host-processors

It was also obvious that it would not be sufficient to just develop a compiler. It was rather necessary to develop an entire PEARL translation system for distributed systems which consited of the following components:

- Compiler-generator
- Compiler front-end
- Code generator
- Assembler
- Library management
- Modular operating system
- Linking loader
- Test and Integration aids

The construction principles of that system, and details about its implementation have already been published several times (3, 4, 5, 6).

2. The Language PEARL

The development and the properties of PEARL have also already been rather broadly published, e.g. in (7, 8). For the purposes of this paper it is therefore sufficient to concentrate on the properties of the implementation by DORNIER-Systems.

3. The PEARL-Implementation by Dornier System

As already mentioned above, the characteristics of the PEARL-implementation by Dornier System are mainly dictated by the requirements of its application area. They are most obviously reflected in the choice of the implemented language subset.

3.1 The Language Subset

For the reasons mentioned above, those language elements were not implemented from which it was known that they would result in poor object code efficiency or unnecessary overhead at runtime.

In particular such elements are:

- File handling (on-board computers usually are not equipped with magnetic background storage devices)
- Formatting (on board there are practically no printing devices and the few which there are, can easily be handled by stream output of character strings)
- Absolute time (time is usually counted relative to 'mission start')
- Signals (exception handling is a source of huge overhead and it is mandatory that unplanned software conditions do not occur during the operational phase of a system)
- Structures (Application studies showed that measurement data are usually of homogeneous type).

On the other hand certain extensions had to be provided for the programming of distributed systems. However, it was a strict policy to keep them very small in order not to deviate too much from the original PEARL. Another important design criterium for these multicomputer extensions was that they had to be 'strategy independent', i. e. the user should be enabled to implement whatever concept be deemed optimal for the safety - or redundancy-strategy of his application. These considerations resulted in the following extensions:

- Declaration of entities with the attribute 'NET GLOBAL' of types 'variable', 'semaphore' and 'task'. These entities are then either copied into or made known to every processor in the distributed system.
- Operations on such entities. This was achieved without additional statements or operators, just by extending the semantics of existing operations (overloading).

Besides, there is a facility for the connection to 'external' tasks or procedures, which may e.g. be written in Assembler. Last, but not least, runtime checks can be inserted on a statement-by-statement basis by means of 'check/nocheck' statements.

3.2 The Compiler Front-End and its Technology

The technology, which had to be used for the translator, was determined by the requirements of adaptability to various target processors and easy transportability with respect to the host processor. This led to the usual separation into a 'front-end' which is independent of the target machine and translates PEARL into machine-independent intermediate code.

The compiler front-end is written in FORTRAN for the following reasons:

- FORTRAN translators are available for nearly every possible host computer
- A compiler, written in FORTRAN, is much more readable and much easier to maintain than any other one which is constructed according to an elaborate bootstrapping technology.

It turned out that this decision was the right one. The front-end could be adapted to the following host-computers with an effort of a few man-days each:

DEC PDP-11/70 and 11/44
AEG-Telefunken 80-20/4
Siemens 7760
DEC PDP 10

Fig. 1 shows an overview over the structure of the entire translation system.

The intermediate representation had to be chosen according to the requirement of maximum code efficiency. Therefore it was not possible to use one of the usual virtual machine representations, because these usually do not contain any more all the information which was there in the source program and which is necessary for optimization. Besides, modern target processors usually have a more powerful instruction set than the one which happens to be implemented in a particular virtual machine architecture. This, too, leads to codeinefficiencies.

Therefore it was decided to use a completely target-independent intermediate representation, the so-called 'triple-code'. In principle it is a numeric representation of the program, where the individual operation is of the form:

operator, operand 1, operand 2

To sum up: the compiler front-end is written in FORTRAN and translates PEARL-Source programs into triple-code. It can detect approximately 200 different syntactical and semantical errors and identifies them by statement number, name of object and additional information, if necessary.

During translation the following listings can be produced on request:

- Source listing
- Cross-Reference listings for the following objects with their respective attributes (e.g. 'GLOBAL')
  - Variables
  - Tasks
  - Semaphores
  - Procedures
  - Labels
  - Dations
- Hierarchies of procedure calls
- Process hierarchy
- Synchronization structure
- Location of variables

3.3 The Code-generator

It produces symbolic assembly code with relative adresses for the target processor in question. This second intermediate layer has the disadvantage of an additional translation step, which may cost some

- Linkage of the operating system components required by the program
- Sorting of task-control-blocks and code segments

- Output of the control sequence for the linking loader

3.6 Linking-Loader

This tool performs the linkage process proper and produces absolute code. In case it cannot be taken from the vendor's software it is delivered together with the PEARL-System and is functionally integrated into the pre-linker.

3.7 Modular Operating System

This is a unique feature of the DORNIER PEARL-System. It allows efficient use of PEARL even in the smallest target configurations. This is achieved by abandoning the concept of an underlying, more or less autonomous and "monolitic" operating system. It is replaced by a set of routines which are automatically linked to the application program according to its requirements. These routines operate on task-control-blocks, time-order-blocks, etc. which are provided by the compiler. Thus it was possible to reduce the size of the operating system kernel to a mere 300 to 500 16-bit words, depending on the quality of the instruction set of the target processor. This kernel includes the following functions:

- Initialization
- Dispatcher
- An exit routine, which is executed if the system knows that there will be no task switching

time during translation, but this is more than balanced by the advantages. So, e.g. the assembler-listing provides an excellent means for final compiler testing and for easy linkage of external routines.

At the moment code-generators exist for the following target processors:

- DORNIER-MUDAS DP 432/433
- AEG-Telefunken 80-20
- DORNIER-MUDAS DP 426 (INTEL 8086-based)

3.4 Assembler

This component is necessary for the reasons given above. It is fully integrated into the translator system, bus usually adopted from the support software provided by the vendor of the target processor.

3.5 Pre-Linker

In case the linking-loader, which is provided by the vendor of the target processor, is not capable of handling the multi-module structure of PEARL-Programs, a pre-linker is provided, which performs the following-functions:

- Identification of program modules to be linked together
- Distribution of code into RAM or ROM
- Distribution of program modules over the various processors of the distributed system
- Completeness check for the definition of global entities

The following functional modules can then be added automatically according to the requirements of the application program:

- Clock-routines
- Interrupt handler
- Activation of tasks
- Task-termination (regular)
- Task-termination (irregular; by 'TERMINATE')
- Suspension of tasks
- Continuation of suspended tasks
- Deletion of a schedule ('PREVENT')
- Inter-processor communication
- User command interface
- Character I/O ('GET','PUT')
- Procedure entry/exit
- Array indexing
- Arithmetic routines for FLOAT and DURATION types
- Comparison routines for FLOAT and DURATION types
- Type conversion routines
- Standard functions (ABS, SIGN)
- Handling of runtime errors

If all operating system services are invoked, it uses up to 4 to 6 K of 16-bit words, depending on the architecture of the target processor.

3.8 Library management

In order to be able to fully exploit the possibilities of the modular structure of PEARL programs and to enable the user to expand his system-library by himself, a special library management package is provided.

It contains the following functions:

- Inclusion of a new module
- Deletion of a module
- Listing of the Directory
- Modification of module names

3.9 Test and Integration Aids

Firstly, these include all the above mentioned listings which are produced by the compiler and serve as reference-documents for the user during test and integration.

Additionally there are runtime checks,which are on request inserted into the program either by the compiler or as operating system routines. The following errors can be monitored:

- Array index overflow
- Division by zero
- Range Violation
- Conversion errors

These runtime checks can be enabled or disabled by the 'check/nocheck' feature.

Furthermore, several trace-routines can be built into the code:

- Jump trace
- Subroutine trace
- Call trace
- Task trace

Another important component is the debugger, which can be loaded together with the object program. It supports the following test functions:

- Activation and continuation of tasks
- Set and reset of breakpoints
- Output of environment information at breakpoints
- Input and display of values of variables
- Exit from Debugger (and return to normal execution of the program)

The design of this debugger allows for two modes of operation:

- Debugging on assembler level
- Debugging on source level

The first mode has already been implemented, the second one is being designed.

4. Application of the System

This PEARL Translator system has already been successfully used in several applications. Two of them are completed:

- A training simulator for the anti-aircraft tank 'Roland' (with 6 physically distributed processors)
- A gust alleviation system for a light aircraft

In both projects PEARL proved highly successful and the translator system fulfilled the expectations.

5. References

[1] S c h n e i d e r, H.-J.: Modulare Software für Flugfuehrung (Modular Software for Guidance and Control), Dornier System, Report, June 1978.

[2] DIN 66253, Part 1, preliminary standard Programmiersprache PEARL, Basic PEARL Beuth Verlag GmbH, Berlin, Köln, 1981

[3] S c h n e i d e r, H.-J.: PEARL-Softwaresystem für gekoppelte Klein- und Mikrorechner (PEARL-Software System for distributed Mini- and Microcomputers); PEARL-Rundschau, Vol. 1, No. 4, Dec. 1980 (pp 3-5).

[4] A m a n n, M.: PEARL für verteilte Systeme (PEARL for distributed Systems), Informatik-Fachberichte 39, 1981, Springer-Verlag (pp 399-403).

[5] G r a f, F.: PEARL für Mikrocomputer (PEARL for microcomputers), Informatik-Fachberichte 39, 1981, Springer-Verlag (pp 413-421).

[6] A m a n n, M., E l z e r, P.: Das PEARL-Übersetzungssystem von Dornier System, Friedrichshafen (The PEARL-Translator system by Dornier Systems, Friedrichshafen), PEARL-Rundschau, Vol. 2, No. 2, March 1981.

[7] PEARL Subset for Avionic Applications; Agard Advisory Report No. 90, Annex J, (A Study of Standardization Methods for Digital Guidance and Control Systems), May 1977.

[8] M a r t i n, T.: PEARL at the Age of three; Proceedings of 4th IEEE Software Engineering Conference, Sept. 1979, (pp 106-109).

The Portable GPP PEARL System

K. Lucas, Munich

1. Implemented Language Scope

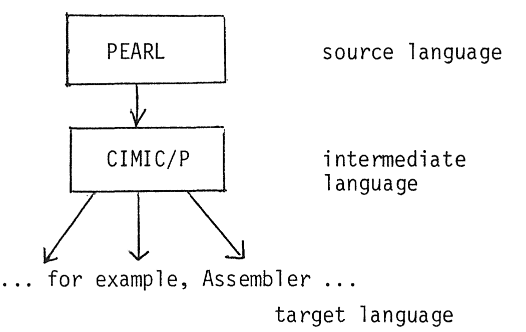
The GPP PEARL compiler implementation completely includes the proposed standard for BASIC PEARL after DIN 66253, part. 1. Moreover, the following language features, which are expected to be adapted by the PEARL Committee to the final standard of BASIC PEARL, are implemented:

- Fields and structures may be initialized (invariant fields and structures must be initialized at their declaration).
- Lists can be initialized. To this end, a 1 : 1 correspondence is used. The last constant is us ed repeatedly if necessary.
- Dynamic initialization is allowed.
- Indicators for selectors have to be unambiguous only within a structure.
- Module names are allowed.
- The declaration of a dation can be supported by an indexed, specified dation.
- In input/output statements, dations may be indexed not only with integer constants, but also with integer expressions.
- In input/output statements, data lists may contain Slices and simple expressions (for example, calls of function procedures).

2. Construction of the PEARL Compiler

The duties of a compiler can be coarsely divided into an analysis phase and a synthesis phase. The analysis phase comprises the tests for syntactic and semantic correctness of the source program. The synthesis phase deals exclusively with the translation of the source programs into the intermediate language.

Corresponding to this point of emphasis, the compiler is divided into a PEARL-oriented section - the so-called frontend and a target machine oriented section - the code generator - appropriate for the target system. As a link between these sections, an abstract PEARL machine is provided. Its assembler is the intermediate language CIMIC/P.



With this procedure, the PEARL-specific assignments are completely taken care of. The frontend is used in the same way in all compilers and is therefore very well tested.

The code generators complete the translation from the intermediate language to the appropriate target language.

The experience until now has shown that the abstract machine can be translated in simple and efficient ways into very different computer architectures.

3. Adaptation to various Process Control Computer Systems

Normally, each process control computer system contains a specially constructed process periphery for the assigned action.

In order to make it possible for the user to easily adapt the compiler to various assigned periphery systems and to the run-time functions (I/O handlers, trigonometric functions) to be used with the respective target machine, a socalled PRELUDE is provided. It contains the required information for compilation and testing of PEARL modules. For this, it deals with the specifications of:

- Devices

  Devices are provided according to their application as INTERRUPT, SIGNAL, or DATION.

  Moreover, an attribute is associated with them for testing the linking in a data way.

- Procedures

  Stated in this way, the application package requires no corresponding specification, that is, they have a "built-in" character.

- Precisions

  The functions whose applicability is implementation dependent are defined for the basic types FIXED, FLOAT, BIT and CHARACTER.

The syntax of the Prelude is oriented essentially to the PEARL notation. Consider the following example:

```
PRELUDE;
SPC
ZE 330 DATION INOUT
       BIT (16)            ROOT GLOBAL,
E 605  SIGNAL              LEAF GLOBAL,
INT INTERRUPT              LEAF GLOBAL,
NOW ENTRY RETURNS (CLOCK) GLOBAL,
LENGTH        FIXED (15),
              FLOAT (27),
              BIT   (1),
              CHAR  (1),
PREEND;
```

In this case the information of the PRELUDE is transferred to the compiler to be taken as "system defined" in the following module:

```
MODULE;
SYSTEM;
              CPU:         ZE 330;
              ZERODIVIDE:  E 605;
              HUPE:        INT*25;
PROBLEM;
              DCL F        FIXED;
MODEND;
```

4. Program Development Support

Early recognition of errors and visual documentation of the tested and compiled program reduce development costs considerably.

- All recognizable violations of BASIC PEARL after DIN 66253 part 1 are flagged.
- All error messages are placed as exactly as possible in the line concerned.
- All errors recognizable at compilation time are identified.
- An effort is made to find as many errors as possible in one compilation pass. After recognition of an error, the smallest possible error neighbourhood is isolated by further tests to prevent error propagation. Only with certain lexical or syntactic errors it is terminated after the appropriate compilation pass.

The experience to date with the GPP PEARL compiler shows that fast progress of program generation is acctieved because all statically recognizable errors are found already at compilation, not at run time.

5. Testing Support

For the dynamic test of a PEARL program, the Test and Service, System must be supplied with information about the modules that make up the PEARL program system. To this end, the compiler makes

  an association of identifiers
  from the source program to the
  memory

available.

Further, the following items in the program list corresponding to the code are marked:

- Definition blocks
  level of nesting of ranges of validity
- Statement blocks
  level of nesting of compound statements
- Flow control blocks
  statements at which the flow of control can be stopped.

These markings and the association list mentioned above allow for the test of a PEARL program system using only the source program.

6. Control of the Compiler

For control of the PEARL compiler, the following possib ilities are provided:

- Characterization of the input
  Statement of the files, channel numbers,... that are contained in the module to be compiled.
- Control of the program listing.
  The protocol can be limited to a section in a module. Lines that contain errors are printed in any case.

- Options
  A name list of all identifiers with attributes and clock membership can otionally be generated. In the same way, the marking for Test and Service Support can be shown in the protocol if desired.

7. Requirements on the Minimum Configuration

The determination of the minimum configuration of a computer system on which the compiler is still capable of running can proceed from the following requirements:

a) Auxiliary Memory

- for the input of the source program
- for the storage of the intermediate and final results produced during compilation,and
- for the storage of the BASIC PEARL Compiler in object form (for example on magnetic discs).

b) Output device for the program listing and for error messages (for example, high speed printer, teletype ...).

c) Run area in the working memory for the compiler ($\geqslant$ 20 K words of a least 16 bit length).

8. Existing Components of the Compiler System

The compiler system consists

- in essence of the translator itself, the so-called frontend of the compiler. It fullfils all language oriented translation work and compiles into the intermediate language CIMIC/P
- of a growing number of code generators. These take up the translation of CIMIC/P into the assembler of the target system in use
- of a consistency check program. This program is checking the global references of all modules of the program for completeness and compatibility in view of the PEARL semantics. The consitency check program also takes these visibilities into account which may occur in sequented programs. After this check error free PEARL-program systems can be processed further-on with the target computer dependent linker.

At present, the PEARL compiler system is available for the following equipment:

SIEMENS 330

INTERDATA 7/32

DATA GENERAL NOVA

PDP 11

VAX (in preparation)

SIEMENS 7.531

Runnable PEARL programs can be produced at present for the following target processors:

PDP-11/03/23

INTEL 8086

MICRONOVA

In general, the compiling computer (host machine) is not associated with one particular target computer. Instead, they are mutually interchangeable, that is, on each of the given compiling computers the PEARL compiling system for one of the given target computers could be installed. The actual availability is shown in the following table.

Compiling Computer (Host Machine) / Target Computer

*) = inpreparation

	PDP-11/03/23	INTEL 8086	MICO NOVA
DATA GNERAL NOVA			X
INTERDATA 7/32		X	
PDP - 11/23/34	X	X	
VAX	*)	*)	*)
SIEMENS 330	X	X	X
SIEMENS 7531	X	X	X

9. Services

Form of Delivery

The compiler system is delivered in the target code of the host computer to be used.

Training material

Training will be given as desired

User handbooks

Good documentation of the compiler systems relating to

Description, Construction,
Work methods and usage

is available.

Maintenance and Further Development

Maintenance and further development of the compiler system is assured and done by GPP, based on central procedures.

10. References to Implementations

BGT Bodenseewerk Gerätetechnik GmbH Überlingen	PEARL for flight control
Lehrstuhl für angewandte Informatik Transport und Verkehrssysteme, Universität Karlsruhe	Test and Service System for PEARL
Standard Elektrik Lorenz AG Stuttgart	PEARL for spacelab applications
VFW Vereinigte Flugtechnische Werke Fokker GmbH Bremen	PEARL for flight control

PEARL-Rundschau, Heft 1, Band 3, April 1982

The Siemens PEARL Compiler System

Dipl.-Math. H. Schoknecht, Dr. rer. nat. P. Rieder, Karlsruhe

1. Language Subset Implemented

The language subset implemented covers Basic PEARL with the following supplements, all language elements from full PEARL:

- initialization of arrays and structures
- more than three dimensions for arrays
- arbitrary lower dimension bound of arrays (also negative)
- assignements of complete arrays and structures
- multiple assignment is allowed
- operators LWB, UPB also monadic
- bit and character group selection (also on the left side of an assignment)
- with task operations schedule lists are allowed
- ACTIVATE with priority parameter
- Modulo function (REM) is implemented
- String assignments with cutting off (with output of a warning)
- TOFIXED for CHAR (2)
- TOCHAR for FIXED (7)
- B2 format is allowed
- for the following short forms, the corresponding long forms are also permitted:

CHAR	CHARACTER
DECL	DECLARE
DUR	DURATION
IDENT	IDENTICAL
INIT	INITIAL
IRPT	INTERRUPT
PRIO	PRIORITY
PROC	PROCEDURE
SPC	SPECIFY

2. Brief Description of the Compiler Technology Applied

2.1. Integration into the System Software

Considering the later introduction of the PEARL Compiler as a product, from the beginning importance was attached to its full integration into the line of products of the development system for the 300/16 bit computer family (/1/, /2/, /3/). This lead to the following objectives:

- Programming of the PEARL compiler in the available programming languages, the ASS 300 assembly language and the MECO 300 syntax analysis language, which the maintenance department is already well aquainted with.
- Generation of the object code GS 300, the representation of the machine language of the 300/16 bit computer family, suitable for further processing through linkage editors and loaders.
- Compilation of the modules from source language libraries into object code libraries conform with the module and library structure required by the available utility programs of the 300/16 bit computer family.
- Possibility of linking the modules produced by the PEARL compiler to other GS 300 modules.
- Executability of the compiler as a background program in a limited partition of 17 Kwords (16 bit word length).
- Executability of the compiler and of the programs compiled by it under the control of the ORG 300 standard operating system.

2.2. Promotion of the Acceptance by the Users

Important criteria for the compiler design also resulted form the objective to promote a positive user attitude towards the new product by means of an efficient implementation. This lead to the following demands:

- high compilation speed despite of partition limitations.
- detailed and exact compiler messages, informing on the type and the location of the error discovered.
- high efficiency of the object code generated by means of full use of the instruction set of the 300/16 bit computer family without an additional optimization pass.
- integration of high performance runtime test aids into the PEARL compiler system.
- error recognition, if possible at compile time (thorough type testing)
- differentiated processing of inner and outer events (signals, alarms).

2.3. Characteristics of the Realization

The Siemens PEARL Compiler PC 30 was realized as an 8 pass compiler (see Fig.1)

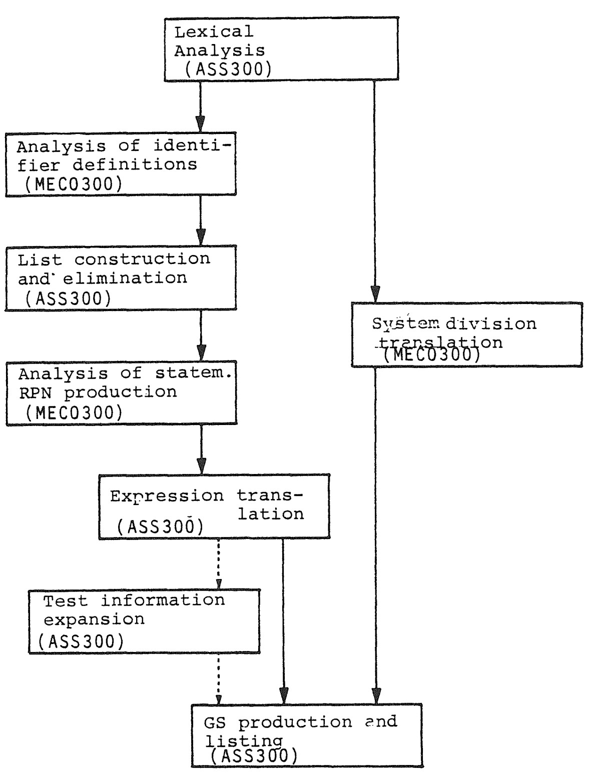


Fig. 1: Passes of the PEARL compiler

and contains, in short, the following characteristics (/4/, /5/, /12/)

- 17 Kwords partition
- lexical analysis by a finite automaton
- division of the syntactic and semantic analysis into four passes: processing of the system division declarations, statements and identifier elimination. The interface between the passes is a common intermediate language, requiring few additional tables. The syntax analysis is implemented according to the syntactic functions method (Glennie syntax machine). Dead ends are avoided by additional bottom-up elements.
- code generation is implemented according to the principle of a stack machine whose first two elements are kept in two register sets. (On principle, the right operand is evaluated before the left one).
- the last pass performs the output of the object code and the listing. If errors are discovered, they are reported in form of a detailed message where they occurred.
- the PEARL source language is directly converted into the object code (300-700 source language lines per minute).

3. Existing Components of the Siemens PEARL Compiler System

3.1. Structure of the Siemens PEARL Compiler system

The structure of the compiler system is to meet the requirements of a high compilation speed and a simultaneous limitation of the available partition. Therefore, the compiler was conceived from the beginning as a multiple pass compiler. Three passes are used for compiling the system division, six for the problem division to output machine code of the 300/16 bit computer family. In the test mode of the compiler, a seventh pass, preparing the test aid information, is executed on compilation of the problem division. The first and the last compilation steps are identical for system division and problem division. Therefore, the compilation is performed in 8 passes (see Fig. 1, /6/, /7/).

3.2. Integration into a compilation system

To produce and execute PEARL programs, several other programs are required apart from the compiler (see Fig. 2). These programs, however, are not PEARL specific but generally applicable utility programs of the 300/16 bit computer family, namely,

- the text editor (MEDIS) for writing and correcting of PEARL source texts.
- the linkage editor (BD 30) for linking PEARL modules together and for linking them to modules of the PEARL run time system or sometimes to modules not written in PEARL, e.g. to assembler procedures.
- the loader of a (standard) operating system for loading of the GS modules produced by the compiler and the production of address references beyond module boundaries.

The entire production path can also be controlled by service masks of the TESEUS software development system.

To execute PEARL programs, the PEARL run time system is necessary apart from a suitable operating system. It contains, in form of procedures, all these instruction sequences and data which, due to their length, are not directly inserted in the code generated by the compiler, but which are addressed via procedure calls.
Most procedures of the run time system are reentrant and therefore, they must be kept in memory only once. To facilitate handling, most of them were collected in the following eight compound modules:

- elementary routines as e.g. routines for block entry, registration of a signal reaction, array addressing etc.
- mathematical functions such as sine, cosine etc.

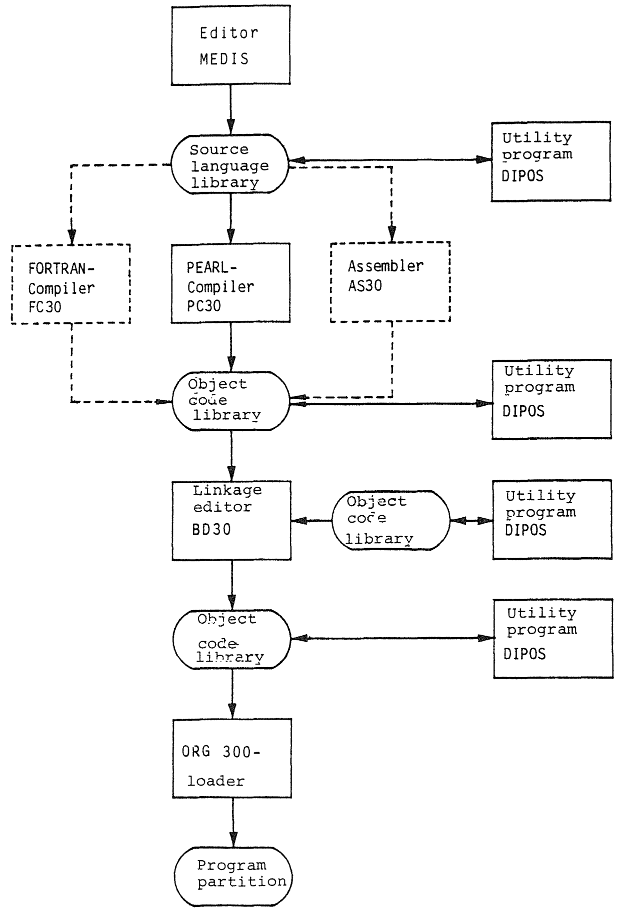


Fig. 2: Production path for PEARL programs

- kernel routines for input and output, especially in binary, unformatted form.
- routines for positioning during input and output.
- routines for realization of the GET statement
- routines for realization of the PUT statement
- routines for input and output into files.

These compound modules can be either linked to PEARL programs or loaded as common code.

Besides the compound modules, there are about 40 reentrant small driver routines which generally are only linked to the task requiring them. For the few non-reentrant run time routines (e.g. the task start routine), the only possiblity to make them available is by linking them to the calling task. For interrupt servicing about 120 standard signals divided into 4 reaction classes and 8 different error classes are at the user's disposal.

3.3. Test aids

The existing high-performance run time test aids make use of the information prepared during the test pass and perform the following dialogue-controlled functions:

- listing the numbers of the lines executed
- lines numbering for the error location in the case of a run time error
- unconditional stops at eligible line beginnings
- stops at eligible line beginnings, depending on the value of a variable
- testing and changing of the value of variables.

In the source language listing, the error messages of the compiler are output at the place where they appeared.

4. Host Computers and Target Computers

There is no distinction between host computers and target computers, since cross-compilation is not necessary.

Host computers and target computers may be:
Siemens 330, R10, R20, R30, R40

Configuration for compilation
- console device
- printer
- 17 Kwords partition for the compiler
- 195 Kwords minimum swapping area on disc

Configuration for target computers
- standard peripherals and process peripherals as applicable (in particular console device, printer, disc, graphic CRT terminal, paper tape, process signal interface).

5. Form of Delivery, Training Material, User Manuals, Maintenance Services

Form of Delivery
The PEARL compiler together with the library is delivered on a disc as a segmented program ready for loading.

Training Material
Siemens offers a two-weeks training-course on the language PEARL.

User Manuals
A user manual as well as a short description for the experienced user are available order number (P71100-D3010-X-X-35). Besides the language description these manuals include the directions for use of the compiler and a detailed error and signal description.

Maintenance Services
The customer has a 12 month warranty on the functioning of the compiler system.

6. References and Applications

The first PEARL compiler system was released in January 1978 (/8/).
Since then, three further product releases were realized. They had become necessary due to our field experience. They cover the enhancements according to the users' suggestions (/9/, /10/, /11/).

References, state 9/81
(The PC 30 has been available since 1978)

Industry:

OBAG	Regensburg
Bayer AG	Krefeld
MBB	Ottobrunn
SDR	Stuttgart
GEW	Köln
Berufsförderungswerk	Heidelberg
Battelle	Frankfurt
BWB	Eckernförde

Verbundwerke	
Haus Aden	Oberaden
DFVLR	Oberpfaffenhofen
NDR	Hamburg
EWAG	Nürnberg
Raubach & Co	Freiburg

Colleges:

Stuttgart, Berlin, Karlsruhe
Göttingen, Darmstadt, Dortmund

Internal References:

Application in several areas

7. Literature References

/1/ Rieder, P.: Effiziente PEARL-Implementierung für den PR 330, Informatik Fachberichte, Band 7, S. 173-183, Springer-Verlag, Berlin/Heidelberg/New York 1977

/2/ Degelow, L., Gottwald, H.-J., Schoknecht, H.: Stand der PEARL Entwicklung. Tagungsbericht der 8. Jahrestagung des Siemens Anwenderkreises, S. 73-89, Fachhochschule Dortmund, 1977

/3/ Schoknecht, H.: PEARL 300: Kompiliersystem und Verwendung für Echtzeitaufgaben. Tagungsbericht der 9. Jahrestagung des Siemens Anwenderkreises, Bericht KfK 2642, S.161-175, Ges. für Kernforschung mbH, Karlsruhe, 1978

/4/ Gottwald, H.-J., Schoknecht, H.: PEARL, eine leistungsfähige Echtzeit-Programmiersprache. Regelungstechnik S. 23-27 (1978)

/5/ Dorn, M., Wenzel, T.: Prozeßsprache PEARL 300 für die Siemens-Systeme 300, Siemens-Zeitschrift, Band 52, S. 23-27 (1978).

/6/ Siemens Erlangen (1980), Übersetzungsprogramme: PEARL 300. Best. Nr.: E-36/2205.

/7/ Siemens Karlsruhe (1980), PC 30/ PEARL 300: Compiler für PEARL, Programmbeschreibung. Best.Nr.: P71100-D3010-X-X-35.

/8/ Bamberger, K.-F.: Das PEARL-Kompiliersystem für die Siemens 300-16 Bit. PEARL-Rundschau, Band 1, Nr. 1, S. 49-64 (1980).

/9/ Struhulla, D.: Erfahrungen mit BASIC-PEARL im Projekt Netzleitstelle Deggendorf. PEARL-Rundschau, Band 1, Nr. 2, S. 13-22 (1980).

/10/ Weber, H.: Einsatz von PEARL bei der Software-Entwicklung für eine Fla-Schließplatz-Automatisierung. PEARL-Rundschau, Band 1, Nr. 4, S. 26-31, (1980).

/11/ Mayr, U.: Funktion und Aufgaben der Netzleitstelle Deggendorf, PEARL-Rundschau, Band 1, Nr. 2, S. 3-12, (1980).

/12/ Rieder, P.: Implementierung eines PEARL-Compilers, PDV Entwicklungsberichte E 148, Ges. für Kernforschung mbH, Karlsruhe, 1980.

The Portable PEARL Programming System of WERUM

Dr. Hans Windauer

1. Implemented Language Features

The implemented subset contains Basic PEARL (DIN 66 253, Part 1) and in addition the following language features of Full PEARL (DIN 66 253, Part 2).

Data Types

- **REF**
- User defined types (**TYPE**)
- **BOLT**
- Arrays with elements of type **SEMA, BOLT, REF,** user defined **DATION, STRUCT,** user defined type (**TYPE**)
- Arrays with more than 3 dimensions
- Arrays with lower bounds < 1
- Structures with components of type array, **STRUCT, REF,** user defined type (**TYPE**)
- B2 bit strings.

Declarations, Specifications, Definitions

- Modules may be identified (e.g. **MODULE** (TEST))
- Global attribute with module identifier (e.g. ... **GLOBAL** (TEST))
- Definition of new data types (**TYPE**)
- Declaration of new operators (**OPERATOR**) with precedences (**PRECEDENCE**)
- Declarations and specifications may be made in arbitrary sequence
- Local procedures, i.e. declaration of procedures also within tasks, procedures, blocks and loops
- Objects of type **SEMA, BOLT, IRPT, SIGNAL, REF** and user defined type (**TYPE**) may be parameters of procedures
- Objects of type **REF, STRUCT** and user defined type (**TYPE**) may be results of function procedures
- Long forms of **INIT** and **IDENT** : **INITIAL** , **IDENTICAL** .

Statements

- Values of reference variables and character string slices at the left side of assignments (e.g. STRING.**CHAR** (J) := 'N' ;)
- The schedule of an activate statement may be combined with a frequence and/or **AFTER** duration
  (e.g. **WHEN** interrupt **AFTER** duration
  ALL duration **DURING** duration
  ACTIVATE task ;)
- **SUSPEND** for other tasks
- **CONTINUE** with priority change
- Lists of **SEMA** variables after **REQUEST** and **RELEASE**
- **BOLT** statements **ENTER, LEAVE, RESERVE, FREE**
- Lists of **BOLT** variables in bolt statements
- **TRIGGER** statement.

Expressions

- Slices of character strings, variable slices of strings
  (e.g. X := INPUT.**BIT** (I : I + 3) ;
  OUTPUT.**CHAR** (J) := STRING.**CHAR** (K) ;)
- Dereferenciation (**CONT**)
- Conditional expression
  (e.g. A := **IF** B < 1 **THEN** B **ELSE** C **FIN** ;)
- Monadic operators LWB and UPB.

Input / Output

- **STRUCT**, user defined type (**TYPE**) and **ALL** may be transfer item type
- Arrays of user defined data stations
- Open parameter CAN, PRM
- Close parameter CAN, PRM.

System Division

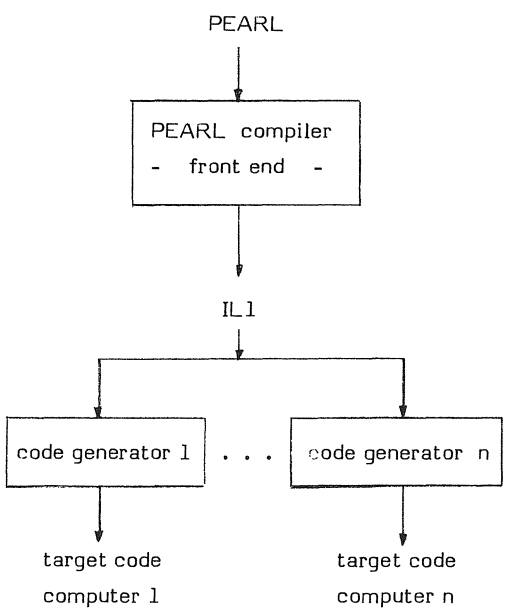
- Arbitrary sequence of connections
- Inverse notation of connections

- Identifier and / step possible after * in connection points.

The subset characterized here is implemented completely in the (portable) compiler of WERUM. There can be restrictions in the various implementations of the run time system on some target computers by reasons of size. E.g., signal handling and file handling are restricted on Siemens 404/3 (64 KB).

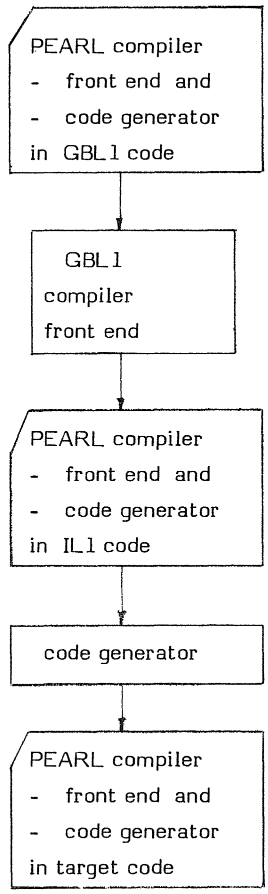
2. Characterization of the Compiler Technology

The **portable** PEARL compiler of WERUM consists of an analytic part ("front end") translating PEARL programs to the computer independent intermediate language IL1, and a code generating part ("code generator") transforming PEARL programs from their IL1 representation to target code (normally assembler or BRF, i.e. binary relocatable format). The front end is computer independent and therefore programmed only once; the computer dependent code generators have to be developed for any type of computer where PEARL programs are to be executed.



The front end and the code generators are programmed in GBL1, a proper subset of PL/I. GBL1 programs can also be translated to IL1 by a front end GBL1⟶IL1. By means of this front end and the corresponding code generator the PEARL compiler front end and the code generator itself are translated to the target code (assembler or BRF) of the target computer.

This compilation is normally performed on one of the production computers of WERUM where the GBL1 compiler is implemented (Siemens 330 and NORD 10 S).



Therefore, the PEARL compiler can be implemented on target computers not having a PL/I compiler.

Of course PEARL specific run time routines and operating system functions have to be implemented too on the target computer in order to execute PEARL programs there.

The PEARL compiler can be also used for **cross compilation :** by reason of the characterized compiler technology it can be installed on every computer having a code generator or a PL/I compiler able to translate GBL1 programs, e.g. IBM or Siemens 7.760.

In addition the PEARL compiler can be transported to FORTRAN or PASCAL computers via a transformer from IL1 to ANSI-FORTRAN or PASCAL in order to work for this FORTRAN or PASCAL computer or to be used there as cross compiler for other target computers.

3. Existing Components of the PEARL Programming System

The portable PEARL programming system of WERUM consists of the following components:

- Compiler (front end and code generator)
- Kernel of the PEARL operating system
- Run time package for binary and process I/O
- Run time package for formatted I/O
- Symbolic debug system
- Real time data base system.

Up to now the run time routines for PEARL specific arithmetics, operators for bit and character strings, procedure organisation, array handling etc. were implemented computer dependent.

In case of HP 1000 a portable PEARL specific linker was implemented by the Technical University of Berlin in order to check the interfaces between the modules of the PEARL program.

Besides this, standard components of the target computer are used.

3.1 Compiler

The PEARL compiler translates PEARL programs to assembler or BRF. Because of its modular structure it only needs a segment of 50 KB to run; therefore it can operate on small computers. In spite of this it is able to translate "arbitrary" big programs.

The handling of the various compiler parameters corresponds to the handling of the other compilers of this computer.

Beside other functions, these parameter can be used to produce listings of the source program and of the translation result. In the assembler listing or BRF listing references to the corresponding source lines are included. In addition the compiler produces a cross reference list of all objects of the program showing their source lines of definition and use.
By compiler parameter index checking and reference checking may be switched on or off.

The compiler analyses programs thoroughly and exactly. The error messages consist of a text together with a reference to the source line causing the error.

A preprocessor allows to include program pieces from text files (%INCLUDE) and to compile conditionally (%IF).

The evaluation of the system parts of PEARL programs is driven by a so-called configuration list describing all configuration possibilities of the target computer. If these possibilities are to be extended, e.g. when adding a new peripheral device, this configuration can be adapted easily by the user himself. The compiler reads the configuration dynamically for any compilation; this is necessary in case of several cross compilations for different target configurations.

On request an optimizing version of the compiler is available performing the following optimizations when setting the corresponding parameter:

- Addresses of components of structures and referenced objects are kept as long as possible in order to avoid more than one address calculation for identical objects.
- Common sub-expressions are calculated only once.
- No index calculation at run time for fixed array indices.

The compiler is programmed in GBL1, a proper PL/I subset.

3.2 Kernel of the Operating System

WERUM has developed a portable kernel of a PEARL operating system, called BAPAS-K, for the PEARL specific organisation and execution of tasks, their synchronisation and process I/O. BAPAS-K is programmed computer independently in GBL1; therefore this kernel can be transported automatically to the target computer where its open interfaces are closed by hand.

BAPAS-K can be added to an existing host operating system; it can also operate without any host operating system.

3.3 Run Time Package for Binary I/O

The run time package BAPAS-FILE contains all run time routines necessary for the PEARL specific organisation of files and execution of READ and WRITE statements. The interface of these computer independent, portable routines to the target computer consists of suitable control blocks and driver calls.

BAPAS-FILE is programmed in PEARL; therefore it can be transported automatically by the PEARL compiler to target computers.

3.4 Run Time Package for Formatted I/O

Analogously to BAPAS-FILE the run time package BAPAS-FORMEA contains all computer independent run time routines necessary to execute PUT and GET statements according to the PEARL semantics.

BAPAS-FORMEA is programmed in PEARL; therefore it can be transported automatically by the PEARL compiler to target computers.

3.5 Symbolic Debug System

The symbolic debug system allows to test interactively PEARL programs by use of PEARL like commands on host and target computers. It is programmed portable in GBL1 and PEARL.

Version 1 for Small Target Computers

The first version offers the following possibilities on PEARL level:

- Line trace
- Breakpoints at lines
- Display of values of variables.

This version is already implemented on HP 3000, NORD 10/100 and Siemens 330.

Version 2 for Medium Target and Host Computers

The second version offers the following possibilities on PEARL level:

- Line, label and call trace
- Breakpoints at
  - Lines and labels
  - Entries and exits of tasks and procedures
- Display and change of values of variables
- Display and change of states of tasks, semaphores and bolts.

This version can be installed on host target computers with 128 KB and more. When being installed on a host computer the PEARL operating system of the target computer and the time scale are simulated (by means of BAPAS-K) in order to handle tasks in the right sequence.

Version 2 is already implemented on HP 3000, NORD 10/100 and Siemens 330.

Version 3 for Host Computers

The third version has been developed for host computers. In addition to version 2 it contains the following aids:

- Simulation of the run time behaviour of the target computer on statement level
- Simulation of the I/O of the target computer by means of
  - Dialogue with the user
  - Anti tasks
  - Files with test data
- Breakpoints at
  - Time events (analog to PEARL schedules)
  - Interrupts
  - I/O statements
- Deadlock analysis
- Interrupt statements.

Version 3 is already implemented on HP 3000, NORD 10 and Siemens 330.

3.6 Open Real Time Data Base System

In order to support the use of PEARL in automation systems with data base oriented problems WERUM has developed the **open real time** data base system BAPAS-DB allowing PEARL tasks and users (via terminal) to access common data concurrently. Important features of BAPAS-DB are:

- Interactive Data Description Language DDL for the data base administrator.
- Interactive Query Language QL for users.
- Data Manipulation Language DML to access the data base in PEARL tasks independently of the chosen access strategies.
- Concurrent access by users and PEARL tasks with implicit synchronisation on record level.
- Different data sets may be accessed by different access strategies.
- Access strategies can be exchanged or added without changing the interfaces to DDL, QL and DML. (The system is open.)

By reasons of these properties BAPAS-DB can be used very flexible in automizing technical processes or it can be adapted to meet special requirements in parallel to the production of the application software.

DDL, QL and DML offer the following functions:

Data Description Language DDL

- Data Base Level
  - Creation and deletion of data bases
  - Definition, modification and deletion of access rights
  - Definition, modification and deletion of administration data
- Data Set Level
  - Creation and deletion of data sets
  - Definition of the structure of the records of a data set
  - Definition, modification and deletion of access rights
- Access Strategies
  - Introduction of new access strategies
  - Attaching access strategies to data sets.

Query Language QL

- Searching records satisfying given conditions which can be complex logical combinations of all components of the records.

In this sense BAPAS-DB is a **relational** data base system.

- Output of found records to terminal, printer or data sets.
- Update of records.
- Deletion of records.
- Insertion of new records.

Data Manipulation Language DML

- Specification of data sets of the data base.
- Searching records satisfying given conditions which can be complex logical combinations of all components of the records.
- Use and update of found records.
- Deletion of found records.
- Insertion of new records.

By standard, BAPAS-DB contains access strategies for sequential (LIFO, FIFO) and direct access (Hash, B*-Tree) together with functions for recovery and chaining data sets. It is programmed portable in GBL1 and PEARL.

Installations have been made on NORD 10, Siemens 330 and Siemens R 30.

The development of BAPAS-DB has been sponsored by the German Ministry for Research and Development within the projects PDV/PFT of Kernforschungszentrum Karlsruhe GmbH.

4. Computers where PEARL Programs Can Be Translated

The PEARL compiler is implemented on the following computers:

- Amdahl 470/6
- Hewlett-Packard HP 1000
- Hewlett-Packard HP 3000
- Norsk Data NORD 10 S and NORD 100
- Siemens 310 and 330
- Siemens R 30
- Siemens 7.760
- Siemens 404/3.

The implementation for

- Intel 8086

is in preparation. The compiler can be transported to PL/I, FORTRAN and PASCAL computers without producing a new code generator.

Number of installations: more than 25.

5. Computers where PEARL Programs Can Be Executed

The compiled PEARL programs can be executed on the following computers:

- Hewlett-Packard HP 1000
- Hewlett-Packard HP 3000
- Intel 8086
- Norsk Data NORD 10 S and NORD 100
- RDC (Really Distributed Computer Control System of the Fraunhofer Institute IITB, Karlsruhe)
- Siemens 310
- Siemens 330
- Siemens R 30
- Siemens 404/3

The following table shows the variant possibilities of installations. In this table, x means that this version is installed, and o means that this version can be installed in short time.

Compilation \ Execution	HP 1000	HP 3000	Intel 8086	NORD 10/100	RDC	Siemens 310	Siemens 330	Siemens R 30	Siemens 404/3
Siemens 404/3									x
Siemens 7.760	o	o	o	o	x	x	x	x	
Siemens R 30	o	o	x	o	x	x	x	x	
Siemens 330	x	x	x	x	x	x	x	x	x
Siemens 310					x	x			
NORD 10/100	o	o	o	x	o	o	o	o	
HP 3000	x	x	o	o	o	o	o	o	
HP 1000	x								
Amdahl 470	o	o	o	o	o	o	o	o	

6. Conditions

6.1 Form of Delivery

The programming system can be delivered partially or at whole on magnetic disk, tape or floppy disk in the form generated by the code generator, FORTRAN, PASCAL or PL/I compiler. Sources and technical documentation can be delivered on request.

6.2 Training, Documentation

The implemented PEARL subset is described in

PEARL Language Reference Manual
Reg. FB 141/8008. WERUM, Lueneburg.

This manual is also published as book:

Wulf Werum, Hans Windauer

Introduction to PEARL
Process and Experiment Automation
Realtime Language
Description with Examples

Braunschweig: Vieweg 1982.XI, 183 S.
ISBN 3-528-03590-0.

General or computer dependent user manuals and training courses are available on request.

6.3 Guarantee, Maintenance

The guarantee time is one year after acceptance.

Afterwards a maintenance contract is offered containing fast removing of errors and delivery of releases.

7. References

Amdahl 470/6 : GRS, Garching: Safety related analysis.

HP 1000 : Technical University of Berlin: Process and experiment automation.

HP 3000 : IRT, Munich: Development of software for broadcasting / television purposes.

Intel 8086 : IITB, Karlsruhe: Process control.

NORD 100 : Halden Reactor Project, Halden, Norway: Disturbance analysis system, reliability investigations.

Technical University of Braunschweig: Control of a data base machine.

NORD 10 S : WERUM, Lueneburg: Development of systems software.

RDC : IITB, Karlsruhe: Process control, programming of industrial robot systems.

Thyssen AG, Duisburg: Process control in steel factories.

Siemens 310 : IITB, Karlsruhe: Development of process control software.

Siemens 330 : IITB, Karlsruhe: Cross compilation for RDC.

WERUM, Lueneburg: Development of systems software and process control software.

Siemens R 30 : IITB, Karlsruhe: Development of process control software, implementation of a software engineering environment for process control engineers.

Siemens 404/3 : Erprobungsstelle 71 der Bundeswehr, Eckernförde.

DFVLR, Oberpfaffenhofen: Experiment automation.

Siemens 7.760 : IITB, Karlsruhe: Cross compilation for RDC.

Industrial Applications of PEARL

Dr. H. Steusloff, Karlsruhe (IITB)

Summary

The value of a programming language may only be determined by application experience. The language PEARL (Process and Experiment Automation Realtime Language) [1] has been designed to be an application programming language for all kinds of real-time systems. To show the potential of this language, four applications from different areas will be described in the following. The different applications cover industrial process control (automation of soaking pit furnaces, control of power utility network), an industrial data communication network and the on-line coordinate transformation for an industrial robot system.

1. Control of 28 Soaking Pit Furnaces by a Distributed Microcomputer system, Programmed in PEARL

One of the big German steel companies, the THYSSEN AG at Duisburg, FRG, decided to replace the analog control instrumentation of their soaking pit furnaces by a distributed computer control system. Since IITB had developed a DISTRIBUTED MICROCOMPUTER SYSTEM (RDC-System) for industrial applications, including a MULTICOMPUTER-PEARL-programming system, a cooperation was established to perform an industrial pilot installation of this system.

Soaking pit furnaces are used for the (re)heating of steel ingots to an uniform temperature, ready for milling. The furnaces are heated by gas (usually blast furnace gas) and contain up to eight ingots of about 10 tons each. Each furnace requires the control of four values:

- furnace (ingot) temperature
- furnace pressure
- percentage of oxygen in the combustion gas
- temperature of the exhaust gas

The controlled variables are the combustion air flow, the gas/air flow ratio, the exhaust gas flow and the flow of cooling air. All of these controlloops should be operated by direct digital control (DDC).

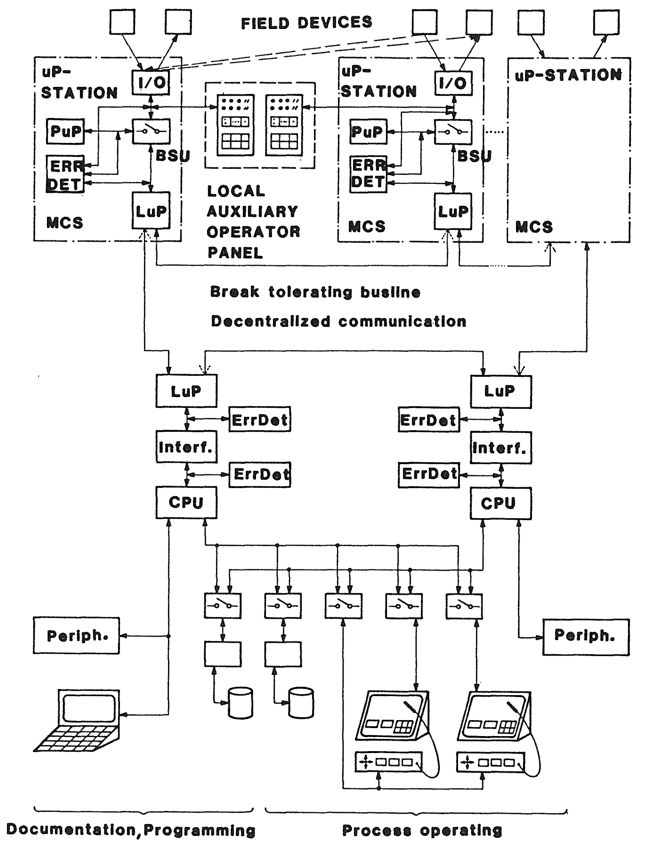
The requirements for this project were defined as follows:

- control of 28 independent furnaces
- one central control-room system
- high control accuracy required
- high availability of the control system required
- open system required
- high flexibility of furnace operation.

The following special requirements had to be met due to special properties of the furnace process:

- adaptive control needed due to
  - changes in ingot status (e.g. cold/hot)
  - changes in the heat equivalent of the heating gas
  - changes in furnace operation mode
- fast reacting control: low time constants of gas heat equivalent and pressurizing fans
- highly disturbed measured values for most process data (oxygen, pressures, gas flow) require fast digital filtering
- remote push-button control of gas and air valves via the central operator panel system, overriding the local DDC control
- safety control of furnace operation to be integrated into the DDC-system.

An outline of the distributed microcomputer system is shown in fig. 1.



Distributed Process Control Computer System RDC (IITB)

Each of the soaking pit furnaces is automated by one microprocessor station (MCS). Each MCS is equipped with a process control microprocessor (P/uP), a set of process I/O-devices, adapted to the requirements of the corresponding furnace and a communication microprocessor (L/uP). As a central unit of each MCS, fig. 1 shows an internal bus-switch with the ability of connecting or separating the three internal partial busses to the I/O-, the P/uP- and the L/uP-unit. This bus-switch unit (BSU) also contains the MCS error detection unit and switches the partial busses according to the actual error conditions. Due to these features, the RDC-system is fault tolerant, employing the principle of dynamic redundancy. To activate this redundancy in a decentralized manner, all status information is communicated to all MCS.

The necessary high transmission rate on the communication busline has been achieved by an optical fiber line with a 350 Kbits/sec transmission rate and a ring-shaped structure. Dynamic, "function-sharing" redundancy is established by this powerful communication link together with error detection equipment in each MCS, alternative data ways and over-dimensioned equipment in each MCS, as well as some spare processor-time of the micro-CPU's.

There are two special computer stations in the system. The first one (lower right hand side in fig. 1) is used for process-operating and is equipped with two color-screen-input/output-systems, employing light-pens and virtual keyboards for the command input. This system serves for operating 28 furnaces as well as for supervising the corresponding 28 distributed computer stations MCS and their communication.

On the lower left hand side, fig. 1 shows the programming and documentation system. Both systems are connected by communication links and cross-over-switches for the peripheral units. The programming system serves as dynamic redundancy for the process-operating system.

The software system within each MCS, completely written in MULTICOMPUTER-PEARL, [2], consists of a local PEARL operating system, a network operating system and the PEARL application programs. MULTICOMPUTER-PEARL especially supports the programming of distributed systems by adding structural description of hardware and program systems as well as I/O-dataways to PEARL. The application programs, comprising adaptive and fast reacting DDC (min. sampling time 100 msec), consist of 8 MODULES, containing 24 TASKS and 26 PROCEDURES. This modularization was necessary, because three programmers were working on the software project in parallel.

The structuring features of PEARL very well supported this team work. Especially the user-defined data types and the efficient access to these data via REFERENCES proved to be very valuable. The possibility of defining arrays of structures allowed a very clear and documentation supporting layout of the data base for the central control-room system. The embedded PEARL-features for I/O, tasking, scheduling and synchronization together with the MULTI-

COMPUTER-PEARL extensions, decreased the coding effort. Concerning the PEARL-application programs (36 KWords), we believe that we saved up to 40 % compared with ASSEMBLER-programming.

At present, the planned 28 soaking pit furnaces are under computer control, and we can sum up the experiences. The control of the furnaces has been improved substantially, compared with analog control; the main reasons are adaptive controllers and digital filtering of the process signals. The improved control also results in saving of heating energy. Another advantage of the new system is the availability of all furnace status information in the control-room via the color-screen-input/output-display system. This facilitates an optimal overall operation of the soaking pit furnace plant. The principal idea of the dynamic redundancy, supported by MULTICOMPUTER-PEARL, has shown its advantages by providing an overall availability of the computer system of 0,9996 during the first two years of operation (6 hours down time during more than 16,000 hours of operation).

2. Industrial Data Communication Network

Big industrial companies with spatially widespread plants have the problem of interconnecting the plant computers to each other and to the central disposition computing center. Again, THYSSEN AG decided to install a data communication network on the basis of the above mentioned RDC-system of IITB. The following requirements had to be met:

- up to 24 lines per network-node, arbitrarily assignable to net-lines and participant-lines
- different protocols on participant-lines
- participants of different "intelligence"
- bit transparent packet-switching
- packet-interleaving
- automatic routing, controlled by table-driven strategy (no automatic strategy)
- max. throughput per node: 480 KBaud
- collection and documentation of throughput data, errors, network status
- automatic downline loading of all nodes.

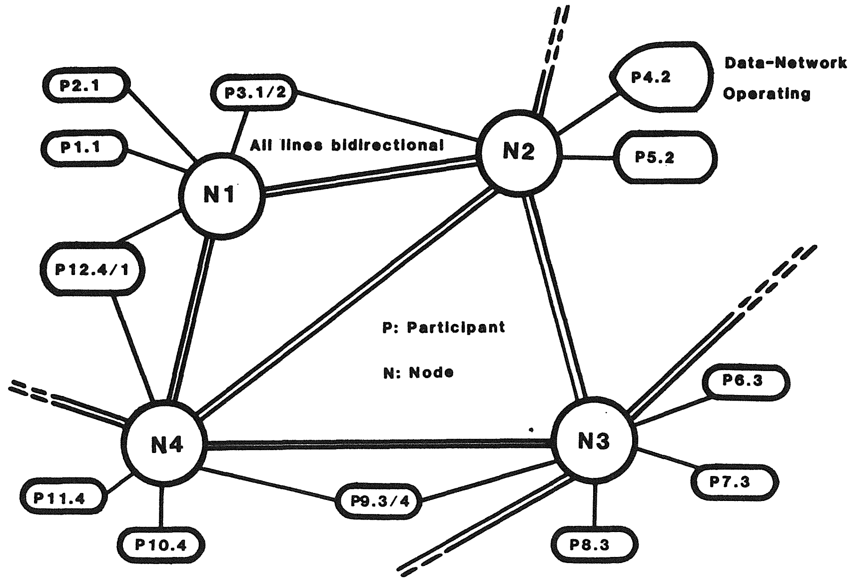
These requirements are met by the following system:

- application of packet-switching communication system, capable of being extended to meet X.25
- application of multi-microprocessor-nodes with central microcomputers for
  - routing
  - end-to-end acknowledgement control
  - intermediate package-storage
  - network supervision
- up to 24 protocol processors per node for various protocols.
- Node-software written in PEARL:
  - 16 MODULES
  - 30 TASKS
  - operating system: 4 KWords
  - packet-buffer: 1 MODULE, 2 KWords
  - application program: 24 KWords.

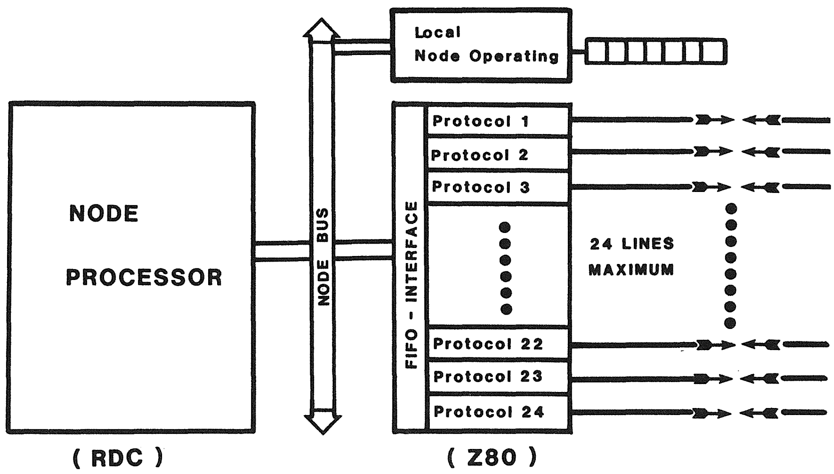
The structure of such a data communication system is shown in fig. 2. There are several nodes, arbitrarily connected by net-lines. The locally neighbored participants are connected to the nodes by participant-lines; this structure allows an application-matched layout of the network topology with respect to high availability of data ways. At present, the network contains four nodes with about 15 participants. The benefits of PEARL for programming these nodes are as follows:

- efficient buffer and queue administration by REFERENCES;
- extensive use of SEMAPHORES for the coordination of asynchronous activities; use of their multi-REQUEST/RELEASE property for the execution of queued orders;
- embedded real-time facilities of PEARL simplify the processing of asynchronous events as well as the programming of the scheduled network testing facilities;
- application-matched data structures facilitate an efficient paging mode of the packet-buffers.

Compared with ASSEMBLER-programming, we believe that we achieved savings of about 30 % during the design phase (the data base was



DATA-NETWORK FOR INDUSTRIAL PLANT COMPUTER COMMUNICATION



CONFIGURATION OF A NODE

designed using PEARL). During the coding phase we saved about 40 % compared with ASSEMBLER, due to the program structuring features of PEARL (up to 4 people programming in parallel). During the test phase we saved about 30 %, because the queue and buffer administration was nearly free of errors. Also this application, though being system-programming-oriented, showed the advantages of the language PEARL with embedded real-time facilities.

3. Control of a Power Distribution Network

OBAG is a regional power utility in eastern Bavaria, FRG. Its distribution area covers about 8.500 sq.miles. Thus OBAG is the second largest power utility in the FRG, with respect to the distribution area. OBAG operates four regional control centers for 110 kV-systems and 20 kV-systems. Each center controls approx. fourty kV-nodes and thirty 110/20 kV-substations [3] .

The control centers perform the following tasks:

- monitoring of the actual system status
- detection of all changes of the system status (messages)
- output of visual and acoustical alarms on receipt of messages
- logging and filing of all events during operation

- boundary checks of all relevant system values
- sorting of messages according to technological and ergonomic criteria
- display of information on semigraphic visual display units (VDU)
- processing and output of commands given via a custom-designed keyboard reflecting the technology of the process.

OBAG imposed the following additional requirements:

- Computer control system for the above mentioned tasks should be easily transferable to further control centers (portability!).
- Since new control centers are built only every five years, the software must be computer-independent to a very high degree, because computer technology showed to change very rapidly.
- The computer system has to maintain very high reliability.
- The software system, especially the data base, should be easily and online adaptable to changes or new installations in the power network.

The solution is a double-computer system which maintains the required high reliability. The concept of dynamic redundancy is applied: During normal system status, computer no. 1 performs all the control tasks and the dialog. During this time, computer no. 2 is used for programming, testing and if necessary, for an interactive changing of the data base and the program system. If computer no. 1 fails, computer no. 2 takes over the control tasks and the dialog. All programs are written in Basic-PEARL.

OBAG reports overall cost savings of up to 40 % compared with ASSEMBLER-programming which is of particular interest because of actually existing experienve with ASSEMBLER-programming of the same application system. The error rate during the programming process was low. At the same time it was possible to reduce the project management cost as well as the maintainance cost. One of the outstanding experiences of OBAG was the very much improved documentation of the program system, especially the documentation of the data base.

4. Coordinate Transformation for Industrial Robots

Advanced industrial robot systems utilize information from the external, Cartesian world, to online determine the actual path coordinates for their movement. This information comes from path-programming systems as well as from e.g. image-processing sensors. It is the task of a robot to move e.g. the hand center point along a desired path or to a desired point, as determined by that external information. In order to do so, a multi-axis robot needs positional information for each single axis as setpoint values for the axis controllers. Thus we have to solve the problem of transforming the external, Cartesian information (related to a x,y,z-coordinate system) into setpoint values for the robot control (fig. 3).

Depending on the desired path velocity and accuracy, path following robots need a very fast online coordinate transformation to provide the controllers with new setpoint values in time. For modern robots with path velocities of more than 50 inch per second, it is necessary to perform the coordinate transformation in less than 100 msec.

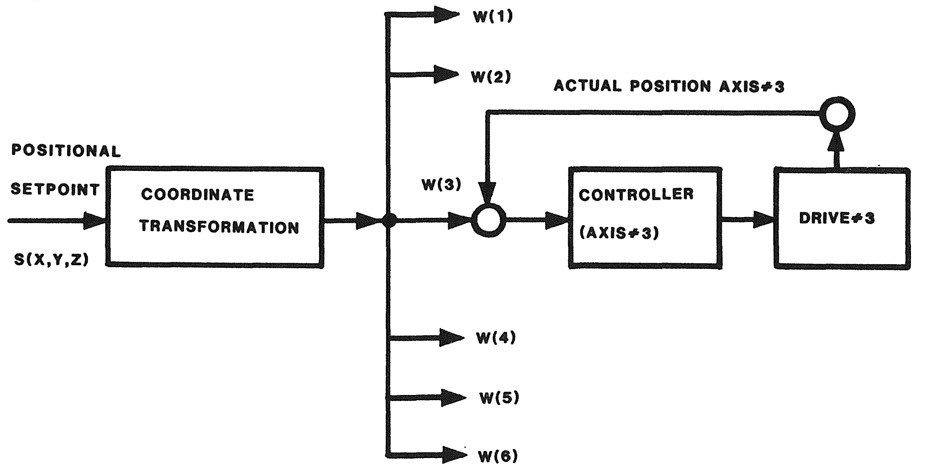
This requirement becomes ambitious, considdering the type of calculations to be performed in coordinate transformation:

$$\theta = \text{ARCSIN} \frac{\sum a\ (i)^2}{a\ (m * a\ (n)}$$

$$\text{COS}\alpha = \text{COS}\ (A + \text{ARCCOS} \frac{B\ \text{COS}\ \theta}{C \sqrt{\text{COS}\theta}}$$

$$+ \text{ARCCOS} \frac{....}{....} \quad ....)$$

The necessary operations are a sequence of transcendent functions like the ARCSIN, the ARCCOS, the ARCTAN as well as the calculation of square roots. These mathematical functions usually are derived from series calculations with floating point numbers as operands. Here we get problems with accuracy as well as with calculation speed.



Coordinate Transformation for Industrial Robots

In programming such a coordinate transformation, it would be desirable to use a higher language for two reasons: It takes some time to fit rather complicated mathematical algorithms to the special robot construction. Therefore, it should be possible to transfer the tested algorithm to other robot controls, servicing the same robot. In addition, it is not easy to program such a complicated algorithm in ASSEMBLER. On the other hand, higher level languages are thought to produce programs with low time efficiency. With its robot project IITB wanted to find out what could be achieved, using the RDC-system and PEARL.

The solution is a very standard PEARL-program for the coordinate transformation of a 5-axis-robot. The computation time for this coordinate transformation is approx. 50 msec. All the mathematical functions are calculated by series; for the sake of calculation accuracy and storage efficiency no function-tables for the trigonometric functions were used.

The comparison with an ASSEMBLER-program of equal functions shows that the run-time of the PEARL-program is about 1.8 times longer than the ASSEMBLER-program. On the other hand, the achieved 50 msec cycle-time is sufficient, and the savings in program development-time are considerable when using PEARL. Some of the reasons are, that during the design of the program the mathematical equations directly can be written in PEARL. Therefore, the coding time decreases significantly, and the test phase is more concerned with tuning parameters than with debugging. The overall savings were about 40 % by the use of PEARL for the implementation of the coordinate transformation program.

5. Conclusions

The status of PEARL, its language elements as well as numerous applications have shown that PEARL is a general purpose process automation programming language ready to use, available on the market, and well proven. So far the experience with PEARL shows that all the approached application problems have been solved efficiently, and that the language concept is able to support new scientific findings too; especially the DATION concept still will show its capabilities concerning distributed and synchronized data communication up to message systems and rendezvous-techniques. In addition, the advantages of language defined real-time features provide for a uniquely advantageous position of PEARL in the field of modern application-oriented real-time programming languages.

6. References

[1] DIN 66253, part 1 (1978). Programmiersprache (Programming language) PEARL, Basic-PEARL. Tentative Standard. Deutsches Institut für Normung (DIN), Berlin, FRG.

DIN 66253, part 2 (1980). Programmiersprache (Programming language) PEARL. Full-PEARL. Draft Standard. Deutsches Institut für Normung (DIN), Berlin, FRG.

[2] Steusloff, H.U. (1977). Zur Programmierung von räumlich verteilten, dezentralen Prozeßrechensystemen. (Programming Spatially Distributed, Decentralized Process Computing Systems). Doctoral Dissertation, Universität Karlsruhe, FRG.

[3] U. Mayr (1981). New Tools for a more Flexible Control of Power Systems with Process-Control Computers. PEARL and Interactive Systems. OBAG, Regensburg, FRG.

Literature on PEARL

Looking for literature is tedious, particularly when it isn't known where the information ist to be found, or when it isn't known whether anything at all has been published in the area of interest. The PEARL Association would therefore like to offer a running literature service to its members. This is a project that must be allowed to grow. Sources must be developed, material must be collected, and a classification system must be established. In this area, we would be happy to get your support. Perhaps you remember a few good publications that we could use about PEARL or its applications that have particular technical or historic interest. We thank you for any references you can offer.

Now for the beginning:

General Descriptions

T. Martin:
"Die Entwicklung der Realzeitprogrammiersprache PEARL im Rahmen des Projekts PDV"
KfK-Nachrichten, Volume II, 1/79, Karlsruhe
A survey paper about the development of PEARL, its basic concepts, and available compilers. Includes an illustrative application example.

T. Martin:
"PEARL at the Age of Three"
Proceedings of the 4th IEEE Conference on Software Engineering, München, 1979
IEEE Cat. No. 79 Ch 1479-5C, pp 100 ff.

T. Martin:
Experience with PEARL, in:
REAL-TIME DATA HANDLING AND PROCESS CONTROL;
H. Meyer (ed.), North Holland Publishing Co., Brussel, Luxembourg, 1980, pp 375 - 391

Gives a summary about available PEARL compilers and about a few typical applications.

T. Martin:
Realtime Programming Language PEARL; concept and characteristics
Proceedings Compsac 1978,
IEEE Cat. No. 78 CH 1338-3C, pp 301 - 307

Discusses the general concepts of PEARL and gives an application example.

T. Martin:
Die Förderung von PEARL im Projekt "Prozeß-Lenkung mit Datenverarbeitungsanlagen"
des 2. und 3. DV-Programms der Bundesregierung.
Regelungstechnik, Volume 25, No. 10/1977
Oldenbourg Verlag, München.

Describes the development of PEARL and discusses its relative position in the area of realtime languages.

T. Martin:
Industrielle Erfahrung mit der Realzeit-Programmiersprache PEARL.
Regelungstechnische Praxis, Volume 21,
No. II/1979, pp 63 - 64.

Short report of the conference
"Process control computer" of the 'VDI/VDE-GMR'
March 1979

T. Martin:
Experience with the Industrial Realtime Programming Language PEARL; (paper 7.1)
1979 Canadian Conference on Automatic Control (15 p)

A relatively detailed description of the concepts of PEARL with an application example

Textbooks:

Wolf Werum, Hans Windauer:
Introduction to PEARL
Process and Experiment Automation Realtime Language
Description with application examples
Braunschweig: Vieweg Verlag, 1982 (Engl.Edition)

This book describes the most important language elements of the realtime programming language PEARL (Process and Experiment Automation Realtime Language) and explains them with many examples. It was written primarily for users of process control computers who have already written realtime programs in a higher order language.

Axel Kappatsch, H. Mittendorf, P. Rieder:
PEARL
Systematische Darstellung für den Anwender
(Systematic Description for the Application Engineer)
With 50 figures, 26 tables and a comprehensive application example.
München: Oldenbourg Verlag, 1979

This book is aimed primarily at users who have certain familiarity with practical programming. The description of the language medium is kept informal and application-independent, although many examples are used to clarify the ideas.

The description of the process environment required in PEARL is explained by means of examples of three existing implementations. The last chapter is dedicated to a detailed example of a real application (with addition of a detailed description of the problem and the complete PEARL listing).

D. Heger, H. Steusloff, M. Syrbe:
"Echtzeitrechnersystem mit verteilten Mikroprozessoren"
(Realtime Computer System with Distributed Microprocessors)
Forschungsbericht des Bundesministeriums für Forschung und Technologie, BMFT-FBDV 79-01, April 1979

This report describes the structure of a decentralized computer system for the control of industrial processes. It comprises hardware, redundancy concepts, man-machine-communication and software support. This software support includes PEARL for multi-computer systems, dynamic loaders, processor and network operating systems.

G. Bonn, L. Lorzen:
Control, Synchronization and Communication with Parallel Processors
Mitteilungen aus dem Fraunhofer-Institut für Informations- und Datenverarbeitung, 2-80, Karlsruhe, April 1980, pp 36 - 40.

The report describes the principles of cooperation between parallel processes and examines the properties of PEARL with respect to control, synchronization and communication between parallel processes.

W. Hinderer:
Reconfiguration and Restart in Fault-Tolerant Systems

Mitteilungen aus dem Fraunhofer-Institut für Informations- und Datenverarbeitung, 2-80 Karlsruhe, April 1980.

Fault tolerance is becoming an increasingly important characteristic of complex systems. In the RDC system (Really Dristributed Control System) fault tolerance was achieved by means of dynamic redundancy obtained by distribution of functions. A fault-tolerant distributed process control system programmed with multicomputer PEARL was implemented. Amongst other ways, it is shown how the restart of such systems can be handled through a transformation of (PEARL) programs into Petri nets and through the establishment of "dynamic cuts" in these nets.

PDV Reports on PEARL

Only those reports that are neither out of print nor out of date have been listed, exept the reports marked with an asterisk which are fundaments for all proceeding PEARL Implementations.

* KFK-PDV 1,	1973	Timmesfeld, K.-H. (12 Co-Autoren) PEARL-A Proposal for a Process and Experiment Automation Realtime Language
KFK-PDV 56,	1975	SCS, Hamburg MULI - Multi Level Dialog System
KFK-PDV 75,	1976	Arbeitskreis ASME Spezifikation CIMIC/1
KFK-PDV 76,	1976	ESG, München ASME-PEARL-Subset/1

KFK-PDV 100, 1976 ASME
Programmieranleitung für das
ASME 1-PEARL-Subset

KFK-PDV 110, 1977 KfK, Karlsruhe
Tagungsband zum Aussprachetag PEARL
(2.Auflage)

*KFK-PDV 120, 1977 PEARL-Arbeitskreis
Basic PEARL Language Description

KFK-PDV 129, 1977 Martin.T.
The Development of PEARL

*KFK-PDV 130, 1977 Full PEARL - Language Description

KFK-PDV 141, 1977 Kappatsch,A.
PEARL - Survey of Language Features

KFK-PDV 155, 1978 Alt, M.
Programmpaket zum Testen von Basic
PEARL-Implementationen

KFK-PDV 164, 1978 Wiedenmann, R.
Untersuchung der Eignung der Prozeßrechnersprache PEARL zur Programmierung von Automatisierungsverfahren der zyklischen Prozeßdatenerfassung

KFK-PDV 171, 1979 Martin (Hrsg.)
Industrielle Erfahrungen mit der
Programmiersprache 'PEARL'

KFK-PDV 179, 1979 Gmeiner, Hommel (Hrsg.)
Testen und Verifizieren von
Prozeßrechnersoftware

PDV Development Notes

Only those development notes that are neither out of print nor rendered out of date by technical development are listed.

PDV-E 63 Kluttig, Alt:
Beschreibung des Macroübersetzers STAGE 2 als Hilfsmittel zur Realisierung der Prozeßrechnersprache PEARL auf dem Prozeßrechner Dietz "mincal 621"

PDV-E 64 Helfert:
ASME-PEARL/1
Implementation des PEARL-Compileroberteils auf der AEG 60-50 des IRP

PDV-E 65 Wiedenmann:
ASME-PEARL/1
Beschreibung der Schnittstelle PEARL-Compiler-Assembler (AEG 60-50)

PDV-E 66 Zeh:
ASME-PEARL/1
Beschreibung der Binder- und Organisationsprogramme im PEARL-Compilersystem des IRP

PDV-E 67 Ghassemi:
ASME-PEARL/1
Beschreibung des Codegenerators im PEARL-Compilersystem des Instituts für Regelungstechnik und Prozeßautomatisierung

PDV-E 83 Holleczek:
Das Filehandling für den "Erlanger" ASME-PEARL-Subset

PDV-E 103 Alt, Mayer, Geiger:
Testprogrammsystem für BASIS-PEARL-Implementierungen

PDV-E 104 Prester:
Die graphische Ein-/Ausgabe des Erlanger ASME-PEARL-Subsets

PDV-E 107 Winkler:
Ein Vergleich von Pascal-E mit PEARL

PDV-E 112 Brock, Kremer:
Anwendung der höheren Programmiersprache PEARL im komplexen Modell eines flexiblen Fertigungssystems

PDV-E 119 Rössler:
PEARL-Betriebssystem für den Z80

PDV-E 122 Elzer:
Das Sprachentwicklungsprojekt des US-Verteidigungsministeriums

PDV-E 125 Inderst:
PEARL-Test- und Bedien-System für die ASME

PDV-E 126 Ghassemi:
Untersuchung der Eignung der Prozeßprogrammiersprache PEARL zur Automatisierung von Folgeprozessen

PDV-E 128 Eichenauer, Lucas, Zeh:
Schlußbericht über die Entwicklung eines portablen Compiler-Oberteils für Basic PEARL nach DIN 66253E

PDV-E 131 Alt:
Programm Package for Testing Basic PEARL Implementations

PDV-E 133 Eichenauer, Henn, Lucas, Zeh:
Spezifikation der Zwischensprache CIMIC/P

PDV-E 134 Eichenauer, Henn, Lucas, Zeh:
Anpassung von CIMIC/P an Basic-PEARL

The PDV development notes and the PDV reports are available from:

Kernforschungszentrum Karlsruhe
GmbH
Projekt PDV/PFT
Postfach 3640
7500 Karlsruhe

A complete catalogue of all PDV-reports (up to Feb. 1979) is available under the number

KFK-PDV 167