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. 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,

[1] Currently with Brown Boverie & Cie., Ladenburg

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).