KRULL81

Krull, Fred N., "Experience with ILIAD: A High Level Process Control Language", General Motors Research Laboratories, (CACM, Vol. 24, No. 2, Feb. 1981, pp. 66-72) In his paper, Fred Krull describes experience in using a high-level language ILIAD to program industrial control applications. ILIAD was designed at General Motors Research Laboratories in order to provide application programmers with a well-structured language as an alternative to assembler and Fortran (two languages most frequently used in the field of industrial application control). Results of using the language and its run-time system are summarized and conclusions drawn from these results. A critical review of some of the implementation techniques that were used to develop the system is also presented. ILIAD is lexically similar to PL/I. The language contains all standard control structures, such are while- and for- loops, if-then-else and select-case statements etc. Like many languages, ILIAD has fixed and float arithmetic, bit and character string manipulation. Among other things, ILIAD makes provisions for type definitions, calls by name and value, and external compilation. Since most industrial applications involve control of concurrent industrial processes, ILIAD incorporates a number of concurrency features. An ILIAD program consists of parallel tasks, procedures and shared data. Synchronization is provided through two statements LOCK and UNLOCK to lock/unlock a data structure and thus serialize its access. Task activation/deactivation is dynamic, and so the amount of concurrency in a program can vary. A typical industrial application control program performs I/O operations on a number of devices other than the usual computer ones such as disks, printers and terminals. It is the diversity and unorthodox I/O protocols of these devices that led to introducing what Krull calls 'device declarations' into the language. A typical device declaration consists of the name of the device followed by a list of operations on it. A library of such declarations is maintained to keep track of the devices already supported. The library currently contains declarations of terminals, printers, disks, optical scanners, analog-to-digital converters, modems, sensors, and actuators. ILIAD is intended to be reliable. As a result of that, it incorporates type compatibility, disallows parameter passing by name among tasks (thus tasks can only communicate through shared storage), forces shared variables to be declared in every task that uses them, and each such declaration must indicate whether the variable may be changed (this is equivalent to Concurrent Euclid's import lists). Moreover, ILIAD has no goto, for-loops can only increment by one and each select-case must have an otherwise (default) clause. The run-time system of ILIAD is two-layered. The outer layer is the object code in the form of 'virtual machine' instructions produced by the ILIAD compiler. The inner layer consists of a real-time monitor which interprets these instructions and executes them. The reason for implementing the run-time system in such a way is to make the ILIAD compiler as application and machine independent as possible despite so many potential I/O devices. A run-time ILIAD system is typically generated in two steps. Firstly, the real-time monitor is ported onto the target computer where it (the monitor) may serve as an operating system for the bare machine, or execute under a host operating system. Secondly, a set of ILIAD programs is cross-compiled (a typical target computer is used strictly for industrial application control, and is not provided with a wide range of hardware and software facilities to support an ILIAD compiler) and loaded to the target machine. Since its birth in the late 70's, ILIAD has been used successfully in three production process control applications and one demonstration. Structured design concepts have reduced error and program development time to such an extent, that after a compilation a task would frequently execute correctly the first time. Having been designed for industrial process control applications, ILIAD seems to serve its purpose well. We hope that the trend for introducing high-level languages into this area of computer applications will continue.