Abstract

Ada is a high level language, designed for implementing large embedded systems. It became an ANSI standard in 1983 and in 1988 the Ada Joint Program Office established the ADA-9X project for the revision of the Ada standard. One of the specific areas addressed by the revision team deals with requirements posed by real-time applications and introduces a new building block for real-time Ada: the protected record. Protected records provide a low level data-oriented synchronization mechanism and are a primary support for real-time systems. They can be used to efficiently program solutions for real-time problems such as mutual exclusion, conditionally shared data, counting semaphores and signals. This paper compares two different approaches for the solution of a classical real-time problem: conditional critical regions. The first solution uses Ada83 tasking and the second one uses Ada9X protected records. Finally we will discuss the runtime support and some implementation issues for protected records on multiprocessor target systems.

1 Introduction

Ada is a modern, block structured algorithmic language designed for implementing large complex software systems. It supports real-time programming with facilities to define the synchronization and timing of parallel tasks, interrupt binding and low-level operations such as machine code insertion, unchecked conversion and representation clauses. In 1988, after some years of use, the Ada Joint Program Office established the Ada-9X project. The project goal is to revise the Ada standard in order to reflect current essential requirements with minimum problems and maximum advantage to the Ada community. The draft versions of the Ada 9X Mapping Documents, released in August 1991, give a snapshot of the work in progress and an in depth analysis of the enhancements of the language. The major change areas include real-time and parallel programming, object oriented programming and hierarchical libraries.

Although Ada83 was originally designed for real-time systems many people in the real-time community felt that special language constructs are necessary for more efficient programming. E.g. with the standard rendez-vous the calling task is always blocked until the server is finished. Moreover the rendez-vous server will take calls in FIFO order. This is often not desirable in real-time systems. On the other hand many modern real-time operating systems provide direct and fast support for mutual exclusion, non blocking communication and other system level functions. Some Ada compilation systems use a subset of such a real-time operating system for their basic runtime support (tasking, exception handling and resource management). In this case, the compilation system could also supply full access to the other features of the real-time operating system via Ada packages without violating the syntax and the semantics of Ada. Although this approach gives additional capabilities for one
A particular compilation system it leads to the use of non standard, non portable and implementation dependent systems for the support of Ada83's real-time needs. A first step towards the standardization of this support is presented in the Catalogue of Interface Features and Options (CIFO 3.0) by the Ada RunTime Environment Working Group (ARTEWG). The main objective of this document is to propose a common set of user-runtime interfaces in advance of some of the more general Ada-9X solutions. The effect of CIFO is to create an "extended runtime library" to supplement the original runtime library. The compilation system can draw the necessary routines from both libraries as directed by the user code.

In Ada-9X, the enhanced functionality is achieved by generalizing existing Ada83 features and adding new syntax elements to the language. This "core" language, which must be implemented in its entirety, will be upward compatible for most existing Ada applications. The extended features for specific application areas are defined in the Special Needs Annexes. These annexes do not define new syntax; they just provide tighter semantics for additional features by defining standards for specialized packages and implementation dependent pragmas and attributes.

2 Shared objects

2.1 Problem domain

Normally the communication between two Ada tasks is achieved through the use of parameters in the entry call. Other languages that support concurrent processing often use global shared data for this purpose. The use of shared data is normally not considered good programming practice and is not encouraged by the Ada language design. If despite this we need global shared data, their correct use requires extreme care (as in any other language) to provide the synchronization and mutual exclusion characteristics. While real-time operating systems provide semaphores to solve this problem, the Ada programmer has to realize both synchronization and mutual exclusion using the rendez-vous and some extra control task. This is a typical situation where the high level of abstraction of Ada tasking forces a programmer to use a combination of higher-level abstractions to express a lower level abstraction. This is known as "abstraction inversion" because the runtime system itself will require the use of mutual exclusion, scheduling and queuing for the implementation of the rendez-vous. Some real-time applications cannot tolerate the overhead of Ada tasking for simple resource control and need a lower-level mechanism that can be implemented in a very simple efficient way. The problem gets even more complicated if a task does not simply want exclusive access to the shared data but that it wants exclusivity only after a certain relation exists among the elements of the shared data.