Special Report

CMU/SEI-88-SR-3

A Practical Application of the Ceiling Protocol in a Real-Time System

C. Douglass Locke,
IBM Federal Systems Division

John B. Goodenough,
Software Engineering Institute

March 1988

Special Report

CMU/SEI-88-SR-3

March 1988

A Practical Application of the Ceiling Protocol in a Real-Time System

C. Douglass Locke

IBM Federal Systems Division

John B. Goodenough

Software Engineering Institute

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

A Practical Application of the Ceiling Protocol in a Real-Time System

Abstract. This paper briefly discusses some of the real-time design issues that arise when using the priority celing protocol for real-time systems. The paper shows a small real-time system design and shows how the code in the system could be structured to satisfy the requirements of the ceiling protocol.

This paper will be presented at the 2nd International Workshop on Real-Time Ada Issues in May 1988.

1. Introduction

This position paper explores the application of a new priority management protocol for bringing a heretofore unknown level of response time predictability to the design of real-time programs in Ada. While the protocol is not itself dependent on the use of Ada, we believe that when used in an Ada design, it will enable a much more complete use of modern software engineering in the implementation of real-time software, enhancing not only performance, but also quality, reliability, and maintainability. Thus, the protocol helps to separate timing issues from the logical structure of a set of tasks, thereby helping to provide a better basis for engineering real-time software.

This new protocol, called the ceiling protocol, is described in a companion position paper [2]. Although the protocol provides what we believe are extremely important benefits to the design of real-time systems, it is not without cost; the ceiling protocol imposes several significant limitations on the use of Ada in software design. It is the purpose of this paper to explore some of the ramifications of these limitations, illustrating the issues raised by the use of an actual problem encountered in modern avionics systems. We hope to demonstrate the likelihood that, in spite of these limitations, there are many existing real-time problems which could greatly benefit from the application of this protocol.

2. Real-Time Design Issues

The design of a real-time system requires a sequence of design decisions, each representing a tradeoff against a set of generally conflicting requirements. The choice of Ada as the implementation language for such a system determines many of the important design decisions, and these choices are further limited by the decision to take advantage of some of the current real-time scheduling research such as the ceiling protocol.

Our current understanding of the ceiling protocol, which emphasizes predictability in the resulting application response characteristics, defines four restrictions on the use of Ada tasks. While these limitations restrict the range of design decisions, we contend that there is a useful class of real-time problems that can be solved within these limitations.

1. All accept statements in a task must be contained in a single select statement that is the only statement in the body of an endless loop. There must be no guards on the select alternatives, and no nested accept statements.

2. There must be no conditional or timed entry calls.
3. Each task must be assigned a priority.
4. Tasks containing accept statements must have a priority lower than the tasks that call them.

The restrictions on how accept statements can be used ensure a simple server design that helps make response time predictable. The inability to use guards need not be crucial. The principal use of guards is to maintain the integrity of the data structure(s) encapsulated by the server, e.g., to prevent table overflow. If guards are not used, integrity must be guaranteed in other ways. For example, if table overflow occurs, either the new data can be ignored, or old data can be overwritten. Either approach is acceptable in many real-time systems; in fact, the client task's time constraints will generally prohibit allowing a server to ignore an entry call to prevent data loss. There are, of course, situations in which such simple solutions will not be adequate, and these will require further study.

The prohibition against conditional and timed entry calls further simplifies response time analysis but might make it difficult to compensate for temporary overload conditions by avoiding excessive rendezvous in periods of high utilization. However, the protocol provides for a guarantee of schedulability which will, under many conditions, guarantee that overload will not occur.

The requirement for priorities to be assigned to all tasks is no restriction at all, since only the correct use of priorities makes it possible to handle response time requirements predictably. The use of low priorities on server tasks is reasonable, since the ceiling protocol ensures that the effective priority of a server will be at the client's level or higher whenever the server is acting on behalf of a client or is blocking a higher priority client.

3. A Real-Time Avionics Example

A simple example of a real-time problem in modern avionics systems is the increasingly common requirement for ''sensor fusion,'' in which information from functionally different sensors is combined to create a coherent, integrated picture of the external environment. For example, a single radar return might suggest the presence of a single target (called a ''track''), but an electromagnetic sensor might indicate the presence of multiple sources (such as several aircraft flying in close formation).

While it is preferable for many reasons to use a straightforward object-oriented design for such a system, such designs are not generally used today. One major difficulty has been the cost of arbitrarily long mutual exclusion delays, regardless of the language used, in addition to the possibility of deadlock. These possibilities have frequently dictated the use of more *ad hoc* designs (such as simply managing a shared track table and accepting occasional inconsistency and system maintenance fragility). The result has been to trade design simplicity in favor of meeting time constraints and avoiding deadlock.

Using Ada to implement a natural object-oriented design means an Ada track table task would encapsulate the table of tracks currently being monitored in the aircraft, while a separate multisource track correlator task would periodically attempt to detect anomalies among the tracks, consolidating or separating consolidated tracks when changes to correlations are detected. Such a change in correlation might occur, for example, when an apparently new track is identified by one of the sensors. In addition to the track table task and the correlator task, there would be a set of sensor tasks, one for each separate sensor; and there would be tasks that use the track information, such as display tasks and data link tasks. The structure of such a system is shown in Figure 1.

Figure 1: Example Task Structure

Each sensor task detecting a contact enters a rendezvous with the correlation task to identify the new track. The correlation task must then rendezvous with the track table task, retrieving the relevant track data and ensuring a consistent view of the tracks in the track table. During the correlation-driven track update, the relevant track entries must remain appropriately locked to ensure a consistent view is presented to other modules; this correlation is provided by a rendezvous with the track table task. The challenge is to handle this mutual exclusion without an adverse impact on overall real-time response and without generating the potential for a deadlock.

Normally in designing entry calls, the implementer arranges to make the rendezvous as short as possible to avoid excessively blocking other tasks, but this reasoning is significantly modified under the priority ceiling protocol. What is important under this protocol is the total time needed to accomplish each of the sensor actions. If the table must be accessed once per sensor period, the time to access the table is logically attributed to the sensor task's processor utilization and belongs inside the rendezvous. If the priority protocol analysis shows that all deadlines will be met when this time is attributed to the sensor task, it doesn't matter whether the call occurs inside the rendezvous. If the analysis shows that all deadlines may not be met, the required corrective measures are unchanged.

In this example, the track correlator and the track table tasks may both be considered as examples of servers that provide common support to a set of client tasks, represented in this case by the sensor tasks and the keyboard, data link, and display tasks. The natural design (omitting all but the basic structure of the task) of one of these server tasks (the correlator task) is shown here.

```
task body CORRELATOR is
    loop
        select
            accept TRACK_UPDATE do          -- Called by one or more of the sensors.
              -- Note the identifying information about this track.
              -- Rendezvous with the track table task to error check
              --     the information provided relative to that already known.
              -- Identify possible matches between this track and current tracks
              --     in the track table, making or revising correlation decisions.
              -- Rendezvous with the track table task to update the track(s).
            end TRACK_UPDATE;
        or
            accept DELETE_TRACK do          -- Called by the keyboard task.
              -- Note the identifying information about this track.
              -- Revise associated correlation decisions.
              -- Rendezvous with the track table task to delete the track(s).
            end DELETE_TRACK;
        or  terminate;
        end select;
    end loop;
end CORRELATOR;
```

It should be noted that this design includes steps (e.g., rendezvous with the track table task to update the track) which might, in a natural design, be performed outside of the rendezvous, although still within the loop. Such a structure would violate one of the constraints imposed by the ceiling protocol and is logically unnecessary, since such steps are performed on behalf of a client *at the client's priority*. These steps are therefore part of the client's predicted processing load for which the schedulability has been predicted. Therefore all processing has been placed within the rendezvous code. Further ramifications of this decision will be evaluated in future work.

Typically in such a system, the sensor tasks execute periodically, driven by the need to sample the sensor (e.g., a magnetic sensor) or by the actual cyclic nature of the sensor hardware (e.g., a radar). Since their periodic rates vary, their priorities also must vary proportionally, using the simple rate monotonic scheduling algorithm which provides for a straightforward analysis of the schedulability of the system as a whole [3]. However, the schedulability of the system is rendered very difficult if the higher priority sensors (the fast ones) can be repeatedly blocked by the lower priority sensors due to rendezvous.

Assuming that the server tasks are indeed structured similarly to the one in the example above, the server task is simply an instance of a semaphore, with its accept statements providing critical sections guarded by the semaphore. This situation is therefore exactly the same as that required for the ceiling protocol [2]. Using the ceiling protocol described in [2], it becomes clear that this approach, rather than resulting in a difficult scheduling problem, guarantees that at most one low-frequency task will block a high-frequency task; in addition, in more complicated cases, the protocol completely eliminates the possibility of deadlock due to multiple rendezvous dependencies.

4. Conclusion

Space limitations for this position paper make it impossible for us to give a detailed analysis of how the ceiling protocol can affect the design of real-time Ada systems. Nonetheless, it appears that the protocol allows a modern avionics problem to be rendered simply into an object-oriented Ada design that can result in an efficient implementation of the required functions. The protocol ensures that the required mutual exclusion can be provided, not only with minimum delay of high-priority tasks and within predictable time constraints, but free of deadlock—one of the most critical conditions to be avoided in a real-time system.

References

[1] Cornhill, D.
Task Session Summary.
In *Proceedings of ACM International Workshop on Real-Time Ada Issues* **VII***, 6*, pages 29-32. *Ada Letters*, 1987.

[2] Goodenough, J. B. and Sha, L.
The Priority Ceiling Protocol: A Method for Minimizing the Blocking of High Priority Ada Tasks
2nd International Workshop on Real-Time Ada Issues, May 1988.

[3] Liu, C. L. and Layland, J. W.
Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment.
Journal for the Association of Computing Machinery 20, 1:46-61, 1973.

Table of Contents

List of Figures