Closed Errata AADL V2.1 (No Change)

Feb 2016

Table of Contents

8 Features and shared access

Semantics of data ports with data having subprogram access

Issue: it is legit to have a data port in a thread/subprogram if this data has subprogram access features? If yes, what is the underlying semantics?

Response: Subprogram access indicates that the data can be accessed via “methods”, i.e., the application inside the thread receiving the data makes calls to the appropriate access methods, e.g., as they would in Java.

Status: Closed 7/14.

Blocking Times for Shared Data Objects

Issue: Timing analysis of threads that use real-time semaphores to control access to shared data objects uses a blocking time that is determined from the worst-case locking times by threads for shared objects. The standard does not clearly state how locking times by a thread for a shared data object are determined. Here are three possible ways to specify locking times in AADL.

1. Infer locking times from the compute times of subprograms that require access to data objects that have a Concurrency_Control_Protocol property. Whether a particular thread calls such a subprogram and locks the object could be specified either in a calls subclause or assumed to call any contained or required subprogram. This could still be ambiguous about which among multiple nested subprograms actually locks the object. Subprograms might access a shared data object but not lock it, which might not be model-able.

2. Infer locking times from subprograms provided by the shared data object itself. Such subprograms would either be specified in a calls subclause or assumed to be called by their containing or accessing thread. Threads might call subprograms that access a data object having a Concurrency_Control_Protocol that do not lock that data object, which might not be model-able this way. A data object might provide subprograms that do not lock it, which might not be model-able. Also, the current "calls" syntax only allows a required subprogram in the data type declaration to be named, not a subprogram from a specific data object as would be needed.

3. Use a behavior annex specification. The behavior annex includes a way to show when Get_Resource and Release_Resource are called. However, it seems awkward to require support for an otherwise optional annex to specify a property value for a capability (shared objects and real-time semaphores) that is part of the core standard.

All of these are wordy for early conceptual modeling where brief specifications for early analysis are desired.

Proposed correction: Option one above is just a clarification to the meaning of the standard. To resolve the ambiguity when there are nested subprograms, state that by default the innermost accessing subprogram that has a Compute_Execution_Time property determines the locking time.

Allowing a subprogram for a specific data object (not just a subprogram for a data type) to be named in a calls declaration would enable the second option above (the object-oriented pattern that a shared data object provides access to the subprograms that operate on it). (The pattern of a subprogram requiring access to a shared data object that it locks is supported by the current calls syntax.)

To support more explicit modeling, an option would be to add a Locking_Time property for subprograms, subprogram groups, threads, and thread groups to explicitly specify a locking time for that software object when it accesses that shared data object. This could be used in combination with the other options to provide a default when there is no explicit Locking_Time property association, in which case it could be stated that an explicit null locking time means that a subprogram does not lock a shared object that it accesses to model those cases. The Locking_Time property would need to specify both a locking Time_Range and the accessed data object being locked by that software object. Here is a possible example:

```
        Locking_Time_Record: type record (
                Data_Object: reference (data);
                Locking_Time: Time_Range;
        );
        Locking_Time: list of THISPROPERTYSET::Locking_Time_Record
        applies to (subprogram, subprogram group, thread, thread group,
process);
```

A specification would be erroneous if it declares a thread that accesses a shared data object that has a Concurrency_Control_Protocol specified but none of the above rules allow a locking time to be determined.

PHF: Note, we already have a property called Access_Time that effectively specifies the locking time.

Status: Closed.