“Position Paper: Clock Support in Ada”

Kristoffer Nyborg Gregertsen

Citation:
K. N. Gregertsen, “Position Paper: Clock Support in Ada” (2018) . Ada User Journal, 2018, 39, 2 pp 133-135

Position paper: Clock support in Ada

Kristoffer Nyborg Gregertsen
Department of Mathematics and Cybernetics
SINTEF Digital, Trondheim, Norway
kristoffer.gregertsen@sintef.no

Abstract

This position paper briefly revises the clock support in the Ada programming language, including execution time clocks and timers for tasks and interrupts. It is argued that the standard library would benefit from a more coherent handling of clocks, in particular with in the lack of a shared interface for clocks and timers, and that better support for high-precision real time (TAI/UTC) should be provided.

1 Background

Most real-time control systems are of a distributed nature and need to have a shared system time of sufficient precision between nodes. Examples of such systems can be found within aerospace, robotics and automation domains. Having a shared system time allows measurements from different nodes to be correctly time-stamped and correlated, and synchronous events to be orchestrated. As all clocks have a certain drift due to inaccuracies in the oscillators and temperature changes, it is necessary to *discipline* them by distributing a time signal to compensate for this drift.

The passing of time is defined by International Atomic Time (TAI) reference, that is made up of a number of atomic clocks around the world distributing time signals between them. The measurements from these clocks are compensated for that time passes at slightly different pace due to differences in gravity. The TAI time reference is *monotonic* – it does not have any jumps. UTC time is based on TAI, but is compensated for the variances in earths rotation by adding an occasional leap seconds at the end of the year. Leap seconds are announced in advance, and special care needs to be taken to handle them properly in systems that use UTC time. As of May 2018, TAI is 37 seconds ahead of UTC due to leap seconds.

For most purposes the Network Time Protocol is the most convenient way of getting access to UTC. This protocol continuously retrieves time from time servers and estimates the drift of the computers internal clock, continuously adjusting it so to avoid abrupt changes. The precision of NTP is in the millisecond range, but can be subjected to offset and jitter if the latency of the network is not consistent. The Precision Time Protocol (PTP) achieves sub-microsecond precision by measuring the latency through the network, but also requires special Ethernet switches and network cards. PTP is used to keep distributed computer systems time synchronized, for instance to acquire scientific measurements. For instance, the White Rabbit Project [4] at CERN uses PTP and special hardware to synchronize experiments over a large area.

To get access to a high-precision UTC signal that can be distributed in the local network it is most convenient to use a GNSS receiver, listening to GPS/Galileo/GLOSNASS/BeiDou satellites that distribute UTC time. Such clocks will typically be synchronized with UTC at sub-microsecond offset, and distribute time signals with protocols such as PTP or NTP. These protocols are supported by operating systems such as GNU/Linux, Mac OSX, Windows, QNX and others. It is also possible to add support in embedded systems such as the Xilinx UltraScale+ MPSoC by using PTP IP cores, or by using a combination of time messages and Pulse Per Second (PPS) signals in embedded systems to correct the internal oscillator.

2 UTC support in Ada

The package `Ada.Calendar` is the standard way of getting the system time in Ada, and has been present since Ada 83. The language defines the type `Duration` for time duration, and the type `Time` for time instants that can be retrieved by the `Clock` function, and functions for getting the year, month, day and intra-day second of that time instants. The standard requires that the value of `Duration'Small` should be no greater than 20 microseconds, but it could also be of nanosecond precision if the implementation allows. The constant `System.Tick` gives the duration of which the clock will return the same value. Sub-packages of `Ada.Calendar` support time-zones, retrieval of UTC offset, and time arithmetic including historical leap seconds.

For use in real-time systems, annex D of the standard has had support for high-precision monotonic time since Ada 95. This package has more stringent requirements for precision, and also requires the clock to be monotonic with no abrupt changes as this would lead to problems for real-time applications, for instance when using periodic tasks with **`delay until`**. The package supports getting the unit for both time instants and time spans, and the standard requires that these shall be no greater than 20 microseconds. Also, it is required that the tick of the real-time clock shall be no greater than 1 milliseconds. Implementations are required to document the metrics of the clock compared to the TAI reference, such as drift and maximal clock jumps. As stated before, a clock that is not disciplined by a time signal will drift over time, and this drift rate will vary among individual clocks and with temperature. It will thus be hard for implementations to give tight bounds on the drift for undisciplined clocks, and for disciplined clocks it would be more relevant to give a bound to the maximal offset to UTC and information of how time is adjusted. For general purpose operating systems like GNU/Linux, it would of course be hard to document this at all as it depends on how the system administrator has configured the system.

The developer should be able to know whether a reliable clock synchronization is available or not. This is must be done as a query at run-time, for instance using a routine as `Last_UTC_Synchronization` to get the time point at which the clock was last synchronized, or a function for querying the a more general state of the clock synchronization. It does not suffice to know that the the system *supports* time synchronization, as this synchronization can be broken due to a range of different errors that can not be predicted in advance, such as network error or snow on the GNSS antenna. Also, it may take some time at start-up to achieve time synchronization.

Furthermore, even though the real-time clock may be implemented by a system clock that is kept in sync with TAI, there is no standard way of retrieving an UTC timestamp from the real-time clock. Such a timestamp could be made from the calendar package, but even though the standard advises that the two clocks could be transformations of each other, there does not seem to be a straightforward way of doing this transform. Also, the calendar package does not take leap seconds into account, something that is needed for correct UTC timestamps. Another problem for real-time applications is that the much used Ravenscar profile disallows the use of the calendar package, leaving no standard method of getting absolute time.

It is argued that means should be provided for real-time applications to acquire high-precision UTC timestamps, given that the implementations supports a clock that is disciplined relative to TAI, and that the metric of this time synchronization should be documented. Specifically, a set of functions should be provided to translate between UTC timestamps and the `Time` type of Annex D. Such a feature would be of great value to distributed systems that need to communicate correlated measurements, and execute actions at specific points time.

3 Coherent clocks in Ada

Related to real-time clocks and timers, there should be a discussion of how to make these more coherently represented in the language. In addition to the standard calendar and real-time clock described earlier, Ada 2005 defined timing events for the real-time clock, clocks and timers for the execution time of tasks, and group execution time budgets for tasks. Also, Ada 2012 addressed concerns about the accuracy of execution time measurement for tasks, by adding support for measuring the execution time of interrupts, both for individual interrupt ID's and the total execution time for interrupt handling. However, full execution time control for interrupts is not possible as interrupt execution time timers and group budgets are not supported. At IRTAW-16 the author proposed to add execution time timers for interrupt handling as a new type extending the existing task timers [3]. This non-standard feature was earlier implemented on GNATforAVR32 [1, 2]. However, IRTAW-16 found that the proposal needed further work, and that a more coherent type system with a new root type should be specified [5]. At, IRTAW-18 the author proposed a revised class hierarchy for timers and group budgets to support interrupts in a more coherent way, and showed how the deferrable server for interrupt handling could be implemented with this feature. This proposal was rejected on breaking backward compatibility by adding a root class for `Timer`.

In the authors opinion, the different clocks and timers in Ada are defined in a fragmented way, that does not allow for reuse and abstraction through object-orientation. The different clocks packages are defined in a similar way, but have no common interface, and even though the timers and timing events are quite similar and are defined as tagged types, there is no common base class or shared interface that allow them to be used in an object-oriented manner. It would seem that the only real benefit of using tagged types for these types is that it allows for dot notation on routine calls. By redesigning the clock definitions using common interfaces, it would be possible to have reuse of timing related functionality such as time servers, and also to define bespoke clocks for real-time systems if needed. Such as object-oriented design is followed for the Real-Time Specification for Java (RTSJ) that also allows custom clocks [6]. The new package `<chrono>` of C++11 also supports different clocks as defined types, and the coming C++20 will also support TAI, GPS and other clock sources. As of now RTSJ and C++ seem ahead of Ada in terms of advanced and coherent clock support, and it is argued that the value of backward compatibility should be weighted against the need to keep Ada at the forefront within real-time systems.

References

[1] K. N. Gregertsen and A. Skavhaug. Implementing the new Ada 2005 timing event and execution time control features on the AVR32 architecture. *Journal of Systems Architecture*, 56(10):509–522, oct 2010.

[2] K. N. Gregertsen and A. Skavhaug. Implementation and Usage of the new Ada 2012 Execution Time Control Features. *Ada User Journal*, 32(4):265–275, 2011.

[3] K. N. Gregertsen and A. Skavhaug. Execution time timers for interrupt handling. *ACM SIGAda Ada Letters*, 33(2):87–96, nov 2013.

[4] J. Serrano, M. Lipinski, T. Wlostowski, E. Gousiou, E. van der Bij, M. Cattin, and G. Daniluk. The White Rabbit project. *Proceedings of IBIC2013*, 2013.

[5] T. Vardanega and R. White. Session summary. *ACM SIGAda Ada Letters*, 33(2):126–130, nov 2013.

[6] A. Wellings and M. Schoeberl. User-Defined Clocks in the Real-Time Specification for Java. *Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems - JTRES '11*, page 74, 2011.