An Experience Report: The Role of Distributed, Real-Time Ada & C++ on the Airborne Surveillance Testbed (AST) Program

Henry Lortz & Tim Tibbetts
Airborne Surveillance Testbed Program
Boeing Space and Communications Group

Overview

The Airborne Surveillance Testbed (AST) program, managed by SMDC for BMDO, is a technology demonstration program that supports development, test, and evaluation of defensive systems to counter intercontinental and theater ballistic missiles (ICBMs and TBMs) and their warheads. The heart of the AST program is a Boeing 767 aircraft equipped with a Raytheon-built, large-aperture, multiband, high data rate infrared sensor and a wide variety of processing equipment designed to detect, track, and discriminate ballistic missiles at long ranges. A Raytheon interceptor seeker (part of a Navy risk reduction effort) has recently been integrated onto the aircraft; a staring medium wave infrared (MWIR) camera is currently being added as well. Onboard processing capabilities include a Concurrent TurboHawk (multi-CPU PowerPC flight computer) along with a variety of custom and off-the-shelf signal processing equipment, SGI workstations, DEC Alphas, and PCs, largely programmed in Ada and C++. These systems are linked via SCRAMNet, Ethernet, 1553B, RS422 and RS232, and communicate externally via various radio systems. Since the start of the program in 1984, AST has been making use of Ada83, Ada95, and C++ for both simulations and embedded flight software. During that time, we have gathered a lot of experience in the use of Ada for real-time distributed systems, especially concerning:

- The pitfalls of task scheduling algorithms and priorities
- The benefits of the (careful) use of generics
- The importance of some changes between Ada83 and Ada95
- Interfacing Ada software to hardware (and standardized interrupt handling)
- The importance of proper use of exception handling to ensure fault tolerance

Background

The AST program began (as the Airborne Optical Adjunct) in 1984, as part of the Strategic Defense Initiative. As such, most of the onboard hardware and software (including operating systems and tools) was custom built, mil-spec, ruggedized, and not easily modifiable. Simulations, software tools and flight software were largely programmed in C, Pascal, Assembler, and Fortran. Early on, it became apparent that programmatic success required the development of reliable, repeatable, end-to-end simulations of the core detection and tracking systems to support both algorithm development and flight software implementation. The first end-to-end simulation was written in Fortran, followed by a successor written mostly in Fortran, with some Pascal flight software embedded. However, it became clear that the development of a more powerful emulation environment would be required to support testing and debugging of the flight software. Thus, the first use of Ada on the AST program was for the development of an emulation testbed known as the AdaBed. The AdaBed was task-based, and provided an environment very similar to the distributed (25 1750A CPUs) target flight hardware. Algorithmic modules written in Pascal or Fortran could be inserted in the emulator, which provided all the operating system services available on the target system, along with extensive debugging and performance measurement capabilities. This first generation AdaBed allowed considerable development and debugging to proceed, but it had problems, primarily associated with the Ada tasking model, debugging environment support and non-repeatability. After experimentation with the use of multiple processes (instead of tasks) to model the multiple CPU target system, a second generation AdaBed was developed which used tasks in a more limited sense (for synchronization only), and provided a single-threaded and far more controlled and repeatable environment for software development.

However, the single-threaded nature of the system did not affect the quality of emulation; on the first true test flight of the target system, a processor failure occurred. Back on the ground, the results generated by the emulation were identical, once a failure of a single processor (out of 25) at that point in the mission replay was scripted in.

As the program evolved, and a variety of successful missions were flown, the expense of maintaining proprietary (and generally unique/custom) hardware and software became more of a burden. Therefore, a process of transitioning both onboard and ground to Commercial-Off-The-Shelf (COTS) technology was started. This included both hardware and software. The initial improvement was to transition from the custom 25 CPU, full mil-spec ruggedized flight computer and its proprietary O/S and Pascal compiler to an industry standard real-time computer (Harris Nighthawk 4404 with four 25MHz Motorola 88100 RISC processors) and a more viable language (Ada 83). This conversion involved a complete rewrite of the application software, but it resulted in a far more robust and extensible system. The two systems were roughly comparable in performance, but the increased memory size and reduced need for interprocessor communication on the Nighthawk provided big gains from the elimination of highly compressed data structures, thereby saving CPU time, and improving maintainability of the software. In addition, the hardware and software *were commercially supported and could be easily modified*, and had greatly reduced maintenance costs, paying for the upgrade within 1 year. With the advent of Ada-based flight software, the emulation testbed became 100% Ada, which allowed it to migrate from system to system with little effort (it primarily runs on a DEC Alpha under OpenVMS, but versions exist to run on SGIs under IRIX and DEC Alphas running Digital UNIX). More recently, the NightHawk 4404 has been upgraded to a TurboHawk 6804 with four 350MHz PowerPC processors, and the flight software ported from Ada83 to Ada95.

Commonality of hardware and software between the ground and onboard systems has been of great benefit in maintaining and extending the flight software. The software modules used in the flight computers and in the ground simulations are nearly identical; ongoing work developing non-intrusive diagnostics and recording techniques is targeted at eliminating most remaining differences between the lab and the aircraft. In addition, the AST onboard equipment configuration is largely duplicated in the ground-based AST Simulation Center (ASC), except for the absence of real sensors. The ASC is used for closed-hardware-in-the-loop (HWIL) simulation, and also contains additional systems for algorithm and software development, mission data analysis, playback, recording, and software-based sensor emulation. This HWIL capability greatly eases algorithm and software development, and provides a reliable and cost-effective method to assess the effects of new algorithms and equipment (e.g., additional sensors, uplinked data for correlation/fusion, etc.) on overall system performance. AST also uses its closed-HWIL simulation and Distributed Interactive Simulation (DIS) capabilities (either on the aircraft or in the lab) for flight crew training, and providing warfighter support via participation in exercises (wargames) such as Roving Sands, Matador and Coldfire.

Other Ada-based software capabilities on AST include the core simulation tools known as the Case Generation Code (CGC), which is also the primary constituent of the Interactive Case Generation Capability. This DEC Alpha/Digital UNIX-based system provides sensor emulation for flight crew training and support of DIS activities. The other main component of the DIS capability is the pilot/DIS network which is programmed in Ada95, using Motif/X-windows on an SGI Indigo^2; it provides interactive control of the flight of the (simulated) aircraft as well as a situational awareness display for battle management. Together with the C++-based SGI Control & Display (C&D) consoles, these systems make up the robust, reliable AST mission and lab systems.

In addition to the important points mentioned earlier, a variety of lessons have been learned about:

- The importance of using a validated, standard language (Ada83 and Ada95) which supports the development of portable and reusable code
- The importance of commonality of software for lab and operational use to reduce risk, cost, and flow time
- Interfacing Ada to other languages (e.g., commingling Ada and C/C++ in a single distributed system)
- The value (and validation) of COTS systems incorporating Ada
- Methods for instrumenting and monitoring/measuring performance of Ada-based real-time systems
- Methods for debugging distributed Ada-based real-time systems

- Validation of Ada-based systems vs. validation of non-Ada systems