Legacy Software Migration

By Dr. Robert Dewar

Ada matters!

The computing industry is rather peculiar: If a technology is not the latest and greatest, people think it has vanished entirely. You can, for example, often meet otherwise knowledgeable people who think that mainframes and COBOL have completely disappeared.

Here at AdaCore, we often find people who think that Ada has disappeared and are surprised to find out that not only is it still around – in the domain of large, critical applications it is especially alive and well. For example, the new air-traffic control system for the UK, iFACTS (www.praxis-his.com/news/radarMar2007.asp), is being written from the ground up in Ada using the SPARK formal methods approach.

So why is Ada still around when many college students are learning Java, and there are even people around who think C and C++ are disappearing? The answer is remarkably simple. A recent National Academy of Sciences report, "Software for Dependable Systems: Sufficient Evidence?" (http://books.nap.edu/catalog.php?record_id=11923), addresses the issue of security-critical software, and states:

> "The overwhelming majority of security vulnerabilities reported in software products – and exploited to attack the users of such products – are at the implementation level. The prevalence of code-related problems, however, is a direct consequence of higher-level decisions to use programming languages, design methods, and libraries that admit these problems."

There are no silver bullets when it comes to safety- and security-critical software. No tools can guarantee success, but as this quote indicates, it helps to use appropriate technology, including the right programming language. Last year's High Confidence Software and Systems (HCSS) conference, sponsored by NSA to address security-critical issues, featured an interesting presentation from Microsoft addressing such issues in the context of Windows. The primary sources of problems in Microsoft's experience are buffer overruns and integer overflow problems. Reasonable enough, so how can a programming language help?

The answer is: quite a bit. However, if we look at C or C++, these languages notoriously have absolutely nothing to help avoid overruns. Microsoft is trying to add assertions to C to help, and found about 100,000 possible buffer overflows in the Vista code. Ada, by contrast, fully checks all array references as a matter of course, to eliminate the possibility of buffer overflow.

Although Java is often cited as a safer alternative to C and C++, when it comes to integer overflow, the semantics for Java can introduce some dangerous vulnerabilities. Overflows wrap silently, so that adding 1 to a positive number can suddenly make it negative. In Ada, every integer value can have a sensible range assigned, and automatic checks ensure that values do not go outside this range.

> "Ada ... fully checks all array references as a matter of course, to eliminate the possibility of buffer overflow."

These are just a couple of small examples of how a language can help. It's not at all surprising that Ada should address issues like these; it was designed for the purpose of writing large critical programs, something that cannot be said of languages like C, C++, or Java. If you search language standards for features related to safety and security, you will draw a blank, unless you are looking at the Ada standard. With Ada, you will find a whole annex devoted to high-integrity programming.

Ada does not guarantee success, but as the NAS study indicates, it makes sense to use tools suitable for the purpose; if one's purpose is writing large safety- or security-critical programs, then Ada is a natural choice. That's why it should not be surprising that Ada continues to play a critical role and that Ada is a programmer's frequent companion. Many might not be aware of that fact, however. An example, though, is the avionics system of the new Boeing 787 Dreamliner, which makes extensive use of Ada. Of course, travelers won't be aware of this when they fly, and that's the point. Software that works properly is out of sight, and out of mind. Programmers who build such systems know the advantages of Ada – and continue to choose it as one key element of critical systems.

Dr. Robert Dewar *is cofounder, president, and CEO of AdaCore; he also has had a distinguished career as a professor of Computer Science at the Courant Institute of New York University. He has been involved with the Ada programming language since its inception in the early 1980s and, as codirector of both the Ada-Ed and the GNAT projects, led the NYU team that developed the first validated Ada compiler. Robert was one of the authors of the requirements document for the Ada 95 revision, and he served as a distinguished reviewer for both Ada 83 and Ada 95. He has coauthored compilers for SPITBOL (SNOBOL), Realia COBOL for the PC (now marketed by Computer Associates), and Alsys Ada. He is also a principal architect of AdaCore's GNAT Ada technology. He has also written several real-time operating systems for Honeywell Inc. and frequently shares his thoughts on computers and on open-source software at conferences. He can be reached at dewar@adacore.com.*