Major Trends (Security, Open Source Software, Lowered Development Time) and Ada

David A. Wheeler

December 10, 2003 (revised)

SIGAda 2003 Keynote

Outline

- Major Trends
  - Security
  - Open Source Software
  - Development time as key driver
- Major Questions (for each Trend)
  - How do Ada and the trend match well?
  - How do Ada and the trend match poorly?
  - How can you use Ada to support the trend?
- What can the Ada community do?

Major trend: User desire for secure applications

- Computers increasingly targeted
  - More systems computerized & networked, increasingly valuable
  - Permits "pleasure" attacks, asymmetric attack
- Attacks getting faster
  - Only seconds to exploit every vulnerable Internetted computer in the world (and there are fewer non-Internetted computers)
  - There might be no patch for the vulnerability; might be unknown
- Common languages (esp. C, C++) make it *very* easy to accidentally create security-relevant bugs
  - It's hard to secure programs without changing technology
- Microsoft saying it's spending $$$ on security
  - Are they serious? I hope so, but even if you don't believe them. . .
  - They're saying this because they think it's what users want

How do Ada and security match well? (1 of 3)

- Buffer overflow protection without performance compromise
  – Buffer overflow current #1 security problem
  – Many other languages trade performance
- Avoids some other security-related flaws
  – Format strings, common mistakes (= vs. = =)
  – Strong typing finds bugs early/automatically
- International rigorous standard specification
  – Supports analysis (esp. formal methods)
- Easy-to-read => easier to review

How do Ada and security match well? (2 of 3)

- Ada’s properties nicely support security standards such as the “Common Criteria”
  - ALC_TAT.1 “Well-defined development tools” (EAL4)
    - “All development tools used for implementation shall be well-defined”
    - Documentation shall unambiguously define meaning of all statements & implementation-dependent options
    - Having a standard mostly meets (so C & C++ okay too)
  - ADV_INT.2 “Reduction of complexity” (EAL7)
    - “design & structure the TSF in a layered fashion that minimized mutual interactions between layers of the design.”
  - Ada *not* required.

How do Ada and security match well? (3 of 3)

- Ada’s “safety & security” annex H
  - Pragma Normalize_Scalars
  - Document implementation decisions
  - Pragma Reviewable
  - Pragma Inspection_Point
  - Pragma Restrictions (to simplify object code)
  - But:
    - Object code analysis often impractical (scale), especially vs. denial-of-service attacks
    - There’s much more to security than this annex!

How do Ada and security match poorly?

- Limited standard Ada libraries
  - Extra work to check inputs, use cryptography, etc.
- C buffer overflow protection can interfere with Ada
  - One approach to countering C buffer overflows is to prevent execution of code on the stack (& move other code)
  - Ada implementations typically need to execute code on the stack ("trampolines", e.g., for access values to nested subprograms)
  - Protection can be disabled for Ada programs (Ada code checks for overflows already)… but if Ada calls a C library, the *Ada* program is vulnerable while C programs using the library *aren't.* In this case, *Ada introduces a new buffer overflow vulnerability*
  - Possible to permit trampolines (OpenWall does) but adds complexity and may create security loophole. Needs analysis
  - Red Hat Fedora uses *exec-shield*: protect, or trampoline, *not both*

How can you use Ada to implement secure programs?

- Apply general secure programming techniques
  - Define security requirements (confidentiality, etc.)
  - Good design (secure interface, minimize privilege)
  - Validate inputs and secure the interface
- Beware gotchas
  - All languages have gotchas; Ada has relatively few
  - Beware using Unbounded_String to store secrets (passwords, private keys); the data lingers
  - Ada strings can hold ASCII 0; problematic if passed to C libraries (including kernels)
  - String type first index value doesn’t always start at 1
- See http://www.dwheeler.com/secure-programs

Outline

- Major Trends
  - Security
  - ➡ Open Source Software
  - Development time as key driver
- Major Questions (for each Trend)
  - How do Ada and the trend match well?
  - How do Ada and the trend match poorly?
  - How can you use Ada to support the trend?
- What can the Ada community do?

Major Trend: Open Source Software

- Free (Libre) Software (FS)
  - Freedom to run for any purpose, study/adapt program, redistribute program, improve & release to public. Requires access to source.
  - *Not* “no charge” - “free as in freedom”
- Open Source Software (OSS)
  - New term to reduce confusion (not all accept this term)
- May be commercial or non-commercial
  - Antonyms: proprietary, closed source
- Usual development approach similar to scientific method
- Many widely-used programs are OSS/FS
  - Apache (#1 webserver), bind (#1 DNS), sendmail (#1 mail server), Linux kernel (#2 server)
- Many licenses; GPL most popular (50%-70%)
- More info: http://www.dwheeler.com/oss_fs_why.html

How do Ada and open source software match well? (1 of 2)

- OSS based on peer review – ease-of-reading helps
- GNAT helped “save” Ada technology
  - Before GNAT, Ada compilers cost 10-100x vs. C, C++
  - GNAT made it easier to get Ada into schools, industry
  - Hasn’t eliminated competition. In retrospect should have been obvious, gcc hasn’t eliminated all C and C++ compiler vendors
  - GNAT gave users more comfort that at least one Ada compiler would be around (so using other Ada compilers would be safer)
  - Some OSS/FS developers want to use *only* OSS/FS tools (in contrast, full Java & C# implementations are still in development!)
  - Red Hat Linux’s inclusion of GNAT disseminated Ada technology; millions of CD-ROMs with Ada distributed worldwide

How do Ada and open source software match well? (2 of 2)

- Many in OSS community view Ada positively
  - Not all. Raymond: “Bondage & discipline language”
  - But GNAT received openly by gcc community, GTK+ developers loved the Ada binding GTKAda
- OSS community is interested in tools to improve reliability
  - OSS incentivizes reliability: Bugs increase effort (not revenue); developers also users; OSS invokes personal reputation
  - OSS dominated by experienced developers who usually understand the value of such tools
    - BCG: Average OSS developer age 30, 11yrs experience
    - Linus Torvalds creating C checking toolset to help find errors in Linux kernel. “I am a big fan of typechecking” (Linux Lunacy 2003)

How do Ada and open source software match poorly?

- Too few know Ada
  - OSS works best with *many* developers
  - Mailman stunted at first because few Python developers
  - Need a serious OSS application in Ada to encourage Ada use by developers who don't yet know Ada
    - Examples: C: Unix kernel; Python: MailMan and Zope
    - Need a domain killer, not a poor half-done re-implementation
    - Something many users will want to use because it's *useful*, not just because it's Ada. *Not* office suite: Open Office owns niche
    - Ada has many strengths, but developers have no reason to try it
- Not enough bindings & too hard to create
- No Perl-CPAN-like facility for OSS Ada libraries
  - there *are* websites and reusable code

How can you use Ada to implement OSS?

- Yes, just release Ada code you write under an OSS license
  – No issue if license is MIT, BSD-new, LGPL, GNAT-modified GPL (GMGPL)
- Are there issues releasing Ada binaries under GPL?
  – GPL forbids combining GPL'ed code with proprietary libraries unless they are "normally distributed with the major components (compiler, kernel, and so on) of the OS"
  – Most Ada compilers' libraries are proprietary (except GNAT)
  – No problem if GNAT used to compile (library not proprietary)
  – No problem if proprietary compiler implements OS (embedded)
  – GPL FAQ says no problem if dynamically linked & the "run-time library normally accompanies the compiler or interpreter"
  – Although it's commonly done, there *is* some debate
    - Authors can state (or imply) that this is okay in license, but that may make it problematic to combine with other GPL software

Outline

- Major Trends
  - Security
  - Open Source Software
  - ➡ Development time as key driver
- Major Questions (for each Trend)
  - How do Ada and the trend match well?
  - How do Ada and the trend match poorly?
  - How can you use Ada to support the trend?
- What can the Ada community do?

Major Trend: Development time as key driver

- Proprietary world dominated by time-to-market
  - "First mover" often has significant advantages
    - Users will use the first one available, and probably won't switch afterwards
    - Related tools will start integrating with the first mover
  - Other drivers (performance, memory use) less critical
- The rise of "worse is better" (Richard Gabriel)
  - Gain acceptance, condition users to expect less, then incrementally improve until "almost right thing"
  - "Better to get half of the right thing available so it spreads like a virus. Once people are hooked on it, take the time to improve it to 90% of the right thing"

How do development time and Ada match well?

- Many language features in *one* language
  - Generics (missing in Java)
  - Threads (missing in C and C++ as built-in)
  - Object-orientation (missing in C)
  - Typically compiled so high performance (missing in Python, Perl)
- Stable specification ("learn once")
- Detects bugs via compiler and run-time
  - If you want to maximally eliminate bugs in a short time, compiler/runtime catch them more efficiently

How do development time and Ada match poorly? (1 of 3)

- Absurdly sparse standard library/framework
  - "Batteries not included." I have to create my own stack?!?
  - Missing collections (inc assoc arrays), regex, i18n, portable GUI,...
  - Everyone creates their own, producing incompatibilities
    - Need code to translate *my* stack into *your* stack
  - Compare to libraries of C++ (STL), Java, C#, Python, Perl
  - Strive for big standard (& growing) library, small stable language
- Bindings
  - Mutually dependent types (WG9 ARG), simpler thin bindings, C++ & Java nonstandard, C functions often can't map to Ada functions

How do development time and Ada match poorly? (2 of 3)

- Garbage collection (optional=not implemented)
  - Irrelevant/undesirable for some applications (real-time)
  - Important for others
  - Incorrect “free”ing can be a security vulnerability
- Languages to improve reliability only matter if you really care about reliability
  - Most companies say they do… but they don’t/can’t
  - “Good enough” (in other words, will you buy it?)
  - Perverse incentive: if there are bugs, vendor can probably make you to pay for the upgrade/repairs. More bugs means more $$$

How do development time and Ada match poorly? (3 of 3)

- "Not C" – most languages now use C-like syntax
  - Non-C syntax slightly increases initial training time
  - Ada's advantages are due to its syntax, so shouldn't be "fixed"
  - But there *are* cases where borrowing might help, e.g., Object.Procedure(1,2,3)
- Too wordy
  - Development/review time (and error rate!) strongly correlated to number of lines of code. More lines & text means more development time & increases probability of error
  - Ideal = short as possible while clear & error-resistant (*not* APL)
  - Especially obvious in (awkward) strings, OO, with-and-use
  - Especially obvious comparing with Python, Java, C#, Perl
  - Syntax should be enhanced to make simple/common things short
    - A spoonful of syntactic sugar makes the medicine go down

Examples of syntactic sugar to reduce wordiness

- With-and-use:
  - with asdfg; use asdfg; with asdrg; use asdrg; with asdft; use asdft; with asdfb; use asdfb;
  - with use asdfg, asdrg, asdft, asdfb;
- Unbounded_String:
  - Auto-promote "x" or +"x" to Unbounded_String
  - Handle String/Unbounded_String interactions more intelligently

Common Ada OO constructs are too wordy

The Ada version is so long that it *impedes* reliability & clarity!

- Ada skeleton (11 lines):

```
with Package_Of_Parent;
use Package_Of_Parent;
package Package_Name is
  type My_Type is new Parent_Type with private;
  type My_Type is access all My_Type'Access;
private
  type My_Type is new Parent_Type with
  record
    Some_Data: Data_Type;
  end record;
end package;
```

- C++ skeleton (4 lines):

```
#include "parent_type.h"
class My_Type(Parent_Type) {
  private Data_Type Some_Data;
};
```

- Python skeleton (1 line):

```
class My_Class(Parent_Type):
```

Should the Ada specification be changed for wordiness?

- Yes; program wordiness is a serious impediment
  - Increases development time/cost
  - Decreases reliability/maintainability
- But it must be very limited
  - Any change has a cost to compiler/tool authors
  - Any addition may have a training cost (esp. if it's Ada-unique)
- These are just examples & ideas. Use this process:
  - Look at lots of Ada code
  - Find a *few* (2-6) syntactic-sugar cubes that give the biggest bang-for-the-buck

Outline

- Major Trends
  - Security
  - Open Source Software
  - Development time as key driver
- Major Questions (for each Trend)
  - How do Ada and the trend match well?
  - How do Ada and the trend match poorly?
  - How can you use Ada to support the trend?
- What can the Ada community do?

What can the Ada community do? (1 of 3)

- Fix exec-shield/Ada trampoline incompatibility
- Create large “batteries included” *standard* Ada library
  - Collections, internationalization, portable GUI, etc. Cover at least C++ STL, probably Java’s library
  - Designed to *work together,* eliminate “translations”
  - Release implementation with OSS license permitting proprietary *and* OSS use (esp. GPL). Suggest GNAT-modified GPL or MIT
  - Implement libraries, review, use, fix, *then* standardize
  - Need a community process with stages, feedback from users, continuous improvement. Probably not ISO standardization (at least at first), maybe ISO TRs later on
  - Don’t try to standardize everything at once
  - Maybe “Ada Standard Container Library” & some GNAT libraries as first step. GNAT isn’t the only Ada compiler!

What can the Ada community do? (2 of 3)

- Create easy download/update OSS Ada libraries
  - “Ada-CPAN”
- Ada community should work together to build at least one OSS “killer app” in Ada
  - *Not* just for Ada users; *give developers a reason to learn Ada!*
  - GNAT Programming System? (Problem: many competing Ada & non-Ada environments, inc.Eclipse)
  - A game (maybe based on military simulators/systems)?
  - Genealogy program?
  - Music?
  - Automotive or robot control?

What can the Ada community do? (3 of 3)

- Ada 200Y should fix trouble spots
  - Reduce wordiness. Add a little syntactic sugar!
  - Easier bindings (C functions should map to Ada functions, standard C++ & Java bindings)
  - Slowly expand standard library
  - As standard libraries developed, feed syntax problems to 200Y
  - Bug Jim Moore & Pascal Leroy! Even better, help them

Some relevant articles/books I’ve written

- Open Source
  - “Why OSS/FS? Look at the Numbers!” http://www.dwheeler.com/oss_fs_why.html
  - Statistics showing GPL common: http://www.dwheeler.com/essays/gpl-compatible.html
- Security
  - *Secure Programming for Linux and Unix HOWTO* http://www.dwheeler.com/secure-programs
- Ada
  - Lovelace Tutorial http://www.dwheeler.com/lovelace

http://www.dwheeler.com is my personal web site and is not endorsed by my employer, government, or guinea pig

Backup Slides

GPL Text Constraining Distribution

- Point 3: "You may copy and distribute the Program (or work based on it…) in object code or executable form… provided that you also do one of the following: a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms… above… b) Accompany it with a written offer [to meet (a)], or c) Accompany it with the information you received as to the offer [noncommercial distribution only]… For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and execution… However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

GPL FAQ (1 of 2)

- “Q: I’m writing a Windows application with Microsoft Visual C++ (or Visual Basic) and I will be releasing it under the GPL. Is dynamically linking my program with the Visual C++ (or Visual Basic) run-time library permitted under the GPL?”
- “A: Yes, because that run-time library normally accompanies the compiler or interpreter you are using.”

GPL FAQ (2 of 2)

- “Q: If a programming language interpreter has a license that is incompatible with the GPL, can I run GPL-covered programs on it?”
- “A: When the interpreter just interprets a language, the answer is yes. The interpreted program, to the interpreter, is just data; the GPL doesn’t restrict what tools you process the program with. However, when the interpreter is extended to provide “bindings” to other facilities (often, but not necessarily, libraries), the interpreted program is effectively linked to the facilities it uses through these bindings. The JNI or Java Native Interface is an example of such a facility; libraries that are accessed in this way are linked dynamically with the Java programs that call them. So if these facilities are released under a GPL-incompatible license, the situation is like linking in any other way with a GPL-incompatible library. Which implies that:
  - 1. … you can state an explicit exception…
  - 2. If you wrote and released the program under the GPL, and you designed it specifically to work with those facilities, people can take that as an implicit exception permitting hem to link it with those facilities. But if that is what you intend, it is better to say so explicitly.
  - 3. You can’t take someone else’s GPL-covered code and use it that way, or add such exceptions to it. Only the copyright holders of that code can add the exception.”

Thunk vs. Trampoline (from The Jargon File)

- Thunk: (1) (original definition, per P.Z. Ingerman, inventor of thunks in 1961) "a piece of coding which provides an address"; created as a way of binding actual parameters to their formal definitions in Algol-60 when a procedure is called with an expression in the place of a formal parameter. The compiler generates a thunk to compute the expression and leave the address of the result in some standard location. (2) (generalization) an expression, frozen together with its environment, for later evaluation if/when needed (similar to closure). The process of unfreezing thunks is called "forcing". (3) a stubroutine… that loads and jumps to the correct overlay (compare trampoline)"
- Trampoline: An incredibly hairy technique… that involves on-the-fly generation of small executable (and, likely as not, self-modifying) code objects to do indirection between code sections. Under BSD and possibly other Unixes, trampoline code is used to transfer control from the kernel back to user mode when a signal (which has a handler installed) is sent to a process… it is said… that the trampoline that doesn't bend your brain is not the true trampoline.