# Correctness by Construction of High-Integrity Software

Rod Chapman

**Praxis High Integrity Systems**

# Contents

- Correctness by Construction
- The Catch...
- Languages
- SPARK
  - Design goals and features
  - Security
  - Projects & Theorem proving performance
- What's next
- Conclusions

# Correctness by Construction

- Observation:

- We can't rely on testing alone as the primary verification activity - much too expensive and risk prone.

- Also, for the most critical systems, testing can never generate sufficient evidence.
  - Some high-integrity standards call for probability of failure of $10^{-9}$ per hour.
  - $10^{9}$ hours is...?

# Correctness by Construction

- Observation 2:
- We normally have to produce evidence of fitness-for-purpose *before* any in-service experience.
  - For the NSA, FAA, MoD for example.
- "Patch it later" is *not* possible!

- We cannot depend on the evolution of ultra-reliability over many years and releases.

# Correctness by Construction (2)

- A design approach characterized by:
  - Use of static verification to prevent defects at all stages.
  - Small, verifiable design steps.
  - Appropriate use of formality.
  - “Right tools and notations for the job” approach.
  - Generation of certification/evaluation evidence as a side-effect of the development process. E.g. for a safety- or security-case.

# Some Praxis CbyC projects

- Typical defect rate in industry is > 5 defects per KLOC
- Typical productivity rate in industry is < 10 LOC per day
- Sample Praxis rates (for deployed, certified code, including all lifecycle phases and management overhead):

| Year | Project | Integrity | Size (sloc) | Defect/ ksloc | loc/day |
|---|---|---|---|---|---|
| 1992 | ATC display | SIL2 | 197,000 | 0.75 | 13 |
| 1997 | Helicopter landing system | SIL 4 | 27,000 | 0.22 | 7 |
| 1999 | Smart card security | ITSEC E6 | 100,000 | 0.04 | 29 |
| 2002 | Aircraft test set | SIL 0 | 35,000 | <0.1 | 28 |
| 2003 | Secure biometrics | CC EAL 5+ | 10,000 | 0.00 | 38 |

# A big new CbyC Project...

BBC NEWS | UK | England | Hampshire | New air traffic system unveiled

BBC

Home | News | Sport | Radio | TV | Weather | Languages

UK version | **International version** | About the versions

BBC NEWS

OPEN **The News in 2 minutes**

**News Front Page**

**Africa**
**Americas**
**Asia-Pacific**
**Europe**
**Middle East**
**South Asia**
**UK**
**England**
Northern Ireland
Scotland
Wales
UK Politics
Education
Magazine
**Business**
**Health**
**Science/Nature**

Last Updated: Wednesday, 7 March 2007, 10:44 GMT

E-mail this to a friend | Printable version

## New air traffic system unveiled

**A new £50m computer system that will revolutionise air traffic control was unveiled in Hampshire on Wednesday.**

The system will operate at the Swanwick air traffic control centre

National Air Traffic Services (NATS) at Swanwick, near Fareham, said the system will enable controllers to increase the amount of traffic they can handle.

Known as iFACTS (Interim Future Area Control Tools Support), the system has been described as the biggest change to air traffic control since radar.

It alerts controllers earlier to planes that are not on the right flight path.

# A big new CbyC Project...

- iFacts is probably the biggest active "Formal Methods" project in the UK, if not Europe.

- We *won* by bidding correctness-by-construction, formal methods, and SPARK, against very tough competition.

- All we have to do now is deliver it... ☺

 

# Correctness by Construction (3)

- Let's focus on what's achievable now.
  - Real languages with real tools that are fielded in industry right now.
  - Stuff that we know works at the highest safety-integrity/evaluation levels and is acceptable to the regulatory authorities.
  - Most high-integrity systems are also hard real-time and embedded.

 

# The Catch...

- Our ability to perform static verification critically depends on the language or notation under analysis.
- In particular, ambiguity in the definition of the language severely limits what is achievable.
- Ideally, languages and notations should be as unambiguous as possible.

# Ambiguity in Computing Languages

- This idea is not new…

*"… one could communicate with these machines in any language provided it was an exact language …"*

*"… the system should resemble normal mathematical procedure closely, but at the same time should be as unambiguous as possible."*

# Ambiguity in Computing Languages

- This idea is not new…

*"… one could communicate with these machines in any language provided it was an exact language …"*

*"… the system should resemble normal mathematical procedure closely, but at the same time should be as unambiguous as possible."*

*Alan Turing (1947)*

# Ambiguity in Software Engineering

- Unfortunately, ambiguity plagues us at every turn:
  - English requirements
  - UML and other “OO” notations
  - Programming languages
    - Does anyone understand C++ Templates?!?
- Machine code is often the first unambiguous representation we get, which can be tested but not much else…oh dear…

# Programming Languages...

- Standard languages?  C, C++, Java?
  - All fall down on ambiguity and therefore verifiability.
  - "Modern" language design is going the wrong way! E.g. OO polymorphism, exceptions etc.
- Special purpose languages?
  - Ever heard of "NewSpeak"?  Nope...

# Programming Languages...

- High-Integrity Language subsets?
  - Potentially combine the best of both worlds: desirable properties for H-I, using standard compilers, tools, staff etc.
  - Integrity achievable critically depends on selection of base language.
  - For the highest integrity levels, subsetting alone may not be enough.  Addition of annotations to strengthen the language ("design by contract"™) may be required.

# So…What is SPARK?

- The "SPADE Ada Kernel"
  - What does the "R" stand for?
- A sub-language of Ada95 with particular properties that make it ideally suited to the most critical of applications:
  - Completely unambiguous
  - All rule violations are detectable
  - Formally defined
  - Tool supported
- SPARK facilitates Correctness by Construction

# SPARK Design Goals

"Design goals….hmm…yes…

…you should definitely have some…"

Guy L Steele Jr, ACM PLDI 1994

# SPARK Design Goals

- Logical Soundness
- Simplicity of Language Definition
- Expressive Power
- Security and Integrity
- Formal definition
- Verifiability
- Bounded Space and Time
- Verifiability of Compiled Code
- Minimal Runtime Library

# SPARK Features

- Base language: ISO-8652:1995 Ada95
- Removes: Tasking, Generics, lots of tricky stuff…
- Limits: Some control flow structures, visibility rules etc.
- Adds: a language of annotations to allow efficient and deep static analysis, including information-flow analysis, and mathematical proof of program properties.
- Tool support: The SPARK Examiner, Simplifier and Checker

# SPARK Features (2)

- SPARK is statically free from all
  - Aliasing
  - Function side-effects
  - Erroneous behaviour
  - Implementation-dependent behaviour

- These analyses are all decidable in polynomial time. i.e. tool is very fast! This enables constructive use.

# Static Analysis of SPARK

- The Examiner tool implements a number of analyses, again all in P-Time:
  - Subset checking and static semantics
  - Information flow analysis
  - Verification Condition Generation - allows proof of properties such as exception freedom, partial correctness, and safety properties.
- Theorem prover tool (the Simplifier) does a good job of proving VCs.

# Exception freedom

- Exception freedom proof - why is it important?
  - Can be attempted without a formal spec., or explicit pre- and post-conditions, so is approachable.
  - Provides evidence that compiler-generated checks can be turned off with justification, or left on for "belt and braces."
  - Forces you to really think about your code. Correctness emerges.
- You mainly need CPU cycles for theorem proving - and these are cheap.

# SPARK and Secure Systems

- SPARK has many properties that make it ideal for the implementation of secure, embedded systems:
  - No data-flow errors.  A subtle and possibly covert source of information flow.
  - Verification of required information flow.  Very useful to support system and software partitioning.
  - Proof of the absence of exceptions.  Virtually free given theorem proving, and very worthwhile.
  - SPARK can be compiled with absolutely no COTS run-time library or operating system.  No acquisition or evaluation problem!

# SPARK Projects

- Military Aerospace:
  - EuroFighter Typhoon - nearly all critical systems are SPARK - about 5 Million lines of code.
  - Harrier II SMS.  Partly specified in Z and 100% implemented in SPARK. Approx 5000 VCs discharged in proof work.
  - SHOLIS - First Def Stan 00-55 SIL4 project. 9000 VCs proved, including top-level safety-properties, partial correctness, and exception freedom.  200 pages Z spec.
  - Aermacchi M346 – “Fly by wire” fast jet trainer

# SPARK Projects (2)

- Commercial Aerospace:
  - Lockheed Martin C130J Mission Computers and Bus-Interface units.
  - Rolls-Royce – Trent 1000 FADEC and EMU, Trent 900 EMU.
  - ARINC ACAMS (aircraft health monioring)

# SPARK Projects (3)

- Security:
  - The MULTOS CA.
  - All Praxis-generated deliverables to ITSEC E6.
  - Formal Security Policy in Z
  - Functional spec in Z (500 pages)
  - Concurrency design in CSP + Model Checking
  - 100,000 lines of code (mixed-language), 3500 person-days, 27 loc per day.
  - Only 4 defects 1 year after delivery, corrected under our warranty of course!

# So What's Wrong with SPARK?

- It's unfashionable, and British...
- "But we can't hire Ada programmers..."
- Selling an approach that slows coding (but speeds up whole-lifecycle) is very hard.
- Fear of formality. (Don't mention the "P" word!)
- Adopting CbyC and/or SPARK is a major life-style change for most organisations and engineers.

# SPARK at Universities

- So why should you/could you use SPARK in a University degree programme?
- How about...
  - High-Integrity Software Engineering (Safety, Security...)
  - Design-by-Contact
  - Embedded/Real-Time Software
  - "Formal methods"
  - Static Analysis
- Tools are free (as in “free beer”) to universities.
- There’s a really good book.

# SPARK at Universities

- Current research using SPARK
  - Proof Planning (Heriot-Watt)
  - Use of SMT decision procedures with SPARK VCs (Edinburgh)
  - Proof of floating point VCs (Aston)
  - Specification refinement (Virginia)
- “Interesting” research ideas
  - Counter-example finding
  - Auto test case generation
  - Language expansion: generics, interface types, polymorphism, ???

# SPARK at Universities (2)

- But no-one teaches Ada, right?
- Well…errr…the following *are* teaching SPARK (but don't tell 'em it's Ada!)
  - Manchester, (Old) York, (New) York, Virginia, Northern Iowa, Oakland, James Madison, Idaho, Roger Williams…and many more…

# Final Quote

"There is still no silver bullet, but dramatic improvements in software quality can be achieved through the rigorous and systematic application of *what we already know…"*

Martyn Thomas - the founder of Praxis.

# Resources

- Book: “High-Integrity Software: The SPARK Approach to Safety and Security” by John Barnes.
  ISBN 0-321-13616-0

- www.sparkada.com
  – Information
  – White papers and publications

# Praxis High Integrity Systems

20 Manvers Street
Bath BA1 1PX
United Kingdom
Telephone: +44 (0) 1225 466991
Facsimilie: +44 (0) 1225 469006

Website: www.praxis-his.com, www.sparkada.com

Email: sparkinfo@praxis-his.com