Going Large with Formal Methods on iFACTS

Roderick Chapman, Altran UK
Contents

- What is iFACTS?
- Formal Methods – Why Bother?
- Metrics and Issues
- Going Large?
- Conclusions
What is iFACTS?

- iFACTS provides advanced tools support to en-route air-traffic controllers at the London Area Control Centre
  - Trajectory Prediction
  - Medium-Term Conflict Detection
  - Electronic Flight Strip Management

- Or more clearly…
Two Control Centres – Prestwick and Swanwick
Picture credits: NATS.
Swanwick Area

Handles on average 5,500 flights each and every day of the year

Controls 200,000 square miles of airspace above England and Wales including the complex airspace of London
Swanwick Centre
Before iFACTS…
After iFACTS...spot the difference...
iFACTS Functions

- Advanced electronic prediction and decision support tools.
- Changed method of operation.
- Increased capacity.
- Reduced fuel burn through less interaction.
- Introduction must cause minimal ATC delay and disruption to the 24/7 service.
iFACTS – Medium-Term Conflict Detection: Separation Monitor Window
Contents

- What is iFACTS?
- Formal Methods – Why Bother?
- Metrics and Issues
- Going Large?
- Conclusions
Formal Methods on iFACTS

- Two main uses of “Formal Methods” in iFACTS
  - Functional Specification in Z with English commentary
  - Implementation in SPARK 2005
    - Strong static verification and proof of properties

- Why bother?
So why bother with FM?

\[ \Delta IDStation; \Delta RealWorld | \\
\text{TISOThenUpdate} \\
\land \ latched ^\text{locked} \land \ latched' = \text{unlocked} \\
\|- \\
(\exists \text{ValidToken} \bullet \ \text{goodT}(\theta\text{ValidToken}) = \text{curi} \\
\land \ \text{UserTokenOKNoCurrencyCheck} \\
\land \ \text{FingerOK}) \\
\lor \\
(\exists \text{TokenWithValidAuth} \bullet \ \text{goodT}(\theta\text{TokenWith} \\
\land \ \text{UserTokenWithOKAuthCertNoCurrencyCheck} \\
\lor \\
(\exists \text{ValidToken} \bullet \ \text{goodT}(\theta\text{ValidToken}) = \text{curi} \\
\land \ \text{authCert} \neq \emptyset \land (\text{the authCert}).role \\
\text{See: TISOThenUpdate (p. 5), UserTokenOKNoCurrencyCheck, UserTokenWithOKAuthCertNoCurrencyCheck (p. ..) \right)\]
So why bother with FM?
Thinking and Tooling Exposes…

Ambiguity…
Thinking and Tooling Exposes…

Contradiction…
Thinking and Tooling Exposes…

...particularly *assumptions* that you didn’t know about…but really should be written down and validated…
Contents

- What is iFACTS?
- Formal Methods – Why Bother?
- Metrics and Issues
- Going Large?
- Conclusions
iFACTS Timeline


- October 2006 – Implementation Project starts

- December 2011 – Fully Operational
  › 24/7 on all sectors with all controllers

- January 2012 and ongoing – Maintenance and upgrades.
Headcount...

- How many “Formalists” do you need?

- Specification team – key “FM skills”
  - requirements elicitation
  - Abstraction
  - Z authoring

- Peak size: 12 people, including 4 NATS employees.

- Now 3 people during maintenance phase.
Headcount…

- How many “Formalists” do you need?

- Implementation team – key “FM skills”
  - reading Z
  - test case design
  - SPARK design, implementation and proof.

- Peak size: 130, spread across 4 sites, in 3 timezones.

- Now: 7 people.
Specification Size

- Specification: what do you count?

- We found that “Delta Z” (Added and Modified lines of Formal Text) was an excellent proxy measure that correlated with effort for changes.

- If you printed it all out, the Z functional specification is over 4000 pages.
Training experience

- **Z reader** and **writer** training are separate and very different courses.

- **Z Reader Training:**
  - 3 day course. We find reasonably fluency after 1 week on the job
  - 57 Engineers trained to read Z, including contractors
  - Also trained NATS Domain Experts and Controllers to read Z so they could review the specification – essential
Training experience

- Z reader and writer training are separate and very different courses.

- Z Writer Training:
  - 3 day course. Fluent and productive with 3 months on the job
  - 11 Engineers trained, including NATS staff
Code Size

- Implementation is a mix of
  - SPARK 2005
  - Full Ada (a few modules impractical to write in SPARK – e.g. OS library interfaces)
  - MISRA C (small GUI “Glue” layer)
Code Size

- The SPARK and Ada Code is:
  - 890k “raw” lines of code
    - of which
      - 116kloc blank
      - 171kloc comments
      - 74kloc SPARK contracts
      - 529kloc “code”
    - of which
      - 250kloc declarations and statements (aka “logical loc”)
SPARK Analyses and Proof

- Data- and Information-Flow
  - No uninitialized variables
  - Verification of intended information flow

- Concurrency
  - No deadlocks
  - No priority inversion or unbounded blocking
  - (See Ada’s “Ravenscar Profile”)

- Memory consumption
  - No pointers, no “heap”, so no worries!
  - Worst case stack usage analysis
SPARK Analyses and Proof

- Proof of “no runtime errors” aka “type safety” in addition to all of SPARK’s type checking rules:
  - Prove no buffer overflow, arithmetic overflow, division by zero etc.

- SPARK Code generates
  - 152927 Verification Conditions
    - of which
      - 151026 (98.76%) are proven automatically
      - 1701 proven by a user-defined lemma
      - 200 “reviewed”
SPARK Analyses and Proof

- All coders *must* prove 100% VCs OK *before check-in.*

- * Entire proof can be reproduced in less than 15 minutes.*
  
  ›  Strict Modularity
  
  ›  Parallelization (Got 152927 processor cores? Great!)
  
  ›  Distributed and persistent caching of proof results.

- “Overnight” proof run clears the cache and rebuilds all analyses and proofs from scratch.
Contents

- What is iFACTS?
- Formal Methods – Why Bother?
- Metrics and Issues
- Going Large?
- Conclusions
Going Large?

- So what does “Going Large”? Mean

- For us...the fact that *no one person understands everything* on a project.

- Some have a broad but shallow understanding of the whole system and its context.

- Some have very deep knowledge of some components.
Contents

- What is iFACTS?
- Formal Methods – Why Bother?
- Metrics and Issues
- Going Large?
- Conclusions
Conclusions – Formal Methods on iFACTS

- It can be done!

- Tools and Languages must be designed to scale up. This does not happen by chance.

- Training people to read and write formal notations is achievable, even for customers.
  - It’s only discrete math after all...
  - The notation may seem like a barrier at first, but it’s not really.
  - It’s the thinking that counts.
  - Abstraction remains the key skill of system and software engineering.