Assurance of COTS software towards mathematical validation

Victor Yodaiken Finite State Research Austin TX yodaiken@fsmlabs.com

Plan:

What is the problem we want to solve?

What are some fun properties?

Why formal methods are so weak

Steps forward

What are we doing?

1)Trying to understand our design objectives

- 2)Trying to get some assurance that the design meets the design objectives and find places where we need to compensate
- 3)Trying to get some assurance that the code implements the design
- 4) Managing and evaluating risk.

What are we doing?

- 1)We're interested in commercial enterprise systems.
- 2)Those systems are constrained by cost and time to market
- 3)Those systems are not going to be written from scratch to DO-178B/EAL7 specs (assuming the unproven claim that such efforts produce better code)
- 4)We're going to be piling Linux, Oracle, Apache, Windows, MySQL, Java ... together.

What are we not too doing?

- 1)Eliminating all doubt and attaining 100% certainty
- 2)Getting rid of the need for good design: some designs will be more assurable than others.
- 3)Getting rid of the need for good programming: see above

Most ignored true statement in programming:

THERE IS NO SILVER BULLET

Why is this so hard?

- 1)Engineering is inherently failure prone: bridges still fall down.
- 2)Complex systems have ridiculously high numbers of states. Interesting software is large and complicated.
- 3)Components of discrete state systems do not have nice dumb additive properties like those found in physics.
- 4)Discrete mathematics is either intractably hard or in a primitive condition.

Fun properties

- 1)Live: Every Linux/Windows process in set P gets at least 1 millisecond of compute time every second.
- 2) No Linux process ever writes on the password file unless it is running an executable in set S
- 3)The highest priority real-time thread is never waiting more than T microseconds to run and once it starts runs to self-suspension unless a higher priority threat preempts it.

Use traditional engineering methods

- Break problem into more tractable parts
- Defense in depth
- Cross check
- Regression tests and coverage
- Informal testing.
- Try for precision and arithmetic in specs

An enterprise configuration may be small



An enterprise configuration may be large



Assurance in depth strategy

- RTMS is 80% coverage with extensive regression and stress test. Increase this. Add static checker.
- Linux/BSD base is stressed and used to run software which can fail. Develop automated regression and static check – there is no excuse for the absence of these.
- RT layer software monitors for correct system operation and can force reboot
- Security add-ons like SELinux software can run on the Linux layer for interlocking security

Assurance in depth strategy

- Validate what can be validated take advantage of the smaller and simpler RT base.
- Place critical code in validated environment
- Cross-check between domains and on semantic levels.

Numbers

- Try to express properties numerically: delays, max processes, frequency of an event, ...
- State machines what everyone is converging on anyways
- Use recursion to describe state change instead of listing states: After an A transition the output of X is equal to F(a,current output of X current output of Z)
- Use recursion for composition
- www.yodaiken.com/w

Final comments

FSMLabs sales@fsmlabs.com yodaiken@fsmlabs.com www.fsmlabs.com (347) 404-5376 Fax: (866) 724-4435