Automatic Formal Verification for EPICS

Jonathan Jacky, Stefani Banerian,
Michael Ernst, Calvin Loncaric, Stuart Pernsteiner,
Zach Tatlock, Emina Torlak

Department of Radiation Oncology
University of Washington Medical Center
Department of Computer Science and Engineering
University of Washington
jon@uw.edu, http://staff.washington.edu/jon/

Topics

Four related projects:

- Radiation therapy machine control system, a safety-critical medical application, that uses EPICS.
- New tool for finding functional errors in EPICS databases, made with *satisfiability checker* technology.
- Intensive review and test of the EPICS database engine, using a new purpose-built re-implementation of (parts of) EPICS, made with *theorem prover* technology.
- New tool for compiling an EPICS database to a standalone program that runs without (parts of) EPICS, made with *verified compiler* technology.

Nothing is ready to distribute yet, but we are seeking collaborators to help use and evaluate the tools.

UWMC Clinical Neutron Therapy System (CNTS)

Hospital-based cyclotron and neutron radiation therapy

CNTS Control System: Safety requirements

We must ensure that the we satisfy this overall safety requirement:

- *The neutron beam can only turn on or remain on when the machine setup matches a prescription that has been selected by the operator.*

This overall requirement is composed of hundreds of detailed requirements, for example:

- *The actual gantry angle must match the prescribed angle within a given tolerance, when the machine is in therapy mode and that setting has not been overridden and . . .*

Checking all these requirements is a formidable task.

EPICS and safety-critical systems

Is it advisable to build a safety-critical system with EPICS?

Conventional wisdom says no:

2008: "(EPICS) code is not rigorously audited to the standards . . . that would be needed (for medical applications). . . ."
`epics/tech-talk/2008/msg00803.php`

2012: "EPICS should never be relied on for safety-critical operations . . ."
`epics/tech-talk/2012/msg01836.php`

But we did anyway — explanation follows.

We did audit some EPICS code – in an innovative way.

CNTS Control System: History

We have more than thirty years of experience with three generations of control systems, including two that we developed:

1984 Scanditronix PDP11/RSX/Fortran
with UWMC custom RTP via DECNET

1999 UWMC VME(68k)/VxWorks/C
with UWMC custom Prism RTP via NFS
new functional spec, workflow, UI, hw, sw

2015 UWMC X86/Linux/EPICS
with commercial Pinnacle RTP via DICOM
new hw, sw *but* very similar functional spec etc.

All use relays, nonprogrammable hardware, or PLCs where feasible.

For prescription data, general purpose computing is unavoidable.

CNTS Control System: Limit Complexity

Re-implement almost the same functionality as the 1999 system. But EPICS is more complicated than C on Vxworks. So —

Use a simple configuration:

- Therapy control program runs on one soft IOC
- Therapy IOC is the only application running on its computer
- Therapy IOC does not require any clients or other IOCs to maintain or achieve safe state

Use a minimal set of EPICS constructs:

- Only database records, StreamDevice .proto files, st.cmd
- Database DB links only, no CA links
- Data flow is all "push": SCAN PASSIVE, INPx NPP, OUT PP, FLNK
- No SNL, no subroutine records, no custom device support
- Only these record types: ai, ao, bi, bo, mbbo, longin, longout, stringin, stringout, calc, calcout, acalcout, scalcout, fanout, dfanout, seq, asyn

Automated Formal Verification

We *believe* good design and programming practices prevents errors.

Do we have *evidence* that we have met the safety requirements?

Concern: Reviews are *subjective*, testing is *sampling*, so coverage is incomplete.

Remedy: Use *automated formal verification*: express requirements in *formal* notation, check them against pertinent source code *automatically*, so coverage can be more thorough.

A multi-year research effort by several computer scientists experienced in formal verification techniques and technology.

Includes review, analysis, and re-implementation of parts of EPICS.

Automated Formal Verification Tools for EPICS

The project is producing several tools, including:

To check application programs : The *Symbolic Evaluator* detects errors in EPICS databases.

To check the EPICS database engine: The *Verified Interpreter* is re-implementation of the EPICS database engine, that checks the standard distribution by differential testing.

To reduce the EPICS trusted core: The *Verified Compiler* compiles an EPICS database to a standalone program, removing or replacing parts of the EPICS runtime.

EPICS symbolic evaluator

New tool for finding functional errors in EPICS databases.

Inputs:

- EPICS database: *.db .substitutions .template st.cmd*
- Property: assertion relating PVs, for example –

  (> (diff gantry-prescribed gantry-actual) gantry-tolerance)
  (=> . . . interlock set . . .)

Output:

- *everything is ok!* – property is satisfied – *OR . . .*
- Counterexample: PVs with values that violate property –

  Iso:GantryCouch:Gantry:Prescribed.VAL = 312 [64-bit]
  Iso:GantryCouch:Gantry:Actual.VAL = 48 [64-bit] . . .

Status:

- Working, found a serious error missed by reviews and testing
- Still improving feature coverage, user interface, workflow

EPICS symbolic evaluator (2)

Testing: Try to guess a test case that violates the assertion.
Passing tests are *not conclusive*.

Symbolic evaluator: Finds the test case (counterexample) that violates the assertion (property), if there is one.
Verified properties are *conclusive*.

How it works:

- Uses *satisfiability checker*: *Rosette* with *Z3* SMT solver.
- From EPICS database, generate a *symbolic program* where every PV value is represented by a *formula* that includes all its potential inputs upstream.
- Submit this symbolic program (full of formulas) and the property to check (more formulas) to the SMT solver.
- SMT solver searches for values of PVs that satisfy database formulas but violate property formulas.

EPICS verified interpreter

Re-implementation of EPICS database engine, made with *theorem prover* technology, for checking standard EPICS distribution.

- Review *Record Processing* sections from RRM
- Express as definitions for *Coq* theorem prover
- Write interpreter in Coq programming language
- Prove interpreter correct with respect to definitions
- Extract executable interpreter (in Haskell) from Coq proof
- Build harness for differential testing against EPICS distribution
- Test cases are randomly generated databases of five records

Status:

- Working, over 20 million test cases, dozens of discrepancies
- Most discrepancies resolved by rereading docs and EPICS code

Conclusion:

- EPICS works like RRM says (except a few corner cases?)

EPICS verified compiler

New tool for compiling an EPICS database to a standalone program that runs without (parts of) EPICS, made with *verified compiler* technology.

Motivation: Use smaller trusted core (runtime)

Plan:

- Start from same semantic definitions as verified interpreter
- Use *CompCert* verified C compiler back end
- ...

Status:

- Early development, no compilation yet

Conclusions

EPICS databases (data flow programs) can be formally verified —

EPICS databases can be used for safety-critical computations —

— *When used in our restricted programming style.* It ensures:

- No unbounded allocation of resources.
- Processing of every event terminates.
- No loops.
- etc. . . .

Similar restrictions are often recommended for safety-critical programming. EPICS database programs readily support them.

Tools are not ready for general distribution, but we are seeking collaborators to help use and evaluate them.

Contact *jon@uw.edu*.