Checking Model Specifications with CrossCheck™

SBIR funded by Air Force Research Laboratory

Jonathan Springer James Ezick
Reservoir Labs, Inc.

Matthew Craven Rick Buskens
Lockheed Martin

CrossCheck

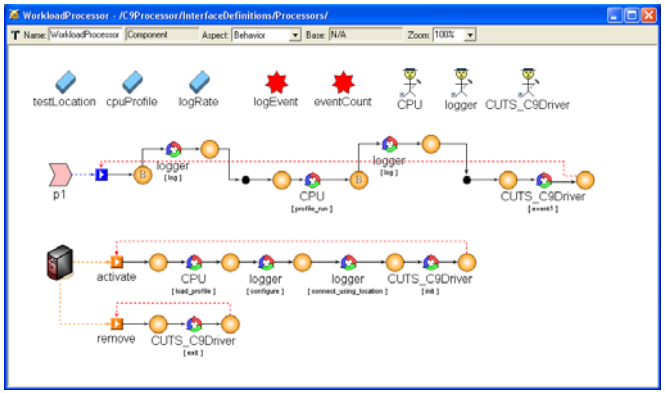
CrossCheck: a dynamic specification checker [1]

- Specifications defined in terms of events transmitted by application
- Events can be normal application output, or abstractions generated with code inserted into the application
- CrossCheck checking engine can be remote, over the network

Modeling with CUTS

Problem: It is difficult to verify the behavior of a complex design prior to implementation. But design errors can be very costly.

- **Model based simulation** allows early validation of designs
- CUTS (Component Workload Emulator Utilization Test Suite) is a model-based tool for emulating systems of components to evaluate workloads at the design stage [2]



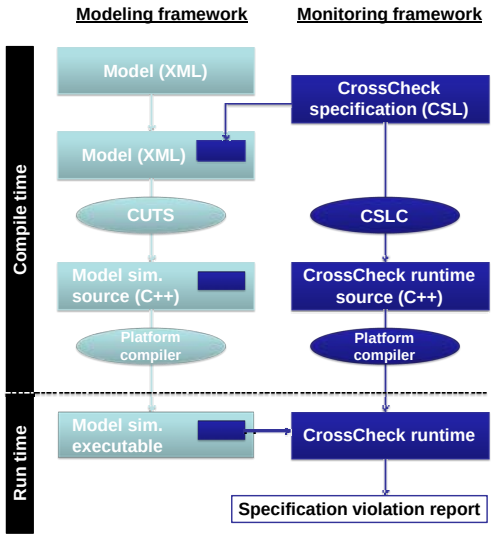
- CUTS model is described in XML
  - Model is created using GME, a GUI editor
  - Larger models are created programmatically using the GAME API
- CUTS can "compile" model to code that will simulate it
  - Allows behavior to be examined
  - Still difficult to analyze this behavior, mine the data

  → Motivates use of *CrossCheck*

SPRUCE Challenge Problem

- **SPRUCE**: a collaborative workspace hosting challenge problems for software intensive systems [3]
- **Challenge problem**: an avionics system specification
  - A collection of ~11,000 signals transmitted among ~20 processors
  - About 2000 signals are periodic, requiring a minimum rate
- **We encoded a model of the signal traffic in CUTS, using GAME**
- http://www.sprucecommunity.org

Model Instrumentation



The CrossCheck dynamic specification checker is integrated to check properties of the simulation as it executes.

- Specifications written in CrossCheck Specification Language (CSL)
- Instrumentation is added to the model to turn model messages into CrossCheck events
  - ***We created reusable CUTS components that generate CrossCheck events, easing instrumentation***
- Specifications are checked dynamically as the model is simulated

Property Specification

Specifications for the challenge problem are written in **CSL**.

- Check that minimum message rate is maintained

CSL describes:

- Event structure
- Sequences of events to match
- Specific checking actions in a C-like syntax

The CrossCheck engine **maintains state** about the progress of the simulation.

```
P17_P21_1(timestamp_ms:uint64);;
P17_P21_2(timestamp_ms:uint64);;
%%
SpruceMsgPred <-
  <predicate_p_true>?:<msg_pred_f> ;;
SpruceMsgRateRule := SpruceMsgPred,
  group::1,
  attr::{oldest_only, rollback},
  recover::<rate_sanity_recover_f>,
  desc::"Check message rates from model" ;;
%%
DECLARE_PREDICATE_F(msg_pred_f, m, C, s) {
  switch (s->type) {
    case ET_P17_P21_1:
      …
      rate = ceil (db_count.u.Uint32 /
                   (total_time/1000.0));
      if (rate < min_rate) {
        C = context_update(C, "RATE", rate);
        return C;  // spec violation
      }
      …
  }
  …
}
```

Example CSL block with embedded C code from a SPRUCE specification

Simulation Results

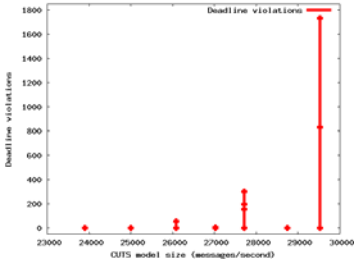


Figure 1: Number of deadline violations given SPRUCE CUTS model size in term of messages exchanged per second

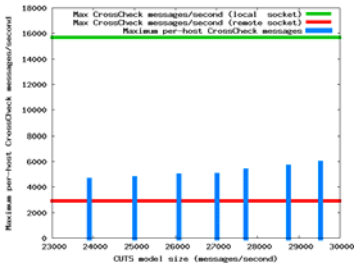


Figure 2: Maximum rate of CrossCheck messages sent per-host

Conclusions

- Through CSL, CrossCheck provides a mechanism to **specify important properties** of the SPRUCE CUTS simulation model
- **Integrating CrossCheck with CUTS is straightforward** and relies upon components with well-defined interfaces that can be reused in any CUTS model
- As the CUTS model size increases, the number of messages exchanged per second increases, leading to message deadline violations in some simulation runs. **CrossCheck correctly detects and reports these violations to indicate whether or not the simulation was successful (Figure 1)**
- By including a CrossCheck Runtime instance on each simulation node, there is more than ample processing capability for the SPRUCE CUTS model. **CrossCheck is suitable for the verification of models with high per-second message rates such as the SPRUCE CUTS multiprocessor avionics model (Figure 2)**

References

1. J. Springer, J. Ezick, D. Wohlford, M. Craven, and R. Buskens, "CrossCheck: Improving System Confidence through High-Speed Dynamic Property Checking", in *High Performance Embedded Computing Workshop*, Sept. 2008.
2. J. Hill, H. Turner, J. Edmondson, and D. C. Schmidt, "Unit Testing Non-Functional Concerns of Component-based Distributed Systems," in *Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation* (ICST), Denver, Apr. 2009.
3. P. Lardieri, R. Buskens, W. McKeever, S. Drager, "SPRUCE: A Web Portal for the Collaborative Engineering of Software Intensive Systems Producibility Challenge Problems and Solutions," in *Proceedings of the 2009 IEEE Conference on Collaborative Technologies and Systems*, Baltimore, May 2009.

Acknowledgement: Thanks to Dr. James Hill, Department of Computer Science, Indiana University/Purdue University at Indianapolis for help with the CUTS framework.