Checking Model Specifications with CrossCheck

Jonathan Springer, James Ezick
Reservoir Labs, Inc.
{springer, ezick}@reservoir.com

Matthew Craven, Rick Buskens
Lockheed Martin
{matthew.craven, rick.buskens}@lmco.com

Summary

Model simulation frameworks are becoming more common as a way to evaluate the behavior of a complex system prior to its actual implementation. Simulation uncovers design errors early, especially important in embedded environments where post-development errors are costly to fix. While the formalization of a system as a model is becoming more systematized, the characterization of its behavior is still *ad hoc*. We have applied our dynamic specification checking technology CrossCheck to the CUTS [2, 3] model simulation framework as a way of verifying properties of interest in a model.

CrossCheck [1] utilizes a specialized language to express specifications, which are properties about the behavior of the system under check. CrossCheck specifications are compiled to generate a very efficient checking runtime, which receives and processes events from the instrumented CUTS model environment. By combining CrossCheck with the CUTS framework, we were able to develop specifications alongside the model that could be automatically verified during simulation runs.

CUTS Model Simulation Framework

It is difficult to evaluate the viability of an embedded system design prior to its actual implementation, and after it is implemented, unforseen problems result in very costly reimplementation and schedule overruns. CUTS is a model-based tool for emulating systems of components to evaluate workloads at the design stage. In CUTS, a model of the system is written, from which a complete simulation environment can be created automatically. This model uses a collection of related technologies, including domain specific modeling languages (DSMLs) and related tools (CoSMIC) and CORBA. The model itself, which has an XML representation, may be manipulated with a GUI editor or generated programmatically. An example view of part of a model (for an avionics system, described subsequently) is given in Figure 1.

After the model has been created, the CUTS code generator is invoked. This phase examines the model and generates C++ code that, when compiled and run, will simulate the model. This run takes place in a simulation environment which resembles the deployment environment of the system being modeled, to the extent practical.

Unclassified, Distribution Statement A: Approved for public release; distribution is unlimited. This material is based upon works supported by the Department of Defense under contract numbers FA8750-06-C-0133 and FA8750-07-C-0049. Any opinions, findings and conclusions expressed in this material are those of Reservoir Labs, and do not necessarily reflect the views of the Department of Defense.

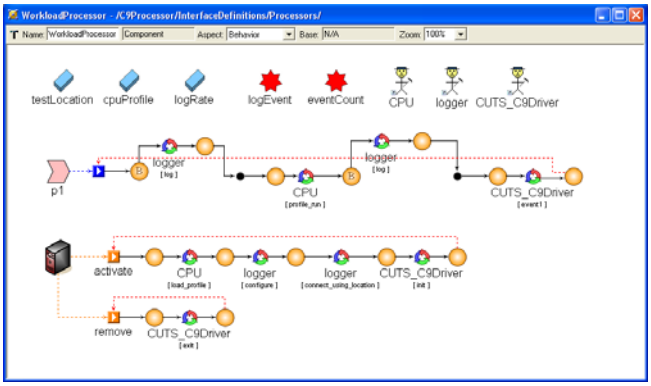

Figure 1: Signal processor model component (viewed with the GME model editor).

Instrumentation with CrossCheck

While CUTS automates the synthesis of the emulation environment, checking for most properties of interest must be done manually. The simulation itself provides a check only on the basic operation of the system. Essentially, CUTS provides the means to generate a tremendous amount of data about the actions of the system being modeled, but it is up to the user to filter or interpret this data.

One obvious approach to applying CrossCheck to CUTS is to modify CUTS to insert calls to CrossCheck in the simulation code that it generates. However, this is not ideal as it requires a level of indirection to address. CrossCheck driver calls are conceptually specific to the model, and different models will want to emit different events. Thus it would be necessary to develop some abstraction that could take a CrossCheck specification together with some annotation about where it should be plugged into the model.

Instead, we opted to apply the strength of the modeling environment itself, creating a model component for the CrossCheck driver. This CrossCheck model component can be hooked up in a user model using regular modeling tools (such as GME). To support this usage mode, we added a CUTS-aware wrapper to the standard CrossCheck C/C++ driver library. This wrapper interfaces between the modeling software environment and the CrossCheck driver, allowing calls to CrossCheck to be edited into a model from within the model development environment. This integration architecture is outlined in Figure 2.

Evaluation

We applied CrossCheck to a CUTS model of an avionics system. This model consists of a collection of communicating components, with signal rates and processor workloads. This model was developed as a part of the SPRUCE project [4], led by Lockheed Martin, and was made according to a system specification provided by the

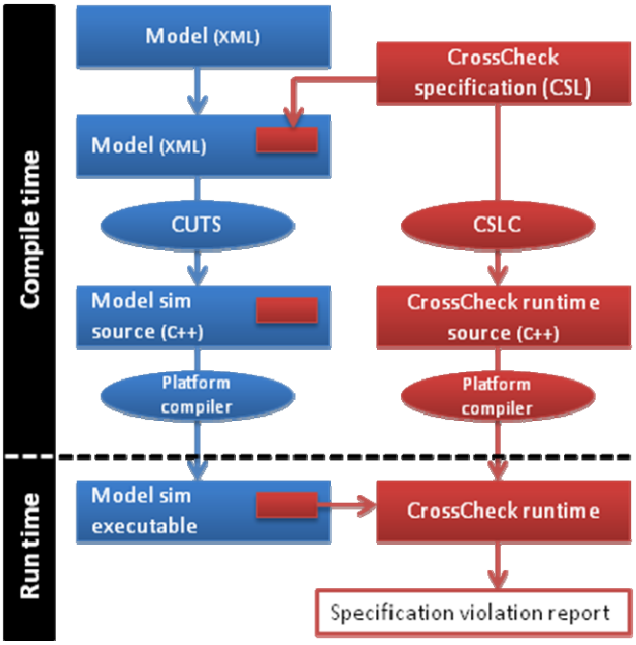

Figure 2: Architecture for integration of CrossCheck and CUTS model environment.

project. The specification defines system components and their communications; these components and signals represent all high-level flight control operations. The model thus serves as a basis for evaluating the viability of the flight control design, so that problems such as exceeding guaranteed response times can be found early, in time to correct the design before costly development is done.

In the system specification to which the model was written, a certain class of signals (communications between components) is required to sustain a certain rate (i.e. messages per second). We wrote a CrossCheck specification to check that particular property of interest; Figure 3 shows a representative portion of the specification. In this example, the model must transmit signals from one model component to another. When the signals arrive at a component, they pass through a series of "action" states that allow model behavior to be triggered. We added an action state that generated a CrossCheck event representing the fact that a signal (message) had been received. Thus, signals become CrossCheck events, relayed to the CrossCheck runtime by the wrapper driver. The runtime checks the timestamps on the events, as directed by the specification, and together with its knowledge of the history of the event traffic (i.e. signals received by the component in the model), is able to determine if the specified rate is being maintained. If not, it triggers a specification failure.

We tested our instrumented model in a series of simulation runs, and were able to verify the specification property. To determine that CrossCheck was able to keep up with the simulation, we gradually increased the model message frequency from the default of 50Hz up to 500Hz. Running with and without CrossCheck, we observed that specification checking completed successfully and did not affect the simulation rate. CrossCheck response time over the runs averaged 350µs.

```
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
      }
      …
} … }
```

Figure 3: Specification extract.

Conclusions

A key challenge in embedded system development is catching design errors early, to minimize development time wasted on unworkable designs. Being able to model the system is a powerful tool, but only part of the solution. By combining model simulation with specification checking, simulation runs can be interpreted more meaningfully.

In applying a specification checking framework, usability is an important and often-neglected consideration. We found that by building a reusable wrapper, a model builder could easily connect to CrossCheck without any custom coding, using the modeling tool environment.

Writing the Crosscheck specification was straightforward, though manual. Future work in this area might focus on even tighter integration, creating modeling domain-specific tools that build the specifications into the model.

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] J. Hill, J. M. Slaby, S. Baker, D. C. Schmidt, "Applying System Execution Modeling Tools to Evaluate Enterprise Distributed Real-time and Embedded System QoS," in *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Sydney, Aug. 2006.

[4] 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.