Modeling Time-Triggered Protocols and Verifying Their Real-Time Schedules

Lee Pike
leepike@galois.com

Galois Inc.

November 14, 2007

Time-triggered systems

Time-triggered systems are distributed systems in which the nodes' local clocks stay synchronized within some bound.

Characteristics include:

:) Behavior is driven according to the passage of time and a globally-known schedule.

- Protocols execute in *rounds*; each round has a communication and computation phase.
- Opposed to event-triggered behavior, which is driven by the occurrence of events.

:) Allows real-time behavior & and fault-tolerance to be treated at the platform level rather than being application-specific.

- Predictable and analyzable since they're "almost synchronous."
- Relieves application programmers from dealing with these issues.
- Makes reasoning about fault-tolerance easier.

:(Worse performance under unusual/peak workloads.

| galois |

How the paper came about

- I was formally verifying NASA Langley's SPIDER time-triggered protocols.
- ...And I thought I'd apply John Rushby's paper, *Systematic formal verification for fault-tolerant time-triggered algorithms* (IEEE TSE, 1999).
- ...But I realized I needed to extend the theory.
- ...So starting with Rushby's original specs and proofs, I added my own axioms and generalized some of the existing theory.

The story (continued)

- …Worried that I'd introduced an inconsistency, I did a formal theory interpretation (i.e., show the axioms have a model) in PVS to prove the consistency of my added axioms.
- …But I could find no satisfying model!
- …But my axioms *looked* right.
- …After longer than I'd like to admit, I realized Rushby's axioms were inconsistent: 3 of the 4 system assumptions were inconsistent.
- …So I wrote a note to IEEE TSE (2006) mending the axioms.

An example inconsistent axiom

- *Inverse Clock*: a total function from *realtime* to *clocktime*: $C_p : \mathbb{R} \rightarrow \mathbb{N}$.
- The drift of nonfaulty clocks is bounded by a realtime constant $0 < \rho < 1$.
- *Clock Drift Rate* axiom: For all realtimes t_1 and t_2, $(1 - \rho)(t_1 - t_2) \leq C_p(t_1) - C_p(t_2) \leq (1 + \rho)(t_1 - t_2)$.

The axiom is inconsistent, in three separate ways!.

Proof.
One proof is...

$\square$

An example inconsistent axiom

- *Inverse Clock*: a total function from *realtime* to *clocktime*: $C_p : \mathbb{R} \rightarrow \mathbb{N}$.
- The drift of nonfaulty clocks is bounded by a realtime constant $0 < \rho < 1$.
- *Clock Drift Rate* axiom: For all realtimes t_1 and t_2, $(1 - \rho)(t_1 - t_2) \leq C_p(t_1) - C_p(t_2) \leq (1 + \rho)(t_1 - t_2)$.

The axiom is inconsistent, in three separate ways!.

Proof.

One proof is... By contradiction. Let $t_2 > t_1$.

$\square$

An example inconsistent axiom

- *Inverse Clock*: a total function from *realtime* to *clocktime*: $C_p : \mathbb{R} \to \mathbb{N}$.
- The drift of nonfaulty clocks is bounded by a realtime constant $0 < \rho < 1$.
- *Clock Drift Rate* axiom: For all realtimes t_1 and t_2, $(1 - \rho)(t_1 - t_2) \leq C_p(t_1) - C_p(t_2) \leq (1 + \rho)(t_1 - t_2)$.

The axiom is inconsistent, in three separate ways!.

Proof.
One proof is... By contradiction. Let $t_2 > t_1$. Then $(1 - \rho)(t_1 - t_2) > (1 + \rho)(t_1 - t_2)$, so there is no $i \in \mathbb{N}$ between the bounds. $\square$

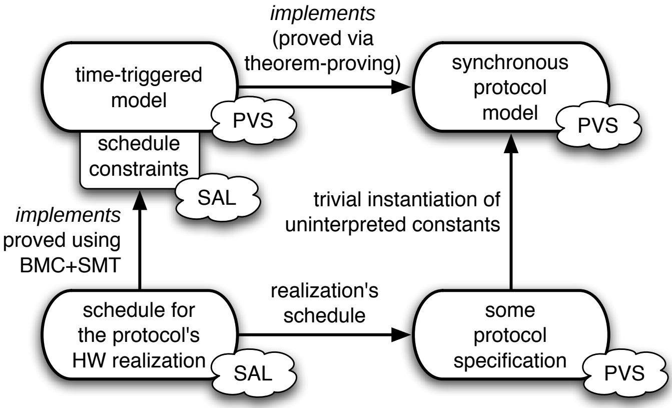
John Rushby

Please keep in mind

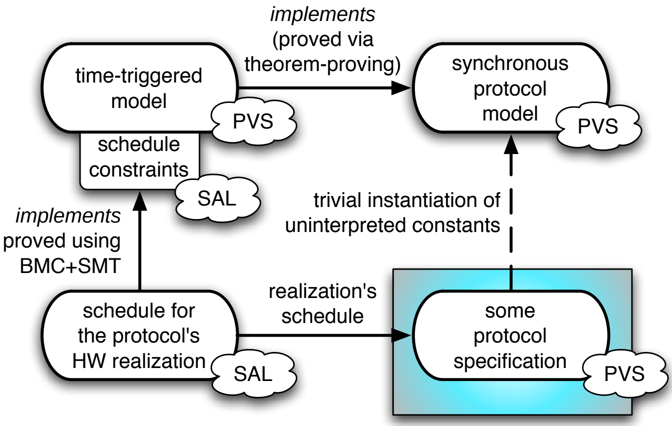
- The concepts behind Rushby's theory were correct.
- I uncovered the errors because Rushby publicized his specs (thanks!).
- Referees (twice) overlooked the errors, as well as researchers citing the work (including me).
- Rushby's work in time-triggered system verification is seminal (like his work in security, mechanical theorem-proving, etc.).

Okay, onto the actual methodology. . .

Verification approach



Verification approach



galois

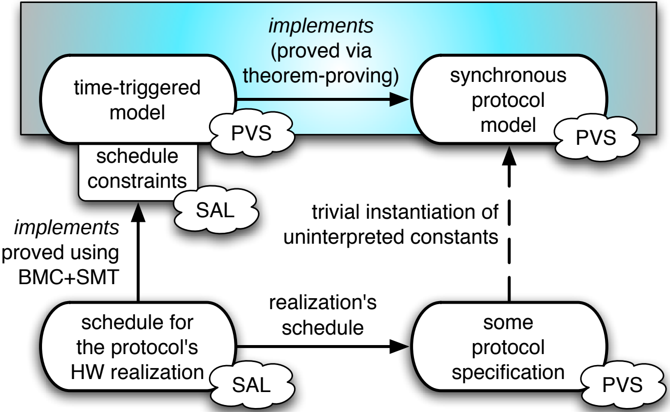
Synchronous specification

$$
\begin{aligned}
& run(r, s) \stackrel{\text{df}}{=} \\
& \quad \text{if } r = 0 \text{ then } s \\
& \quad \text{else } \lambda p.\ trans_p(run_p(r-1, s), \\
& \qquad\qquad\qquad\qquad \lambda q.\ msg_q(run_q(r-1, s), p)), \\
& \text{where } q \in in\_nbrs_p
\end{aligned}
$$

$trans_p$: state-transition function for node p.
msg_q: message-generation function for node q.
in_nbrs_p: inbound-neighbor-nodes for node p.

galois

Part I: the theory



Part I: the theory

The theory Rushby built and I expanded upon is an axiomatic theory of time-triggered protocols. The axioms fall into the following categories:

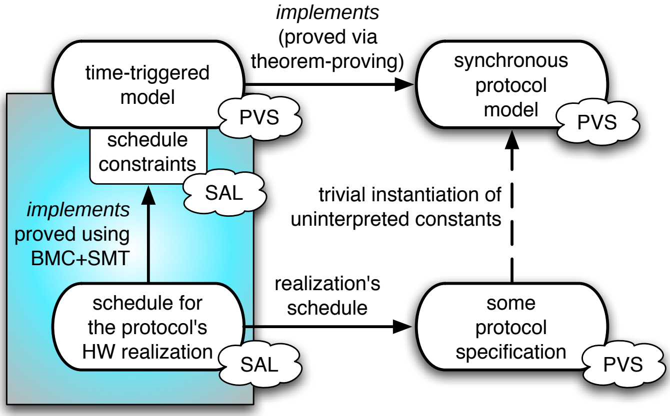
1. System assumptions (4 axioms)
   Assumptions made about the underlying clocks—e.g., clock monotonicity, drift rate, skew bound, & communication delay.
2. Schedule constraints (6 axioms)
   Constraints on the local schedules—e.g., computation phase, communication phase, reception windows, & pipelining.
3. Semantics (9 axioms)
   The semantics of time-triggered behavior is a transition system, the states of which are (s, t) where s is the global state of the system and t is a realtime. The transitions between states are constrained by the axioms.

The simulation theorem

Via taking "cuts" at regular points in time:

$$
\begin{array}{ccccccccccc}
s_0 & & \longrightarrow & & s_1 & & \longrightarrow & & s_2 & \rightarrow & \cdots \\
\Uparrow & & & & \Uparrow & & & & \Uparrow & & \\
(s_0, t) & \rightarrow & \cdots & \rightarrow & (s_1, t') & \rightarrow & \cdots & \rightarrow & (s_2, t'') & \rightarrow & \cdots
\end{array}
$$

Part II: the implementation



SAL in a slide

SRI's Symbolic Analysis Laboratory (SAL) is a GPL open-source high-level language interface & model-checking tools:

- The language includes predicate subtypes, higher-order functions, algebraic datatypes, etc.
- To a family of model checkers (symbolic, explicit, and bounded).
- And other tools like a deadlock checker, path-finder, test-case generator, etc.
- And a SMT solver (mainly SRI's Yices).

Verifying the schedules

1. The six schedule constraints are reformulated from PVS into SAL (nearly verbatim).

Schedule constraints

1. Offset constraint $0 < P(r) < sched(r+1) - sched(r)$.
2. Communication Constraint #1
   $D(r) \geq \Sigma(r) + \Lambda(r) - \lfloor (1-\rho) \cdot (\delta_{nom} - e_l) \rfloor$.
3. Computation offset constraint
   $P(r) > D(r) + \Sigma(r) + \Lambda(r) + \lceil (1+\rho) \cdot (\delta_{nom} + e_u) \rceil$.
4. Pipeline constraint $\neg independent(r)$ implies $D(r) \geq 0$.
5. Communication constraint #2 $r > 0$ implies
   $D(r) \geq P(r-1) - sched(r) + sched(r-1)$.
6. Reception window constraint
   $0 \leq R(r) \leq D(r) + \lfloor (1-\rho) \cdot (\delta_{nom} - e_l) \rfloor - \Sigma(r) - \Lambda(r) + 1$.

$sched(r)$: clocktime round r begins.

$P(r)$: clocktime offset computation begins in round r.

$D(r)$: clocktime offset when communication begins in round r.

$\Lambda(r)$: maximum schedule discrepancy in round r.

$\Sigma(r)$: maximum clock skew in round r.

galois

Verifying the schedules

1. The six schedule constraints are reformulated from PVS into SAL (nearly verbatim).
2. Schedules for the hardware are specified as simple infinite-state machines (variables & constraints from the reals and integers).
   - State variables include the round counter and schedule-variables (e.g., computation offset, communication offset, reception window opening time, etc).
   - State variables may be nondeterministically updated.

Verifying the schedules

1. The six schedule constraints are reformulated from PVS into SAL (nearly verbatim).
2. Schedules for the hardware are specified as simple infinite-state machines (variables & constraints from the reals and integers).
   - State variables include the round counter and schedule-variables (e.g., computation offset, communication offset, reception window opening time, etc).
   - State variables may be nondeterministically updated.
3. We prove the constraints using infinite-state bounded model checking via k-induction.

   $k =$ number of rounds of the protocol.

Lessons (re)-learned

- Don't be a verification purist! Or as J. Moore put it. . .

Quoting J. Moore

February 27, 2007, `acl2@lists.cc.utexas.edu`

> *I think of theorem provers sort of like vehicles. What is the best kind of vehicle to buy? Isn't that silly question? If you're hauling kids to soccer practice, it might be a minivan. If you're hauling hay to the cattle, it might be a pickup.*

Lessons (re)-learned

- Don't be a verification purist! Or as J. Moore put it...
- Prove the consistency of your axioms (or your hypotheses)!

Lessons (re)-learned

- Don't be a verification purist! Or as J. Moore put it...
- Prove the consistency of your axioms (or your hypotheses)!
- Figure out what correctness means formally first (for time-triggered systems, it's "implementing a synchronous specification")!

Lessons (re)-learned

- Don't be a verification purist! Or as J. Moore put it...
- Prove the consistency of your axioms (or your hypotheses)!
- Figure out what correctness means formally first (for time-triggered systems, it's "implementing a synchronous specification")!
- A benefit of FV is to give engineers confidence that aggressive optimizations are provably correct.

Lessons (re)-learned

- Don't be a verification purist! Or as J. Moore put it. . .
- Prove the consistency of your axioms (or your hypotheses)!
- Figure out what correctness means formally first (for time-triggered systems, it's "implementing a synchronous specification")!
- A benefit of FV is to give engineers confidence that aggressive optimizations are provably correct.
- Try out infinite-state bounded model checking for real-time verification!

Towards a complete verification story

Abstraction levels:

- Application properties.
- Synchronous protocol abstraction. } Miner et. al., FTRTFT'04
- Time-triggered protocol abstraction.
- Real-time constraints.

} This paper

- Distributed communicating state-machines.
- HW realization & physical-layer protocols.

} Schmaltz, FMCAD'07; Knapp&Paul, LNCS4444

SPIDER is open-spec and is a great testbench for new approaches!

|galois|

Thanks

Steve Johnson, Paul Miner, Geoffrey Brown, Larry Moss, and Wilfredo Torres-Pomales.

I was extraordinarily impressed with the thoroughness of my anonymous reviewers. Thanks!

Web resources

Slides, specifications, and proofs

`http://www.cs.indiana.edu/~lepike/pub_pages/fmcad.html`

Google: lee pike fmcad

NASA SPIDER

`http://shemesh.larc.nasa.gov/fm/spider/`

Google: spider nasa

PVS & SAL

`http://fm.csl.sri.com/`

Google: formalware (please ignore tuxedo-related Google ads)