A Framework for the Formal Verification of Time-Triggered Systems

Lee Pike
`leepike@galois.com`

Indiana University, Bloomington
Department of Computer Science
Advisor: Prof. Steven D. Johnson

December 12, 2005

Acknowledgments

- Professors Steven Johnson, Geoffrey Brown, and Lawrence Moss
- Dr. Paul Miner and the SPIDER Research Team (Alfons Geser, Jeffrey Maddalon, Mayhar Malekpour, Wilfredo Torres-Pomales)
- Ricky Butler and the NASA Langley Research Center Formal Methods Group

Talk Goals

I present a framework for the formal verification of a class of safety-critical embedded systems.

- Introduce the domain of time-triggered embedded systems for fly-by-wire and drive-by-wire systems.
- Overview the verification challenges.
- Describe a framework for carrying out verification based on temporal abstraction.

Safety-Critical Embedded Systems

Digital control systems for commercial aircraft are *safety-critical*. Failure rates must be on the order of 10^{-9} per hour of operation (about the same probability as being hit by lightning in a given hour).

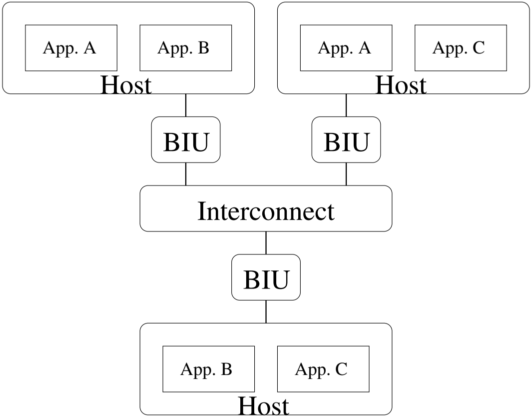
Time-Triggered Systems

To achieve fault-tolerance, control systems are implemented as *distributed systems*. The nodes in a distributed system must coordinate their behavior.

- *Event-triggers* signal the occurrence of some event.
- *Time-triggers* signal the passage of time, demarcated by a *schedule*.

I focus on time-triggered systems.

A Generic Fault-Tolerant Bus Architecture



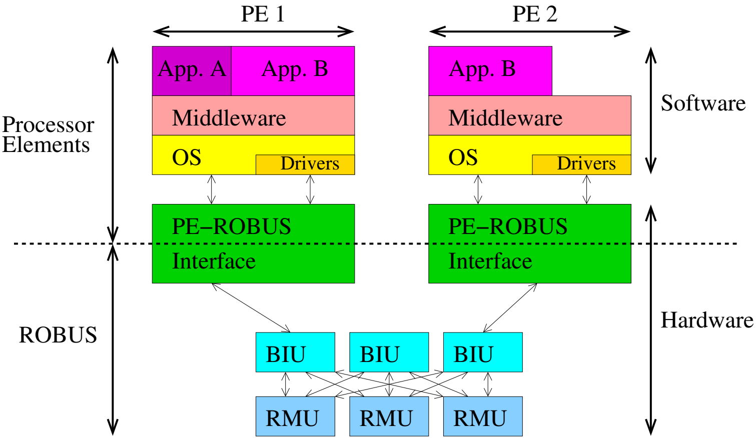
Bus Architecture Desiderata

- Integration
    - Off-the-shelf application integration
    - Off-the-shelf fault-tolerance
    - Eliminate redundancy
- Partitioning
    - Fault-partitioning
    - Modular certification
- Predictability
    - Hard real-time guarantees
    - A virtual time-division multi-access bus

SPIDER

“Time turns the improbable into the inevitable”

SPIDER Architecture



BIU/RMU Modes of Operation

- Self-Test Mode
- Initialization Mode
  - Initial Diagnosis
  - Initial Synchronization
  - Collective Diagnosis
- Preservation Mode
  - Clock Synchronization
  - Collective Diagnosis
  - PE Communication
- Reintegration Mode

Continuous on-line diagnosis. . .

Formal Methods

Formal methods are used to prove the correctness of digital systems.

$$\frac{\varepsilon_3}{\varepsilon_1} = \frac{A'}{A^2}\beta^2$$

$$\varepsilon_1 = \left(\frac{A}{A+1}\right)^2 E_1$$

$$\mu_3 = \mu$$

$$\frac{\varepsilon_4}{\varepsilon_1} = \frac{A'}{A+1-A'}\frac{\varepsilon_3}{\varepsilon_1}$$

$$\mu_4 = \mu$$

$$\longleftrightarrow$$

- Failure rates on the order of 10^{-9} make design assurance via testing infeasible.
- Design errors dramatically and unexpectedly raise the failure rate.
- Certification documents (will) require formal verification.
- A “best practice” in the design of complex safety-critical systems.

Caveat

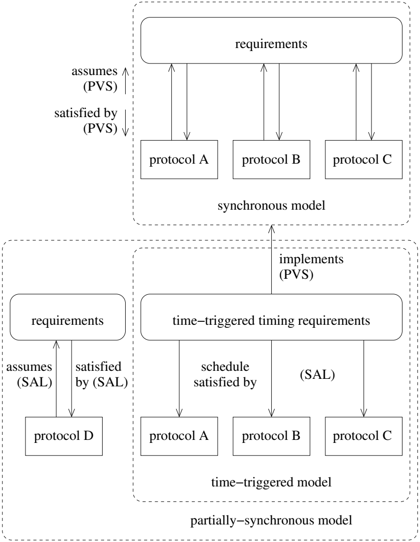
A verification of a fault-tolerant protocol guarantees only that if a *maximum fault assumption* (MFA) holds, then the protocol is correct. Experimental data and statistical analysis determines the probability of the MFA holding.

Verification Technologies

- Mechanical Theorem-Proving (PVS)
- Induction proofs via infinite-state bounded model checking (SAL)
- Interactive hardware derivation methods are also used for SPIDER, but not in this work.

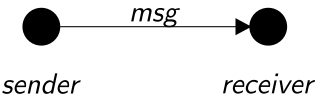
Industrial verification challenges depend on the judicious combination of tools and methods.

Verification Strategy for Time-Triggered Systems



Essential characteristics of messages for verifying fault-tolerance

- Its corruption
- Whether an arbitrary receiving process can detect its corruption



Message Classifications

- *Benign Message* Any non-faulty process receiving it could determine the message is corrupted, e.g.,
  - The message arrives at the wrong time (in a synchronized system).
  - The message fails error-detection.
- *Accepted Message* Any other message.

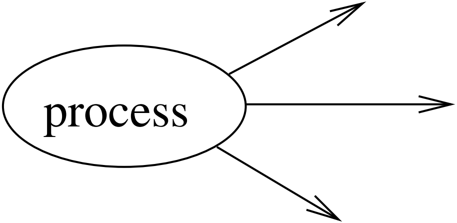
Two Ways Faults are Abstracted

- *Fault-Location Abstractions* Where in a system the fault occurs.
- *Fault-Type Abstractions* How a system is affected by the fault.

The Hybrid Fault Model[1]

Let V be the uncorrupted message to be sent.

- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.



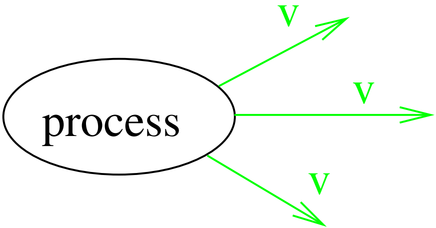
---

[1] Thambidurai and Park. Interactive consensus with multiple failure modes. *7th Reliable Distributed Systems Symposium*, 1988.

The Hybrid Fault Model[1]

Let V be the uncorrupted message to be sent.

- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.



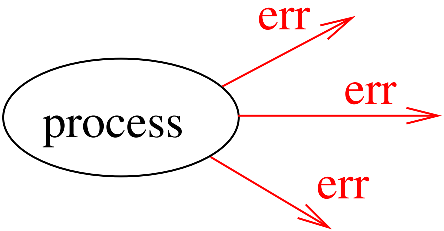
---

[1] Thambidurai and Park. Interactive consensus with multiple failure modes. *7th Reliable Distributed Systems Symposium*, 1988.

The Hybrid Fault Model[1]

Let V be the uncorrupted message to be sent.

- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.



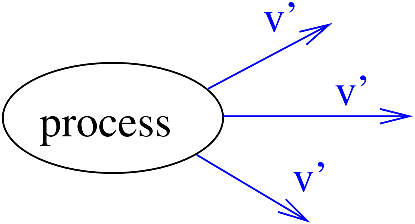
---

[1] Thambidurai and Park. Interactive consensus with multiple failure modes. *7th Reliable Distributed Systems Symposium*, 1988.

The Hybrid Fault Model[1]

Let V be the uncorrupted message to be sent.

- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.



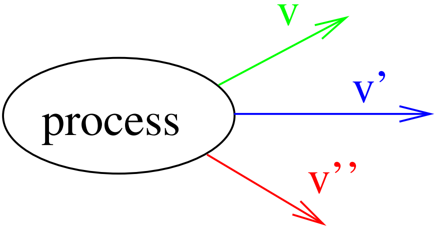
---

[1] Thambidurai and Park. Interactive consensus with multiple failure modes. *7th Reliable Distributed Systems Symposium*, 1988.

The Hybrid Fault Model[1]

Let V be the uncorrupted message to be sent.

- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.



---

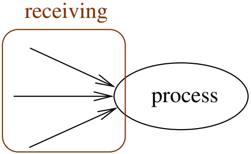
[1] Thambidurai and Park. Interactive consensus with multiple failure modes. *7th Reliable Distributed Systems Symposium*, 1988.

Abstracting the Location of Faults

- A process can perform three basic actions.
  - Receive messages
  - Compute messages
  - Send messages
- All of which can suffer faults.
- Reception and computation faults are abstracted as sending faults.

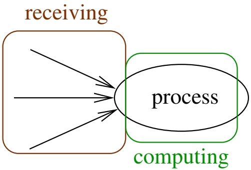
Abstracting the Location of Faults

- A process can perform three basic actions.
  - Receive messages
  - Compute messages
  - Send messages
- All of which can suffer faults.
- Reception and computation faults are abstracted as sending faults.



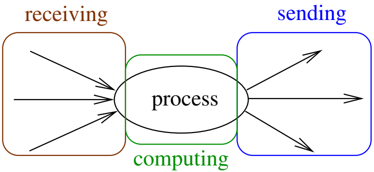
Abstracting the Location of Faults

- A process can perform three basic actions.
    - Receive messages
    - Compute messages
    - Send messages
- All of which can suffer faults.
- Reception and computation faults are abstracted as sending faults.



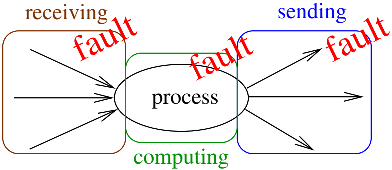
Abstracting the Location of Faults

- A process can perform three basic actions.
  - Receive messages
  - Compute messages
  - Send messages
- All of which can suffer faults.
- Reception and computation faults are abstracted as sending faults.



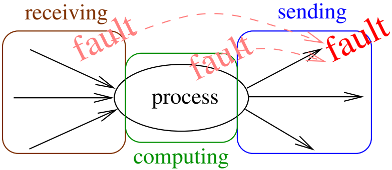
Abstracting the Location of Faults

- A process can perform three basic actions.
    - Receive messages
    - Compute messages
    - Send messages
- All of which can suffer faults.
- Reception and computation faults are abstracted as sending faults.



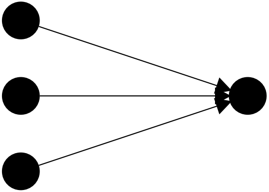
Abstracting the Location of Faults

- A process can perform three basic actions.
    - Receive messages
    - Compute messages
    - Send messages
- All of which can suffer faults.
- Reception and computation faults are abstracted as sending faults.



Comparing Incoming Messages to Mask Faults

- In fault-tolerant protocols, processes receive redundant messages from other processes.
- Messages are compared to ensure the selected message is within the range of those sent by non-faulty processes.



Two Means to Compare Messages

- *Majority Voting* The item that shows up most often is chosen (if one exists).
- *Middle-Value Selection* The sequence of messages is put into sorted order; then the item with the middle index is chosen.

- Majority of $\{1, 1, 1, 2, 2\}$ is 1.
- Middle-Value of $\{1, 1, 3, 4, 7\}$ is 3.

If a majority value exists, then majority voting and middle-value selection are equivalent.

A Relational Model of Communication and Voting

- A single relational model can be implemented by different functional specifications.
- Independent of the architecture and fault-classifications.
- Simplifies specifications and proofs in the functional models.

Relational Models of Inexact and Exact Sampling

We formulate two similar relational abstractions determined by the kind of function sampled.

- *Inexact Function* Approximating (sampling) a function's value.
  Example: Temperature (a function of time) is approximated by a digital thermometer.
- *Exact Function* Computing some function exactly.
  Example: Ordering a set of values.

Prove: If the MFA is satisfied by the sending nodes, then the computed result is within the range of non-faulty messages.

Examples

The verifications of the following protocols is based on these abstractions:

- SPIDER Interactive Consistency Protocol
- SPIDER Distributed Diagnosis Protocol
- SPIDER Clock Synchronization Protcol

The Time-Triggered Model

Synchrony is an abstraction.

- In an independently-clocked distributed system, skew, drift, latency, etc. place constraints on the scheduling of the system.
- In a *time-triggered model*, these constraints are explicit.

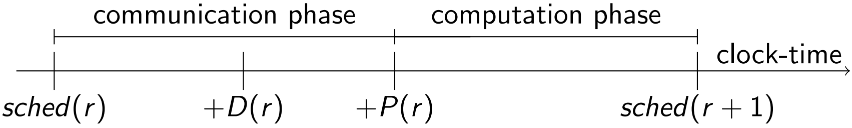
Goal: Demonstrate that the protocols verified in the synchronous model are correctly implemented under the time-triggered constraints. The original model is developed by John Rushby.[2]

---

[2] Systematic formal verification of time-triggered algorithms. *IEEE Transactions on Software Engineering*. 1999.

Time-Triggered Communication and Computation

```
run(rnd, state) =
  IF r = 0 THEN state
  ELSE LAMBDA p. comp(p)(run(r - 1, state)(p),
                         LAMBDA q. msg(q)(run(r - 1, state)(q), p))
```



Inconsistencies

- Three of the four system axioms are inconsistent, despite a formal specification and verification in PVS.
- The problem: no model is given to demonstrate the consistency of the axioms.
- Example: (*Clock Monotonicity*) Let C_p be a total function from $\mathbb{R}$ to $\mathbb{N}$. Then $t_1 < t_2$ implies $C_p(t_1) < C_p(t_2)$.
- Inconsistent: there is no injection from $\mathbb{R}$ to $\mathbb{N}$.

Amendments to the Model

The model is augmented to reason about

- event-triggered behavior,
- communication delays,
- reception windows,
- non-static clock skew,
- pipelined rounds.

Verification

The theory is formulated in PVS, and two verifications are given:

- The model is shown to satisfy the *synchrony hypothesis*: a simulation relation exists between the time-triggered model and the synchronous model.
- A *theory interpretation* is given to show relative consistency.

The time-triggered model demonstrates that provided a uninterpreted algorithm satisfies the scheduling constraints, then it implements a synchronous protocol.

Schedule Verification

Bounded model-checking and decision procedures are used to prove automatically that a protocol schedule satisfies the theory constraints.

1. State the system assumptions (maximum drift rate, minimum and maximum delays, skew, etc.).
2. State the implemented schedule for the protocol as a state machine, and check the satisfaction of the scheduling constraints in each round.

Examples

- SPIDER Distributed Diagnosis Protocol schedule verified.
- SPIDER Clock Synchronization Protocol schedule verified.
- Optimized and parameterized schedules verified.

Reintegration Protocol: an Unsynchronized Protocol

The protocol allows a faulty node to rejoin the operational nodes.

- Preliminary Diagnosis Mode
- Frame Synchronization Mode
- Synchronization Capture Mode

Safety Properties:

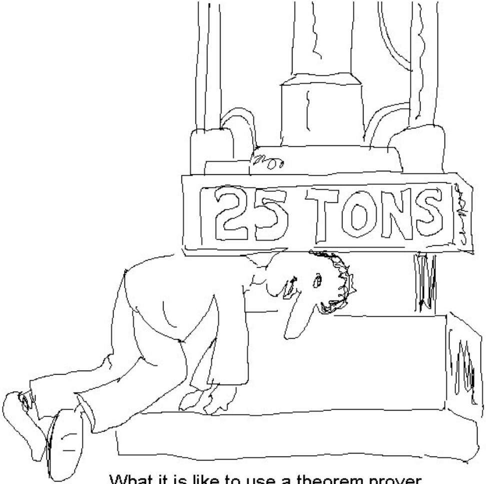
Theorem (No Operational Accusations)

The reintegrator never accuses an operational node.

Theorem (Synchronization Acquisition)

The reintegrator's clock is synchronized with those of the other nodes, up to the nominal skew.

Why Not Theorem-Proving?[3]



What it is like to use a theorem prover.

---

[3] Credit: NASA Langley Formal Methods Humor Page

Infinite-State Bounded Model-Checking

- Combines SAT solving and decision procedures to *prove* safety properties.
- Strengthens the induction schema by inducting over trajectories of fixed length rather than just single transitions: k-induction.

No Free Lunch

k-induction is exponential, so discovering the sufficiently-strong inductive invariant is still difficult.

Goal: reduce the number of steps necessary in the induction step.

Time-Triggered Simulation

- Typically, a state transition is taken each time the state changes.
- Another approach: “time-triggered simulation.”
- At fixed intervals of time
    - Determine the events observed by the reintegrator (i.e., after the reintegrator’s current timeout and before its next timeout).
    - Update the state of the reintegrator based on these observations simultaneously.

Summary

- Time-triggered bus architectures are being designed to provide fault-tolerance, coordination, and a communication infrastructure for embedded control systems.
- A strategy for the formal verification of these systems has been presented based on temporal abstraction.
- Judicious use of verification tools eases the difficulty of verification.

Further Information

More Details

`http://www.cs.indiana.edu/~lepike/`

Google: lee pike

SPIDER Homepage

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

Google: formal methods spider

NASA Langley Research Center Formal Methods Group

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

Google: nasa formal methods