Abstractions for Fault-Tolerant Distributed System Verification

Lee Pike[1]
Reporting joint work with
Jeffrey Maddalon[1] Paul Miner[1] Alfons Geser[2]

[1]Formal Methods Group, NASA Langley Research Center
{lee.s.pike, j.m.maddalon, paul.s.miner}@nasa.gov

[2]National Institute of Aerospace
geser@nianet.org

September 13, 2004

Desiderata

- The formal specification and verification of safety-critical embedded systems.

Desiderata

- The formal specification and verification of safety-critical embedded systems.
- In particular, SPIDER, an ultra-reliable embedded platform, under development at the NASA Langley Research Center.

SPIDER

Desiderata

- The formal specification and verification of safety-critical embedded systems.
- In particular, SPIDER, an ultra-reliable embedded platform, under development at the NASA Langley Research Center.
- Systematic and reusable specifications.

Desiderata

- The formal specification and verification of safety-critical embedded systems.
- In particular, SPIDER, an ultra-reliable embedded platform, under development at the NASA Langley Research Center.
- Systematic and reusable specifications.
- Specifications that facilitate proof in higher-order mechanical theorem-provers.

Principles of Abstraction

Good abstractions

- Dispose of irrelevant detail.
- Are simple, general, and comprehensible.

Example: A *set* abstracts a *sequence* when the relevant property is simply membership.

Level of Abstraction of Specifications

- Behavioral system specification.

Level of Abstraction of Specifications

- Behavioral system specification.
- Fault-tolerant distributed protocol specification, e.g.,
  - Passing data
  - Diagnosing faults
  - Synchronizing local clocks
  - Start-up/Restart
  - Reintegration

SPIDER

Level of Abstraction of Specifications

- Behavioral system specification.
- Fault-tolerant distributed protocol specification, e.g.,
  - Passing data
  - Diagnosing faults
  - Synchronizing local clocks
  - Start-up/Restart
  - Reintegration
- NOT protocol scheduling.
- NOT block-level processor design.

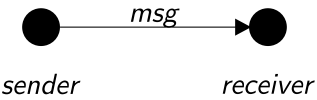
SPIDER

Contributions

Our contribution is the organization, explanation, and library support in PVS of the abstractions described herein.

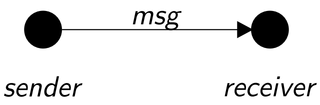
What is essential about a message in a fault tolerance context?

- Whether it is corrupted or not.



What is essential about a message in a fault tolerance context?

- Whether it is corrupted or not.
- Whether or not a process can detect this corruption.



Message Classifications

Benign Message Any non-faulty process receiving it could determine the message is corrupted, e.g.,

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.

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.

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.

Note An accepted message is not necessarily an uncorrupted 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.

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.

Two Ways Faults are Abstracted

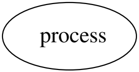
Fault-Location Abstractions *Where* in a system the fault occurs.

Fault-Type Abstractions *How* a system is affected by the fault.

SPIDER

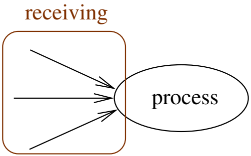
Abstracting the Location of Faults

- A process can perform three basic actions.



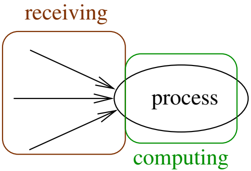
Abstracting the Location of Faults

- A process can perform three basic actions.
  - Receive messages



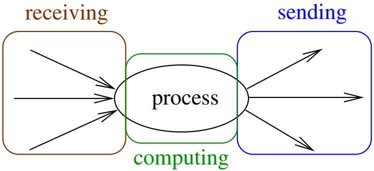
Abstracting the Location of Faults

- A process can perform three basic actions.
  - Receive messages
  - Compute messages



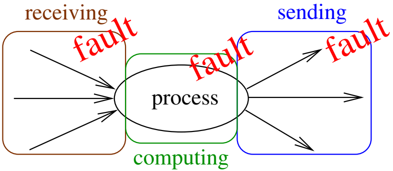
Abstracting the Location of Faults

- A process can perform three basic actions.
  - Receive messages
  - Compute messages
  - Send messages



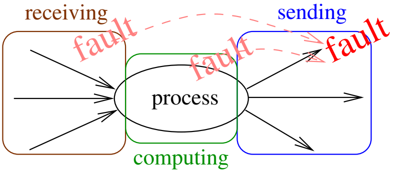
Abstracting the Location of Faults

- A process can perform three basic actions.
  - Receive messages
  - Compute messages
  - Send messages
- All of which can suffer faults.



Abstracting the Location of Faults

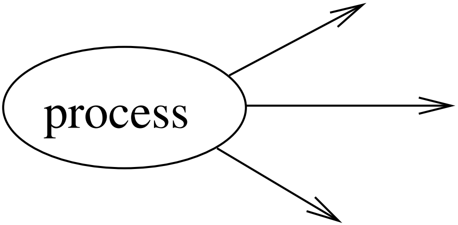
- A process can perform three basic actions.
  - Receive messages
  - Compute messages
  - Send messages
- All of which can suffer faults.
- Reception and computation faults are abstracted as sending faults.



The Hybrid Fault Model

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.

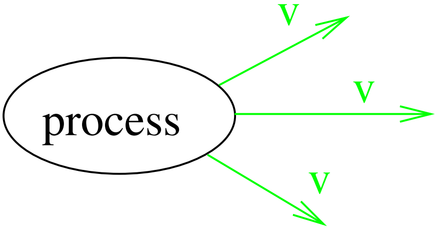


SPIDER

The Hybrid Fault Model

Let V be the uncorrupted message to be sent.

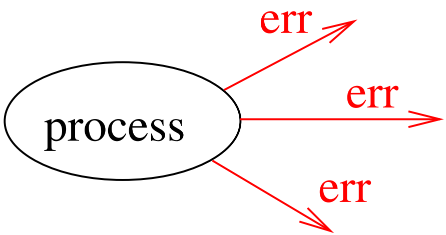
- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.



The Hybrid Fault Model

Let V be the uncorrupted message to be sent.

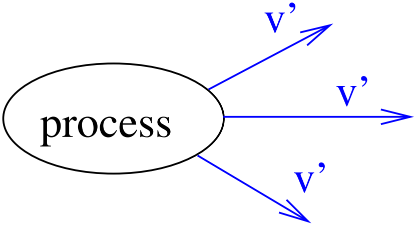
- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.



The Hybrid Fault Model

Let V be the uncorrupted message to be sent.

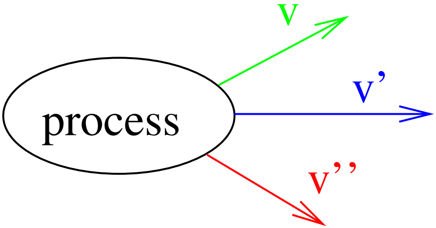
- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.



The Hybrid Fault Model

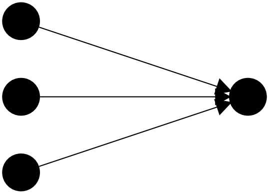
Let V be the uncorrupted message to be sent.

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



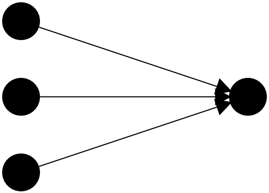
Comparing Incoming Messages to Mask Faults

- In fault-tolerant protocols, processes receive redundant messages from other processes.



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.

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.

SPIDER

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.

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

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.

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

A Relational Model of Communication

A Relational Model of Communication

A *relational* specification of a protocol is more abstract than a *functional* specification:

SPIDER

A Relational Model of Communication

A *relational* specification of a protocol is more abstract than a *functional* specification:

Example:

Assume: Most of the values in a majority vote are from good processes.

Prove: The voted value is from a good process.

A Relational Model of Communication

A *relational* specification of a protocol is more abstract than a *functional* specification:

Example:

Assume: Most of the values in a majority vote are from good processes.

Prove: The voted value is from a good process.

A functional model of the protocol can then be shown to satisfy the preconditions of the relational model.

SPIDER

Benefits of a Relational Model

Benefits of a Relational Model

- A single relational model can be implemented by different functional specifications.

Benefits of a Relational Model

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

Benefits of a Relational Model

- 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.

Benefits of a Relational Model

- 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.
- Maximizes proof-reuse between functional models.

SPIDER

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.

SPIDER

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.

SPIDER

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.

Ongoing Work

- On-going development of a generalized fault-tolerance library of results in PVS.

Ongoing Work

- On-going development of a generalized fault-tolerance library of results in PVS.
- Joint work with Holger Pfeifer (Univ. of Ulm) to
  - Extend these abstractions (e.g., a more refined fault model).
  - Verify TTA using these abstractions and our library.

Ongoing Work

- On-going development of a generalized fault-tolerance library of results in PVS.
- Joint work with Holger Pfeifer (Univ. of Ulm) to
  - Extend these abstractions (e.g., a more refined fault model).
  - Verify TTA using these abstractions and our library.

Software engineering didn't succeed without good abstractions and library support.
Neither will theorem-proving.

SPIDER

Links

PVS Files for the Paper

http://shemesh.larc.nasa.gov/fm/spider/tphols2004/

Google: tphols abstractions

SPIDER Project

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

The Beamer and PGF Classes for LaTeX

This presentation brought to you by the wonderful Beamer class.

Beamer Website

`http://latex-beamer.sourceforge.net/`

Google: beamer class

Formalizing Messages

Let $m \in MSG$:

Constructors	Extractors	Recognizers
$accepted\_msg[m]$	$value$	$accepted\_msg?$
$benign\_msg$	none	$benign\_msg?$

Formalizing Faults: A Send Function

$$send(msg\_map, sender\_status, s, r) \stackrel{\text{df}}{=}$$
$$\begin{cases} accepted\_msg[msg\_map(s)] & : \quad sender\_status(s) = good \\ benign\_msg & : \quad sender\_status(s) = ben \\ sym\_msg(msg\_map(s), s) & : \quad sender\_status(s) = sym \\ asym\_msg(msg\_map(s), s, r) & : \quad sender\_status(s) = asym \end{cases}$$

Formalizing Majority Voting Relationally

$$ms : \; V \rightarrow \mathbb{N}$$

$$maj\_set(ms) \stackrel{\text{df}}{=} \{v \mid 2 \times ms(v) > |ms|\}$$

$$majority(ms) \stackrel{\text{df}}{=} \begin{cases} no\_maj & : \quad maj\_set(ms) = \emptyset \\ \epsilon(maj\_set(ms)) & : \quad \text{otherwise} \end{cases}$$

Formalizing Middle-Value Selection Relationally

$$lower\_filter(ms, v) \stackrel{\text{df}}{=} \lambda i. \begin{cases} ms(i) & : \quad i \preceq v \\ 0 & : \quad \text{otherwise} \end{cases}$$

$$upper\_filter(ms, v) \stackrel{\text{df}}{=} \lambda i. \begin{cases} ms(i) & : \quad v \preceq i \\ 0 & : \quad \text{otherwise} \end{cases}$$

$$mid\_val\_set(ms) \stackrel{\text{df}}{=} \left\{ v \;\middle|\; \begin{array}{l} 2 \times |lower\_filter(ms, v)| > |ms| \wedge \\ 2 \times |upper\_filter(ms, v)| \geq |ms| \end{array} \right\}$$

$$middle\_value(ms) \stackrel{\text{df}}{=} \epsilon(mid\_val\_set(ms))$$

Middle-Value Selection and Majority Voting Equivalence

Theorem (Middle Value is Majority)

$$majority(ms) \neq no\_maj \ \ implies \ middle\_value(ms) = majority(ms).$$

The Exact Validity Property

Exact Validity: Exact Validity: A good receiver's fault-masking vote is equal to the value of the function good processes compute.

Pre-Conditions to Satisfy Exact Validity

First, most of the sending processes must be good.

$$
\begin{aligned}
&\mathit{majority\_good}(\mathit{good\_senders}, \mathit{eligible\_senders}) \stackrel{\mathrm{df}}{=} \\
&\quad 2 \times |\mathit{good\_senders}| > |\mathit{eligible\_senders}| \wedge \\
&\quad \mathit{good\_senders} \subseteq \mathit{eligible\_senders}
\end{aligned}
$$

SPIDER

Pre-Conditions to Satisfy Exact Validity

Second, the all good sending processes must send correctly.

$$exact\_message\_error(good\_senders, ideal, actual) \stackrel{\text{df}}{=}$$
$$\forall s.\ s \in good\_senders \implies ideal(s) = actual(s)$$

Pre-Conditions to Satisfy Exact Validity

Third, all good sending processes compute the same function.

$$\begin{aligned}&\mathit{function\_agreement}(\mathit{good\_senders}, \mathit{ideal}) \stackrel{\text{df}}{=}\\ &\quad \forall s_1, s_2.\ s_1 \in \mathit{good\_senders} \wedge s_2 \in \mathit{good\_senders}\\ &\qquad \Longrightarrow\ \mathit{ideal}(s_1) = \mathit{ideal}(s_2)\end{aligned}$$

A Technical Detail...

$$make\_bag(eligible\_senders, actual) \stackrel{\text{df}}{=}$$
$$\lambda v. \left| \{s \mid s \in eligible\_senders \wedge actual(s) = v\} \right|$$

Formally Stating the Exact Validity Theorem

$$\begin{aligned}&exact\_validity(eligible\_senders, good\_senders, ideal, actual) \stackrel{\text{df}}{=}\\ &\quad \forall s.\, s \in good\_senders \implies\\ &\qquad ideal(s) = majority(make\_bag(eligible\_senders, actual))\end{aligned}$$

Theorem (Exact Validity)

$$\begin{aligned}&majority\_good(good\_senders, eligible\_senders) \wedge\\ &exact\_message\_error(good\_senders, ideal, actual)) \wedge\\ &message\_agreement(good\_senders, ideal)\\ &\qquad \implies\\ &exact\_validity(eligible\_senders, good\_senders, ideal, actual)\end{aligned}$$