Assured Reconfiguration: An Architectural Core For System Dependability

ICSE 2005
Workshop on Architecting Dependable Systems

John Knight
University of Virginia

Joint work with Elisabeth Strunk
The Challenge

Safety-Critical Applications

- Desired Functionality

- System Hardware Volume
- System Software Volume

Hardware Costs

Complexity

May 2005

UVA Dependability Research Group

Assured Reconfiguration
Implications Of The Challenge

- **System:**
  - Distributed processing/Integrated Modular Avionics
  - High data communications demand

- **Hardware:**
  - Replication to meet MTBF demands

- **Software:**
  - Increased volume, complexity, functionality

- And it is bound to continue for the foreseeable future…
Meeting The Challenge?

- All defects can have serious consequences in typical systems but...
- Hardware replication:
  - Expensive, bulky
  - Increased weight, power, space, shielding
- Software complexity:
  - Mostly outside the realm of assurance techniques
- Trying to deal with this by restricting amount of function in systems is naïve
- Can we continue with “business as usual”?
Business As Usual For **Hardware**?

Hardware Is *Much* More Reliable Than It Used To Be

- Business as usual unnecessary
Business As Usual For **Software**?

- Why is software so difficult?
  - Fluid mechanics:
    - Continuous mathematics
    - Navier-Stokes equation
  - Structural analysis:
    - Continuous mathematics
    - Finite element method
  - Software:
    - Discrete mathematics
    - ?

Business as usual unlikely to succeed
Claim

Hardware Degradation Faults Are Much Less Frequent Than In The Past

Maintaining Complete Functionality With Ultra High Assurance Is Unnecessary

Occasional Operation With Reduced But Safe Functionality Is Satisfactory

Basing System Design On These Assumptions Reduces Complexity And Cost

ASSURED RECONFIGURATION
What Is Assured Reconfiguration?

- *Explicit decision at specification level* to define a tradeoff between system dependability and function
- *Explicit decision by system stakeholders* to accept alternative functionality if errors do occur
- *Because:*
  - Complete hardware masking is too expensive
  - *Adequate software fault avoidance/removal is infeasible*
What Is Assured Reconfiguration?

Reliability, Availability

Assured Reconfiguration

Target Configuration Depends On Conditions
Example: Modern Avionics Systems

- Aircraft flight control **software**
- FAA software development standard:
  - **Minor:**
    - Anticipated to occur one or more times during the entire operational life of each airplane
  - **Major:**
    - Not anticipated to occur during the entire operational life of a single random airplane
  - **Catastrophic:**
    - Not anticipated to occur during the entire operational life of all airplanes of one type
    - Failure rate of $10^{-9}$ per hour of operation
Example: Modern Avionics Systems

- These requirements:
  - Cannot be assured with current approaches
  - Are essentially impossible to demonstrate

- *But*, some (most?) functionality:
  - Does not need to be reliable
  - Needs to be *fail-stop* with ultra high dependability

- Assured reconfiguration is an option to achieve system goals
Prior Work on Reconfiguration

- Survivability in critical information systems
  - Different requirements for embedded systems
- Alternative functionalities (Shelton and Koopman)
  - Provides a model of system utility
- Graceful degradation
  - Maximum utility with working components
Prior Work on Reconfiguration

- Quality of service
  - Specific aspects of a system
- Simplex architecture (Sha)
  - Assumes analytic redundancy
- Current systems, e.g., Boeing 777
  - Ad-hoc
  - Are built using facilities already provided by the system
Vision

Reconfiguration As Architectural Foundation

Assured System Reconfiguration

Assurance By Proof
Proposed Approach

- **System architecture:**
  - Fully distributed, arbitrary layout and number of parts
  - Ultra-dependable data bus, e.g., TTP

- **Computing and storage hardware:**
  - Allow computers to fail, *but*
  - Use ultra-dependable fail-stop machines

- **Software:**
  - Allow application software to fail, *but*
  - Use ultra-dependable, fail-stop applications

**Ultra-dependable reconfiguration mechanism**
Proposed Approach

Common Components

Components Added As Needed

High Speed Data Bus

Avionics Application
Operating System
General Purpose Computer
Proposed Approach

High Speed Data Bus

Fail Stop General Purpose Computer

Avionics Application
Operating System
General Purpose Computer

Avionics Application
Operating System
General Purpose Computer

Avionics Application
Operating System
General Purpose Computer
Proposed Approach

Ultra Dependable, Reconfigurable High Speed Data Bus

High Speed Data Bus
Proposed Approach

Reconfigurable Fail-Stop Avionics Application

Operating System
General Purpose Computer
Avionics Application

High Speed Data Bus
Distributed Reconfigurable System Architecture

- Avionics Application
- Operating System
- General Purpose Computer
- BIU
- High Speed Data Bus
- Subsystem Control Reconfiguration Analysis & Management (SCRAM) Software
- Operating System
- General Purpose Computer
- BIU
- Special Purpose Device

Crucial Software

May 2005
Assured Reconfiguration
Crucial Software Development

One

Reconfiguration Definition

Equivalence Proof

Many

SCRAM Software (Common)

State Machine Specification (System Specific)

Analysis & Synthesis

Reconfiguration Specification
Application Programming
Fail-Stop Processors

- Introduced by Schlichting and Schneider
- Building block for critical systems
- Fail-stop processor:
  - Processing units
  - Volatile storage
  - Stable storage
- Stable storage preserved on failure
Reconfigurable FTAs

- Fault-tolerant actions (FTAs)
  - In S&S work, recovery must complete original action
  - In our work, recovery could be reconfiguration
    - Complete some different function
Reconfigurable Fail-Stop Systems

- Software building block is a reconfigurable application
- Reconfigurable application has:
  - A predetermined set of specifications
  - A predetermined set of FTAs for each specification
- Application function exists in system context:
  - Recovery must be appropriate to system
  - Failure in one application could cause failure in another
- Not a problem in S&S work since failures were masked, sufficient resources assumed
Application and System FTAs

- **Application FTAs**
  - Execution of a single application

- **System FTAs**
  - Composed of a set of AFTAs
    - Affected applications’ actions and recovery protocols
    - Standard AFTAs for the other applications
  - Coordinates stages of AFTAs
  - Stages have time bounds
  - S & S can guarantee liveness
  - Safe configuration enables real-time guarantees
Reconfiguration Software Architecture

Specifications
S_{i,1}: desired functionality
S_{i,2}: intermediate functionality
\ldots
S_{i,m}: crucial functionality

(System calls) (Software fault signals) (Reconfiguration Signals) (System calls)
Reconfiguration Assurance
Reconfiguration Properties

- Reconfiguration:
  - Begins with a signal generated by some application
  - Ends either with a second signal, or when all applications have finished initialization

- The new configuration is appropriate for the circumstances

- All reconfigurations complete within their required time bound

- The system invariant holds during reconfiguration

- Additional restriction on sequences of reconfiguration signals
Assurance Technology

Based on PVS specification notation and PVS theorem-proving system

PVS:

- Language is a higher-order logic based on type theory
- Subtypes are defined by adding a predicate to a supertype
- Predicate must hold over any instance of subtype
- Type properties can be used in proofs
- In some cases, type properties are undecidable
- Produces type-correctness conditions (TCCs), a kind of proof obligation
- PVS system mechanically checks proofs
Proof Structure

Reconfigurable Properties

Reusable PVS Proof Using Type Constraints

Interaction Specification (State Sequences)

Abstract Reconfiguration Specification

System-specific Proof by Type System

Application Abstract Specification

Used In

System-Specific Specification Instance

Application Specification Instances

System-Specific Configuration, Environment, Transition Information

Abstract Reconfiguration Specification
Reconfiguration Specification

- System applications
- Operating environment
- System configurations
- System transitions
- Valid system implementation generates a valid sequence of system states
Proof Sample

Proofs are scripts that can be mechanically checked using the PVS system

assured_reconfig.CP5: proved - complete [shostak](13048.43 s)

(""
(skosimp)
(split)
("1"
  (lemma "reconf_length")
  (inst -1 "s!1" "r!1")
  (typepred "r!1")
  (typepred "s!1\`tr")
  (expand "get_reconfigs")
  (hide -2 -3 -4)
  (flatten)
  (case "r!1\`end_c - r!1\`start_c = 1")
("1"
  (lemma "reconf_halt")
  (expand "reconfig_end?")
  (split -6)
  ("1"
    (expand "reconfig_start?")
    (skosimp)
    (inst -1 "app!1")
    (inst -2 "s!1" "r!1" "app!1")
    (hide -4 -5 -6 -7 -8)
    (grind))
  ("2" (propax)))
("2"
Reconfiguration Example
Example

- **UAV system**
- **Four applications:**
  - Sensors, flight control system
  - Autopilot, pilot interface
- **Complete reconfiguration interface,**
  multiple functionalities
- **Three reconfiguration triggers:**
  - Electrical power
  - Rudder
  - Autopilot
## Example Configurations

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Power</th>
<th>Rudder</th>
<th>Autopilot</th>
<th>FCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Service</td>
<td>alternator</td>
<td>working</td>
<td>normal</td>
<td>normal</td>
</tr>
<tr>
<td>Altitude Hold Only</td>
<td>alternator</td>
<td>working</td>
<td><strong>altitude hold only</strong></td>
<td>normal</td>
</tr>
<tr>
<td>Flight Control Only</td>
<td>alternator</td>
<td>working</td>
<td><strong>nonfunctional</strong></td>
<td><strong>normal</strong></td>
</tr>
<tr>
<td>Flight Control Only</td>
<td>battery</td>
<td>working</td>
<td><strong>disabled</strong></td>
<td><strong>normal</strong></td>
</tr>
<tr>
<td>Rudder Hard-Over L/R</td>
<td>alternator</td>
<td>hard-over left/right</td>
<td>normal</td>
<td>adjusting for rudder</td>
</tr>
<tr>
<td>Rudder Hard-Over L/R, Altitude Hold Only</td>
<td>alternator</td>
<td>hard-over left/right</td>
<td><strong>altitude hold only</strong></td>
<td>adjusting for rudder</td>
</tr>
<tr>
<td>Rudder Hard-Over L/R, Flight Control Only</td>
<td>alternator</td>
<td>hard-over left/right</td>
<td><strong>nonfunctional</strong></td>
<td>adjusting for rudder</td>
</tr>
<tr>
<td>Rudder Hard-Over L/R, Flight Control Only</td>
<td>battery</td>
<td>hard-over left/right</td>
<td><strong>disabled</strong></td>
<td>adjusting for rudder</td>
</tr>
</tbody>
</table>
Example SFTA
In Full Service configuration when the rudder becomes stuck hard-over to the left

<table>
<thead>
<tr>
<th>Frame</th>
<th>Action</th>
<th>Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (start)</td>
<td>Sensors: signal generated</td>
<td>Sensors: invariant</td>
</tr>
<tr>
<td></td>
<td>All other apps: normal execution</td>
<td>All other apps: invariant</td>
</tr>
<tr>
<td>2</td>
<td>Apps anticipate possible reconfiguration</td>
<td>App postconditions</td>
</tr>
<tr>
<td>3</td>
<td>FCS: prepare to adjust for rudder</td>
<td>FCS: transition condition</td>
</tr>
<tr>
<td></td>
<td>All other apps: normal execution</td>
<td>All other apps: invariant</td>
</tr>
<tr>
<td>4 (end)</td>
<td>All apps: normal execution</td>
<td>All apps: invariant</td>
</tr>
</tbody>
</table>
Example Status

- Specified in PVS
- Type-checked against the abstract specification
- 75 TCCs generated
  - Most resulted from specific PVS approach
  - Most others trivial to prove
  - Nontrivial proofs could be generated using state-space search
  - Proofs could be more difficult for larger systems
- Proof obligations discharged
  - Reconfiguration properties hold
Conclusion

- Exploit potential of fully distributed target
- Hardware MTBFs:
  - Much higher
  - Less replication needed, accept rare failures
- Software Volume:
  - Increasing and assurance remains difficult
  - Fail-stop software less difficult to develop
- Base architecture on assured reconfiguration
- Assurance via comprehensive formal proof
Contact Information

- John Knight – knight@cs.virginia.edu
- Elisabeth Strunk – strunk@cs.virginia.edu
- Papers available at:
  http://www.cs.virginia.edu/~jck/recentpapers.htm