Design of safety-critical distributed applications

Claudio Pinello
(pinello@eecs.berkeley.edu)

Prof. Sangiovanni-Vincentelli
Chess Review
May 8, 2003
Berkeley, CA

Motivation

- Drive-by-Wire applications
Problem Overview

- System failure rate can't be worse than traditional systems
- Fault tolerance: redundancy is key

Proposed Design Flow

FaultBehavior → Mapping → [Diagram]

Chess Review, May 8, 2003
Outline

- Background
- Current approach
- Advantages of synthesis
- Contributions to enable synthesis
- Future work
- Conclusions

Fault Tolerant Control Systems

- Plant Faults (plant, sensors, actuators)
  - estimation and control algorithms
- Architecture faults (channels, ECUs)
  - hardware redundancy
  - software replication
  - redundancy management
- Application faults: bugs
  - can be reduced by disciplined coding
  - code generation from formal models
  - simulation
  - formal verification
(Architecture) Fault Model

<table>
<thead>
<tr>
<th>Fault</th>
<th>Error</th>
<th>Detection</th>
<th>Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silent</td>
<td>omission</td>
<td>timeout</td>
<td>any other</td>
</tr>
<tr>
<td>Detectable</td>
<td>detectably</td>
<td>integrity check (e.g.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>corrupted</td>
<td>CRC)</td>
<td>any other</td>
</tr>
<tr>
<td>Non-silent</td>
<td>value errors</td>
<td>voting</td>
<td>majority</td>
</tr>
<tr>
<td>Byzantine</td>
<td>malicious</td>
<td>signatures + voting</td>
<td>majority</td>
</tr>
<tr>
<td></td>
<td>attacks,</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Available building blocks

- Self-testing processors
- Fault tolerant communication protocols
  - TTP (Time-Triggered Protocol)
  - FlexRay
- OSEK FTcom operating system
- etc...
Outline

- Background
- Current approach
- Advantages of synthesis
- Contributions to enable synthesis
- Future work
- Conclusions

Common Practice (ECU design)

- Pros:
  - design once
- Cons:
  - N-x costs, 1x speed

- Pros:
  - reduced cost
- Cons:
  - degradation, 1x speed
  - multiple designs
Advantages of Synthesis

- Decouple application design from implementation of fault tolerance
- Synthesize glue-code and customizable replication mechanisms
  - adapt to different fault models
  - exploit available building blocks
- Treat architecture as a distributed execution machine
  - must have redundancy, why not exploit parallelism to speed up execution?

Contributions

- Programming model
- Fault behavior specification
- Schedule synthesis tool
- Verification Tools
Fault Tolerant DF
Pendulum Example

Synchronous

Programming and simulation environment

- Metropolis library to model FTDF netlists
- Support for simulation, fault injection and visualization
- Early assessment of closed loop behavior in degraded modes
Contributions

- Programming model
- Fault behavior specification
- Schedule synthesis tool
- Verification Tools

Architecture Model

- Connectivity:
  - bipartite graph \(\textit{Arch}\)
- Performance:
  - matrix of actor/ECU execution times
  - matrix of data/channel transmission times
- Actuator/Sensor location
Fault Behavior Specification

- Failure patterns $P_i \subseteq Arch$
  - subsets of $Arch$ graph that may fail simultaneously (in a same iteration)

- For each $P_i$, specify which functionalities must be guaranteed
  - typically functionality chosen based on criticality

Contributions

- Programming model
- Fault behavior specification
- Schedule synthesis tool
- Verification Tools
Static Schedule

- At design time (off-line)
  - devise order of execution/transmission
- At run-time
  - each ECU executes/transmits in given order
  - each ECU skips actors that cannot fire (localized reconfiguration)
- Low overhead, predictable, localized

Credits: in collaboration with Catalin Dima (INRIA)
Generating Schedules

1. Full architecture ($P_0$)
   1. Load + schedule all functionalities

2. For each $P_i$
   1. Mark faulty components (critical functionalities cannot run there)
   2. Load + schedule critical functionalities
      (constrain non critical as in $P_0$)

3. Merge loadings
4. Schedule merged loadings

Add redundancy only to critical

FT scheduling tool chain

Parse.exe $\rightarrow$ SynDEx

Merge.exe $\rightarrow$ Schedule.exe
Schedule optimization

- Exploit redundancy as a performance boost (in absence of faults)
- Techniques:
  - deallocation of unneeded replicas
  - graceful degradation: reduced functionality (and resource demands) under faults

Active Replicas

Behavior:

Architecture:

Active Replication:

ECU1

A
B
C
D

ECU2

A
B
C
D

Chess Review, May 8, 2003 24
**Deallocation & Degradation**

**Behavior:**

**Architecture:**

**Deallocation:**

**Code Generation**

- Run-time support for FTDF semantics
  - for a Linux UDP/IP based implementation (in collaboration with Mark McKelvin)
- Provides synchronization mechanisms, as well as redundancy management
- Enables 1-to-1 translation of actors into C procedures for execute() and firingrule() methods
Conclusions

- Proposed design flow enables
  - greater separation of concerns
    - application, architecture, fault behavior
  - formal specification and verification of fault tolerant systems
  - architecture space exploration