Experiences with CANoe-based Fault Injection for AUTOSAR

Patrick E. Lanigan, Priya Narasimhan
Electrical & Computer Engineering
Carnegie Mellon University

Thomas E. Fuhrman
Research & Development
General Motors Company
What to Expect

• A brief overview of automotive systems / tools
  – AUTOSAR
  – FlexRay
  – Vector CANoe
• A description of a proof-of-concept software-implemented fault-injection framework
• An example application of the framework
• A qualitative discussion of the what worked well, as well as what did not work so well.
What Not to Expect

• A dependability evaluation of the...
  – AUTOSAR specification
  – AUTOSAR implementation
  – FlexRay protocol
  – demo application

• A discussion of specific fault-models
• A coverage assessment
• A quantitative analysis
ROADMAP

• Introduction
  – Overview of automotive systems
  – Motivation
  – Goals

• Fault-injection framework

• Runtime evaluation

• Conclusion
**Automotive Systems**

- **What is an “automotive system”**?
  - Many mechanical, hydraulic and electrical (incl. hardware and software) components interact to perform vehicle functions
  - Operates in a dynamic environment (other vehicles, pedestrians, animals, etc)
- **Our focus is on the embedded computing architecture**
  - Electronic Control Units (ECUs) and software
  - Distributed, serial communication (e.g. CAN, FlexRay)
  - Sensors and actuators
**BACKGROUND: AUTOSAR**

Electronics account for significant and increasing proportion of innovation as well as cost.

- **Standard software-architecture for automotive applications**
  - Encourage reuse of software
  - Reduce development costs
- **Layered architecture**
  - Basic Software (BSW) layers provide hardware abstractions
  - Application layer implements high-level functionality
  - Runtime Environment (RTE) layer enables information exchange

---

**BACKGROUND: FlexRay**

- High-speed, synchronous serial communication protocol
  - Communication schedule is divided into equal-length time slots and executed periodically
  - Slots are statically assigned to nodes
  - Support for dual channels (up to 10 mbps each)

- FlexRay nodes are synchronized to a global time-base

---

[Diagram of FlexRay communication schedule with slot counters and frame IDs]


http://www.flexray.com
An increasing amount of control and autonomy is being delegated to embedded computing architectures.

- Adaptive cruise control
- Forward collision warning
- Curve speed control
- Side blind zone alert
- Lane keeping / lane centering control
- Cross traffic collision avoidance
Motivation for Our Framework

- AUTOSAR is likely to be a key enabler of functional safety systems
- Fault-injection plays an important role in the dependability analysis of such systems
  - “Highly recommended” by upcoming ISO 26262 standard
- Hardware-based fault injection requires specialized equipment (e.g., TTX Disturbance node)
  - Availability, cost, complexity can limit applicability
  - Cannot accurately target specific software modules
- How far can we go with existing software tools (e.g., Vector CANoe)?
**BACKGROUND: VECTOR CANoe**

- CANoe is a simulation and evaluation environment for automotive applications
- Supports a variety of bus protocols (e.g., FlexRay)
- The behavior of simulated nodes can be defined in multiple ways
  - External DLL using CANoe programming interface
  - CAPL scripting language
- Provides graphical and text-based output windows
THE “-ILITIES”
The “-ilities”

Functionality

(1) Exercise error handling
(2) Runtime configuration
(3) Runtime visualization
THE “-ILITIES”

Functionality
(1) Exercise error handling
(2) Runtime configuration
(3) Runtime visualization

Controllability
Repeatable scenarios with respect to time, location and duration.
THE “-ILITIES”

Functionality
(1) Exercise error handling
(2) Runtime configuration
(3) Runtime visualization

Controllability
Repeatable scenarios with respect to time, location and duration.

Observability
Distinguish masked faults from faults with no effect.
The “-ilities”

**Functionality**

1. Exercise error handling
2. Runtime configuration
3. Runtime visualization

**Controllability**

Repeatable scenarios with respect to time, location and duration.

**Observability**

Distinguish masked faults from faults with no effect.

**Portability**

Minimize changes required to existing codebase(s).
THE “-ILITIES”

Functionality
(1) Exercise error handling
(2) Runtime configuration
(3) Runtime visualization

Controllability
Repeatable scenarios with respect to time, location and duration.

Observability
Distinguish masked faults from faults with no effect.

Portability
Minimize changes required to existing codebase(s).

Flexibility
Support a wide range of scenarios, faults and errors.
Experiences with CANoe-Based Fault-Injection for AUTOSAR

Patrick E. Lanigan (planigan@ece.cmu.edu)

THE “-ILITIES”

**Functionality**

1. Exercise error handling
2. Runtime configuration
3. Runtime visualization

**Controllability**

Repeatable scenarios with respect to time, location and duration.

**Observability**

Distinguish masked faults from faults with no effect.

**Portability**

Minimize changes required to existing codebase(s).

**Flexibility**

Support a wide range of scenarios, faults and errors.

**Avoid Probe Effects**

Fault-injection process should not produce unintended side effects.
ROADMAP

• Introduction

• Fault-injection framework
  – Architecture
  – Fault-injection hook locations
  – Example use of fault-injection hooks

• Runtime evaluation

• Conclusion
Fault-injection Strategy

- Hooks are inserted into AUTOSAR functions
  - Manipulation hooks modify function arguments
  - Suppression hooks cause functions to return error codes
- Fault-injection scenarios are defined by six parameters

<table>
<thead>
<tr>
<th>Manipulation parameters</th>
<th>Location</th>
<th>Argument</th>
<th>Mask</th>
<th>Offset</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suppression parameters</td>
<td>Location</td>
<td>Active</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Experiences with CANoe-Based Fault-Injection for AUTOSAR

Patrick E. Lanigan (planigan@ece.cmu.edu)
Experiences with CANoe-Based Fault-Injection for AUTOSAR

Patrick E. Lanigan (planigan@ece.cmu.edu)

Fault-Injection Framework
**Fault-Injection Framework**

- **CANoe**
  - Node 1
    - AUTOSAR library
  - Node 2
  - Node 3
  - Node 4
Experiences with CANoe-Based Fault-Injection for AUTOSAR

Patrick E. Lanigan (planigan@ece.cmu.edu)
FAULT-INJECTION FRAMEWORK

Fault-injection library

CANoe

Node 1

AUTOSAR library

CAPL script

Node 2

Node 3

Node 4
Experiences with CANoe-Based Fault-Injection for AUTOSAR
Patrick E. Lanigan (planigan@ece.cmu.edu)
Experiences with CANoe-Based Fault-Injection for AUTOSAR
Patrick E. Lanigan (planigan@ece.cmu.edu)
Fault-injection library
  
  Fault-injection parameters
  
  Fault-injection hook implementations
  
  CAPL extensions

CANoe
  
  Node 1
    
    AUTOSAR library
    
    CAPL script
  
  Node 2

Node 3

Node 4
Experiences with CANoe-Based Fault-Injection for AUTOSAR
Patrick E. Lanigan (planigan@ece.cmu.edu)
Experiences with CANoe-Based Fault-Injection for AUTOSAR

Patrick E. Lanigan (planigan@ece.cmu.edu)
Experiences with CANoe-Based Fault-Injection for AUTOSAR

Patrick E. Lanigan (planigan@ece.cmu.edu)

Fault-Injection Framework

- Fault-injection library
- Fault-injection parameters
- Fault-injection hook implementations
- CAPL extensions
- AUTOSAR library
- CAPL script
- Node 1
- Node 2
- Node 3
- Node 4
Fault-Injection Framework

CANoe

Fault-injection library

- Fault-injection parameters
- Fault-injection hook implementations
- CAPL extensions

Node 1

- AUTOSAR library
- CAPL script

Node 2

Node 3

Node 4

Patrick E. Lanigan (planigan@ece.cmu.edu)
Fault-Injection Framework

- Fault-injection library
  - Fault-injection parameters
  - CAPL extensions
- AUTOSAR library
- CAPL script
- Node 1
- Node 2
- Node 3
- Node 4
- Fault-injection hook implementations
Hook Locations in AUTOSAR

- Application layer
- AUTOSAR Runtime Environment (RTE)
  - Services Layer
  - ECU Abstraction Layer
  - Complex Drivers
  - Microcontroller Abstraction Layer
  - Microcontroller (ECU)
Hook Locations in AUTOSAR

Application layer

AUTOSAR Runtime Environment (RTE)

Services Layer

ECU Abstraction Layer

Complex Drivers

Microcontroller Abstraction Layer

Microcontroller (ECU)
Hook Locations in AUTOSAR

- Application layer
- AUTOSAR Runtime Environment (RTE)
  - System Services
  - Communication Services
  - ECU Abstraction Layer
  - Complex Drivers
  - Microcontroller Drivers
  - Memory Drivers
  - Communication Drivers
  - I/O Drivers

Microcontroller (ECU)

Experiences with CANoe-Based Fault-Injection for AUTOSAR
Patrick E. Lanigan (planigan@ece.cmu.edu)
Hook Locations in AUTOSAR

Application layer

AUTOSAR Runtime Environment (RTE)

System Services

Communication Services

ECU Abstraction Layer

Complex Drivers

Microcontroller (ECU)

Microcontroller Drivers

Memory Drivers

Communication Drivers

I/O Drivers
Hooke Locations in AUTOSAR

- **System Services**
  - WdgM

- **Communication Services**
  - Com

- **Microcontroller Drivers**
- **Memory Drivers**
- **Communication Drivers**
- **I/O Drivers**

---

Experiences with CANoe-Based Fault-Injection for AUTOSAR
Patrick E. Lanigan (planigan@ece.cmu.edu)
Hook Locations in AUTOSAR
Pass target arguments by reference for manipulation inside the hook. Arguments are only modified if the current location is active.

```c
SWIFI_Hook_Fr_Tx(&channel, &slotId, &baseCycle, &cycleRep, &frameStatus, 
    (uint16 *)&Fr_LSduLength, (uint8 *)Fr_LSduPtr);

CANoeAPI_SendFlexRayMessage(Fr_CtrlIdx, channel, slotId, baseCycle, 
    cycleRep, frameStatus, Fr_LSduLength, Fr_LSduPtr);
```

If the suppression hook is active, immediately return an error code.

```c
if (SWIFI_Suppress_Fr_Tx()) {
    return E_NOT_OK;
}
```
Modifiable arguments \((id, buffer)\) are passed by reference, while non-modifiable arguments \((bLength)\) provide additional context.

```c
EXPORT_C int SWIFI_Hook_Com_Write(uint16 *id, uint16 bLength, uint8 *buffer) {
  if ((gFi_Location == SWIFI_LOC_COM_WRITE) && (gFi_Operation != SWIFI_OP_NONE)) {
    switch(gFi_Argument) {
      case SWIFI_ARG_PAYLOAD:
        apply_mask(buffer, bLength, gFi_Mask, gFi_Offset, gFi_Operation);
        return 1;
      case SWIFI_ARG_SLOTID:
        apply_mask((uint8 *)id, sizeof(uint16), gFi_Mask, gFi_Offset, gFi_Operation);
        return 1;
      default:
        return 0;
    }
    return 0;
  }
  return 0;
}
```

Suppression hooks return a boolean value that indicates whether or not the instrumented function should abort.

```c
EXPORT_C int SWIFI_Suppress_Com_Write() {
  return (gFi_Location == SWIFI_LOC_COM_WRITE) && gFi_Suppress;
}
```
ROADMAP

• Introduction
• Fault-injection framework
• Runtime evaluation
  – Configuration interface
  – Demo application
  – Example scenarios and fault manifestations
• Conclusion
CONFIGURING A FAULT-INJECTION SCENARIO
Select how the mask is applied to the argument.
CONFIGURING A FAULT-INJECTION SCENARIO

Select how the mask is applied to the argument.

Activate a suppression hook.
CONFIGURING A FAULT-INJECTION SCENARIO

Select how the mask is applied to the argument.

Select the target argument.

Activate a suppression hook.
Experiences with CANoe-Based Fault-Injection for AUTOSAR

Patrick E. Lanigan (planigan@ece.cmu.edu)

Configuring a Fault-Injection Scenario

Select how the mask is applied to the argument.

Select the target argument.

Activate a suppression hook.

Define the mask...

...and offset.
**CONFIGURING A FAULT-INJECTION SCENARIO**

Select how the mask is applied to the argument.

Select the target argument.

Activate a suppression hook.

Define the mask...

...and offset.

Select the target location.
CONFIGURING A FAULT-INJECTION SCENARIO

Select how the mask is applied to the argument.

Select the target argument.

Activate a suppression hook.

Define the mask...

...and offset.

Select the target location.
**Demo Application**

- The framework was applied to a “by-wire” application
  - Sensing ECU reads throttle and brake inputs
  - Front and rear ECUs calculate wheel speed independently
  - Front ECU is the target of fault injection

- Originally developed as a simple CANoe demonstration
  - No fault-tolerance mechanisms
  - No application-level error handling

![Diagram of the application](image)
EXAMPLE FAULT-INJECTION SCENARIOS
EXAMPLE FAULT-INJECTION SCENARIONS

No active faults.
Experiences with CANoe-Based Fault-Injection for AUTOSAR

Patrick E. Lanigan (planigan@ece.cmu.edu)

**Example Fault-Injection Scenarios**

- **No active faults.**
- **Front & rear wheel speed changes equally with application of throttle.**
EXAMPLE FAULT-INJECTION SCENARIOS (CONT.)
EXAMPLE FAULT-INJECTION SCENARIOS (CONT.)

Suppression hook is active.
**EXAMPLE FAULT-INJECTION SCENARIOS** (CONT.)

- FlexRay frame reception function is the target.
  - Suppression hook is active.
EXPERIENCES WITH CANOE-BASED FAULT-INJECTION FOR AUTOSAR

Patrick E. Lanigan (planigan@ece.cmu.edu)

EXAMPLE FAULT-INJECTION SCENARIOS (CONT.)

- Front wheel speed no longer responds to changes in throttle application.
- FlexRay frame reception function is the target.
- Suppression hook is active.
EXAMPLE FAULT-INJECTION SCENARIOS (CONT.)

Experiences with CANoe-Based Fault-Injection for AUTOSAR
Patrick E. Lanigan (planigan@ece.cmu.edu)
EXAMPLE FAULT-INJECTION SCENARIOS (CONT.)

Suppression hook is active.
Com signal reception function is the target.

Suppression hook is active.
EXAMPLE FAULT-INJECTION SCENARIOS (CONT.)

Suppression hook is active.

Com signal reception function is the target.

Front wheel speed drops immediately and no longer responds to changes in throttle application.
ROADMAP

• Introduction
• Fault-injection framework
• Runtime evaluation
• Conclusion
  – Lessons learned
  – Summary
  – Questions
THE “-ILITIES” (LESSONS LEARNED REMIX)
The “-ilities” (Lessons Learned Remix)

- ✓ Excercise AUTOSAR fault-handling mechanisms
- ✓ Observe application-level manifestations
**The “-ilities” (Lessons Learned Remix)**

- **Functionality**
  - ✓ Excercise AUTOSAR fault-handling mechanisms
  - ✓ Observe application-level manifestations

- **Portability**
  - ✓ Fault-injection logic is modularized away from application code
  - ✓ Requires no changes to existing configuration files or autogenerated code
## THE “-ILITIES” (LESSONS LEARNED REMIX)

| Functionality | ✓ Excercise AUTOSAR fault-handling mechanisms  
|               | ✓ Observe application-level manifestations |
| Portability   | ✓ Fault-injection logic is modularized away from application code  
|               | ✓ Requires no changes to existing configuration files or autogenerated code |
| Controllability | ✓ Good control of fault location by layer, component, function, data structure |
|               | X Manual control makes duration and time of faults very difficult to control |
THE “-ILITIES” (LESSONS LEARNED REMIX)

- **Functionality**
  - ✓ Excercise AUTOSAR fault-handling mechanisms
  - ✓ Observe application-level manifestations

- **Portability**
  - ✓ Fault-injection logic is modularized away from application code
  - ✓ Requires no changes to existing configuration files or autogenerated code

- **Controllability**
  - ✓ Good control of fault location by layer, component, function, data structure
  - X Manual control makes duration and time of faults very difficult to control

- **Observability**
  - X Observability is limited to application-level behavior

Experiences with CANoe-Based Fault-Injection for AUTOSAR
Patrick E. Lanigan (planigan@ece.cmu.edu)
Experiences with CANoe-Based Fault-Injection for AUTOSAR

Patrick E. Lanigan (planigan@ece.cmu.edu)

THE “-ilities” (LESSONS LEARNED REMIX)

- **Functionality**
  - ✓ Excercise AUTOSAR fault-handling mechanisms
  - ✓ Observe application-level manifestations

- **Portability**
  - ✓ Fault-injection logic is modularized away from application code
  - ✓ Requires no changes to existing configuration files or autogenerated code

- **Controllability**
  - ✓ Good control of fault location by layer, component, function, data structure
  - X Manual control makes duration and time of faults very difficult to control

- **Observability**
  - X Observability is limited to application-level behavior

- **Flexibility**
  - X Protocol-level errors cannot be injected directly because the simulated communication controller is not accessible from software
## The “-ilities” (Lessons Learned Remix)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Functionality** | ✓ Excercise AUTOSAR fault-handling mechanisms  
                     ✓ Observe application-level manifestations |
| **Portability**   | ✓ Fault-injection logic is modularized away from application code  
                     ✓ Requires no changes to existing configuration files or autogenerated code |
| **Controllability** | ✓ Good control of fault location by layer, component, function, data structure  
                     X Manual control makes duration and time of faults very difficult to control |
| **Observability** | X Observability is limited to application-level behavior |
| **Flexibility**   | X Protocol-level errors cannot be injected directly because the simulated communication controller is not accessible from software |
| **Avoid Probe Effects** | X Certain faults cannot be isolated to simulated nodes (e.g., memory-access faults crash the entire simulation) |
THE “-ilities” (LESSONS LEARNED REMIX)

- **Controllability**
  - ✓ Good control of fault location by layer, component, function, data structure
  - X Manual control makes duration and time of faults very difficult to control

- **Observability**
  - X Observability is limited to application-level behavior

- **Implementation limitations**

- **Flexibility**
  - X Protocol-level errors cannot be injected directly because the simulated communication controller is not accessible from software

- **Avoid Probe Effects**
  - X Certain faults cannot be isolated to simulated nodes (e.g., memory-access faults crash the entire simulation)
**The “-ilities” (Lessons Learned Remix)**

- **Controllability**
  - ✔️ Good control of fault location by layer, component, function, data structure
  - ✗ Manual control makes duration and time of faults very difficult to control

- **Observability**
  - ✗ Observability is limited to application-level behavior

- **Flexibility**
  - ✗ Protocol-level errors cannot be injected directly because the simulated communication controller is not accessible from software

- **Avoid Probe Effects**
  - ✗ Certain faults cannot be isolated to simulated nodes (e.g., memory-access faults crash the entire simulation)

---

Experiences with CANoe-Based Fault-Injection for AUTOSAR
Patrick E. Lanigan (planigan@ece.cmu.edu)
SUMMARY

- CANoe is a suitable environment for fault-injection, provided that the faults injected fall within the level of abstraction provided by the simulation.
- This proof-of-concept framework shows promise for rapid-prototyping of new applications.
- A combination of techniques are likely required for a robust analysis.
Questions?

Patrick E. Lanigan
planigan@ece.cmu.edu