Safety-critical systems

Simin Nadjm-Tehrani

www.ida.liu.se/~rtslab
Department of Computer & Information Science
Linköping University, Sweden
and
University of Luxembourg
Linköping group @NODES

- Real-time systems laboratory
  - Dependability, Distributed systems, Formal analysis
  - Four PhD students, 5 examined PhDs in 2005-07
  - Recruiting 2 PhD students and a post doc ...

- Intelligent information systems laboratory
  - Security, P2P systems, databases & web information systems
  - Five PhD students, 5 examined PhDs in 2005-07
• How can we produce computer systems that do their job, and how to prove or measure how well they do their jobs?
Engineers: Fool me once, shame on you – fool me twice, shame on me
Software developers: Fool me N times, who cares, this is complex and anyway no one expects software to work...
"If you have a problem with your Volkswagen the likelihood that it was a software problem is very high. Software technology is not something that we as car manufacturers feel comfortable with."

Bernd Pischetsrieder, chief executive of Volkswagen
• “Automaker Toyota announced a recall of 160,000 of its Prius hybrid vehicles following reports of vehicle warning lights illuminating for no reason, and cars' gasoline engines stalling unexpectedly.”

Wired 05-11-08

• The problem was found to be an embedded software bug
Angel Eck, driving a 1997 Pontiac Sunfire found her car racing at high speed and accelerating on Interstate 70 for 45 minutes, heading toward Denver ...

... with no effect from trying the brakes, shifting to neutral, and shutting off the ignition.
### Driver support: Volvo cars

<table>
<thead>
<tr>
<th>Year</th>
<th>Feature Description</th>
<th>Year</th>
<th>Feature Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1984</td>
<td>ABS Anti-lock Braking System</td>
<td>2004</td>
<td>Blind Spot Information system (BLIS)</td>
</tr>
<tr>
<td>1998</td>
<td>Dynamic Stability and Traction Control (DSTC)</td>
<td>2006</td>
<td>Active Bi-Xenon lights</td>
</tr>
<tr>
<td>2002</td>
<td>Roll Stability Control (RSC)</td>
<td>2006</td>
<td>Adaptive Cruise Control (ACC)</td>
</tr>
<tr>
<td>2003</td>
<td>Intelligent Driver Information System (IDIS)</td>
<td>2006</td>
<td>Collision warning system with brake support</td>
</tr>
</tbody>
</table>
Early space and avionics

- During 1955, 18 air carrier accidents in the USA (when only 20% of the public was willing to fly!)

- Today’s complexity many times higher
- Integrated modular avionics (IMA), with safety-critical digital components, e.g.
  - Power-by-wire: complementing the hydraulic powered flight control surfaces
  - Cabin pressure control (implemented with a TTP operated bus)
What is safety?

- IFIP WG 10.4 definition: **Safety**: Absence of catastrophic consequences on the user(s) and the environment  
  [Avizienis et al]

- Freedom from exposure to danger, or exemption from hurt, injury or loss  
  [Bowen and Stavridou]
Programs are always safe!

- According to these definitions software can only **contribute** to unsafe behaviour

- Safety is a system level property, and can be claimed/assured at system level
- Differs from reliability

- Closely related to **risk**
System safety & Hazards

• Safety: achieved by anticipating accidents, and eliminating their causes

• Hazards are potential causes of accidents

*Conditions in a system which together with other factors in the environment inevitably cause accidents.*
Fault to Accident

- Fault
- Error
- Failure
- Hazard
- Accident
Safety & risk management

- Means anticipating accidents...
- hence anticipating hazards ...
- which means quantifying/classifying the potential ...
- Must reduce risks which are not tolerable!
- Result: construction of safety Case

Think negative!
Structure of SC systems

IEC 61508

- Equipment under control (EUC)
- Control system
  - Safety functions
- Protection system
  - Safety functions
Overall safety lifecycle

1. Concept
2. Overall scope definition
3. Hazard and risk analysis
4. Overall safety requirements
5. Safety requirements allocation

D
Overall planning of:

6 O & M
7 Safety validation
8 Installation & Commissioning

Realisation of:

9 Safety-related E/E/PES
10 Other technical safety-related systems
11 External risk reduction facilities

12 Overall installation & commissioning
13 Overall safety validation
14 Overall operation, maintenance & repair
15 Decommissioning or disposal
16 Overall modification & retrofit
But how does this fit in classical (software) systems development process?
Violation of safety

Patterns for safety analysis?
Traditional Safety Analysis

Fault Tree Analysis (FTA)

Top event
### Traditional Safety analysis

**Failure modes and events analysis (FMEA):**

- What are the consequences of some particular component’s failure?

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Failure Mode</th>
<th>Effects of failure</th>
<th>Cause of failure</th>
<th>Actions</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Value Failure</td>
<td>?</td>
<td>Sensor Malfunction</td>
<td>Duplicate sensors</td>
<td>...</td>
</tr>
</tbody>
</table>

...
• Adaptive Cruise Controller (ACC)
• Extension to a traditional cruise control
  – adapts vehicles speed to the speed and distance of
    the vehicle in front

• Identify the hazards and their risks
Fault tree analysis

Collision

ACC enabled

+ No output signal

ACC disabled

Faulty output signal

Undesired output signal
No output signal

- ACC enabled
  - +
  - No output signal
    - +
    - Communication failure
    - Logic error
    - Physical fault
    - Faulty sensor input
    - Absent sensor input
Undesired output signal

ACC disabled

Undesired output signal

Communication failure

Logic error

Physical fault
Growing complexity

FTA:

Top event

Software/Digital hardware
Focus on safety

- Faults that are probable and may cause failures that lead to hazards are in focus.

- The system should be shown to avoid hazardous failures even in presence of these faults.
Pattern: Functional verification

Formal Verification bench

Component

Environment

Observer

Alarm
Pattern: Fault mode analysis
• A fault library can be created in design tools

• Fault mode classification:
  • Value faults
  • Omission faults
  • Commission faults

Examples of faults
• Stuck-at
• Bit-flips

Diagram:
- Fault trigger
- Input
- Output
- Block diagram with labeled components.
Adding components (upgrades)

The pattern works if:

- The system is developed in one organisation
- All source code (all models) are available
- Formal analysis of the composition is not prohibitive (size, time)
Component-based Development

- CBD is an emerging trend in software systems

- Problem: no component models address safety properties!
Components & Interfaces

- *Software component Interfaces* provide all information needed for *composition*.

- How should the interface look like in order to capture safety?
• A safety property $\varphi$ is typically defined at system-level

• Our approach:
  - Interface captures information about behavior of component in presence of faults in the system

$$\vdash \varphi \Rightarrow \exists + \vdash \varphi$$
• $\varphi$ : When the ACC is in ACC-Mode, the speed is higher than 50 km/h and there is a vehicle in front closer than 50 m, the ACC should not accelerate
• Formal definition: $C = \langle SI^\phi, M \rangle$

For a given set of faults

How do single and double faults in the environment of $M$ affect the safety property $\phi$?

• Given a set of faults $F$, a safety property $\phi$, and a model $M$, the safety interface $SI^\phi$ describes the single and double faults in $F$ that $M$ is resilient to
Environment Abstraction

• Dilemma with CBD:
  – The fewer assumptions about the environment the more useful the notion of component
  – In order to guarantee something, assumptions must be made

• Solution: include some assumptions about the environment in the safety interface $SI^\varphi$

$$C = \langle SI^\varphi, M \rangle$$

$SI^\varphi = \langle E^\varphi, \text{single, double} \rangle$ where

- single $= \langle \langle F_1^s, A_1^s \rangle, ..., \langle F_m^s, A_m^s \rangle \rangle$ and
- double $= \langle \langle F_1^d, A_1^d \rangle, ..., \langle F_k^d, A_k^d \rangle \rangle$
And ...

• Provide help in generating them!
Environment Generation Algorithm

- Support for computing the Interface implemented in SCADE
Environment Abstraction

\[ C = \langle SI^\varphi, M \rangle \]

\[ SI^\varphi = \langle E^\varphi, \text{single, double} \rangle \]

\[ E^\varphi \text{ is the weakest environment in which } C \text{ will be "safe" with no faults} \]

\[ E^\varphi \parallel M \models \varphi \]
Environment Abstractions

\[ C = \langle SI\phi, M \rangle \]

or \[ \langle E\phi, \text{single, double} > \] where

- single = \[ \langle \langle F_1^s, A_1^s \rangle, \ldots, \langle F_m^s, A_m^s \rangle \rangle \]
- double = \[ \langle \langle F_1^d, A_1^d \rangle, \ldots, \langle F_k^d, A_k^d \rangle \rangle \]

\[ F_i^d \text{ is a pair of faults} \]

\[ F_i^s \text{ is a single fault} \]

Abstraction of the environment in which \( C \) will tolerate the single fault \( F_m^s \)

Abstraction of the environment in which \( C \) will tolerate the double fault \( F_k^d \)
Component-Based Safety Analysis

$SI_1^\phi = \langle E^\phi, \text{single, double} \rangle$

single = $\langle F_1^s, A_1^s \rangle, \ldots, \langle F_m^s, A_m^s \rangle$

double = $\langle \langle F_1^d, A_1^d \rangle, \ldots, \langle F_k^d, A_k^d \rangle \rangle$

If $F_3$ appears in single, then it suffices to prove that the environment of $M_1$ is more constrained than $A_3$

- However, infeasible to compose all components and check $M_2 || \ldots || M_m \leq A_3$

- Solution: Assume-Guarantee reasoning
Assume-Guarantee reasoning

To show that environment of $M_1$ is more specific than $A_3$, show that:

- $C_1$ with the fault $F_3$ at its input still satisfies environment requirement of every other component

For all $j$: $A_3 \circ F_3 \parallel M_1 \leq E_j^\varphi$

- individual components and their weakest environments are more specific than $A_3$

For all $j$: $M_j \parallel E_j^\varphi \leq A_3$
Resilience to double faults

- At system level is proved similarly

- Proof rules that take account of:
  - Double faults in one component
  - Two single faults affecting two different components
Safety analysis using $SI^\phi$

$C = \langle SI^\phi , M \rangle$

System integrator
Safety engineer

Workflow

Component modeling
Generating safety interfaces
EGA

Component developers

feedback
ACC: Safety Analysis Result

- Of the 20 fault modes considered the ACC is resilient to:
  - 8 single faults
  - 2 double faults

- Parts of safety analysis from one fault can be reused later

- However: safety analysis is not finished here!