A Framework for Designing Reliable Software-Intensive Systems

Irem Y. Tumer, Oregon State University
Carol S. Smidts, Ohio State University

AFRL Safe and Secure Systems and Software Symposium
Beavercreek, OH
June 15-17, 2010
Introductions & Outline

Research funded by AFOSR BAA, Software & Systems Program

A Framework for designing reliable software-intensive systems
  - Oregon State University PI: Dr. Irem Tumer: FA 9550-08-1-0158
  - Ohio State University PI: Dr. Carol Smidts: FA 9550-08-1-0139

- Motivation: Integrated Complex System Analysis & Design
- Hardware world: FFIP Approach
- Software world: FPSA Approach
- Integration of hardware and software worlds
Design of (Safety-Critical) Complex Systems

Complex systems are everywhere....

....engineering systems with complex interactions and behavior!
Design of Complex Systems

Importance of “systems thinking”

- Emergent behavior attributed to the entire system due to interactions
- Key characteristic of complexity causing failures!

Importance of design stage analysis

- First and best place to catch errors
- Majority of publicized failures should have been caught early in design!

Importance of model based design

- Models play a key role for designers to analyze systems
- Many different types with different fidelity levels to work with!
Design of Complex Systems

Focus on hybrid systems composed of software & hardware

✓ Significant interactions and dependencies are difficult to capture!
✓ Traditional adherence to separate design processes is a problem!

ISHM/PHM has been a good example of poor integration!

Focus on failure/reliability analysis during early design stages

✓ Lack of detailed data and knowledge is a problem!
✓ Traditional methods need more detailed information about systems!

Ability to reason with qualitative information is needed!
Motivation:
- Hardware and software typically designed separately
- HW and SW failure/reliability analysis done separately

Overall Goal:
- Develop a formal framework to design and analyze software and hardware systems concurrently, focusing on safety and reliability

Specific Objectives:
- Formalization and automation of failure identification and propagation analysis during early design
- Extension to software domain including semantics & syntax
- Addition of failure quantification for SW and HW
- Application to real-world software-intensive systems
Function-Failure Identification & Propagation (FFIP) Analysis Framework

BEFORE a design is realized:
- Explore what-if types of failure scenarios
- Explore effects on overall system functionality under off-nominal conditions
- Reason about effects from causes using higher-level reasoning to map system behavior and interactions to functional failures
FFIP Analysis Framework

Mapping System Representations

Representation of Framework
FFIP – System Representation

Functional Modeling

- Block diagram representation of system functions
  - Generic representation of different technical domains
- Uses the Functional Basis developed for NIST project
  - Semi-formal ontology of Functions (verb-noun pairs)
  - Flows go in and out of each block (Energy, Material, Signal)
FFIP – System Representation

Configuration Flow Graph (CFG)

- Generic component implementation of functions
- Hierarchal system representation
- Related to Functional Model based on EMS Flows
Each component in CFG has a set of nominal and failure modes

- For each mode, a specific behavior defines output flow levels in terms of inputs flow levels

Flows discretized into qualitative levels

- e.g., Zero, Low, Nominal, High

Mode changes in simulation occur by:

- Injection of critical events by user – e.g. “fly wheel cut off”
- Input flow level changes – e.g. input power loss
- Internal component logic – e.g. timeout

Models implemented with Statecharts or if-then-else rules

Simulation done in ModelCenter
**FFIP – Function-Failure Logic**

FFL relates output flow levels determined by component behavioral simulation to the health state of functions.

**Component**

**Function Failure Logic: Guide Electrical Energy**

**State: Healthy**

Control Signal \{Level\} = Output Electrical Energy \{Level\}

**State: Lost**

Control Signal \{Level\} \neq \{Zero\} \& Output Electrical Energy \{Level\} = \{Zero\}

**State: Degraded**

Control Signal \{Level\} \neq Output Electrical Energy \{Level\}

**State: Lost Recoverable**

Control Signal \{Level\} = \{No Signal\} or Input Electrical Energy \{Level\} = \{Zero\}
International Collaboration with Aalto University & Finland Safety Authority: Apply FFIP to nuclear reactor turbine hazard study

FFIP – Validation Case Study

Nuclear Reactor

Preliminary Models

Extension to Designing Control Algorithms
Collaboration with NASA Ames: FFIP-Based Design and Development of Electro Mechanical Actuator Test Stand Design

FFIP – Validation Test Stand

Typical EMA usage

C-17 Platform

UH-60 Platform

Preliminary testbed design
Integration with Software Analysis

Hardware domain status:

- FFIP works very well for electromechanical systems
- Additional reasoning method developed to capture non-nominal flows
- Simulation currently implemented in ModelCenter; automated simulation
- Demonstration planned through design of EMA teststand
- Applications to rocket engines and nuclear reactor: Role of software critical!

What about the software domain representation?

- Unified Modeling Language for software
- FPSA: fault-failure error propagation and simulation approach
- Case-study: Helium tank subsystem

Proposed integration of FPSA & FFIP

- Currently in-progress
The Unified Modelling Language (UML) is a de facto standard for object-oriented software modelling.

UML Research includes:
- Formalizing UML to capture the dynamic aspects of software systems
- Performing fault diagnosis
- Risk assessment
- UML model transformation

UML for V&V:
- Transformation of UML models into Petri nets
- Relationship between UML design and fault tree symbology

UML’s Extendibility:
- UML profiles (stereotypes, tag values and constraints)
Classification of UML Diagrams

- UML diagrams are classified into behavioral and structural diagrams.
- There are seven behavioral and six structural diagrams, each diagram conveys different information.

- Show how the system is actually executed and assigned to various pieces of hardware. Used to typically show how components are configured at runtime.
- Capture the flow from one behavior or activity to the next. They are similar to classic flowcharts but are more expressive.
- Capture functional requirements for a system. Focus is on user needs rather than realization details.

UML diagrams are classified into behavioral and structural diagrams. There are seven behavioral and six structural diagrams, each diagram conveys different information.
FPSCA Approach

FPSCA: Failure Propagation Simulation Approach

- Mimics FFIP on the software side
- Unified Modeling Language (UML) based methodology

FPSCA aims at propagating faults through UML diagrams directly. Since each UML diagram captures a particular design aspect, propagation through each diagram allows us to explore particular characteristics of the failure (timing, functionality, state).

- Mapping between different elements of UML diagrams

Mapping between different elements of UML helps to explore the effect of propagation of faults within a particular UML diagram and across different diagrams. Navigation from one diagram to another provides a unique feature of studying all the aspects of software design in single execution.

- Rules and functional failure logic (FFL)

Rules are developed for each component which are further used to build the FFL to study the functional effect of a fault/error on software system.

- Result table

Observations of the simulation are recorded in a result table which is further used for analysis of the failure.
The mapping between different UML diagrams is an initial and necessary step in the development of fault propagation rules in software systems.

- Use case can be mapped to activity and sequence diagram.
- For instance, use-case1 of Use case diagram is related to action 1 and action 2 of activity diagram.
- A more formal representation can be seen in next slide.
Mapping Between UML Diagrams

- A UML meta-model, a formal representation, can be developed to formalize relationships existing between UML diagrams.
- One use case consists of one or many actions and many control nodes.
- One source action can connected to many target action or many control nodes by an activity edge.
Approach: Basic Steps

- FPSA consists of 12 basic steps, which include mapping between different elements of UML diagrams, development of component level rules and functional failure logic (FFL), and simulation.

1. Select use cases and identify functions
2. Map activities to components
3. Map Use Case to Activity diagram
4. Map Activity to Sequence diagram
5. Map objects to Class diagram
6. Map Activity to State Machine diagram
7. Map Component to Class diagram
8. Map Component to Deployment diagram
9. Identify input-output variables for each component
10. Define rules for nominal and faulty behavior of each component
11. Derive the functional failure logic for each component
12. SIMULATION
Approach: Simulation

1. Identify propagation paths in the Activity
   - Identify steps in the form of actions and control nodes
     - Build simulation table
     - Initialize all objects’ classes
     - Start simulation (step 1)
   - Identify active components
     - Identify active use cases through Use Case to Activity mapping
   - Identify active objects through Activity to Sequence mapping

2. Identify active objects’ classes through the Sequence diagram
   - Identify all states of a class through Activity to State Machine mapping
     - Identify input-output variables and assign values
     - Apply rules to components
     - Apply functional failure logic
     - Identify function status
   - Update simulation

3. Increment simulation

4. Last step?
   - Yes
     - RESULT ANALYSIS
   - No
Case Study: Helium Tank

- **System Status**: The isolation valves and pressure regulators are closed
- **Expected function**: OPEN ISOLATION VALVES and OPEN PRESSURE REGULATORS
- **Error**: Incorrect position information is provided by the ‘PLC_interface’ software component i.e. open instead of close
Case Study: Helium Tank

- The Use case diagram consists of seven use cases/functions marked from <F1> to <F7>
- The stick figure in the diagram represents the actors interacting with the system “Tank Control”

The Isolation valve 1 and Isolation valve 2 receive the command to open or close the valve through use case/function <F5>

Use-Case Diagram
Case Study: Helium Tank

- The Deployment diagram consists of six software components located within the control unit which communicate with the three external systems.
- The input and output variables of each system and component can be depicted on the deployment diagram.

- $db1, p1, t1, q2$: Input variables external to the software system.
- $p3, t3, q5$: Output variables external to the software system.
- Remaining are the internal variables of the software system.
Case Study: Helium Tank

- Mark the simulation steps
- Simulation starts at step 1 and proceeds to successive step. We apply the rules and FFL developed for each component at each step.
- The rules and FFL for the temperature sensor are given below.

Rules and FFL for temperature sensor

**Functional Failure Logic (FFL):**

A) Internal observer

IF INOM1

then F3 = operating

Elseif IFAULTY1

then F3 = Lost

Else F3 = Unknown

B) External observer

IF ENOM1

then F3 = operating

Elseif EFAULTY1

then F3 = Lost

Else F3 = Unknown

---

**Rules:**

A) Internal observer

- Nominal behavior
  - INOM1) IF t1 exist then t2 = t1
- Faulty behavior
  - IFAULTY1) IF t1 exist then t2 ≠ t1

B) External observer

- Nominal behavior
  - ENOM1) IF t1 exist
- Faulty behavior
  - EFAULTY1) IF t1 does not exist

---

**Diagram:**

- Mark the simulation steps
- Simulation starts at step 1 and proceeds to successive step. We apply the rules and FFL developed for each component at each step.
- The rules and FFL for the temperature sensor are given below.

---

**Diagram Elements:**

- Configuration Manager
- Log sensor data
- Enable interrupts
- Configuration software
- Read temperature data
- Read pressure data
- Main_activity - Processing logic
- Temperature sensor
- t1
- t2
- Step 1
- Step 2
- Step 3
- Step 4
- Step 5
- Step 6
### Case Study: Helium Tank

<table>
<thead>
<tr>
<th>STEPS</th>
<th>Step 1</th>
<th>Step 2</th>
<th>...</th>
<th>Step 8</th>
<th>Step 9</th>
<th>Step 10</th>
<th>Step 11</th>
<th>Step 12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component Behavior</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFALITY1</td>
<td>first seen at step 8. This does not affect the behavior of other components till step 12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Isolation valve</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLC interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Object state</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Isolation valve1</td>
<td>Inactive</td>
<td>Inactive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLC interface1</td>
<td>Inactive</td>
<td>Inactive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>State variable</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Close</td>
</tr>
<tr>
<td>q3</td>
<td>blank</td>
<td>blank</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Open</td>
</tr>
<tr>
<td>v2</td>
<td>blank</td>
<td>blank</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>blank</td>
</tr>
<tr>
<td>q5</td>
<td>blank</td>
<td>blank</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>blank</td>
</tr>
<tr>
<td><strong>Function status</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open/Close Isolation Valve(s) (F5)</td>
<td>-</td>
<td>-</td>
<td></td>
<td>Unknown</td>
<td>Operating</td>
<td>Operating</td>
<td>Operating</td>
<td>Lost</td>
</tr>
</tbody>
</table>

- The faulty behavior of ‘PLC interface’ has propagated and its effects is seen much later on the isolation valve, which shows EFAULTY2 behavior.
- The EFAULTY2 behavior results in loss of function F5.
Benefits

Identification of fault source
The simulation results observed can be tabulated. This table can be used to identify the source of fault.

Multiple fault propagation
Multiple faults can be injected in the design and propagated through the software at the same time. This special characteristic of the method can us investigate the effect of CCF.

Both structural and behavioral aspects can be investigated
FPSA records both the structural aspects like class, attributes, components etc and behavioral aspects like function, action, decision etc. of the software. Hence multiple characteristics of the software can be investigated in single execution.

Focus on elements of high importance
The result table can be modified to focus on the elements of high importance such as class, state, timing, function etc.

The FFIP like UML representation
Representation similar to FFIP can be adopted for UML based software design. Hence integration of two methodologies to study the fault propagation and study the effect on the entire system is possible.
Hardware-Software Integration: FFIP vs. UML
Future Work

Planned Extensions:

- Design of an electro-mechanical actuator testbed (NASA)
- Automation within a virtual design environment (NSF)
- Model-based testing (NSF/AFOSR)
- Verification (NASA/NSF/AFOSR)
Contact Information

Dr. Irem Y. Tumer  
Director, Complex Engineered System Design Laboratory  
School of Mechanical, Industrial, and Manufacturing Eng.  
Oregon State University  
irem.tumer@oregonstate.edu

Dr. Carol S. Smidts  
Director, Center for Academic Excellence in Digital Control Systems and Safety  
Department of Mechanical Engineering  
Ohio State University  
smidts.1@osu.edu