Tools and Procedures at GRS to Automatically Compare and Modify Fault and Event Trees

Joachim Herb, GRS
2012-12-10/11
Next Generation PSA Software, Methods, and Model Representation Standards
Paris, France
Motivation

PSA have become state-of-the-art since several decades (e.g. in Germany PSA are obligatory since the 1990s)

⇒ New versions of a PSA are based on already reviewed older versions
⇒ Therefore it is important to identify the differences between different versions of a PSA

Several fault tree analysis cases (e.g. of highly redundant systems) require structurally similar fault trees

- Modeling of common cause failures
- Modeling of digital I&C systems

⇒ Generate fault trees “automatically”, i.e. let the computer do the work
⇒ Create a programming language (domain specific language) to model fault trees

- Most PSA in the world (and all within Germany) use RiskSpectrum®
⇒ Common interface to stored models and data
Review Approaches for PSA Updates

Top-down

- Start from event trees:
  - Compare sequences and consequences
- Recursively check for changes in
  - Function events
  - Fault trees
  - Basic event
  - Parameters
Review Approaches for PSA Updates

Bottom-up
- Start from parameters
- Find effects on
  - Basic events
  - Fault trees
  - Function events
  - Event trees’ sequences and consequences
Architecture of GRS Tools to Support the Review of PSA Changes

Library of classes representing:
- Event trees
- Fault trees (Fault tree pages)
- Parameters

Different tools:
- Extract data, fault/event tree models from RiskSpectrum®
- Identify differences of fault/event trees

Find fault trees which are part of different versions of the PSA

Starting from the TOP gate, check for changes of these fault trees:
- Type of node (AND, OR, K/N, basic event, transfer gate, …)
- Transfer source (fault tree used as input for transfer gate)
- ID of node

Repeat recursively for all children nodes

Output differences-graphs in pdf files
GRS Tool: Generate Differences-Graphs of Fault Trees
GRS Tool: Generate Differences-Graphs of Fault Trees
GRS Tool: Generate Differences - Graphs of Fault Trees
"Expand" fault trees up- or downwards:
Replace recursively all transfer gates through the referenced fault trees

Downward expansion:
- Generate lists of all gates and basic events connected to a specified TOP gate
- Mark a specified fault tree as recursively equal, if:
  - Fault tree (page) itself has not changed
  - All fault trees (pages) referenced by input transfer gates have not changed
- For a not recursively equal fault tree output the differing referenced fault trees (pages)

Upward expansion:
- Generate list of all TOP gates affected by changes in a certain fault tree
Example for Downward Expansion
Example for Downward Expansion
## Example Applications of the GRS Tools

- Review of an updated L1 PSA
- Quality assurance processes during establishment of a Fire PSA

<table>
<thead>
<tr>
<th>Case</th>
<th># of Fault Trees</th>
<th># of Differing Fault Trees</th>
<th># of Event Trees</th>
<th># of Differing Event Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review (PSA L1)</td>
<td>2554/2570</td>
<td>53</td>
<td>37/38</td>
<td>17</td>
</tr>
<tr>
<td>Fire PSA</td>
<td>1902/1928</td>
<td>616</td>
<td>75/75</td>
<td>33</td>
</tr>
</tbody>
</table>
Data Link to Modify RiskSpectrum® Database using GRS RiskLang

Internal data link to RiskSpectrum®
- RiskLang parser generates Ruby objects
- Open source library ActiveRecord is used to map these objects onto the tables in the RiskSpectrum® database
- Data are added to RiskSpectrum® database through ODBC-drivers using SQL commands

Housekeeping
- All imported data are tagged in RiskSpectrum®
- User and edit date is set to new value
- Review/approval information is reset

Export fault trees from RiskSpectrum® in RiskLang
- Generate templates
## RiskLang Specification to Model Fault Trees

Based on the programming language Ruby

Add three commands to create fault trees, fault tree nodes, basic events/gates

<table>
<thead>
<tr>
<th>Command</th>
<th>Argument</th>
<th>Data Type</th>
<th>Required</th>
<th>Event Types</th>
<th>RiskSpectrum® Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT</td>
<td>:ID</td>
<td>String (max length 21)</td>
<td>Yes</td>
<td>:circle</td>
<td>Basic event</td>
</tr>
<tr>
<td>FT</td>
<td>:Top</td>
<td>FTNode</td>
<td>Yes</td>
<td>:diamond</td>
<td>Diamond basic event</td>
</tr>
<tr>
<td>FT</td>
<td>:Text</td>
<td>String</td>
<td>Optional</td>
<td>:house</td>
<td>House event</td>
</tr>
<tr>
<td>FTNode</td>
<td>:Event</td>
<td>Event</td>
<td>Yes</td>
<td>:orgate</td>
<td>OR gate</td>
</tr>
<tr>
<td>FTNode</td>
<td>:Transfer</td>
<td>Event</td>
<td>Yes</td>
<td>:andgate</td>
<td>AND gate</td>
</tr>
<tr>
<td>FTNode</td>
<td>:Pos</td>
<td>Integer</td>
<td>Optional</td>
<td>:kofn</td>
<td>K/N gate</td>
</tr>
<tr>
<td>FTNode</td>
<td>:InLevel</td>
<td>Integer</td>
<td>Optional</td>
<td>:norgate</td>
<td>NOR gate</td>
</tr>
<tr>
<td>FTNode</td>
<td>:Children</td>
<td>Array of FTNode</td>
<td>Optional</td>
<td>:nandgate</td>
<td>NAND gate</td>
</tr>
<tr>
<td>Event</td>
<td>:ID</td>
<td>String (max length 21)</td>
<td>Yes</td>
<td>:xorgate</td>
<td>XOR gate</td>
</tr>
<tr>
<td>Event</td>
<td>:Type</td>
<td>Event Type</td>
<td>Yes</td>
<td>:comment</td>
<td>Comment gate</td>
</tr>
<tr>
<td>Event</td>
<td>:Text</td>
<td>String</td>
<td>Optional</td>
<td>:continue</td>
<td>Continuation gate</td>
</tr>
<tr>
<td>Event</td>
<td>:Model</td>
<td>Integer</td>
<td>Optional</td>
<td>:notfound</td>
<td>-</td>
</tr>
<tr>
<td>Event</td>
<td>:CalcType</td>
<td>Integer</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exemplary Fault Tree
require "RiskRobot/ParseFT"
RiskSpectrumConnection.init('C:\temp\k2.rpp')

FT(:ID=>"EX1", :Text=>"Example fault tree", :top=>
  FTNode(:Event=>Event(:ID=>"TOP1", :Type=>:orgate, :CalcType=>1, 
    :Text=>"TOP of EX1"), :Pos=>1, :InLevel=>0, :Children=>[
    FTNode(:Event=>Event(:ID=>"OR1", :Type=>:orgate, :CalcType=>1), 
      :Pos=>1, :InLevel=>1, :Children=>[
        FTNode(:Event=>Event(:ID=>"B1", :Type=>:circle, :Model=>3, :CalcType=>1, 
          :Text=>"Basic element 1"), :Pos=>1, :InLevel=>1),
        FTNode(:Event=>Event(:ID=>"B2", :Type=>:circle, :Model=>3, :CalcType=>1, 
          :Text=>"Basic element 2"), :Pos=>2, :InLevel=>1)
      ],
    FTNode(:Event=>Event(:ID=>"AND1", :Type=>:andgate, :CalcType=>1), 
      :Pos=>3, :InLevel=>1, :Children=>[
      FTNode(:Event=>Event(:ID=>"B3", :Type=>:circle, :Model=>3, :CalcType=>1, 
          :Text=>"Basic element 3"), :Pos=>3, :InLevel=>1),
      FTNode(:Event=>Event(:ID=>"B4", :Type=>:diamond, :Model=>3, :CalcType=>1, 
          :Text=>"Basic element 4"), :Pos=>4, :InLevel=>1)
    ])
  ])
)
System Overview of Generic Digital I&C Example

AIM | Analogous Input Module
---|---
ASM | Analogous Signal Module
AU | Acquisition Unit
BP | Backplane
CM | Communication Module
CP | Communication Processor
DIM | Digital Input Module
DOM | Digital Output Module
ECCP | Emergency Core Cooling Pump
F | Flow Sensor
L | Level Sensor
LM | Link Module
PM | Processing Module
PU | Processing Unit
SV | Solenoid Valve
T | Temperature Sensor
VU | Voting Unit

Next Generation PSA Software, Methods, and Model Representation Standards, Paris, France
## Generic Fault Trees of Digital I&C System

<table>
<thead>
<tr>
<th>Fault Tree ID (wildcards {r}, {r2}, {m}, {comb})</th>
<th>Description</th>
<th>Number of fault trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>2O4_VU{r}.A_{comb}_{m}</td>
<td>OR of all combinations leading to a failure of 2 out of for 4 voter</td>
<td>372</td>
</tr>
<tr>
<td>AU{r}.A_{m}_FNSS</td>
<td>Non-self-signaling failure AU</td>
<td>12</td>
</tr>
<tr>
<td>AU{r}.A_FSS</td>
<td>Self-signaling failure AU</td>
<td>4</td>
</tr>
<tr>
<td>COM_AU{r}.A_PU{r2}.A</td>
<td>Failure communication AU-&gt;PU</td>
<td>16</td>
</tr>
<tr>
<td>COM_PU{r}.A_VU{r2}.A</td>
<td>Failure communication PU-&gt;VU</td>
<td>16</td>
</tr>
<tr>
<td>ECCP{r}_{m}</td>
<td>Failure to start ECCP</td>
<td>8</td>
</tr>
<tr>
<td><strong>F2M_PU{r}.A_{comb}_{m}</strong></td>
<td>All combinations leading to a failure of second min processing unit</td>
<td>516</td>
</tr>
<tr>
<td>OUT_AU{r}.A_{m}_FNSS</td>
<td>Non-self-signaling failure of output AU</td>
<td>12</td>
</tr>
<tr>
<td>OUT_AU{r}.A_{m}_FSS</td>
<td>Self-signaling failure of output AU</td>
<td>12</td>
</tr>
<tr>
<td>OUT_PU{r}.A_{m}_FNSS</td>
<td>Non-self-signaling failure of output PU</td>
<td>12</td>
</tr>
<tr>
<td>OUT_PU{r}.A_{m}_FSS</td>
<td>Self-signaling failure of output PU</td>
<td>12</td>
</tr>
<tr>
<td>PU{r}.A_FNSS</td>
<td>Non-self-signaling failure PU</td>
<td>4</td>
</tr>
<tr>
<td>PU{r}.A_FSS</td>
<td>Self-signaling failure PU</td>
<td>4</td>
</tr>
<tr>
<td>PU{r}<em>F2M</em>{m}_FNSS</td>
<td>Non-self-signaling failure of second min due to input failures from AU in PU</td>
<td>12</td>
</tr>
<tr>
<td>SIG_RED{r}<em>FSS</em>{m}</td>
<td>Self-signaling failure of analogous input AU</td>
<td>12</td>
</tr>
<tr>
<td>SV{r}_{m}</td>
<td>Failure to control SV</td>
<td>8</td>
</tr>
<tr>
<td>VU{r}.A_FNSS</td>
<td>Non-self-signaling failure VU</td>
<td>4</td>
</tr>
<tr>
<td>VU{r}.A_FSS</td>
<td>Self-signaling failure VU</td>
<td>4</td>
</tr>
<tr>
<td>VU{r}<em>2O4</em>{m}_FNSS</td>
<td>Non-self-signaling failure of 2 out of 4 due to input failures from PU in VT</td>
<td>12</td>
</tr>
</tbody>
</table>

\[ \sum = 1052 \]
Conclusions

Use cases exists besides the „normal“ application of PSA software (FT, consequence, sequence, importance, MCS analyses):

- Provide data for documentation
- Modify existing ET/FT models or create new ones automatically
- Trace chances
- Review
- …

**Lessons learned**: One software product can not foresee all user requirements

⇒ Need an **interface** (open file format or programming interface) to extend PSA software (i.e. RiskSpectrum®):

✔ Import/export PSA models and data (partly implemented by GRS database link/via OpenPSA file format)

✗ Trigger computations

✗ Access results of analyses

✗ Call (external) user defined functions from PSA software