Experiences From Specifying the TCAS II Requirements Using RSML

Mats P.E. Heimdahl
Nancy G. Leveson
Jon Damon Reese

Outline

- Background
  - The Irvine safety research group
  - TCAS II
- Requirements language design
- Requirements State Machine Language (RSML)
- Lessons learned
  - About specification languages
  - In general
Requirements Development

Get the requirements “right”?  

- Rigorous capture  
  • Attempt to capture the correct requirements  
- Inspection  
  • Study the requirements and try to determine if they capture the desired behavior  
- Static analysis  
  • Use tools to determine if the requirements satisfy desirable properties  
- Execution  
  • Simulate or test the requirements to see if they behave as expected  

Goal

Provide an overview of easily implemented improvements

Background

Motivation

- The Irvine Safety Group has developed procedures to identify and eliminate software related hazards  
- The methods were not tightly coupled and are difficult to use together  
- Large case study needed

The Irvine Safety Group

- Nancy Leveson: professor  
  • University of Washington  
- Mats Heimdahl: student  
  • University of Minnesota  
- Jon Reese: student  
  • Safeware Engineering Corp.  
- Holly Hildreth: student  
  • TRW

Our Goal:  

A unified methodology for the development and analysis of safety critical systems
Traffic alert and Collision Avoidance System (TCAS II)

Honeywell

Time-line

- RTCA SC-147 starts work on requirements spec. in English
- MOPS Developed
- FAA Requires TCAS II
- Nancy Leveson contacted by FAA
- UC Irvine group starts working on experimental formal document
- UC Irvine approach formally adopted
- Requirements specification ready for IV&V
- IV&V starts
Pseudo-Code in the MOPS

```
PROCESS Ground_level_estimation;
    IF (O.OGROUND EQ 'TRUE)
        THEN G.ZGROUND = G.ZOWN;
    ELSEIF (G.RADAROUT GT P.RADARLOST)
        THEN G.ZGROUND = -P.ZLARGE;
    ELSEIF (G.RADAROUT GT 0)
        THEN <do not update ZGROUND>
    ELSEIF (G.RADAROUT EQ -P.ZLARGE)
        THEN
            IF (O.ZRADAR LT P.KNOWGROL)
                THEN G.ZGROUND = G.ZOWN - O.ZRADAR;
            ELSE <ground level unchanged>
        OTHERWISE
            IF (O.ZRADAR GT P.KNOWGROH)
                THEN G.ZGROUND = -P.ZLARGE;
            ELSE G.ZGROUND = G.ZOWN - O.ZRADAR;
    END Ground_level_estimation;
```

The Problem Revisited

Robert Buley
**Time-line**

- **1981**: MOPS Developed
- **1989**: FAA Requires TCAS II
- **1990**: Nancy Leveson contacted by FAA
- **1991**: UC Irvine group starts working on experimental formal document
- **1992**: UC Irvine approach formally adopted
- **1993**: Requirements specification ready for IV&V
- **1994**: IV&V starts

**Language Design Criteria**

- Blackbox
- Minimal
- Semantically simple
- Coherent, consistent, and concise
- Unambiguous with a formal foundation
- Simulatable
- Analyzable
- Readable, reviewable, and usable by application experts
- Flexible notation
- Readability given priority over writability
- User needs given priority over personal preferences
Candidate Approaches

- Z
- VDM
- Gypsy
- PVS
- Aslan
- Eves
- HOL/ML

All were found to be too complex for our target audience.

OMT - Booch
  - UML (Rational)
- Hatley/Pirbhai
- Statemate
- Ward/Mellor
- SCR
  - Software Cost Reduction

Informal approaches.
Multiple descriptions.
Too rich.
Too much design.

RSML

Requirements State Machine Language

- Features in common with Statecharts
  - States and transitions
    - Events, guarding conditions, and actions
  - Hierarchies
  - AND decomposition
  - State arrays
  - Conditional connectives

- Additions to Statecharts
  - Directed communication
  - Component state machines
  - Interface definitions
  - Distinction between external and internal events
  - Transition definition readability
  - Transition busses
  - Cross referencing
**TCAS Highest Level**

- **Power-On**
- **Power-Off**
- **Fully-Operational**
  - **Own-Aircraft**
  - **Other-Aircraft [1..30]**
  - **Mode-S-Ground-Station [1..15]**
- **Standby**

**Other-Aircraft**

**Input:**
- Other-Mode-S-Address: Integer
- Other-Alt: Integer
- Other-Bearing: Integer
- Other-Range: Integer

**Output:**
- Display-Arrow-Out: {Up, Down}
- Other-Relative-Alt-Out: Integer
- Other-Range-Out: Integer
- Other-Bearing-Out: Integer
- Advisory-Code: {Other, PA, TA, RA}
Tracked

- Air-Status:
  - On-Ground
  - Airborne

- Intruder-Status:
  - Threat
  - Pot.-Threat
  - Prox.-Traffic
  - Other-Traffic

- Advisory-Status:
  - Resolution-Adv.
    - Climb
    - Descend
  - Traffic-Adv.
    - Prox.-Adv.
    - No-Adv.

Transition Definitions?

- Surv.-Event[Alt < 150ft] / Air-Stat-Event
Transition Definition Example

**Transition:** Airborne → On-Ground

**Location:** Intruder-Aircraft Tracked Air-Status

**Trigger Event:** Air-State-Event

**Condition:** Alt < 150

**Output Action:** None

**Description:** Transition triggered by Air-Stat-Event with the guarding condition (Alt < 150), and no output action.

**Comments:** The transition from the previous slide formatted.

---

Transition Example

**Transition(s):** Potential-Threat → Other-Traffic

**Location:** Other-Aircraft > Intruder-Status

**Trigger Event:** Air-Status-Evaluated-Event

**Condition:**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt-Reporting-101 in state Lost</td>
<td>F</td>
</tr>
<tr>
<td>RA-Mode-Canceled-213</td>
<td>T</td>
</tr>
<tr>
<td>Alt-Reporting-101 in state No</td>
<td>T</td>
</tr>
<tr>
<td>Other-Bearing-Valid-130</td>
<td>T</td>
</tr>
<tr>
<td>Other-Range-Validity-117 = True</td>
<td>T</td>
</tr>
<tr>
<td>Potential-Threat-Range-Test-214 = True</td>
<td>T</td>
</tr>
<tr>
<td>Potential-Threat-Condition-213</td>
<td>T</td>
</tr>
<tr>
<td>Proximity-Traffic-Condition-215</td>
<td>T</td>
</tr>
<tr>
<td>Threat-Condition-224</td>
<td>T</td>
</tr>
<tr>
<td>Other-Air-Status-101 in state On-Ground</td>
<td>T</td>
</tr>
</tbody>
</table>

**Output Action:** Intruder-Status-Evaluated-Event
Language Lessons Learned

- Readability, usability, and simplicity are extremely important
- Model-based languages are much easier to understand than algebraic or axiomatic languages
- The graphical notation is powerful
- It is easy to be blind to the users needs
  - Difficult to overcome personal preferences
- Use suitable notation for different types of information
- Even a readable and usable notation cannot solve the problem with inherent complexity

General Lessons

- Formal modeling has tremendous advantages
  - Common unambiguous model of the system
- Understanding intent is difficult
  - The rationale behind the engineering decisions is lost
- It is easy to include large amounts of design
- Keep an open mind
  - Industry - Be willing to try new (safe) technologies
  - Academia - Listen to industry’s needs and adapt
- We should have started from scratch
- It is difficult to transfer technology
- Requirements specification is tedious and time consuming
## Tool Support

### The benefits of tool support and automated analysis are irrefutable

<table>
<thead>
<tr>
<th>Tool support is necessary</th>
<th>But….</th>
</tr>
</thead>
<tbody>
<tr>
<td>★ Even simple tools are extremely useful</td>
<td>★ Flexibility is dramatically reduced</td>
</tr>
<tr>
<td>★ Editor, parser, type checker, and simple data flow analysis</td>
<td>★ Could impact the usability of the approach</td>
</tr>
<tr>
<td>★ Tools are cheap, people are expensive</td>
<td>★ Detect a lot of problems</td>
</tr>
<tr>
<td>★ Detect a lot of problems</td>
<td>★ It takes time to get all the problems out of a specification</td>
</tr>
<tr>
<td>★ Focus on solving problems, not finding problems</td>
<td>★ It is “too much work”</td>
</tr>
</tbody>
</table>

---

## Current Status

- RSML specification maintained by independent organization
  - Version 7.0 just released
- Method with tool support has been developed
  - SpecTRM
- Language undergone some changes based on TCAS experience
  - SpecTRM-RL
- Commercialized by Safeware Engineering Corporation
References