Supporting Tests of Autonomy: Autonomy Requirements Tester (ART)

Eugene V. McMahon
S&K Global Solutions (NASA)
Houston, TX

Copyright © 2017 by Eugene V. McMahon. Permission granted to INCOSE to publish and use
CONTRIBUTORS

• Carroll Thronesbery, PhD “Principal Investigator”
• Ayman Qaddumi, MS “Software Developer”
• Michael Merta, MA “Human Computer Interaction Designer”
• Mike Monahan, “Business Developer”
• Eugene McMahon, “Testing Designer”
Topics

- SBIR and the ART project
- High-level Diagram
- Test Runner
- Requirements in XML Format
- Generating Test Plan from Requirements
- ART Examples
- Advantages
- Innovations
- Next Steps
NASA Small Business Innovation Research (SBIR) project titled “Autonomy Requirements Tester” (ART) with the following goals:

• Design eXtensible Markup Language (XML) schema to define data models to support app-level testing
• Describe potential approaches for semi-automatic test generation
• Design displays that support the management of requirements, test designs, and test results
• Develop a Concept of Operations (ConOps) for the use of ART that employs the following scenarios:
  • Capture autonomy requirements
  • Generate test specifications
  • Execute the test specs
  • Report results
Recent Survey

A recent survey of software developers revealed some of their top issues:

1. Requirements that are confusing or incomplete
2. Modifying software that is not documented, difficult to understand, or difficult to see the relationship between requirements and software
3. Unrealistic expectations or deadlines

- This SBIR project directly addresses the first two issues
High-level diagram of Autonomy Requirements Tester (ART)

**Requirements Handler**
- Enter traditional requirement information
- Enter expected behavior
- Generate requirements XML

**Test Handler**
- View requirements
- Edit template for the test
- Edit input variables (cutoff values)
- Generate the test plan
- View test results

**Test Runner**
- Execute test
- Record test results

**Flow Diagram**

Requirements (xml) → Test Handler → Test Runner → Test Results (csv)

Test Plan (csv) → Test Runner
Publish / Subscribe Software

- Examples: cFS and ROS
- Component based design
- Publish-subscribe message communication to make component apps independent
• **Publish**: Send specific test data to Application Under Test
• **Subscribe**: Receive test results from Application Under Test
Test Runner: Reads Test Spec, Produces Test Results

Test Spec
(Script Data)
(Adaptation of ATML)

Test1
Input Message:
  x=5
  y=2
  z=7
Expected results
message:
  xx=1
  yy=3
  zz=1
Test2
...

Test Results

Test1 <pass>
Test2
Expected:
  xx=1
  yy=3
  zz=1
Observed:
  xx=0
  yy=3
  zz=1
...

Test Runner

Computer System

Application Component 1

Application Component 2
(Under Test)

Application Component 3

CFE bus (Publish / Subscribe)
Test Runner: Reads Test Spec, Produces Test Results

Test Spec
(Adaptation of ATML)

Test Spec
(Script Data)

Test1
Input Message:
  x=5
  y=2
  z=7
Expected results
message:
  xx=1
  yy=3
  zz=1
Test2...

Test Results

Test1 <pass>
Test2
Expected
  xx=1
  yy=3
  zz=1
Observed
  xx=0
  yy=3
  zz=1
...

Computer System

Test Runner

Application Component 1

Application Component 2 (Under Test)

Application Component 3

CFE bus (Publish / Subscribe)

Test Runner: Reads Test Spec, Produces Test Results

Could potentially act as H/W simulator

Test Runner: Reads Test Spec, Produces Test Results

Test Runner: Reads Test Spec, Produces Test Results
High-level diagram of Autonomy Requirements Tester

---

**Requirements Handler**
- Enter traditional requirement information
- Enter expected behavior
- Generate requirements XML

**Test Handler**
- View requirements
- Edit template for the test
- Edit input variables (cutoff values)
- Generate the test plan
- View test results

**Test Runner**
- Execute test
- Record test results

---

**Requirements**

**Test Plan (csv)**

**Test Results (csv)**
Requirements in XML format

XML Requirements

- No action should be taken if Battery temp okay
- All data needs to have consistent readings over a period of time such as 5 of 6 readings must be the same
- There is a limit to the number of times a command can be sent to the hardware

Requirements Behaviors

**Autonomy shall perform the following tiered response if the battery temperature is above a pre-defined limit:**

A) Soft-reset the PSC
B) Power-cycle the PSC
C) Switch the PSC (via a CIM side switch)
Observation from Solar Probe Plus autonomy requirements:
- Similarities from one requirement to the next
  - Often a tiered response, when first tier doesn’t correct the issue, go to the next tier
  - Rule based behavior: If \{condition\} then \{response\}
- Similarities enable the formation of a template that could be reused for generating tests
- Some additional parameters are needed in addition to the template. A display was developed to collect this data.
AUT-3  Monitor Battery Temperature  

Autonomy shall perform the following tiered response if the battery temperature is above a pre-defined limit:

A) Soft-reset the PSC  
B) Power-cycle the PSC  
C) Switch the PSC (via a CIM side switch)

1. Set nominal spacecraft system state  
2. Verify autonomy takes no action  
3. Inject fault  
4. Verify faulted state (optional, especially level 0)  
5. Verify autonomy response  
6. Repeat steps 3-5 through all possible iterations  
7. For tiered rule:
   1. Inject fault corrected by 1st action  
   2. Inject fault corrected by 2nd action  
   3. Inject fault corrected by 3rd action  
   4. Inject unrecovered fault
Data Entry To Enable Test Generation From Template

Enter Design Values To Construct Initial Test

<table>
<thead>
<tr>
<th>AUT-3</th>
<th>Monitor Battery Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Autonomy shall perform the following tiered response if the battery temperature is above a pre-defined limit:</td>
</tr>
<tr>
<td></td>
<td>A) Soft-reset the PSC</td>
</tr>
<tr>
<td></td>
<td>B) Power-cycle the PSC</td>
</tr>
<tr>
<td></td>
<td>C) Switch the PSC (via a CIM side switch)</td>
</tr>
</tbody>
</table>

Enter values from design file...

- **M**: 5
- **N**: 6
- **Persistence (m of n)**: 9
- **Max fire count**: 2
- **Priority**: Enabled
- **Initial rule state (enabled/disabled)**

- **battery_temp**: Battery temperature variable name (default from reqts xml)
- **160**: pre-defined limit
- **PSC_reset_cmd**: Soft-reset the PSC command name (default from reqts xml)
- **PSC_pwr_cycle_cmd**: Power-cycle the PSC command name (default from reqts xml)
- **change_CIM_side_cmd**: Switch CIM side command name (default from reqts xml)

Data Model Based on IEEE Standards adopted by the Institute of Electrical and Electronics Engineers (IEEE) as a standard (IEEE Std 1671-2010)
### Autonomous Rule System Requirements Overview

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Test Plans</th>
<th>Test Results</th>
<th>Requirement AUT-3 Monitor Battery Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement ID</td>
<td>Title</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUT-1</td>
<td>Detect Loss of Telemetry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUT-2</td>
<td>Detect Invalid Telemetry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUT-3</td>
<td>Monitor Battery Temp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUT-4</td>
<td>Monitor Battery State of Charge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUT-5</td>
<td>Detect Critically Low State of Charge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUT-6</td>
<td>Battery Heater Power On</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUT-7</td>
<td>Battery Heater Power Off</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### ID
- AUT-3

#### Name
- Monitor Battery Temp

#### Categories
- Device Health, Thermal Monitoring

#### Parents
- SC-1

#### Rationale
- Protects against a potential PSC control fault that could cause excessive battery temperature.

#### Details

**Text**
- Autonomy shall perform the following tiered response if the battery temperature is above a pre-defined limit: A) Soft-Reset the PSC, B) Power-cycle the PSC, C) Switch the PSC (via a CIM slide switch).

**Condition**
- `battery_temp > 160`

**Subject**
- Autonomous system

**Tiered Action(s)**
- Soft-reset the PSC, Power-cycle the PSC, Switch the PSC (via a CIM slide switch)

**Object**
- PSC

**Limit Value**
- 160

**Constraint**
- N/A
Examples: Test Plan (Flowchart)

Step 1
- Initialize
- Set Nominal
- Spacecraft System State
- Verify autonomy takes no action

Step 2
- Tier 1 Response
- Inject Fault
- Verify Tier 1 response
- Determine Success

Step 3
- Tier 2 Response
- Inject Fault
- Verify Tier 2 response
- Determine Success

Step 4
- Tier 3 Response
- Inject Fault
- Verify Tier 3 response
- Determine Success

Step 5
- Unrecovered Function
- Cycle Tiered Response
- Verify rule disabled

Back to Step Details
### Test Runner - Active Requirement: AUT - 3

#### Test Plans

<table>
<thead>
<tr>
<th>Step ID</th>
<th>Step Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>Initialize</td>
</tr>
<tr>
<td>Step 2</td>
<td>Tier 1 Response</td>
</tr>
<tr>
<td>Step 3</td>
<td>Tier 2 Response</td>
</tr>
<tr>
<td>Step 4</td>
<td>Tier 3 Response</td>
</tr>
<tr>
<td>Step 5</td>
<td>Unrecovered Function</td>
</tr>
</tbody>
</table>

#### Test Results

**Step 2: Tier 1 Response**

1. Set `battery_temp` = 161, every 1 second

**Verify tier 1 autonomy response**

1. Wait 7 seconds
2. Verify `PSC_reset cmd`
3. Verify `rule_fire_count(AUT3) = 1`

**Success**

1. Set `battery temp` = 159, every 1 second
2. Wait 15 seconds
3. Expect `rule_fire_count(AUT3) = 1`
4. Verify commands
### Initial autonomous rule test Results

**System:** Autonomous Rule System  
**Date:** Aug. 11, 2016 – 9:34 AM  
**Personnel**  
- Operator: Smith Johnson, *sjsop@company.com*, 239-234-4321  
- Quality Assurance: Jane Smith, *qa@company.com*, 239-234-4321

**Test Plan:** Initial autonomous rule test  
**Description:** Performs macro command tests to verify that appropriate commands are being sent when the AUT rule is triggered.  
**Outcome:** Failed

- **Step 1:** Initialize  
- **Step 2:** Tier 1 Response
- **Step 3:** Tier 2 Response
- **Step 4:** Tier 3 Response
  - 1. Inject Fault  
  - 2. Verify tier 3 autonomy response
    - 1. Wait 7 seconds  
    - 2. Verify change, CIM, side
  - 3. Verify tier 3 response
Advantages of this Method

- Start test driven development early
- Express autonomy requirements in terms of expected behavior
- Support pre-integration testing
- Make integration testing time more productive – no logic errors in software
- During integration, if software changes are required:
  - Make the changes
  - Re-run the pre-integration test to ensure no errors were inadvertently entered
  - Resume integration testing
Innovations

• Represent requirements and link with intended behaviors for testing the requirements (Survey issue #1)

• Formal data models for requirements, behavioral expectations, test specifications, and test results (Survey issue #1)

• Use of template to drive the elaboration of test specifications

• Integration of the testing mechanism with the operational environment
  • Enabled by modular architecture w/ pub-sub communications scheme
  • No change to the unit under test between testing and operations
  • Paves the way for runtime checkout routines for selected apps (e.g., sensors for deep-space science operations)

• Reporting of test results – similar appearance to specifications, still linked to requirements (Survey issue #2)
Next Steps

- Complete development of the ART tool and associated user interfaces
- Identify how to support higher levels of integration testing
- Identify how to support additional types of autonomy requirements
- Update overall documentation and documentation for all requirement schemas (formal data models)