Modeling Complex Cross-Systems Software Interfaces using SysML

Sanda Mandutianu, Ron Morillo, Kim Simpson, Otfried Liepack, Kevin Bonanne
Jet Propulsion Laboratory/Caltech
Infotech@AIAA, August 19-22 2013, Boston, MA
Goals and Objectives

- NESC sponsored task to
  - Identify potential ICD gaps
  - Characterize cross-systems behaviors of the critical MPCV-SLS
  - Demonstrate approach for cross-system integration
Systems Engineering Problems

- Describing and Documenting System Interfaces
- Software Architecture Specifications
- System Modeling
Modeling Approach

• Leverage:
  - EFT-1 and EM-1 end-to-end architecture models based on the IEEE 1471 and DoDAF
  - JPL ontologies and interface pattern recommendation

• Support MS-SLS software ICD development

• Modeling scope: architecture descriptions of the software interfaces at different levels of abstractions and their behaviors
Interface Modeling Pattern

- The interface pattern allows to define an abstract interface without specifying any implementation.
- The interface is an abstraction of multiple interactions.
- The intent of interacting is abstracted as an “interface”
  - Specifies characteristics of what is shared
  - May include behavioral features
Interface Modeling Pattern Definition and Usage
Modeling Software Interfaces

SW interface – particular case of interface
Layered approach – loosely coupled
End-to-end Data Exchange Interfaces

Providing mission context
High-level logical connections
Modeling Functions and Behaviors

• Understand behavior by decomposing top-level functions derived from functional requirements specification
• Key: functional architecture allocated to performing structural elements
• Two representations of functional architecture
  ▪ Functional hierarchy
  ▪ Behavioral model
Functional Hierarchy

- Conveys the transformation of interface specifications to interface functions.
- Provides traceability of requirements to products.
Supports the specification of the behavior of functional components
- Sequence of functions (functional flow)
- Data flow
- Control logic for execution (control flow)
- Resources (mapping to structure)
Mapping behaviors to structure – interface pattern

Explicit mapping of the flows

SysML Internal Block Diagram:
- Ground SW Interface: Send command to Spacecraft, Receive telemetry from Spacecraft
- Spacecraft SW Interface: Receive command from Ground, Send telemetry to Ground

Allocate behavior to structure:
- Interfaces (ports) to function (input/output pins)
- Item flows to object flows (transfers in/out to performs relations)
One of the main purposes of modeling: accurate, reliable, automated analysis

Analyses – explicitly represented
- Identify analyses
- Specify relationships with other model elements

Conceptually – a justification

Examples: requirements analysis, trade studies, functional analysis, etc.
# Engineering Analyses

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Documentation</strong></th>
<th><strong>Owned Diagram</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>01. Requirements Allocation Analysis</td>
<td><strong>Purpose:</strong> Find the requirements that are not allocated to a component. &lt;br&gt;<strong>Rules:</strong> &lt;br&gt;1. All requirements have to be allocated to components &lt;br&gt;2. All components have to be specified by one or more requirements.</td>
<td><img src="image" alt="Not Allocated Requirements Analysis" /></td>
</tr>
<tr>
<td>02. Requirement Refinement Analysis</td>
<td><strong>Purpose:</strong> Check for valid requirements refinement graph &lt;br&gt;<strong>Rules:</strong> &lt;br&gt;Except for top-level requirements, every requirement must refine at least one other requirement (rule 1) &lt;br&gt;The requirements refinement graph must not contain cycles (rule 2)</td>
<td><img src="image" alt="SLS-MS Requirements IRD-ICD Derive Table" /></td>
</tr>
</tbody>
</table>

**Method:**<br>**Currently:** Dependency table - requirements allocated to components (can be reversed) - model/tool built-in query<br>**Construct** Built-in dependency tables<br>**More built-in sophisticated queries**
Requirements Refinement Analysis

**Purpose:** Check for valid requirements refinement graph

**Rules:**
- Except for top-level requirements, every requirement must refine at least one other requirement (rule 1)
- The requirements refinement graph must not contain cycles (rule 2)

**Method:**
- Construct built-in dependency tables
- More built-in sophisticated queries

**Results:**
- Display requirements flow-down
- Show gaps

### SLS-MS ICD Requirement vs. MPCV-SLS Traced Requirement

<table>
<thead>
<tr>
<th>SLS-MS ICD Requirement</th>
<th>MPCV-SLS Traced Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/C-Ground 3.41.841.841 MS commanding S/C via InterSys</td>
<td>- InterSys-S/C.2722 InterSys initiating S/C engine shutdown</td>
</tr>
<tr>
<td></td>
<td>- InterSys-S/C.3004 InterSys sending att. rates cmds to S/C while manual steering off-nominal</td>
</tr>
<tr>
<td></td>
<td>- InterSys-S/C.3006 InterSys sending ascent target cmd to S/C</td>
</tr>
<tr>
<td>S/C-Ground 3.41.841.842 S/C receiving commands from Ground via InterSys</td>
<td>- InterSys-S/C.3005 SLS receiving att. rates cmds from InterSys while manual steering</td>
</tr>
<tr>
<td></td>
<td>- InterSys-S/C.3007 SLS receiving ascent target cmd from InterSys</td>
</tr>
<tr>
<td></td>
<td>- InterSys-S/C.3078 SLS receiving notification of SA separation from InterSys</td>
</tr>
<tr>
<td></td>
<td>- InterSys-S/C.3080 SLS receiving notification LAS separation from InterSys</td>
</tr>
</tbody>
</table>
Determine configurations of control paths: connections, characteristics of data exchanges, link options, etc.
Functional Allocation Analysis

Purpose: Find the functions that have not been allocated to a mission component.

Rules:
1. Each function is specified by one or more requirements
2. Each function is performed by exactly one component
3. Each component is bound by at least one requirement

Methods:
Query all components; can be done with a generic table in the first approximation. More sophisticated queries if needed.

Reports/Findings:
- All components that perform a function, by functions accomplished.
- All functions that are not accomplished by any component
- All components that are not performing any functions

<table>
<thead>
<tr>
<th>Interface</th>
<th>Behavior</th>
<th>Component</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/C to Ground CMD</td>
<td>Receive CMD</td>
<td>Spacecraft</td>
<td>Provide Telemetry</td>
</tr>
<tr>
<td>Ground to S/C TLM</td>
<td>Package TLM</td>
<td>Spacecraft</td>
<td>Provide Telemetry</td>
</tr>
<tr>
<td>Ground to S/C TLM</td>
<td>Send TLM</td>
<td>Spacecraft</td>
<td>Provide Telemetry</td>
</tr>
</tbody>
</table>
Automated Generation of SE Artifacts

The output of SE is ultimately specification, not implementation: system architecture, IRD, ICD, analysis results, etc.
Conclusions and Future Work

• **Observations**
  - Communication with stakeholders is critical
  - Product oriented modeling
  - MBSE doesn’t replace established SE processes
    - SE rules implemented in the model
  - Graphics are very powerful

• **Future work** – extend the current capabilities (functional and behavioral), explore more areas
BACKUP
Ontology Hierarchy

Foundation Ontologies
- Base Ontology
- Analysis Ontology
- Project Ontology

Discipline Ontologies
- Mechanical Ontology
- Thermal Ontology
- Physics Ontology
- Electrical Ontology
- Propulsion Ontology
- ACS Ontology

Application Ontologies
- Star Tracker
- Thruster
- Antenna
- Reaction Wheel
- Sun Sensor

Fundamental terms used in all projects, disciplines, and applications

Discipline specific terms specified and owned by cognizant organizations

Focus: integration and interoperation

Kinds of items modeled in a project

Focus: reuse
Foundation Ontology Fragment