Automated System Analysis using Executable SysML Modeling Patterns

Maged Elaasar*
Modelware Solutions

http://magedelaasar.com
melaasar@gmail.com

CEA LIST, Paris, France
March 2016

*In collaboration with Robert Karban (JPL) and Nerijus Jankevičius (NoMagic)
Maged Elaasar

• Ph.D. in Computer Systems Engineering (Canada)
• Senior Software Architect at NASA’s JPL (USA)
  – Leads R&D work in MBSE
• Formerly Senior Software Architect at IBM
  – Led R&D work in MDE (e.g., RSA, DM)
• Founder of Modelware Solutions
  – Consults on MDE technologies
  – Affiliated with several international research labs
• Leading modeling standards at OMG
Systems Analysis

• Decomposing the system into its components for the purpose of studying how well those components work together and interact to meet some objectives or satisfy some constraints
Requirements Verification

- A kind of systems analysis that assesses whether a system design meets the objectives and satisfies the constraints that are implied by the system requirements.
Model Based Systems Engineering

• MBSE is the formalized application of modeling techniques to support system requirements, design, analysis, verification, validation and documentation activities
• MBSE expresses a system using a Systems Modeling Language (SysML), a profile of UML
• MBSE is often applied with a method like Object Oriented System Engineering Method (OOSEM)
Project Question

• How to perform requirements verification on systems models using vanilla SysML?
Proposed Approach

• An extension of OOSEM named Executable Systems Engineering Method (ESEM)
• ESEM produces executable SysML models that verify requirements
• ESEM provides a set of analysis patterns that are specified with various SysML structural, behavioral and parametric diagrams
Executable Models

• Most SysML models today are created for documentation purposes
  – The focus is on syntax and notation

• Some SysML models are created to gain system understanding, explore and validate desirable or undesirable behaviors of a system
  – The focus is on semantics
Model Execution

- Executable SysML models are defined with a subset of the language with well defined execution semantics
  - The subset is called Foundational UML (fUML)
  - SysML inherits the fUML subset from UML

- SysML models are executed with the help of an execution, or simulation, engine
  - Ex.: MagicDraw’s CST
  - Ex.: Papyrus’s MOKA
Cameo Simulation Toolkit (CST)

- A plugin to MagicDraw SysML modeling tool
- A simulation platform based mostly on fUML but plugs in additional execution engines
  - State Chart XML (SCXML)
  - Scripting for the Java Platform (JSR 223)
  - Precise Semantics of Composite Structures (PSCS)
CST Parametric Solver

• Executes SysML parametric diagrams based on the PSCS semantics
  – At object instantiation time, it analyzes the causality of attributes, i.e., determines which attributes are given (pre-bound) and which are target (unbound)
  – Allows the given values to immediately propagate and update any target values after constraints evaluate (with a math solver)
  – Whenever the value of an attribute that is bound to constraint parameter changes, the constraint is re-evaluated and updates all related variables, creating a cascade effect
Object Oriented Engineering Method

• Defines the architecture in terms of:
  – Domain: the context of the solution
    • Enterprise: the ecosystem of the solution
      – System of Interest: the solution being specified
        » Black Box: externally visible specification
        » Conceptual: white box functional specification
        » Physical: white box realization specification
Running Example

• Thirty Meter Telescope (TMT) [http://www.tmt.org/](http://www.tmt.org/)
  – Alignment and Phasing System (APS)
    • Sensor responsible for pre-adaptive optics wavefront quality
• Developed by TMT International Observatory (TIO)
  – JPL participates in several subsystems of TMT
  – JPL delivers the APS based on requirements from TIO
  – APS team uses MBSE to analyze requirements, produce design, and performs analysis
Running Example Objectives

• Use MBSE to define executable SysML model that captures requirements, operational scenarios (use cases), system decomposition, relationships and between subsystems, etc.
• Use the model to analyze the system for power consumption
  – Also mass and duration analysis (but out of scope)
• Produce documents like
  – Design Description Document
  – Requirement Flow Down Document
  – Operational Scenario Document
• Goals: use standard languages and techniques, avoid custom software development
Executable System Engineering Method (ESEM)

- Step 1: Formalize Requirements
- Step 2: Specify Design
- Step 3: Characterize Components
- Step 4: Specify Analysis Context
- Step 5: Specify Operational Scenarios
- Step 6: Specify Analysis Configurations
- Step 7: Run Analysis
- Step 8: Evaluate Requirement Satisfaction
Step 1: Formalize Requirements

• Requirement Pattern
  – Customer Side
    • Define the textual requirement with a Requirement
    • Optionally define a design black box specification with a Block with relevant value properties
    • Optionally refine the Requirement with a Constraint Block on the black box design Block
  – Supplier Side
    • Define a design black box specification with a Block (that refines the customer’s black box Block if any and provides tighter property values)
    • Refine the textual Requirement by a Constraint Block (if not already defined by the customer)
Step 1: Formalize Requirements

```
<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>APS Blackbox Customer</td>
</tr>
<tr>
<td>values</td>
</tr>
<tr>
<td>pwrPeakLimit: W = 8500(unit = watt)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>APS Blackbox Supplier</td>
</tr>
<tr>
<td>values</td>
</tr>
<tr>
<td>pwrPeakLimit: W = 8100(unit = watt)</td>
</tr>
</tbody>
</table>

| requirement                   |
| Peak Dome Load                |
| Id = "REQ-2-APS-23"           |
| Text = "APS shall have a peak load inside the Dome of less than 8.5 kW" |

| constraint                    |
| Peak Power Load Constraint Supplier |
| constraints                   |
| {p(requiredPeakLimitLoad)}     |
| parameters                    |
| p : W                         |
| requiredPeakLimitLoad W       |
```

Both, the value property and the constraint block refine the textual requirement.
Step 2: Specify Design

• Follow OOSEM to define two white box Blocks that specialize the black box Block
  – Conceptual Block (out of scope for brevity)
  – Physical Block

• Decompose the white box designs with Blocks representing the subsystems
Step 2: Specify Design

System Decomposition Hierarchy
Step 3: Characterize Components

• Add relevant patterns to the design Block to make it executable

• Example: Roll-up Pattern
  – Constrained value represents an aggregate value that is propagating up a hierarchy of subcomponents
  – Static roll-up (e.g., mass roll-up)
  – Dynamic roll-up (e.g., power roll-up)
Step 3: Characterize Components

Power Rollup Pattern
Step 3: Characterize Components

Power Roll-up Pattern Variation
Step 4: Specify Analysis Context

• Analysis Context Pattern
  – Abstract analysis context Block composes both the design black box Block and white box Block
  – Analysis properties defined on the analysis context Block (e.g., peak power, power margin)
  – Analysis parametric model on the analysis context that computes and binds analysis values
Step 4: Specify Analysis Context
Step 4: Specify Analysis Context

Analysis Context Parametric Model
Step 5: Specify Operational Scenarios

• Operational Scenario Pattern
  • Concrete analysis context Block that
    – Represents one operational scenario (e.g., power configuration)
    – Specializes the abstract analysis context Block
    – Redefines context’s properties with scenario-specific values
    – Defines an owned behavior (sequence diagram) as scenario driver
      » Changes the states of the different components, by sending them signals, causing the rolling-up to occur automatically
      » Can specify duration constraints to time the injection of signals thus controlling time spent in a certain state
      » Can use state constraints (on components) to verify during execution if a component is actually in expected state
Step 5: Specify Operational Scenarios
Step 5: Specify Operational Scenarios
Step 6: Specify Scenario Configurations

• Scenario Condition Pattern
  – A decomposition tree of instance specifications representing the state of the scenario
    • Can be presented in tabular form
      – Rows represent the instance specifications (e.g., component)
      – Columns represent values (e.g., operating power) from the instance specifications

• Issues
  – Hard to keep instance specifications in sync with BDDs
    • Mitigation: tool automation
  – Instance specifications cannot be displayed in IBDs
    • Mitigation: use full specialization tree of singleton Blocks for each scenario
Step 6: Specify Analysis Configurations

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Operating Power : W</th>
<th>Standby Power : W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>aps.dome installation_1.bench_1</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>aps</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>aps.coordinator_1</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>aps.dome installation_1</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>aps.dome installation_1.bench_1.apt + te ccd_1</td>
<td>300.0</td>
<td>100.0</td>
</tr>
<tr>
<td>6</td>
<td>aps.dome installation_1.bench_1.large motor_1[1]</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>7</td>
<td>aps.dome installation_1.bench_1.large motor_1[2]</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>8</td>
<td>aps.dome installation_1.bench_1.large motor_1[3]</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>9</td>
<td>aps.dome installation_1.bench_1.large motor_1[4]</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>10</td>
<td>aps.dome installation_1.bench_1.large motor_1[5]</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>11</td>
<td>aps.dome installation_1.bench_1.large motor_1[6]</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>12</td>
<td>aps.dome installation_1.bench_1.large motor_1[7]</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>13</td>
<td>aps.dome installation_1.bench_1.large motor_1[8]</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>14</td>
<td>aps.dome installation_1.bench_1.large motor_1[9]</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>15</td>
<td>aps.dome installation_1.bench_1.large motor_1[10]</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>16</td>
<td>aps.dome installation_1.bench_1.boxfs + te ccd_1</td>
<td>150.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Scenario Condition Pattern
Step 6: Specify Scenario Configurations

• Scenario Configuration Pattern
  – Scenario configuration is an Instance Specification of the scenario’s concrete analysis Block
    • References an initial scenario condition with an <<analyzes>> relationship
    • References a final scenario condition with an <<explains>> relationship
Step 6: Specify Analysis Configurations

Scenario Configuration Pattern
Step 7: Run Analysis

- Run the configured analysis with a simulation engine on the initial conditions to get the final conditions:
  - Produce the following views on final conditions
    - Table showing final analysis values (e.g., peak power) and the constraint’s pass/fail status for each scenario
    - Timelines: state changes for components over time
    - Value profiles: total rolled up values over time
Step 7: Run Analysis

Analysis Result Views
Step 8: Evaluate Requirement Satisfaction

- Requirement Satisfaction Pattern
  - Requirement satisfaction Block aggregates the black box Block[1] (customer) and white box Block[*] (one for each operational scenario) from supplier
  - A parametric model on the requirement satisfaction Block checking that all the supplier values conform to the customer value (have lower power level)
  - Instance Specification of requirement satisfaction Block that <<analyze>> all final scenario conditions and <<explain>> a <<satisfy>> relationship from the design black box to the textual requirement
Step 8: Evaluate Requirement Satisfaction

Requirement Satisfaction Pattern
Step 8: Evaluate Requirement Satisfaction

Requirement Satisfaction Parametrics
Summary

• It is possible to automate requirements verification in SysML models

• Introduced a new Executable System Engineering Method that consists of a set of pure SysML analysis patterns

• The method can be executed using an off-the-shelf simulation engine for SysML
Reference
