Lifecycle Modeling – Application to Architecture Development
Overview

- Why a New Language?
- Lifecycle Modeling Language Overview
- Use of LML for Architecture to Systems Design Specification
- Use of LML in Test and Evaluation
- Use of LML in Operations and Support
- Summary
WHY A NEW LANGUAGE?

We already have SysML ... what else do you need!
State of Current “Languages”

• In the past decade, the Unified Modeling Language (UML) and now the profile Systems Modeling Language (SySML) have dominated the discussion

• Why?
  – Perception that software is “the problem”
  – Hence need for an “object” approach

• SysML was designed to relate systems thinking to software development, thus improving communication between systems engineers (SE) and software developers
Why Objects Are Not the Answer

• Although SysML *may* improve the communication of design between SEs and the software developers it does not communicate well to anyone else
  – No other discipline in the lifecycle uses object oriented design and analysis extensively
  – Users in particular have little interest/acceptance of this technique
  – Software developers who have adopted Agile programming techniques want functional requirements (and resent SEs trying to write software)
So What Do We Do?

• Recognize that our primary job as SEs is to communicate between all stakeholders in the lifecycle
• Be prepared to translate between all the disciplines
• Reduce complexity in our language to facilitate communication
LIFECYCLE MODELING LANGUAGE (LML) OVERVIEW
The Lifecycle

Current Operations and Maintenance

Future Operations and Maintenance

Demolition and Disposal

Design & Analysis

Architecture Development

System Design

Hardware/Software Acquisition

Operational T&E and Transition

Integration and Test

Integration & Verification

Program Management
Lifecycle Modeling Language (LML)

- LML combines the logical constructs with an ontology to capture information
  - SysML – mainly constructs – limited ontology
  - DoDAF Metamodel 2.0 (DM2) ontology only
- LML simplifies both the “constructs” and ontology to make them more complete, yet easier to use
- Goal: A language that works across the full lifecycle
LML Ontology* Overview

- **Taxonomy**: 
  - 12 primary element classes
  - Many types of each element class
    - Action (types = Function, Activity, Task, etc.)
- **Relationships**: almost all classes related to each other and themselves with consistent words
  - Asset performs Action/Action performed by Asset
  - Hierarchies: decomposed by/decomposes
  - Peer-to-Peer: related to/relates

*Ontology = Taxonomy + relationships among terms and concepts
** Taxonomy = Collection of standardized, defined terms or concepts
LML Taxonomy Simplifies Classes

• Technical
  – Action
  – Artifact
  – Asset
  – Characteristic
  – Input/Output
  – Link
  – Statement

• Programmatic/Technical
  – Cost
  – Issue
  – Location
    • Physical, Orbital, Virtual
  – Risk
  – Time
    • Duration, Timeframe, Point-in-Time
LML Relationships Provide Linkage Needed Between the Classes

<table>
<thead>
<tr>
<th>ACTION</th>
<th>ARTIFACT</th>
<th>ASSET</th>
<th>CHARACTERISTIC</th>
<th>COST</th>
<th>INPUT/OUTPUT</th>
<th>ISSUE</th>
<th>LINK</th>
<th>LOCATION</th>
<th>RISK</th>
<th>STATEMENT</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>decomposed by related to</td>
<td>references</td>
<td>captures consumes preformed by produces</td>
<td>specified by</td>
<td>incurs</td>
<td>generates receives</td>
<td>causes resolves</td>
<td>-</td>
<td>located at</td>
<td>causes mitigates resolves</td>
<td>based on</td>
</tr>
<tr>
<td>ARTIFACT</td>
<td>referenced by</td>
<td>decomposed by related to</td>
<td>references</td>
<td>specified by referred by</td>
<td>incurs referred by</td>
<td>referenced by</td>
<td>causes referred by</td>
<td>defines protocol for referenced by</td>
<td>located at</td>
<td>causes mitigates resolves</td>
<td>based on</td>
</tr>
<tr>
<td>ASSET</td>
<td>captured by consumed by performs produced by</td>
<td>references</td>
<td>decomposed by related to</td>
<td>specified by</td>
<td>incurs</td>
<td>-</td>
<td>causes resolves responds to</td>
<td>connected by</td>
<td>located at</td>
<td>causes mitigates resolves</td>
<td>based on</td>
</tr>
<tr>
<td>CHARACTERISTIC</td>
<td>specifies references specifies specifies</td>
<td>decomposed by related to</td>
<td>incurs specifications</td>
<td>specifies</td>
<td>causes resolves</td>
<td>specifics</td>
<td>located at</td>
<td>causes mitigates resolves</td>
<td>based on</td>
<td>specified on</td>
<td>occurs</td>
</tr>
<tr>
<td>COST</td>
<td>incurred by</td>
<td>incurred by references</td>
<td>incurred by specified by</td>
<td>decomposed by related to</td>
<td>incurred by</td>
<td>incurs incurred by resolves</td>
<td>incurred by</td>
<td>located at</td>
<td>causes incurred by resolves</td>
<td>causes on incurred by</td>
<td>occurs</td>
</tr>
<tr>
<td>INPUT/OUTPUT</td>
<td>generated by references</td>
<td>-</td>
<td>specified by</td>
<td>incurs decomposed by related to</td>
<td>causes resolves</td>
<td>transferred by</td>
<td>located at</td>
<td>causes mitigates resolves</td>
<td>based on</td>
<td>occurs</td>
<td>INPUT/OUTPUT</td>
</tr>
<tr>
<td>ISSUE</td>
<td>caused by resolved by</td>
<td>caused by references resolved by</td>
<td>caused by</td>
<td>resolved by</td>
<td>caused by</td>
<td>resolved by</td>
<td>caused by resolved by</td>
<td>causes decomposed by related to</td>
<td>located at</td>
<td>caused by mitigates causes</td>
<td>caused by resolved by</td>
</tr>
<tr>
<td>LINK</td>
<td>-</td>
<td>defined protocol by references</td>
<td>connects to</td>
<td>specified by</td>
<td>incurs</td>
<td>transfers</td>
<td>causes resolves</td>
<td>decomposed by related to</td>
<td>located at</td>
<td>causes mitigates resolves</td>
<td>based on</td>
</tr>
<tr>
<td>LOCATION</td>
<td>locates</td>
<td>locates</td>
<td>locates</td>
<td>locates</td>
<td>locates</td>
<td>locates</td>
<td>locates</td>
<td>locates</td>
<td>decomposed by related to</td>
<td>locates mitigates</td>
<td>based on</td>
</tr>
<tr>
<td>RISK</td>
<td>caused by mitigated by resolved by</td>
<td>caused by mitigated by references resolved by</td>
<td>caused by</td>
<td>mitigated by resolved by</td>
<td>caused by</td>
<td>resolved by</td>
<td>caused by resolved by</td>
<td>causes decomposed by related to</td>
<td>located at</td>
<td>mitigated by resolved by</td>
<td>causes on</td>
</tr>
<tr>
<td>STATEMENT</td>
<td>basis of</td>
<td>basis of references sourced by</td>
<td>basis of</td>
<td>basis of specified</td>
<td>basis of</td>
<td>basis of</td>
<td>causes resolves</td>
<td>-</td>
<td>basis of located at</td>
<td>causes located at mitigates resolves</td>
<td>decomposed by related to</td>
</tr>
<tr>
<td>TIME</td>
<td>taken by occurred by</td>
<td>occurred by</td>
<td>occurred by</td>
<td>occurred by</td>
<td>occurred by</td>
<td>data resolves delays occurred by</td>
<td>occurred by</td>
<td>occurred by</td>
<td>occurred by</td>
<td>occurred by</td>
<td>decomposed by related to</td>
</tr>
</tbody>
</table>

- decomposed by/decomposes
- orbited by/orbits
- related to/relates
LML Logic

No constructs – only special types of Actions
LML Action Diagram Captures Behavior

1.1 Serial Element Action
1.2 Request Service Action
1.3 Element in Parallel Action
1.4 Element in Decision Action
1.5 Exit Criteria Action
1.6 Element in Loop Action
1.7 Synchronize Information? Action

Start -> Data 1 -> Data 2 -> Data 3 -> End
Data 1 -> Data 2 -> Data 3
External Input

Loop 3 times

End
LML Physical Block Diagram

Sensor Systems Operator \(\xrightarrow{I.1.3\text{ Operator-Sensor Platform Interface}}\) Sensor Platform

- **connected with/ connects**
  - capacity (10 Mbits/sec)
  - Latency (100 millisec)

Sensor Platform \(\xrightarrow{used\ by/uses}\) Sensor System Memory

- **used by/uses**
  - maximum quantity (6 Gbytes)
  - minimum quantity (10 Kbytes)

- **Asset (Human)**: Sensor Systems Operator
- **Asset (System)**: Sensor Platform
- **Asset (Resource)**: Sensor System Memory
LML Combined Physical Behavior Diagram Enables Instances and Clones

Clones provide multiple instances of an Asset for use in simulation.
LML Summary

- LML contains the basic technical and programmatic classes needed for the lifecycle
- LML defines the Action Diagram to enable better definition of logic as functional requirements
- LML uses Physical Diagram to provide for abstraction, instances, and clones, thus simplifying physical models
- LML provides the “80% solution”
  - It can be extended to meet specific needs (e.g. adding Question and Answer classes for a survey tool that feeds information into the modeling)
USE OF LML FOR ARCHITECTURE TO SYSTEMS DESIGN SPECIFICATION
Architecture Development Process and Products

1. Capture and Analyze Related Artifacts
2. Identify Assumptions
3. Identify Existing/Planned Systems
4. Capture Constraints
5. Develop the Operational Context Diagram
6. Develop Operational Scenarios
7. Derive Functional Behavior
8. Derive Assets
9. Allocate Actions to Assets
10. Prepare Interface Diagrams
11. Define Resources, Error Detection & Recovery
12. Perform Dynamic Analysis
13. Develop Operational Demonstration Master Plan
14. Prepare Interim Diagrams
15. Conduct Trade-off Analyses
16. Generate Operational and System Architecture Graphics, Briefings and Reports

This implementation of the middle-out approach has been proven on a variety of architecture projects.
Key Architecture Products

• DoDAF Diagrams
• Concept of Operations (CONOPS)
• Functional Specifications of Hardware and Software
• Early Design Validation Through Modeling and Simulation
• Test and Evaluation Plans (for T&E)
• Processes and Procedures (for Operations and Support, as well as inputs to training plans)
USE OF LML IN TEST AND EVALUATION
Coming Up the Vee
Measure of Performance (MOP) View

Characteristics

Measure of Effectiveness (MOE)
Type = MOE
MOE 1

Measure of Performance (MOP)
Type = MOP
MOP 1.1

MOP Test Result
Type = MOP Occurrence
MOP 1.1 [System (SW1/HW1)]

MOP Test Result
Type = MOP Occurrence
MOP 1.1 [System (SW2/HW2)]

MOP Test Result
Type = MOP Occurrence
MOP 1.1 [System (SW3/HW3)]

Actions

System Function
Type = System Function
Function

1:1

Tests /
tested by

1:M

decomposed by /
decomposes

1:M

instantiates /
instantiated by

1:M

specifies /
specified by

1:1

MOP Test Result
specifies /
specified by

1:1

MOP Test Result
specifies /
specified by

1:1

MOP Test Result
specifies /
specified by

1:1

Assets

System Instantiation
Type = System Instantiation
System (SW1/HW1)

1:1

instantiates /
instantiated by

1:1

System Instantiation
Type = System Instantiation
System (SW2/HW2)

1:1

instantiates /
instantiated by

1:1

System Instantiation
Type = System Instantiation
System (SW3/HW3)

1:1

instantiates /
instantiated by

1:1

System
Type = System
System

1:M

allocated to /
performs

1:1

tests /
tested by

1:M

specifies /
specified by

1:1

MOP Test Result
specifies /
specified by

1:1

MOP Test Result
specifies /
specified by

1:1

MOP Test Result
specifies /
specified by

1:1

System
Type = System
System

1:M

allocated to /
performs

1:1

tests /
tested by

1:M
USE OF LML IN OPERATIONS AND SUPPORT
LML Support Operations and Support Analyses

• Process Modeling
• Simulation of Operations
• Training Processes and Procedures
• Operations Manuals
• Logistics Analysis
SUMMARY
LML Bottom-Line

• LML provides a simple, complete language for all stakeholders, not just software developers
  – SysML/UML focus on software developers only
• Use of Actions instead of constructs to capture command and control functions explicitly
• Translation from LML to other languages now feasible
• Support for entire lifecycle