Lifecycle Modeling – An Approach to Simplified, Rapid Development, Operations, and Support

Steven H. Dam, Ph.D., ESEP
President, SPEC Innovations
571-485-7805
Steven.dam@specinnovations.com

October 24, 2011
Overview

- Why a New Language?
- Lifecycle Modeling Language Overview
- Suggested LML Processes
- LML Tool Needs
- Use of LML for Architecture
- Use of LML for Systems Design
- Use of LML for Software Development
- 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!
Complexity

With the growth of the Internet and daily changes in IT, systems have become more complex and change more rapid than ever before.

- Systems engineering methods have not kept up with these changes.

- SE has been relegated to the beginning of the lifecycle.

From a presentation by Dr. Michael Ryschkewitsch, NASA Chief Engineer, at CSER Conference 15 April 2011.
How Does SE Typically Respond to Complexity?

• Focus on “architecture”
• More complex languages
• More complex procedures
• More layers of abstraction
  – “Systems of Systems”
  – “Family of Systems”
  – “Portfolio Management”
• Need more time and money!
More Money Is a Problem

Calls for doing more with less continue

Need for lower labor and tool costs essential for acceptance of SE across the lifecycle

Use and Challenges vary throughout the life cycle

- Early Phases; conduct Systems Analysis and trade studies
  - Exercise through CONOPS
  - Rapidly dismiss faulty options
  - Develop feasible, cost effective options
  - Money advanced technology requirements
    - How to enable modeling that provides the needed fidelity yet can be done quickly and cheaply?
    - Current methods tend to be "weeware" intense.
    - Need rapid and effective teaming

- Development
  - Refine Design, support Validation & Verification
  - Enhance manufacturing
    - How to better enable integration of discipline oriented design tools into systems models that capture functional and performance behaviors
    - How to capture system design rationale, assumptions and other "background" data
    - How do we develop the standards that allow lossless integration across organization and tool boundaries?

- Operations
  - Provide data for ops team, resolve in flight issues, address parts obsolescence
    - How do we make the full suite of information captured during design and development available to the operators without having prior knowledge of their needs?

From a presentation by Dr. Michael Ryschkewitsch, NASA Chief Engineer, at CSER Conference 15 April 2011
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
What We Did

• Researched the variety of languages (ontologies) in common use (DM2, SysML, BPMN, IDEF, SREM, etc.)

• Researched the variety of representations (FFBDs, N2, Behavior Diagrams, Class Diagrams, Electrical Engineering Diagrams, etc.)

• Took the best of each of these languages and representations and distilled them down to the essential elements, relationships, attributes, and diagrams

The result: Lifecycle Modeling Language
LIFECYCLE MODELING LANGUAGE (LML) OVERVIEW
The Lifecycle

Current Operations and Maintenance

- Architecture Development
- System Design
- Hardware/Software Acquisition

Design & Analysis

Future Operations and Maintenance

- Operational T&E and Transition
- Integration and Test

Future Operations and Maintenance

Integration & Verification

Demolition and Disposal

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/referenced by</td>
<td>references</td>
<td>captured by/consumed by</td>
<td>specified by</td>
<td>incurred by</td>
<td>generates resolves</td>
<td>-</td>
<td>located at</td>
<td>causes mitigates resolves</td>
<td>based on</td>
<td>takes occurs</td>
</tr>
<tr>
<td>ARTIFACT</td>
<td>referenced by</td>
<td>decomposed by/referenced by</td>
<td>related to</td>
<td>specified by</td>
<td>referenced by</td>
<td>incurs</td>
<td>receives</td>
<td>defines protocol for</td>
<td>referenced by</td>
<td>located at</td>
<td>causes mitigates resolves</td>
</tr>
<tr>
<td>ASSET</td>
<td>captured by/consumed by</td>
<td>performs</td>
<td>produced by</td>
<td>references</td>
<td>decomposed by/oriented by</td>
<td>related to</td>
<td>-</td>
<td>causes resolves responds to</td>
<td>connected by</td>
<td>located at</td>
<td>causes mitigates resolves</td>
</tr>
<tr>
<td>CHARACTERISTIC</td>
<td>specifies</td>
<td>references</td>
<td>specifies</td>
<td>decomposed by/oriented to</td>
<td>incurs</td>
<td>specifies</td>
<td>specifies</td>
<td>causes resolves</td>
<td>specifies</td>
<td>located at</td>
<td>causes mitigates resolves</td>
</tr>
<tr>
<td>COST</td>
<td>incurred by</td>
<td>incurred by</td>
<td>specified by</td>
<td>decomposed by/referenced to</td>
<td>incurred by</td>
<td>-</td>
<td>incurred by</td>
<td>causes incurs by resolves</td>
<td>incurred by</td>
<td>located at</td>
<td>causes incurs by resolves mitigates</td>
</tr>
<tr>
<td>INPUT/OUTPUT</td>
<td>generated by</td>
<td>references</td>
<td>-</td>
<td>specified by</td>
<td>decomposed by/referenced 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>
</tr>
<tr>
<td>ISSUE</td>
<td>caused by</td>
<td>resolved by</td>
<td>references resolved by</td>
<td>caused by</td>
<td>resolved by</td>
<td>caused by</td>
<td>resolved by</td>
<td>causes decomposed by/referenced to</td>
<td>caused by</td>
<td>resolved by</td>
<td>resolved by</td>
</tr>
<tr>
<td>LINK</td>
<td>-</td>
<td>defined protocol</td>
<td>by references</td>
<td>connects to</td>
<td>specified by</td>
<td>incurs</td>
<td>transfers</td>
<td>causes resolves</td>
<td>decomposed by/referenced to</td>
<td>located at</td>
<td>causes mitigates resolves</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>decomposed by/referenced to</td>
<td>locates</td>
<td>mitigates</td>
<td>based on</td>
</tr>
<tr>
<td>RISK</td>
<td>caused by</td>
<td>mitigated by</td>
<td>references resolved by</td>
<td>caused by</td>
<td>mitigated by</td>
<td>resolved by</td>
<td>causes mitigated by</td>
<td>causes resolved by</td>
<td>-</td>
<td>located at</td>
<td>causes mitigates resolved</td>
</tr>
<tr>
<td>STATEMENT</td>
<td>basis of</td>
<td>basic of</td>
<td>references</td>
<td>basis of</td>
<td>specified</td>
<td>basis of</td>
<td>incurs</td>
<td>basis of</td>
<td>causes resolves</td>
<td>-</td>
<td>basis of</td>
</tr>
<tr>
<td>TIME</td>
<td>taken by</td>
<td>occurred by</td>
<td>occurred by</td>
<td>occurred by</td>
<td>occurred by</td>
<td>occurred by</td>
<td>occurred by</td>
<td>date resolved by</td>
<td>occurred by</td>
<td>delays</td>
<td>occurred by</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
LML Physical Block Diagram*

*Note: Work in progress
LML Combined Physical Behavior Diagram* Enables Instances and Clones

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

*Note: Work in progress
LML Translation

- Two types of mapping for tailoring:
  - Map names of classes to enable other “schema” models to be used
  - Map symbols used (e.g., change from LML Logic to Electrical Engineering symbols)

| LML Class | DM2  | SysML | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Activity</td>
<td>Activity</td>
<td></td>
</tr>
<tr>
<td>Asset</td>
<td>Performer</td>
<td>Actor</td>
<td></td>
</tr>
</tbody>
</table>

| LML Symbol | Electrical Engineering | BPMN | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Other Diagrams

• Physical Block Diagrams
  – With option for icon substitution
• Interface Diagrams
  – N2 (Assets or Actions)
• Hierarchy Diagrams
  – Automatically color coded by class
• Time Diagrams
  – Gantt Charts
  – Timeline Diagram
• Location Diagrams
  – Maps for Earth
  – Orbital charts

• Risk Chart
  – Standard risk/opportunity chart
• Organization Charts
  – Showing lines of communication, as well as lines of authority
• Pie/Bar/Line Charts
  – For cost and performance
• Combined Physical and Functional Diagram (?)
Reducing Complexity by Association

<table>
<thead>
<tr>
<th>Action</th>
<th>Programmatic</th>
<th>Other</th>
<th>Picture</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Select Icon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Icon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Key Relationships**

- Parents (decomposes)
- Children (decomposed by)
- Inputs (receives/triggers)
- Outputs (generates)
- Statement (based on)
- Asset (performed by)
- Duration (takes)
Reducing Complexity by Association

- **Action**
  - Name
  - Number
  - **incurs:** Cost
    - List of Costs or None
  - **located at:** Location
    - Map It
    - List of Locations or None
  - **occurs:** Time (TimeFrame or PointInTime)
    - List of Times or None

- **Programmatic**
  - Relations for Risk Analysis
    - **causes:** Issue or Risk
      - List of Issues or Risks or None
    - **resolves:** Issue or Risk
      - List of Issues or Risks or None
    - **mitigates:** Risk
      - List of Risks or None
Reducing Complexity by Association

Peer-to-Peer Relationships

*related to: Action*

- List of Actions or None

*relates: Action*

- List of Actions or None

**references: Artifact**

- List of Artifacts or None

**specified by: Characteristic**

- List of Characteristics or None
Reducing Complexity by Association

Relations for Resource Modeling

- **captures: Resource (Asset subclass)**
  - List of Resources or None

- **consumed by: Resource (Asset subclass)**
  - List of Resources or None

- **produces: Resource (Asset subclass)**
  - List of Resources or None
  - Amount
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)
SUGGESTED LML PROCESSES
Systems Engineering During Design Phase

- Requirements Analysis
  - Top Down
  - Best Use: “Classical SE”

- Functional Analysis and Allocation
  - Middle Out
  - Best Use: Architecture Development (To-Be)

- System Analysis and Control

- Synthesis
  - Bottom Up
  - Best Use: Reverse Engineering (As-Is)

Adapted from EIA-632
**Requirements Analysis**

1. **Decompose Statements**
2. **Critical Issue?**
   - YES: Determine Options and Perform Trade Studies
   - NO: Resolve Issues with Customer
3. **Statement Verifiable?**
   - YES: Identify Risks and Plan Mitigation
   - NO: Coordinate Changes to Make Statement Verifiable
4. **Review Statements and Risks with Customer**
5. **Update Knowledgebase**

**Documents**:
- Source Documents
- Change Requests
- User Needs

**External Interface Database**

**Preliminary Test Requirements**

**Updated Requirements Traceability Matrix**

**Standards Selected**

**Architecture Knowledgebase**

See System Analysis and Control for details.
Key LML Elements for Requirements Analysis Process

• Artifact

• Statement

• Risk
• Issue
Functional Analysis and Allocation

- Develop/Revise Context Diagram
- Develop Series of Scenarios for Analysis
- Create/Update System Behavior Model
- Analyze Behavior Model Performance
- Identify Risks and Plan Mitigation
- Review Model and Risks with Customer
- Allocate Actions to Assets and Input/Outputs to Links

Behavior Model
- Control Flow
- Data Flow (Activity Model)
- Performance Criteria

See System Analysis and Control for details
Key LML Elements for Functional Analysis Process

• Action

• Input/Output
  – Types: Analog, Digital, Event, Mixed, Physical, Product, Verbal

• Time: Duration
Synthesis

1. Identify Component Assets
2. Optional Assets?
   - YES: Determine Options and Perform Trade Studies
   - NO: Allocate Assets
3. Select New Assets in Coordination with Customer
4. Identify Risks and Plan Mitigation
5. Review Design and Risks with Customer

See Functional Analysis and Control for details

Programmatic Roadmap

Link to programs

Technology Insertion Recommendations

Architecture Knowledgebase

Updated Architecture Knowledgebase

Functional Requirements Document (FRD)

Design Diagrams

Transition Plan

SPEC INNOVATIONS
Key LML Elements for Synthesis Process

• Asset

• Link

• Location
• Cost
• Characteristic
Systems Analysis and Control

- Identify Technologies, Methods and Needs for Study
- Develop Metrics for Trades
- Analyze Effectiveness, Use, O&M, Security, and Training Impacts
- Analyze Performance and Cost Effectiveness (Dynamic)
- Recommend Changes and Prepare Reports
- Assess Risk Probability and Consequence
- Develop Risk Mitigation Approach
- Trade Study Reports
- Updated Architecture Knowledgebase
Key LML Elements for Systems Analysis and Control Process

• Characteristic
  – Types: Condition, Data Element, Environmental, Functional, Performance, Physical, Scenario, Security, Verification Category

• Time: subclass Timeframe

• Risk
Break
LML TOOL NEEDS
Tool for Vision for MBSE

- Tool must be designed as a MBSE tool using Lifecycle Modeling Language (LML)
  - All artifacts must be produced from the tool using standard reports
- Tool must enable both seamless data flow and distributive, collaborative teams working on the same model
- Implies need for ability to scale “infinitely” on demand

From a presentation by Dr. Michael Ryschkewitsch, NASA Chief Engineer, at CSER Conference 15 April 2011
Tool for Complexity

- Tool must be designed for use throughout the lifecycle, thus enabling end-to-end systems engineering
  - Legacy systems are a key part of any modeling environment – LML has specific information classes for capturing designs at different times in one knowledgebase
- Tool must embed simulation, which enables real time simulations as well as the ability to scale “infinitely” to hundreds of server instances
- Tool must also reduce complexity by use of special input screens, which enhance configuration management as well

From a presentation by Dr. Michael Ryschkewitsch, NASA Chief Engineer, at CSER Conference 15 April 2011
Tool for Multi-Decadal/Generation

Supporting multi-decadal, multi-generation activities is a major challenge

• The International Space Station will have a lifespan of at least 20 years with evolving uses and constant changes.
• Systems analyses show that as we explore beyond low earth orbit, launch costs will remain a driver and thus put a huge value in re-using systems already moved up the gravity well.
  - We will need to track systems health and status against predictions and threshold
  - Systems will be modified, updated and re-purposed multiple times
  - Operating environments and conditions may change from those used for design
  - Likely to want to use systems well beyond initial life objectives

From a presentation by Dr. Michael Ryschkewitsch, NASA Chief Engineer, at CSER Conference 15 April 2011

• Tool must enable tracking of performance over time as Characteristics of the Assets
• Environments and conditions can also be captured as evolved over time and location (including orbital locations)
• Tool must evolve of the lifecycle of systems, but the data must be captured and reused throughout the lifecycle
Tool for Uses and Challenges

Use and Challenges vary throughout the life cycle

- Early Phases: conduct Systems Analysis and trade studies
  - Exercise through CONOPS
  - Rapidly dismiss faulty options
  - Develop feasible, cost-effective options
  - Identify advanced technology requirements
  - How to enable modeling that provides the needed fidelity yet can be done quickly and cheaply?
  - Current methods tend to be "wetware" intense
  - "Need rapid and effective "teamring"

- Development
  - Refine Design, support Validation & Verification
  - Enhance manufacturing
  - How to better enable integration of discipline oriented design tools into systems models that capture functional and performance behaviors
  - How to capture system design rationale, assumptions and other "background" data
  - How do we develop the standards that allow lossless integration across organization and tool boundaries?

- Operations
  - Provide data for ops team, resolve in flight issues, address parts obsolescence
  - How do we make the full suite of information captured during design and development available to the operators without having prior knowledge of their needs?

From a presentation by Dr. Michael Ryschkewitsch, NASA Chief Engineer, at CSER Conference 15 April 2011

- Tool needs to create a CONOPS report based on scenario modeling
- Capture Issues and Decisions, as well as Cost for cost, schedule and performance tradeoffs
- TRL levels must be easily captured in the tool
- Software as a service provides inexpensive tool, while scalability enables fidelity level required; LML method enables rapid SE design and analysis

- Tool should automate what can be automated to reduce "wetware" requirements and enhance teaming
Tool for Uses and Challenges (continued)

Use and Challenges vary throughout the life cycle

- Early Phases, conduct Systems Analysis and trade studies
  - Exercise through CONOPS
  - Rapidly dismiss faulty options
  - Develop feasible, cost effective options
  - Identify advanced technology requirements
  - How to enable modeling that provides the needed fidelity yet can be done quickly and cheaply?
  - Current methods tend to be “wetware” intense
  - Need rapid and effective “teaming”

- Development
  - Refine Design, support Validation & Verification
  - Enhance manufacturing
  - How to better enable integration of discipline oriented design tools into systems models that capture functional and performance behaviors?
  - How to capture system design rationale, assumptions and other “background” data
  - How do we develop the standards that allow lossless integration across organization and tool boundaries?

- Operations
  - Provide data for ops team, resolve in flight issues, address parts obsolescence
  - How do we make the full suite of information captured during design and development available to the operators without having prior knowledge of their needs?
  - From a presentation by Dr. Michael Ryschkewitsch, NASA Chief Engineer, at CSER Conference 15 April 2011

- Operations data and obsolescence part of LML elements (Assets, Characteristics and Time)

- Use of a scenario-based approach enables better information capture from operators

- Tool should enable design down to detailed level (if desired), as well as V&V support

- Manufacturing enhanced by linking design to CAD/CAM systems (export design information as required)

- Use of XML files to import/export data from other design tools

- Design rationale, assumption and other programmatic information captured as part of Issues, Risk and other program-related elements of LML

Standards should be captured and developed using the tool; enforcement by tailoring “personal trainer” version
Tool for Managing Complexity

• Tool in conjunction with a process should capture a reasonable number of scenarios for modeling and simulation, using a test matrix approach (move away from “peeling the onion”)

• Use “test matrix” approach to provide breadth of problem space, including failure modes, to capture the full functionality required

• Test must become even more dependent on simulation

Managing Complexity

Current approaches tend to reflect human thinking that complex systems are simple extrapolations of simpler systems yet we know this is not true – N2, difficulty to test completely, etc

– We currently divide and conquer, attempting to limit interactions via simple interfaces
– What are the optimal strategies for limiting the undesirable or unrecognized effects
– Are there better test strategies (probabilistic methods???)
– Are there modeling and simulation strategies that allow us to truly model probabilistic behaviors in all their details and exercise them against operational scenarios and yet do so quickly and efficiently so that they become design tools
– move away “peeling the onion”

From a presentation by Dr. Michael Ryschkewitsch, NASA Chief Engineer, at CSER Conference 15 April 2011
Tool for Managing Complexity

Collaborative Systems Engineering

• Current systems processes inherently reflect the need to limit complexity and detail in models, simulation and analyses
  – Driven by time to assembly, compute power, lack of interface standards and techniques
  – Forced to extract behaviors by parameterization or simplifying models
  – Will this always be the case?
    • Just like CAD systems have libraries of detailed parts, can we have methodologies that allow integration of complex sub-models?
    • Can we have methodologies that allow systems models to evolve with initial simple parametric sub-models and evolve to design based sub-models?
    • Can we better enable rapid and diverse teaming unimpeded by technological barriers?

From a presentation by Dr. Michael Ryschkewitsch, NASA Chief Engineer, at CSER Conference 15 April 2011

• Tool should enable sharing of models, thus creating a “library” of component models
• Contributions should be made from academia (tool should be free access to academia) to this library
• Government sponsored research can also be made available to the NASA family via website
Key Tool Features Needed

- Scalable Modeling
- Scalable Discrete Event Simulation
- Functional Behavior Diagrams
- Physical Block Diagrams w/ Icons
- Element Extractor/Automated Parsing
- Splitting and Merging Requirements
- Merge databases from other tools
- Access to the web
- Automated report writing
- Enterprise data search
- Customization
- Collaboration
- Configuration Management
USE OF LML FOR ARCHITECTURE DEVELOPMENT
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 Production 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.

© 2011 Systems and Proposal Engineering Company. All Rights Reserved
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)
# Key LML Elements and Use - Architecture

<table>
<thead>
<tr>
<th>Elements</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Scenario analysis</td>
</tr>
<tr>
<td>Asset</td>
<td>High level for gap analysis</td>
</tr>
<tr>
<td>Cost</td>
<td>High level cost estimates - lifecycle</td>
</tr>
<tr>
<td>Characteristic</td>
<td>Key metrics for performance/acceptance</td>
</tr>
<tr>
<td>Input/Output</td>
<td>Key data elements</td>
</tr>
<tr>
<td>Link</td>
<td>Critical interfaces (between systems of systems)</td>
</tr>
<tr>
<td>Risk</td>
<td>Key schedule, cost, performance risks</td>
</tr>
<tr>
<td>Time - Timeframe</td>
<td>Broad roadmap</td>
</tr>
</tbody>
</table>
USE OF LML FOR SYSTEM DESIGN
System Design Process

- Requirements Analysis
  - Top Down: "Classical SE"
- Functional Analysis and Allocation
  - Middle Out: Architecture Development (To-Be)
- System Analysis and Control
- Synthesis
  - Bottom Up: Reverse Engineering (As-Is)

Adapted from EIA-632
Key Systems Design Products

- System/Segment Specifications
- Interface Control Documents
- Detailed Integration and Test Plans
# Key LML Elements and Use – System Design

<table>
<thead>
<tr>
<th>Elements</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Functional analysis</td>
</tr>
<tr>
<td>Asset</td>
<td>Design packaging</td>
</tr>
<tr>
<td>Cost</td>
<td>Detailed cost estimates - lifecycle</td>
</tr>
<tr>
<td>Characteristic</td>
<td>Specific metrics for performance/acceptance</td>
</tr>
<tr>
<td>Input/Output</td>
<td>Data elements</td>
</tr>
<tr>
<td>Link</td>
<td>Interfaces (between components)</td>
</tr>
<tr>
<td>Location</td>
<td>Identifies differences due to location</td>
</tr>
<tr>
<td>Risk</td>
<td>Schedule, cost, performance risks</td>
</tr>
<tr>
<td>Statement</td>
<td>Key requirements</td>
</tr>
<tr>
<td>Time - Duration</td>
<td>Key time dependences</td>
</tr>
</tbody>
</table>
USE OF LML FOR SOFTWARE DEVELOPMENT
Support Object Analysis (if desired)

- Assets = Objects
- Links = Relationships/Interfaces
- Characteristics = Attributes
- Actions = Methods
  - Can include key logic algorithms
Otherwise – Support Functional Requirements Specification

• Actions
• Input/Output
• Packaging using Assets and Links
## Key LML Elements and Use – Software Development

<table>
<thead>
<tr>
<th>Elements</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Functional requirements for methods</td>
</tr>
<tr>
<td>Asset</td>
<td>Classes and/or packages</td>
</tr>
<tr>
<td>Characteristic</td>
<td>Specific metrics for performance/acceptance</td>
</tr>
<tr>
<td>Input/Output</td>
<td>Data elements</td>
</tr>
<tr>
<td>Link</td>
<td>Interfaces (Inheritance between Classes)</td>
</tr>
</tbody>
</table>
USE OF LML IN TEST AND EVALUATION
Coming Up the Vee

- Design Monitoring
- Integration
- Verification
- I&V Planning

The diagram illustrates the Vee model with arrows indicating the flow and interaction between various engineering processes.
Measure of Performance (MOP) View

Characteristics

Measure of Effectiveness (MOE)
  Type = MOE
  MOE 1
  decomposed by / decomposes
  1:M

Measure of Performance (MOP)
  Type = MOP
  MOP 1.1
  instantiates / instantiated by
  1:M

MOP Test Result
  Type = MOP Occurrence
  MOP 1.1 [System (SW1/HW1)]
  specifies / specified by
  1:1

MOP Test Result
  Type = MOP Occurrence
  MOP 1.1 [System (SW2/HW2)]
  specifies / specified by
  1:1

MOP Test Result
  Type = MOP Occurrence
  MOP 1.1 [System (SW3/HW3)]
  specifies / specified by
  1:1

Actions

System Function
  Type = System Function
  Function
  tests / tested by
  1:M

allocated to / performs
  1:1

Assets

System Instantiation
  Type = System Instantiation
  System (SW1/HW1)
  instantiates / instantiated by
  1:1

System Instantiation
  Type = System Instantiation
  System (SW2/HW2)
  instantiates / instantiated by
  1:1

System Instantiation
  Type = System Instantiation
  System (SW3/HW3)
  instantiates / instantiated by
  1:1

System
  Type = System
  System
  1:M
# Key LML Elements and Use – Test and Evaluation

<table>
<thead>
<tr>
<th>Elements</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Test processes and procedures/ functional requirements for testing</td>
</tr>
<tr>
<td>Asset</td>
<td>Test configurations</td>
</tr>
<tr>
<td>Cost</td>
<td>Test costs</td>
</tr>
<tr>
<td>Characteristic (Technical Measurement subclass)</td>
<td>Specific metrics for performance/acceptance (MOEs, MOPs)</td>
</tr>
<tr>
<td>Input/Output</td>
<td>Data elements for measurement</td>
</tr>
<tr>
<td>Link</td>
<td>Interfaces (between components)</td>
</tr>
<tr>
<td>Statement</td>
<td>Key requirements for testing</td>
</tr>
</tbody>
</table>
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
## Key LML Elements and Use – Operations and Support

<table>
<thead>
<tr>
<th>Elements</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>O&amp;S processes and procedures</td>
</tr>
<tr>
<td>Asset</td>
<td>Configurations</td>
</tr>
<tr>
<td>Cost</td>
<td>O&amp;S costs</td>
</tr>
<tr>
<td>Characteristic (Technical Measurement subclass)</td>
<td>Specific metrics for performance/acceptance (MOEs, MOPs)</td>
</tr>
<tr>
<td>Input/Output</td>
<td>Data elements for decision making</td>
</tr>
<tr>
<td>Location</td>
<td>Differences due to location</td>
</tr>
</tbody>
</table>
SUMMARY
People Considerations for LML

- Large teams make organization and focus on a vision very difficult
  - Need better ways to collaborate
- You need people with a wide variety of skills and personalities
  - Someone with vision
  - Someone who can perform the detailed system engineering
  - Someone who understands the domain
  - Someone familiar with the technique and tools
  - Someone who understands the process
- They need to be trained as a team – including the government personnel
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
• Reduces complexity