Lifecycle Modeling Language (LML) – A Technique for Enhancing Reliability, Availability, and Maintainability (RAM) throughout the Lifecycle

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

October 2013

Overview

- Why is RAM often overlooked until late in the lifecycle?
- What is LML?
- How does LML help enhance RAM?
- What processes and tools work with LML to enhance RAM?
- Summary

Why is RAM often overlooked until late in the lifecycle?

- RAM analysis requires details to estimate uncertainties in estimated values and requirements, which takes time and money
- As such, it often is not addressed at all until the detailed design phase
- However, RAM should be part of the overall scenario analysis at the very beginning of the concept development phase

So what happens? Someone arbitrarily assigns the number of "9's" needed.

Example "Requirements" for FireSAT

From Applied Space Systems Engineering, p. 113

- Reliability: "The FireSAT spacecraft shall have an on-orbit lifetime of at least five years"
- Availability: "The FireSAT spacecraft shall have an operational availability of 98%, excluding outages due to weather, with a maximum continuous outage of no more than 72 hours"
- Maintainability: "The FireSAT spacecraft shall require the removal (or opening) of no more than ten fasteners (panels) to replace any Line Replaceable Unit (LRU) ... during pre-launch ground operations"

Where did these come from? Were they the result of analyses or are they just best guesses?

SPEC INNOVATIONS

Lots of metrics to take into account

- It's not just the RAM metrics – it's all the "illities"
- How can we capture and trace all these metrics?

Commonality

- Physical commonality (within the system)
  - Hardware commonality
    - Number of unique line replaceable units (LRUs)
    - Number of unique fasteners
    - Number of unique cables
    - Number of unique standards implemented
  - Software commonality
    - Number of unique software packages implemented
    - Number of languages
    - Number of compilers
    - Average number of software instantiations
    - Number of unique standards implemented
- Physical familiarity (from other systems)
  - % vendors known
  - % subcontractors known
  - % hardware technology known
  - % software technology known
- Operational commonality
  - % of operational functions automated
  - Number of unique skill codes required
  - Estimated operational training time—initial
  - Estimated operational training time—refresh from previous system
  - Estimated maintenance training time—initial
  - Estimated maintenance training time—refresh from previous system

Modularity

- Physical modularity
  - Ease of system element upgrade
    - Lines of modified code
    - Number of labor hours for system rework
  - Ease of operating system upgrade
    - Lines of modified code
    - Number of labor hours for system rework
- Functional modularity
  - Ease of adding new functionality
    - Lines of modified code
    - Number of labor hours for system rework
  - Ease of upgrade existing functionality
    - Lines of modified code
    - Number of labor hours for system rework
- Orthogonality
  - Are functional requirements fragmented across multiple processing elements and interfaces?
  - Are there throughput requirements across interfaces?
  - Are common specifications identified?
- Abstraction
  - Does the system architecture provide options for information hiding?
- Interfaces
  - # of unique interfaces per system element
  - # of different networking protocols
  - Explicit versus implicit interfaces
    - Does the architecture involve implicit interfaces?
  - # of cables in the system

Standards Based

- Open systems orientation
  - Interface standards
    - # of interface standards/# of interfaces
    - Multiple vendors (more than 5) exist for products based on standards
    - Multiple business domains apply/use standard (aerospace, medical, telecommunications)
    - Standard maturity
  - Hardware standards
    - # of form factors/# of LRUs
    - Multiple vendors (more than 5) exist for products based on standards
    - Multiple business domains apply/Use standard (aerospace, medical, telecommunications)
    - Standard maturity
  - Software standards
    - # of proprietary and unique operating systems
    - # of non-standard databases
    - # of proprietary middleware
    - # of non-standard languages
- Consistency orientation
  - Common guidelines for implementing diagnostics and performance monitoring and fault localization
  - Common guidelines for implementing OMI

RMT

- Reliability
  - Fault tolerance
    - % of mission critical functions with single points of failure
    - % of safety critical functions with single points of failure
  - Critical points of delicateness (system loading)
    - % processor loading
    - % memory loading
      - How critical is this?
    - % network loading
      - How critical is this?
- Maintainability
  - Expected mean time to repair
  - Maximum fault group size
  - Is system operational under maintenance?
  - Accessibility
    - Are there space restrictions?
    - Are there special tool requirements?
    - Are there special skills requirements?
- Testability
  - # of LRUs covered by BIT (BIT coverage)
  - Reproducibility of errors
    - Logging or recording capability
    - Create system state at time of system failure?
  - Online testing
    - Is system operational during external testing?
    - Ease of access to external test points?
  - Automated input or stimulation insertion

FIGURE 5-37. **Commonly Used Metrics for Architectural Assessments.** We use these metrics to assess architectures. These examples are from the Architecture Trade-off Analysis Method (ATAM) Business Drivers (Figure 5-35) or Quality Attribute Requirements (Figure 5-36). Middleware is a layer of software that enables open systems. (RMT is reliability, maintainability, and testability; OMI is operator machine interface; BIT is built-in test.)

From Applied Space Systems Engineering, p. 189

What Is LML?

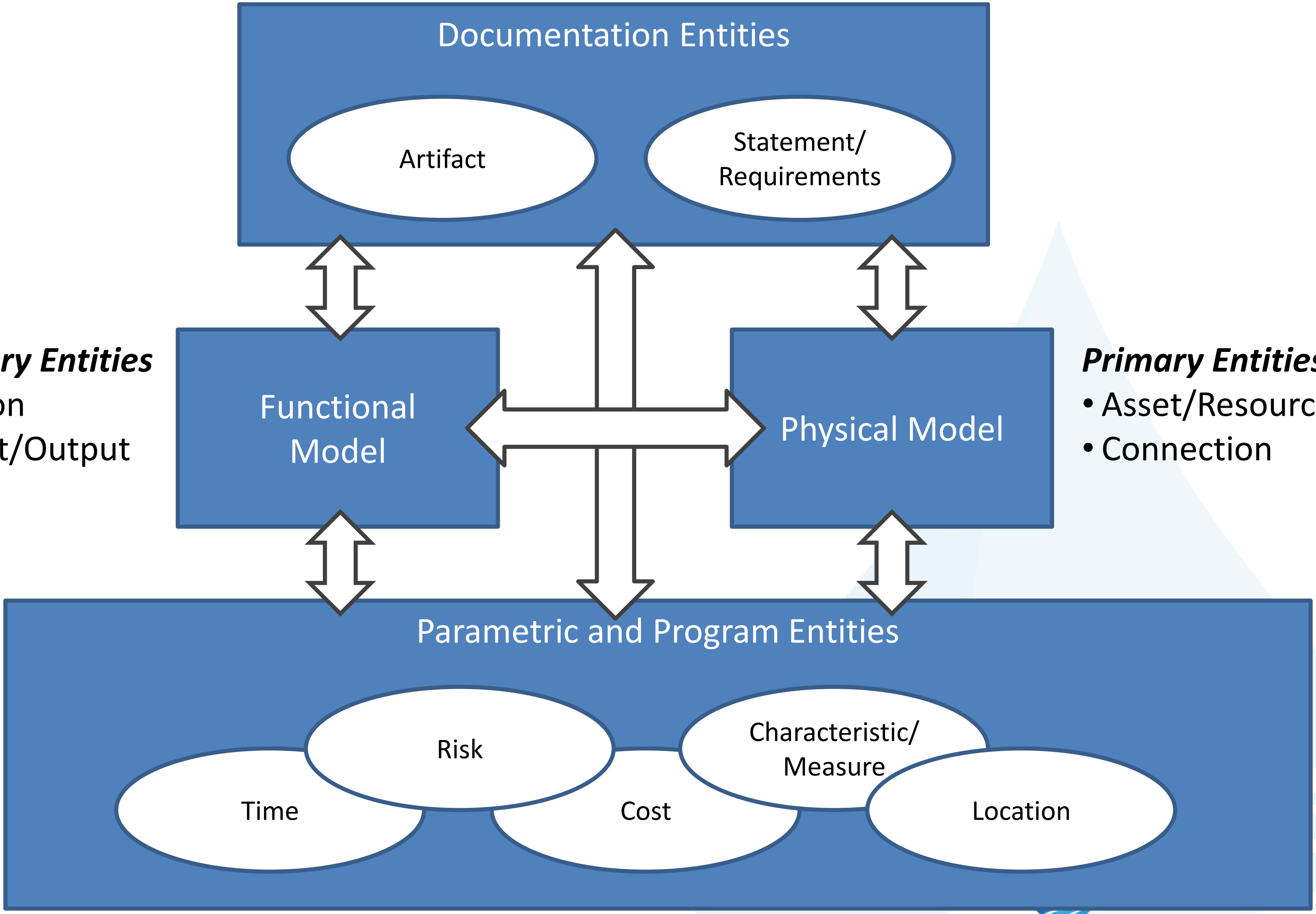
Lifecycle Modeling Language (LML)

- 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

SPEC INNOVATIONS

LML Models

Primary Entities
- Action
- Input/Output



Primary Entities
- Asset/Resource
- Connection

LML Taxonomy Simplifies and Enhances the Semantic Schema Elements

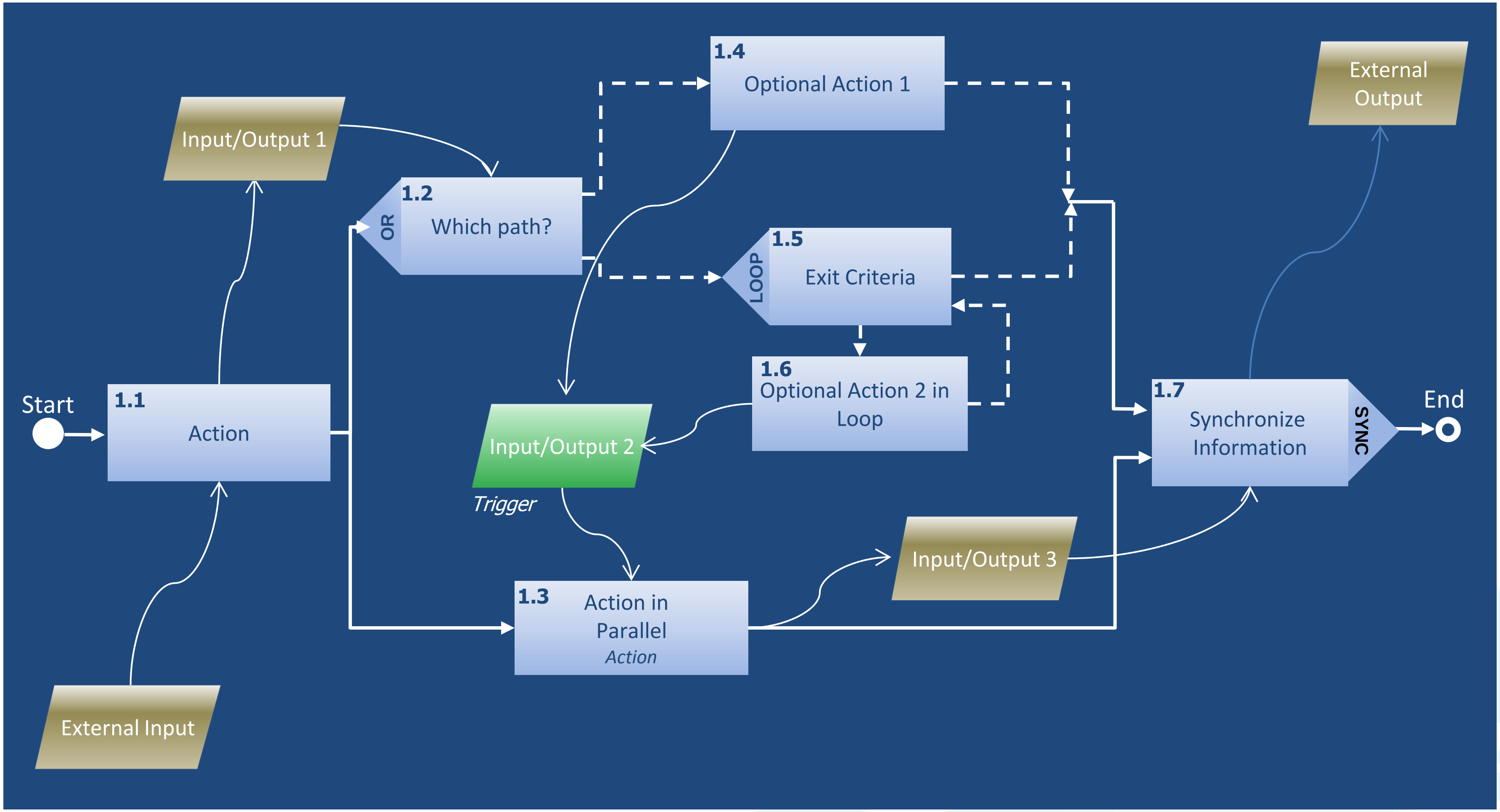
- Action
- Artifact
- Asset
  - Resource
- Characteristic
  - Measure
- Connection
  - Logical
  - Conduit
- Cost
- Input/Output
- Location
  - Physical
  - Orbital
  - Virtual
- Risk
- Software Interface
- Statement
  - Requirement
  - Decision
- Time

LML Relationships Provide Linkage Needed Between the Classes

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

	Action	Artifact	Asset (Resource)	Characteristic (Measure)	Connection (Conduit, Logical)	Cost	Decision	Input/Output	Location (Orbital, Physical, Virtual)	Risk	Statement (Requirement)	
Action	decomposed by* related to*	references	(consumes) performed by (produces) (seizes)	specified by	-	incurs	enables results in	generates receives	located at	causes mitigates resolves	(satisfies) traced from (verifies)	occurs
Artifact	referenced by	decomposed by* related to*	referenced by	referenced by specified by	defines protocol for referenced by	incurs referenced by	enables referenced by results in	referenced by	located at	causes mitigates referenced by resolves	referenced by (satisfies) source of traced from (verifies)	occurs
Asset (Resource)	(consumed by) performs (produced by) (seized by)	references	decomposed by* orbited by* related to*	specified by	connected by	incurs	enables made responds to results in	-	located at	causes mitigates resolves	(satisfies) traced from (verifies)	occurs
Characteristic (Measure)	specifies	references specifies	specifies	decomposed by* related to* specified by*	specifies	incurs specifies	enables results in specifies	specifies	located at specifies	causes mitigates resolves specifies	(satisfies) spacifies traced from (verifies)	occurs specifies
Connection (Conduit, Logical)	-	defined protocol by references	connects to	specified by	decomposed by* joined by* related to*	incurs	enables results in	transfers	located at	causes mitigates resolves	(satisfies) traced from (verifies)	occurs
Cost	incurred by	incurred by references	incurred by	incurred by specified by	incurred by	decomposed by* related to*	enables incurred by results in	incurred by	located at	causes incurred by mitigates resolves	incurred by (satisfies) traced from (verifies)	occurs
Decision	enabled by result of	enabled by references result of	enabled by made by responded by result of	enabled by result of specified by	enabled by result of	enabled by incurs result of	decomposed by* related to*	enabled by result of	located at	causes enabled by mitigated by result of resolves	alternative enabled by traced from result of	date resolved by decision due occurs
Input/Output	generated by received by	references	-	specified by	transferred by	incurs	enables results in	decomposed by* related to*	located at	causes mitigates resolves	(satisfies) traced from (verifies)	occurs
Location (Orbital, Physical, Logical)	locates	locates	locates	locates specified by	locates	locates	locates	locates	decomposed by* related to*	locates mitigates	locates (satisfies) traced from (verifies)	occurs
Risk	caused by mitigated by resolved by	caused by mitigated by references resolved by	caused by mitigated by resolved by	caused by mitigated by resolved by specified by	caused by mitigated by resolved by	caused by incurs mitigated by resolved by	caused by enables mitigated by results in resolved by	caused by mitigated by resolved by	located at mitigated by	caused by* decomposed by* related to* resolved by*	caused by mitigated by resolved by	occurs mitigated by
Statement (Requirement)	(satisfied by) traced to (verified by)	references (satisified by) sourced by traced to (verified by)	(satisified by) traced to (verified by)	(satisified by) specified by traced to (verified by)	(satisified by) traced to (verified by)	incurs (satisified by) traced to (verified by)	alternative of enables traced to results in	(satisified by) traced to (verified by)	located at (satisfied by) traced to (verified by)	causes mitigates resolves	decomposed by* traced to* related to*	occurs (satisified by) (verified by)
Time	occurred by	occurred by	occurred by	occurred by specified by	occurred by	occurred by	date resolves decided by occurred by	occurred by	occurred by	occurred by mitigates	occurred by (satisfies) (verifies)	decomposed by* related to*

LML Action Diagram Captures Functional and Data Flow



Uses Common Diagrams for Every Class

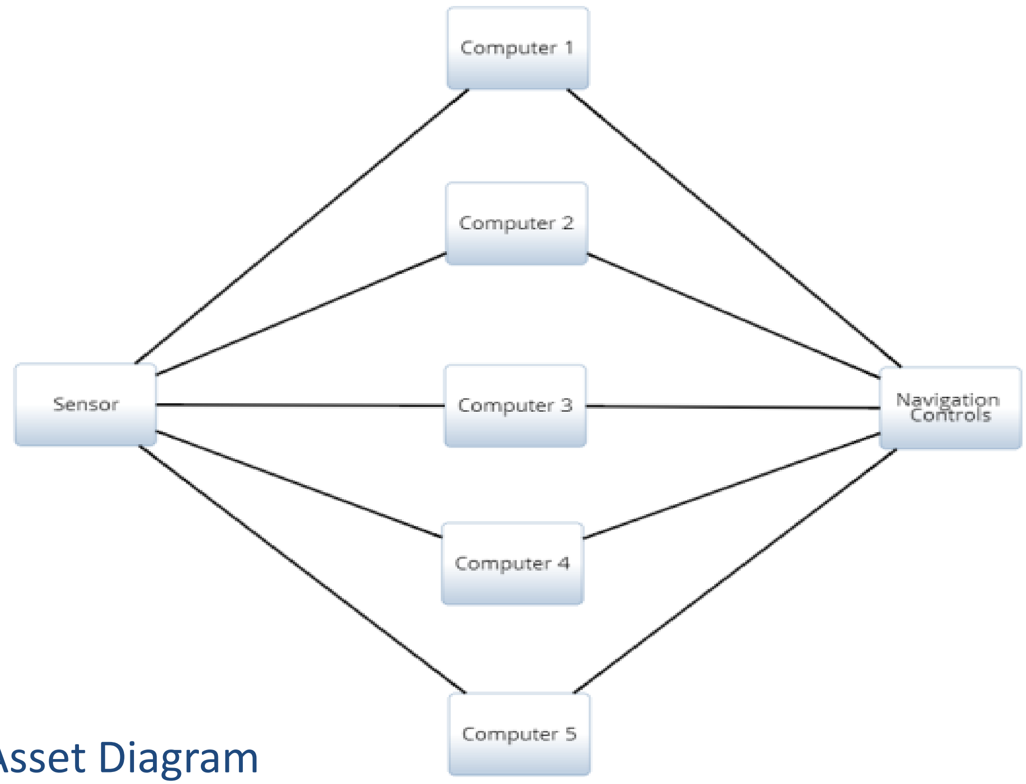
- Physical Block (Asset) Diagrams
  - With option for icon substitution
- Interface Diagrams
  - N2 (Assets or Actions)
- Hierarchy Diagrams
  - Automatically color coded by class
- Time Diagrams
  - Gantt Charts
  - Timeline Diagram
- Location Diagrams
  - Maps for Earth
  - Orbital charts
- Class/Block Definition Diagram
  - Data modeling
- 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

How Does LML Help Enhance RAM?

How does LML help enhance RAM?

- LML was designed with all aspects of systems engineering across the lifecycle
- LML provides:
  - Asset/Resource entities, Asset Diagrams, and Characteristics/Measures entities to capture physical information about the system
  - Action entities, Action Diagrams, and Input/Output to capture and model processes
  - Action Diagrams can be simulated to include Resource use
- As such, LML can support the analyses needed to derive key RAM metrics, such as mean time between failures (MTFB)

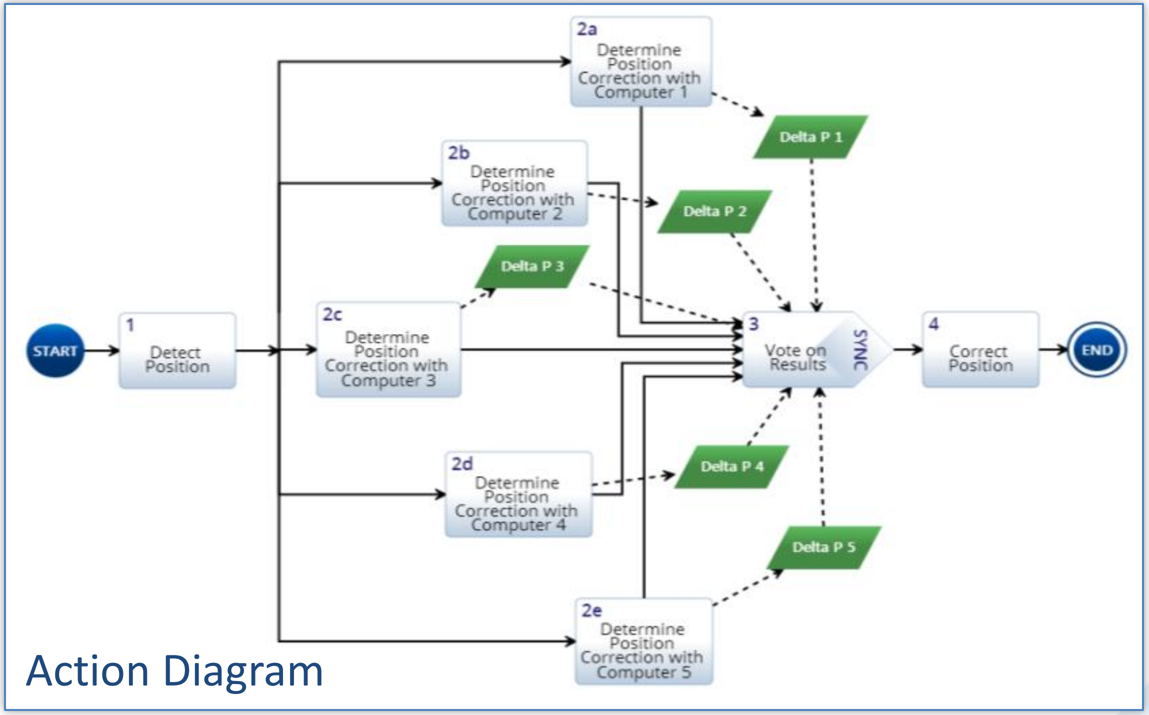
Example: Modeling for Reliability



Asset Diagram

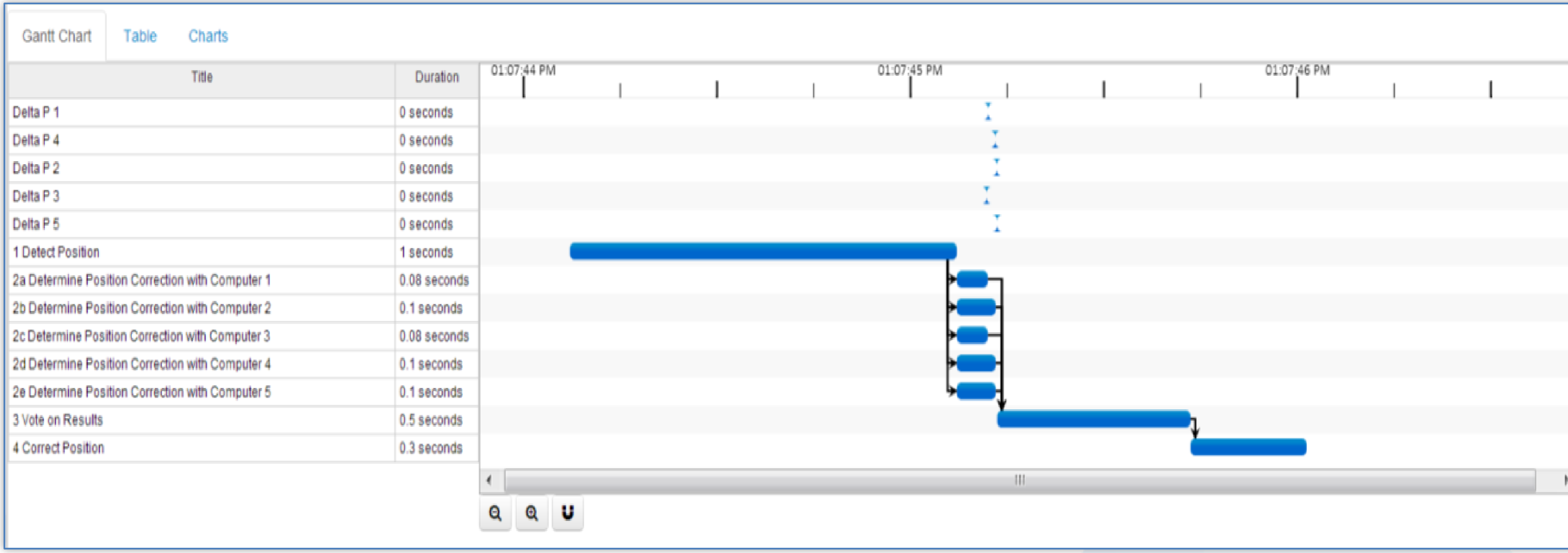
- Use of redundancy to enhance reliability
- Modeling multiple computers that "vote" on a value

Example continued



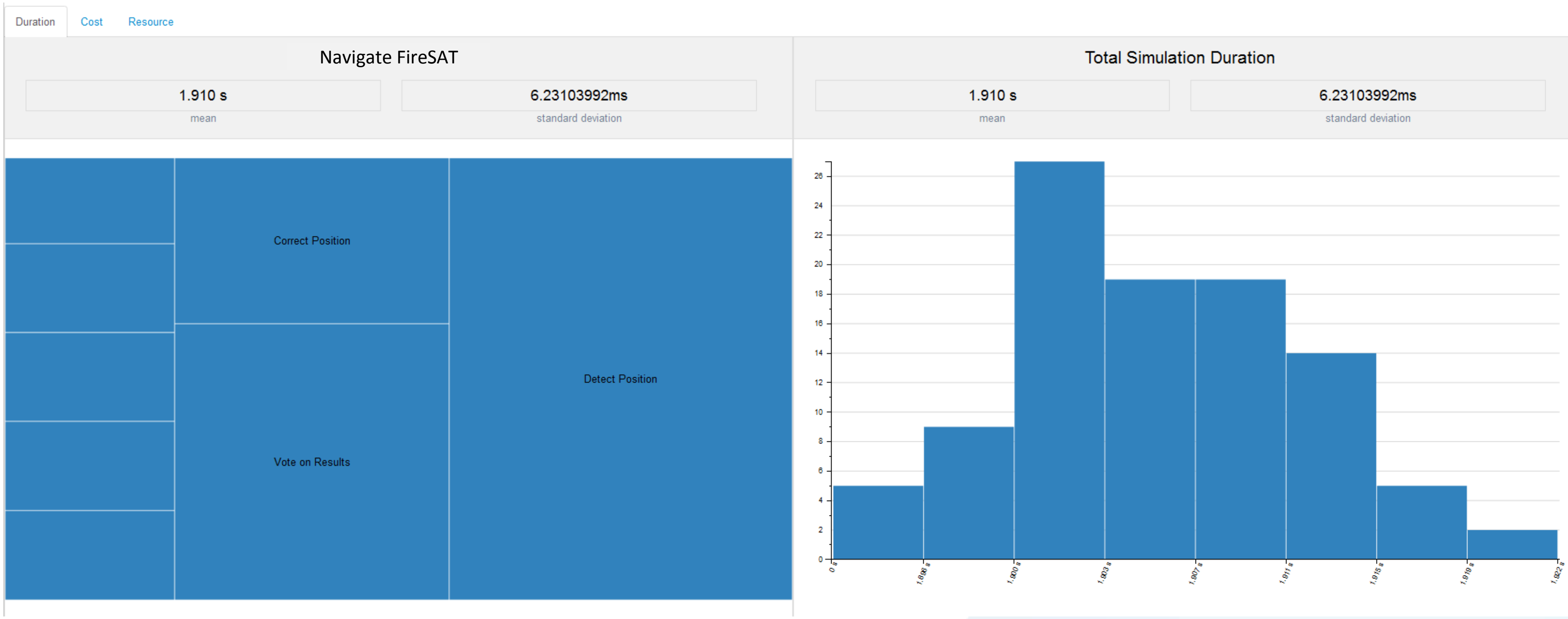
- Functional model equivalent using Action Diagram
- Timing provided for each computer can be a random distribution, as can failure modes

Simulation of Example



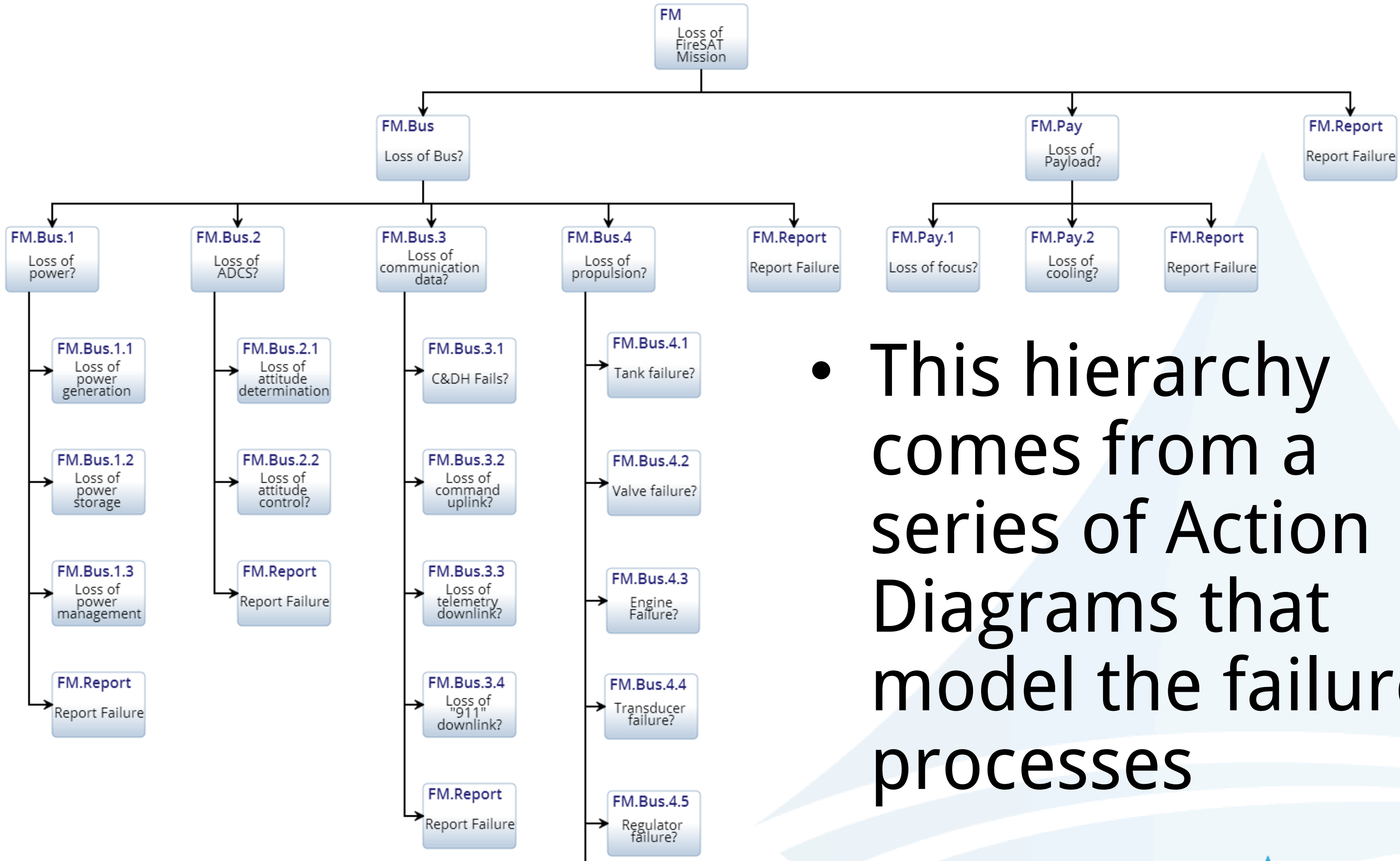
- Discrete Event Simulation of the Action Diagram can show the random nature of timing
- Sensitivity to failure modes can then be identified and mitigated

Monte Carlo simulation of Action Diagram supports reliability analysis



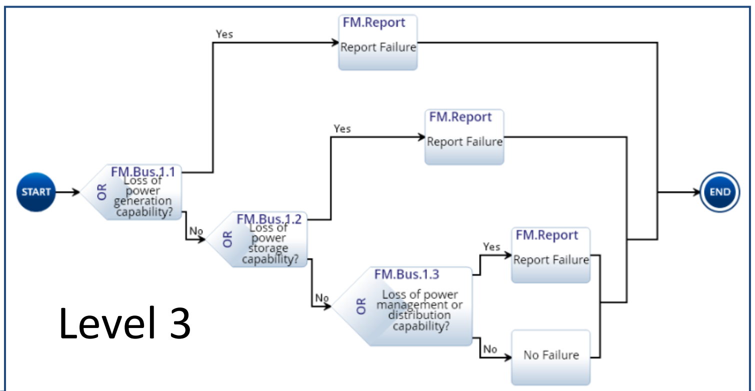
- Executing the model with random time distributions provides way to derive key metric requirements

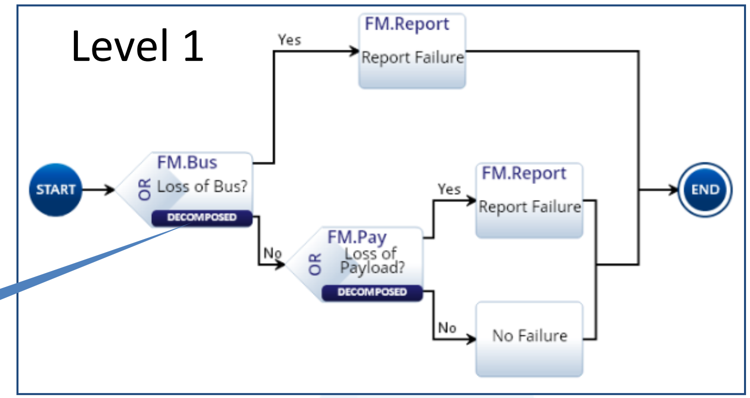
FireSAT Failure Mode Hierarchy

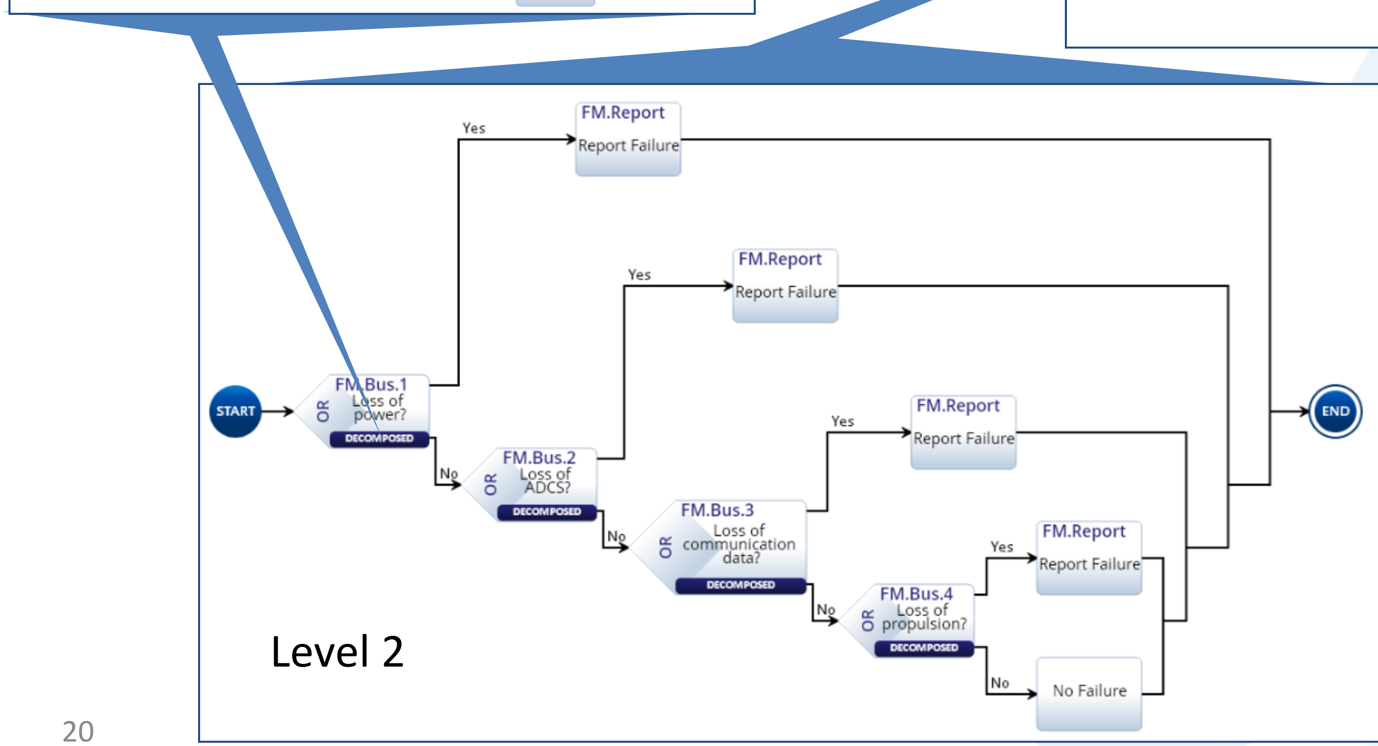


- This hierarchy comes from a series of Action Diagrams that model the failure processes

Action Modeling for FMECA



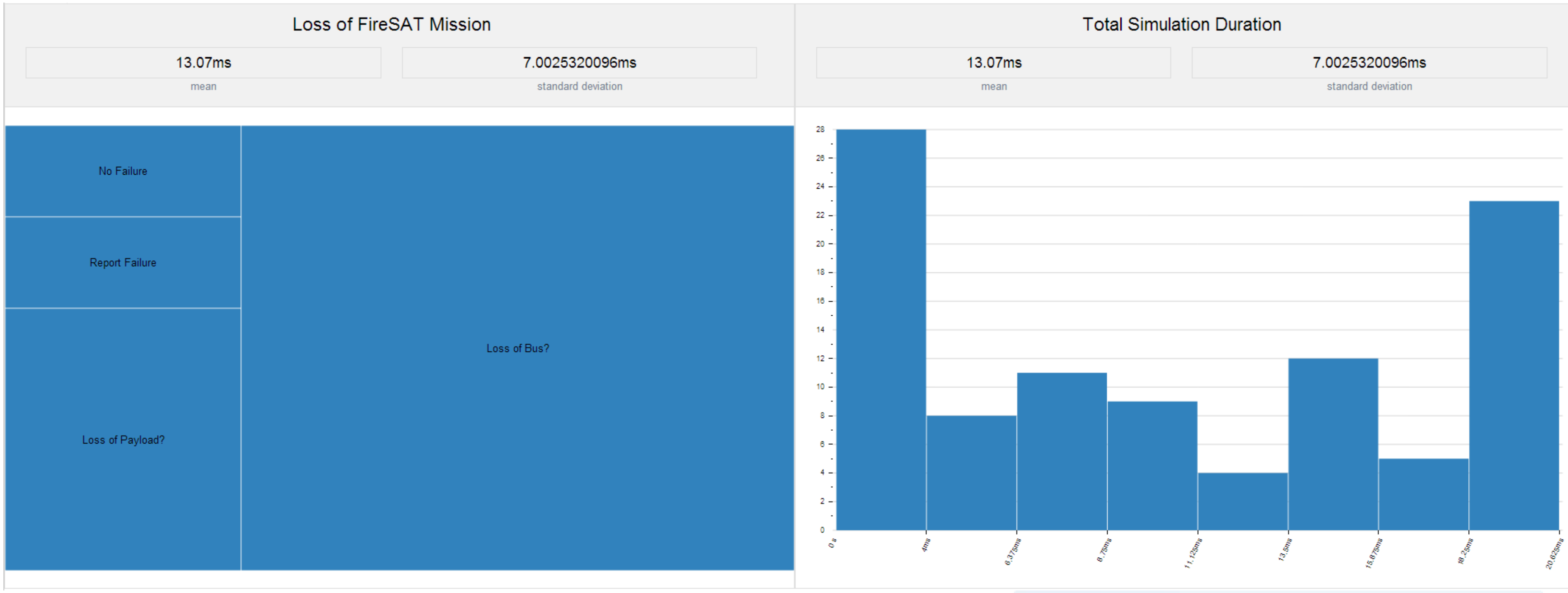




- Modeling failure modes with Action Diagram

SPEC INNOVATIONS

Execution of FMECA Model



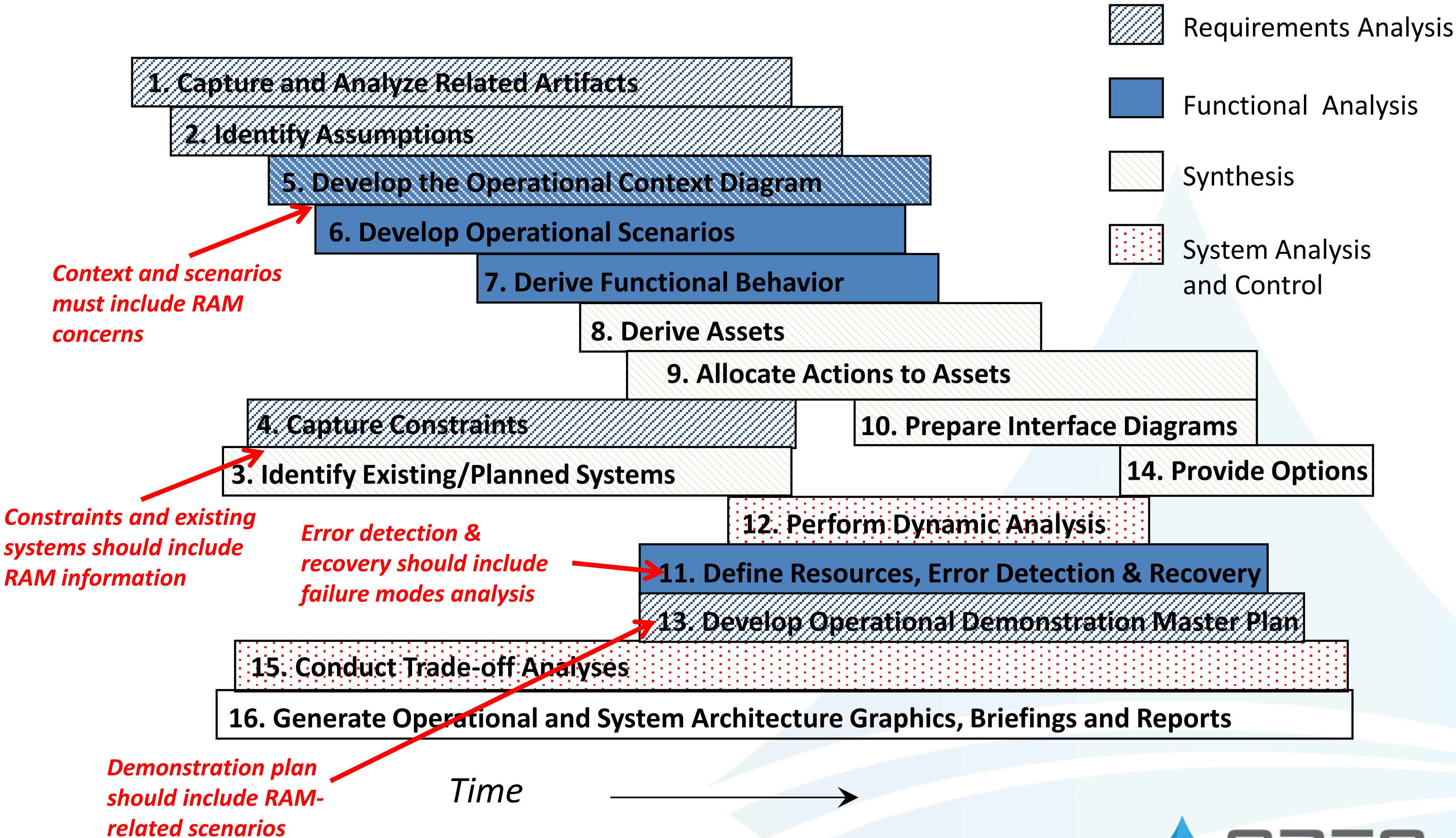
- Monte Carlo simulation shows notional failure distribution for mission
- Realistic probability can now be used to assess the potential impacts of these failure on the systems

What Processes and Tools Work with LML

- We use a "middle-out" process that begins with functional analysis (scenarios) and derives the functional and performance requirements via simulation
- Tools require both discrete event and Monte Carlo simulations of the LML Action Diagram

SPEC INNOVATIONS

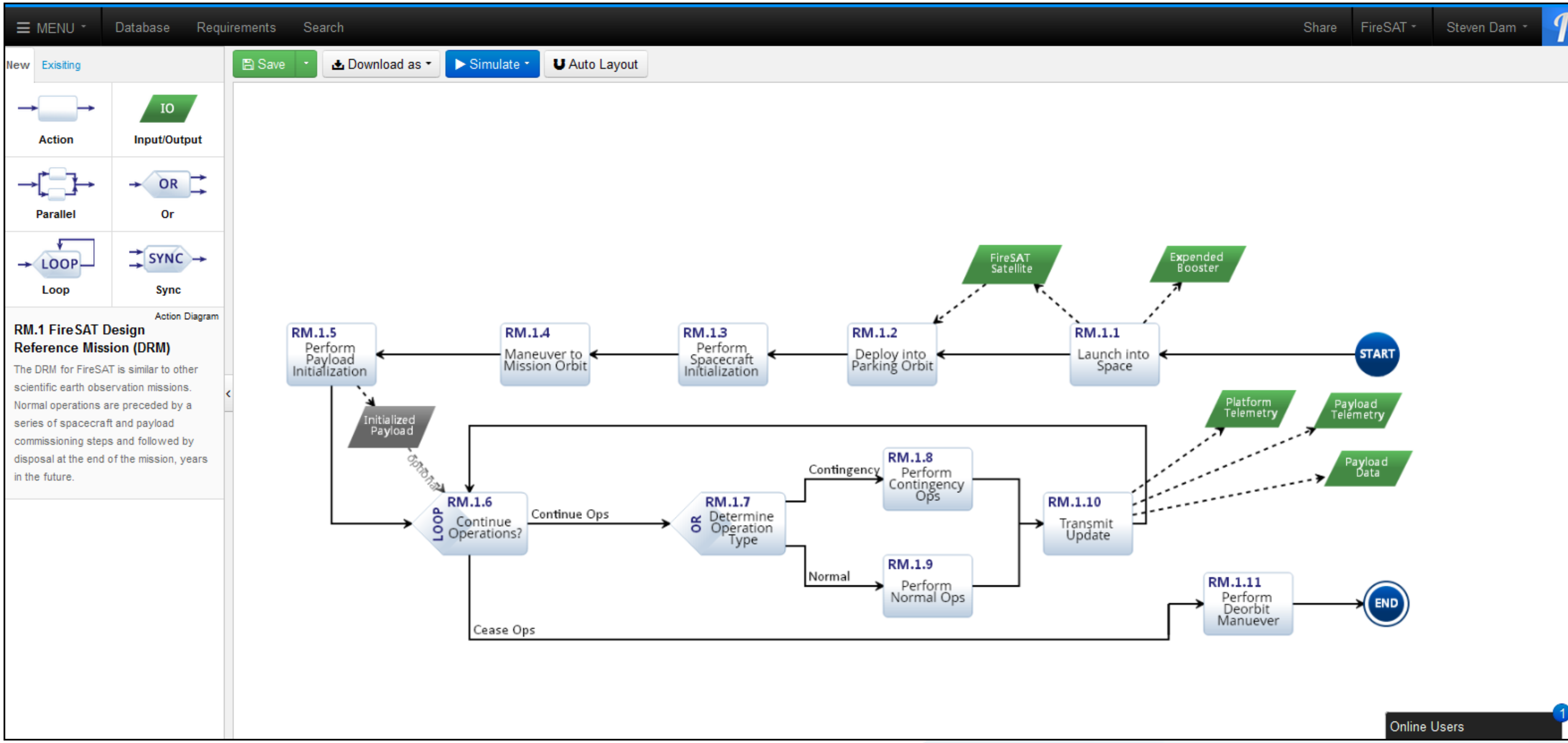
"Middle-Out" Process



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

SPEC INNOVATIONS

Action Modeling with Innoslate



Action Diagrams for functional modeling can be simulated using discrete event and Monte Carlo techniques

Summary

- LML provides the necessary language entities to capture the RAM-related information
- The accompanying tool must implement the language and have the capability to extend it to meet any specific needs
- The process used should emphasize all the "ilities" including RAM