Lifecycle Modeling Language (LML) and Systems of Systems (SoS)

Presented to
Systems of Systems Engineering Collaborators Info Exchange

by

Steven H. Dam, Ph.D., ESEP

May 19, 2015

LML

SPEC INNOVATIONS

Overview

- Why a New Language?
- Lifecycle Modeling Language Overview
- How can LML support SoS?

Why a New Language?

Complex Systems Implications for Systems Engineering

Complexity has been identified by many as a critical problem facing system engineers

- Larger and more complex systems (including systems of systems) development creates a need for:
  - Larger and more distributed teams
  - A clear concise way to express the system design (clear and logically consistent semantics)
  - New tools to enable collaboration across the entire lifecycle

Complexity

- With the growth of the Internet and daily changes in IT, systems have become more complex and change more rapidly than ever before
- Cloud computing gives us new tools to deal with these larger systems
- Systems engineering methods have not kept up with these changes
- SE has been relegated to the beginning of the lifecycle

Complexity is a Major Issue

- Integration of systems create a major problem with complexity
  - Within in a system, interactions grow as N squared or worse
  - Ability to understand and test becomes less certain
  - As more systems are added, the interfaces grow in a non-linear fashion
  - Many of the existing systems are old and not built for these interfaces
  - Conflicting or missing interface standards make it hard to define interface interactions
  - Hardware and software may be re-purposed and "heritage" compromised
  - Future systems will be integrated from multi-organizational, multi-national contributions, adding additional layers of complexity.
- Systems engineering must deal with this complexity
  - End-to-end systems engineering is needed, including "reengineering" of old systems
  - Robust M&S, verification and validation testing are A must

Systems engineering must deal with this complexity
- End-to-end systems engineering is needed, including "reengineering" of old systems
- Robust M&S, verification and validation testing are A must

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

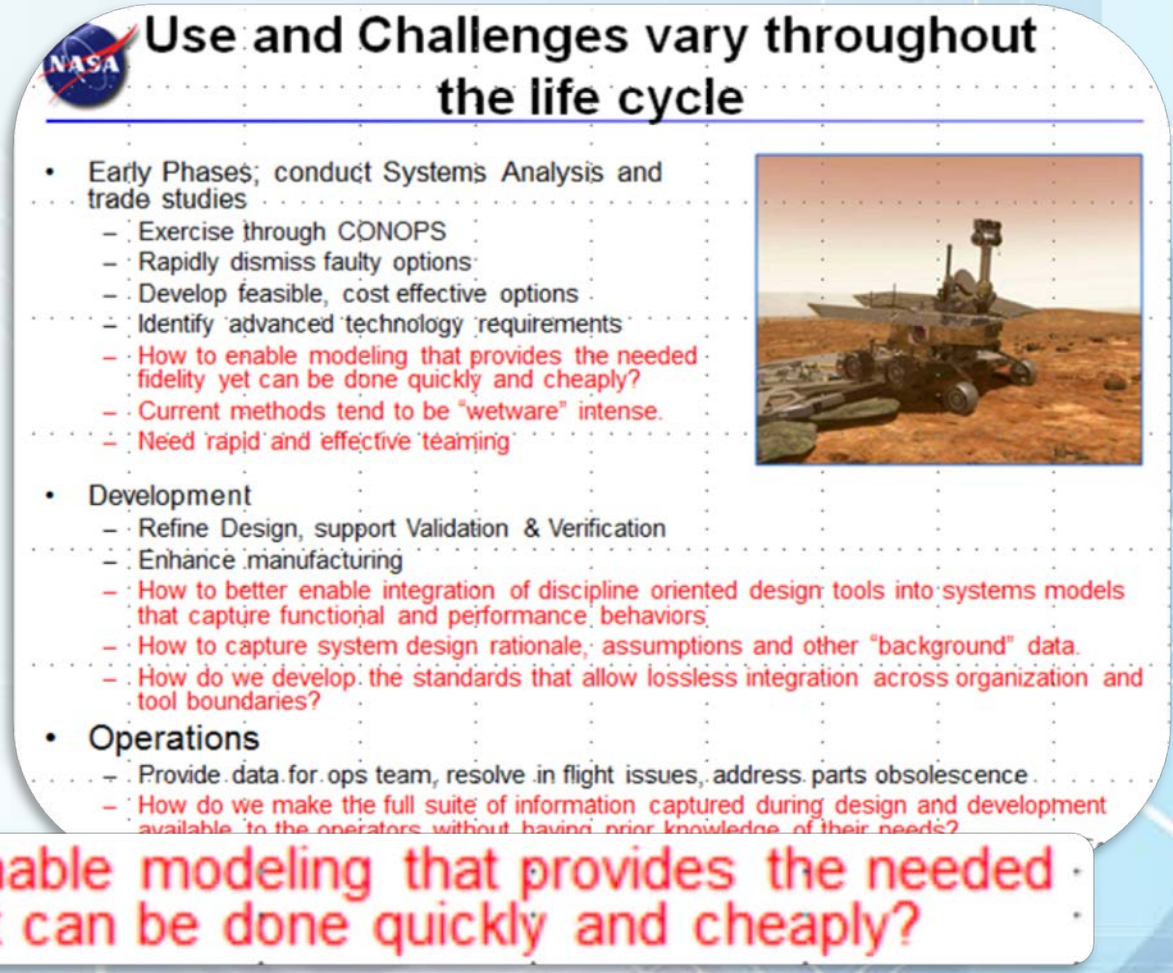
How Does SE Typically Respond to Complexity

- Focus on “architecture”
- More complex languages
- More complex procedures
- More layers of abstraction
  - “Systems of Systems”
  - “Family of Systems”
  - “Portfolio Management”
  - “Capability Views”
- 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

How can we simplify things enable "quicker/ cheaper?" Start with the language we use.



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

State of Current “Language”

- 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)
  - Many software languages are hybrid object and functional

Popular Software Languages

Position Mar 2012	Position Mar 2011	Programming Language	Ratings Mar 2012	Delta Mar 2011	Functional/ Object/Hybrid
1	1	Java	17.110%	-2.60%	Object
2	2	C	17.087%	+1.82%	Functional
3	4	C#	8.244%	+1.03%	Hybrid
4	3	C++	8.047%	-0.71%	Hybrid
5	8	Objective-C	7.737%	+4.22%	Object
6	5	PHP	5.555%	-1.01%	Hybrid
7	7	(Visual) Basic	4.369%	-0.34%	Hybrid
8	10	JavaScript	3.386%	+1.52%	Functional
9	6	Python	3.291%	-2.45%	Hybrid
10	9	Perl	2.703%	+0.73%	Hybrid

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

- In preparing for the cloud computing world of SE we:
  - 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

A language to simplify system design description for the cloud

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

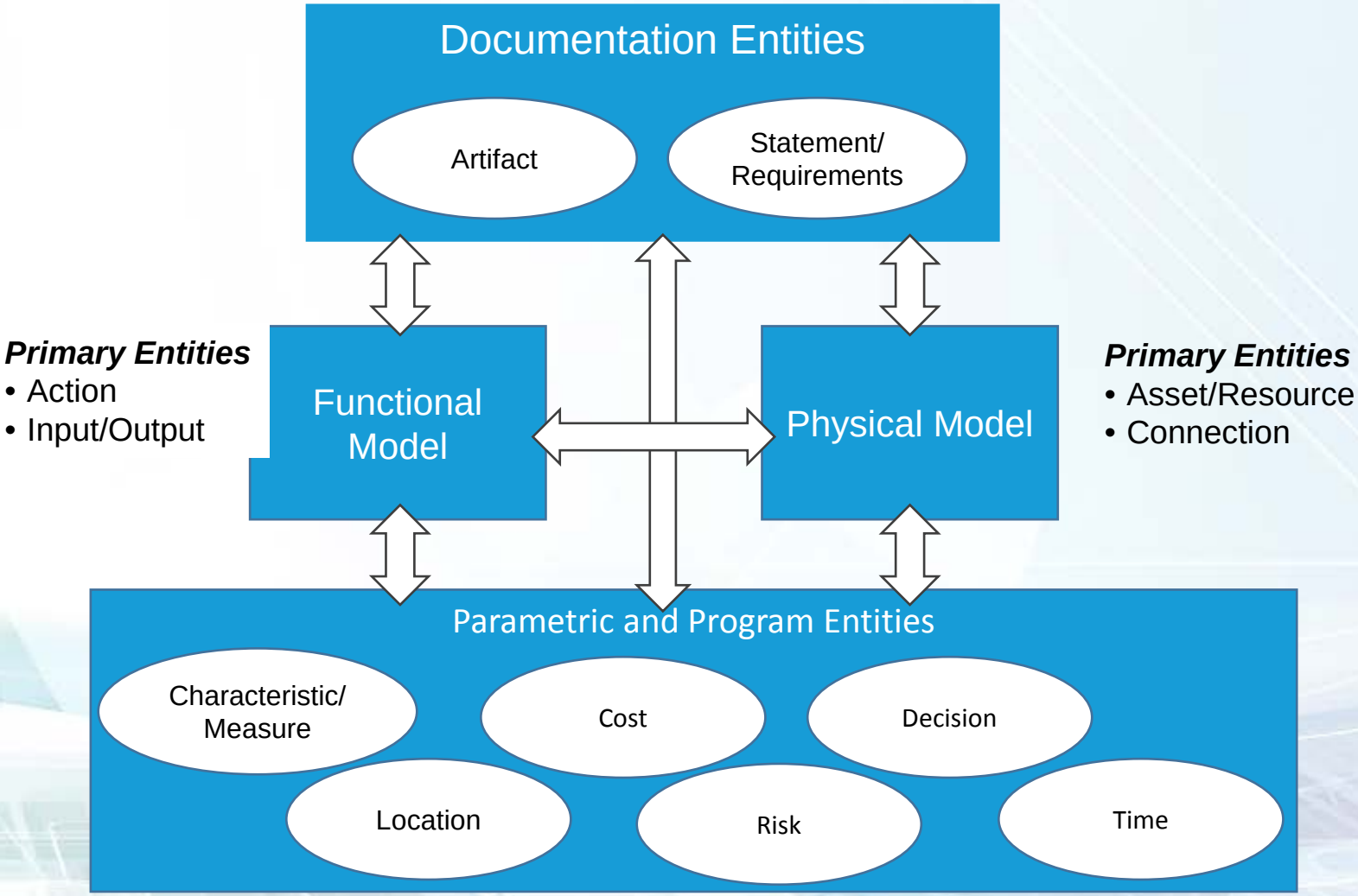
*Ontology = Taxonomy + relationships among terms and concepts
** Taxonomy = Collection of standardized, defined terms or concepts

LML's Simplified Schema

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

Supports capturing information throughout the lifecycle

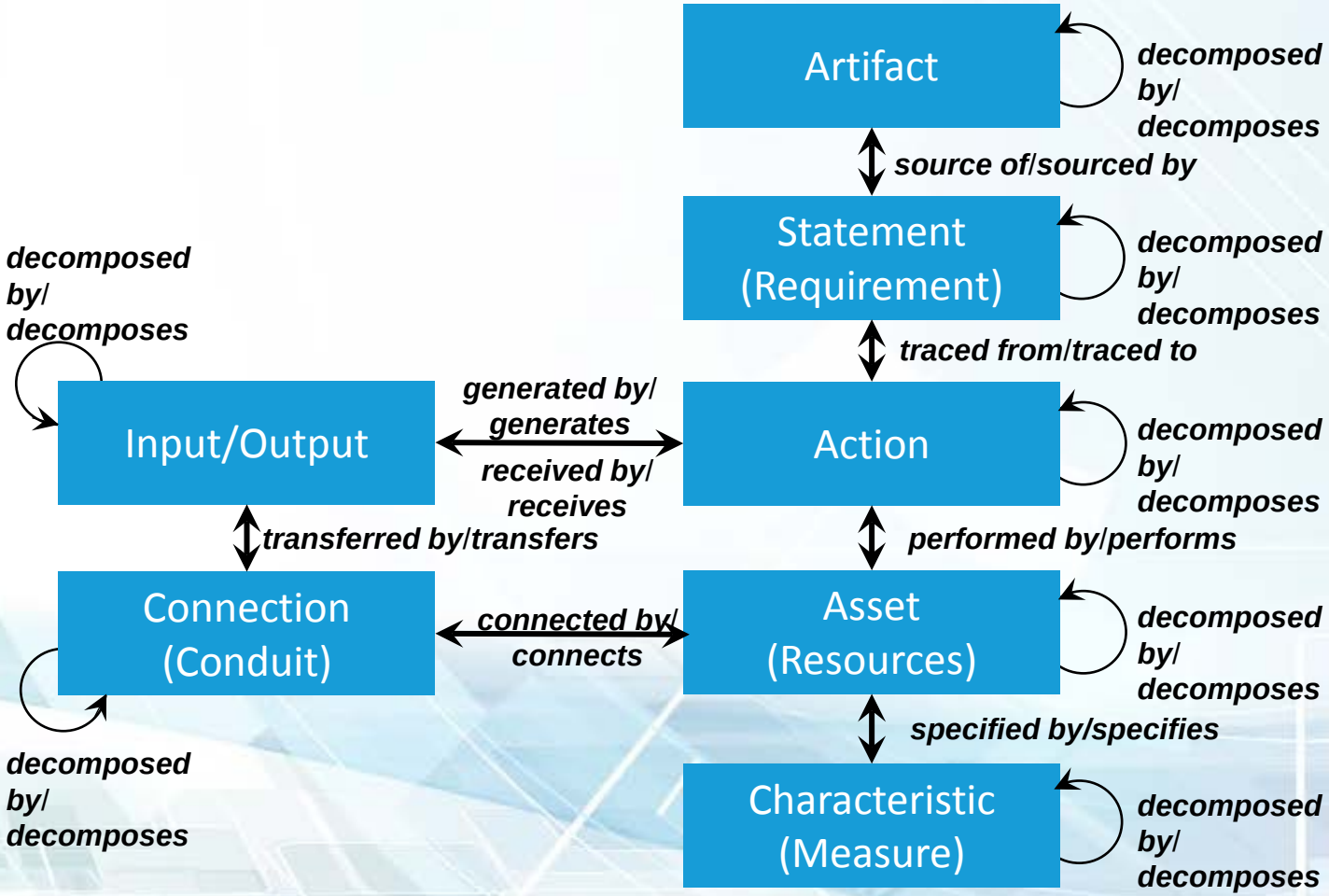
LML Models



Primary Entities
- Action
- Input/Output

Primary Entities
- Asset/Resource
- Connection

LML Primary Entities and Relationships for DoDAF Support



LML Relationships Provide Linkage Needed Between the Classes

	Action	Artifact	Asset (Resource)	Characteristic (Measure)	Connection (Conduit, Logical)	Cost	Decision	Input/Output	Location (Orbital, Physical, Virtual)	Risk	Statement (Requirement)	Time
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*	[illegible] 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)	refer[illegible]ces	decomposed by* orbited by* related to*	sp[illegible]ified by	connected by	incurs	enables made responds to results in	-	located at	causes mitigates resolves	(satisfies) traced from (verifies)	occurs
Characteristic (Measure)	specifies	reference specifies	specifies	d[illegible]mposed 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 (satisfied by) sourced by traced to (verified by)	(satisfied by) traced to (verified by)	(satisfied by) specified by traced to (verified by)	(satisfied by) traced to (verified by)	incurs (satisfied by) traced to (verified by)	alternative of enables traced to results in	(satisfied by) traced to (verified by)	located at (satisfied by) traced to (verified by)	causes mitigates resolves	decomposed by* traced to* related to*	occurs (satisfied 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*

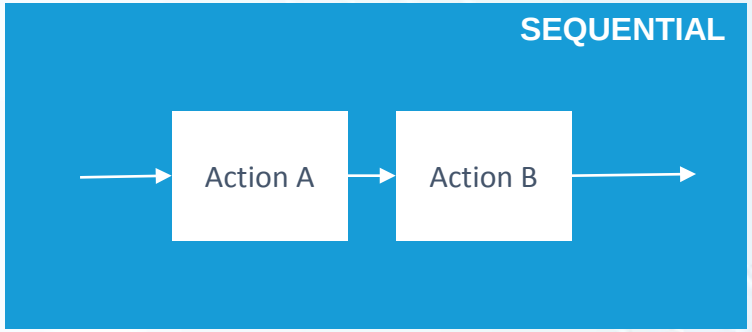
decomposed by*
orbited by*
related to*

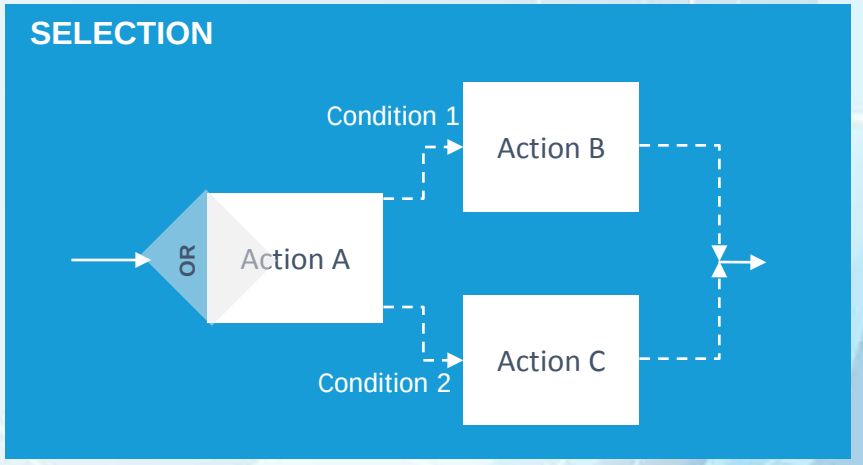
Diagrams Are Needed for Every Class

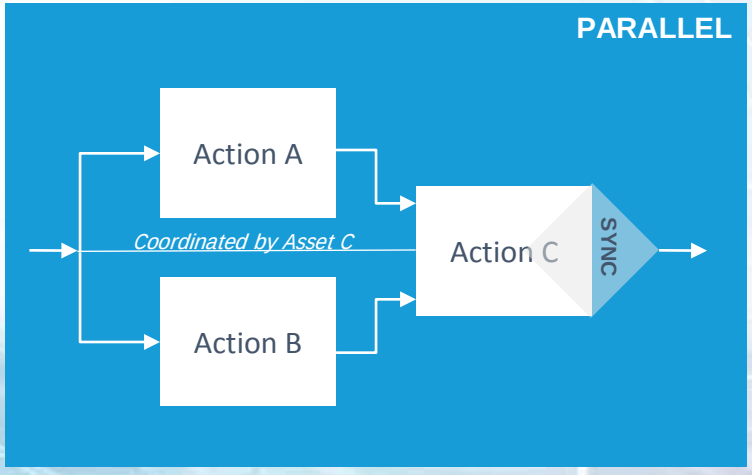
- Action Diagram (Mandatory)
- Asset Diagram (Mandatory)
- Spider Diagram (Mandatory)
- Interface Diagrams
  - N2 (Assets or Actions)
- Hierarchy Diagrams
  - Automatically color coded by class
- Time Diagrams
  - Gantt Charts
  - Timeline Diagram
- Location Diagrams
  - Maps for Earth
  - Orbital charts
- 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
- Combined Physical and Functional Diagram

Action Diagram (Mandatory)

No constructs – only special types of Actions – ones that enable the modeling of command and control/information assurance to capture the critical decisions in your model







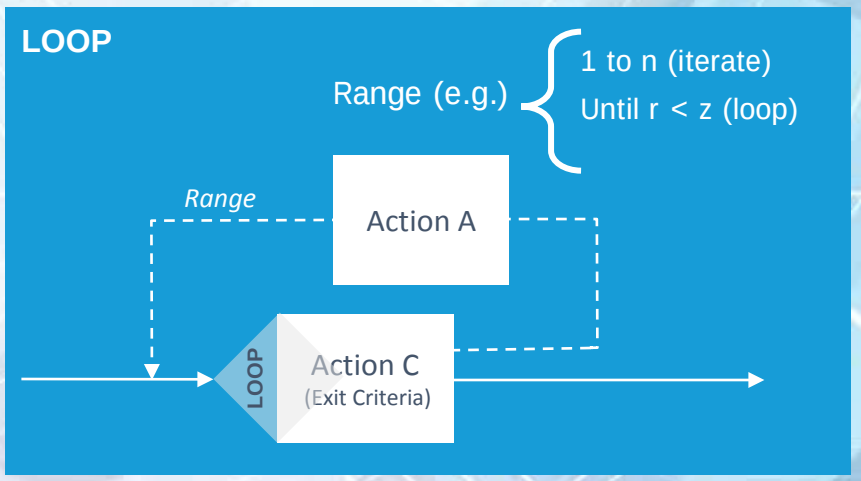


Diagram Comparison: SYSML

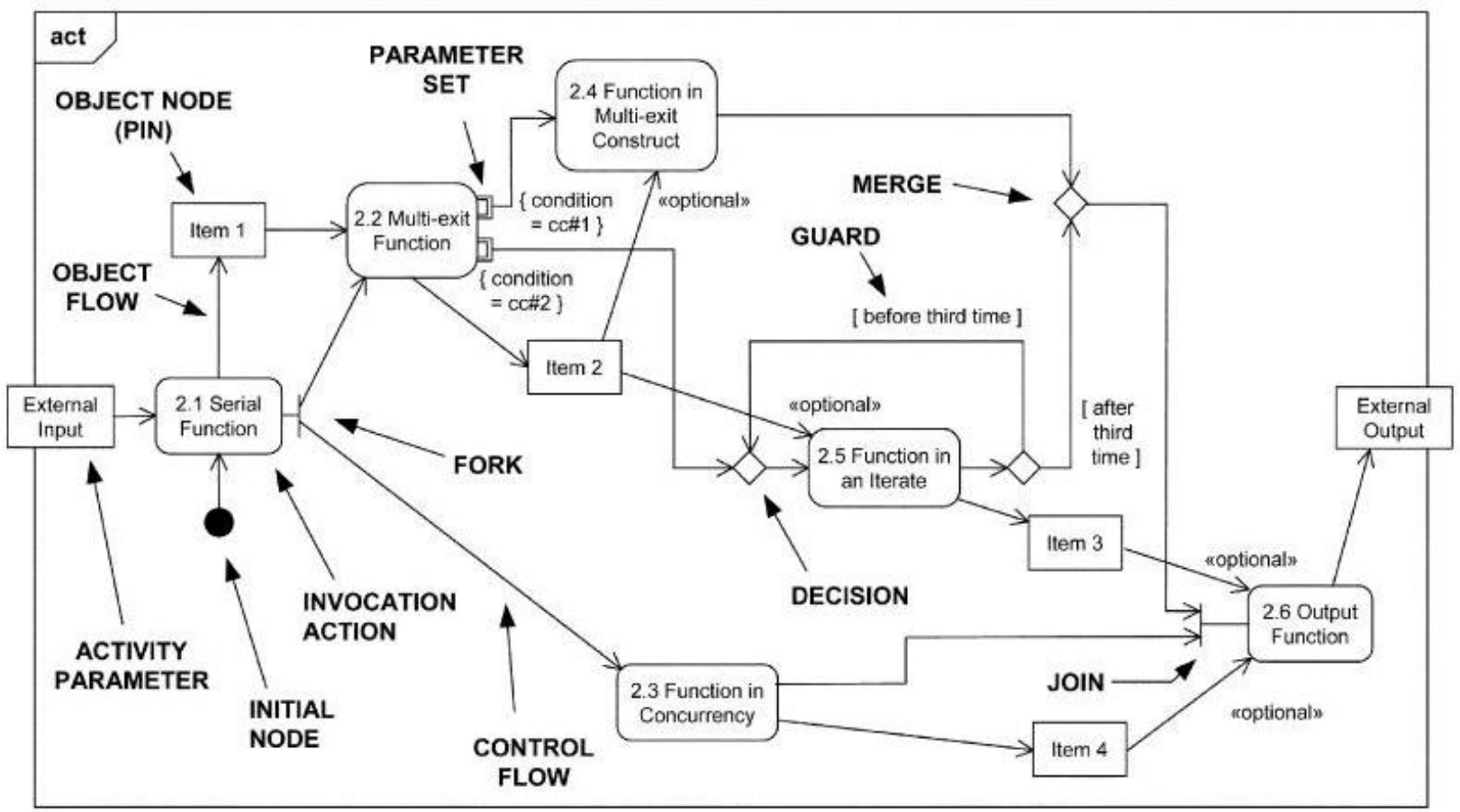
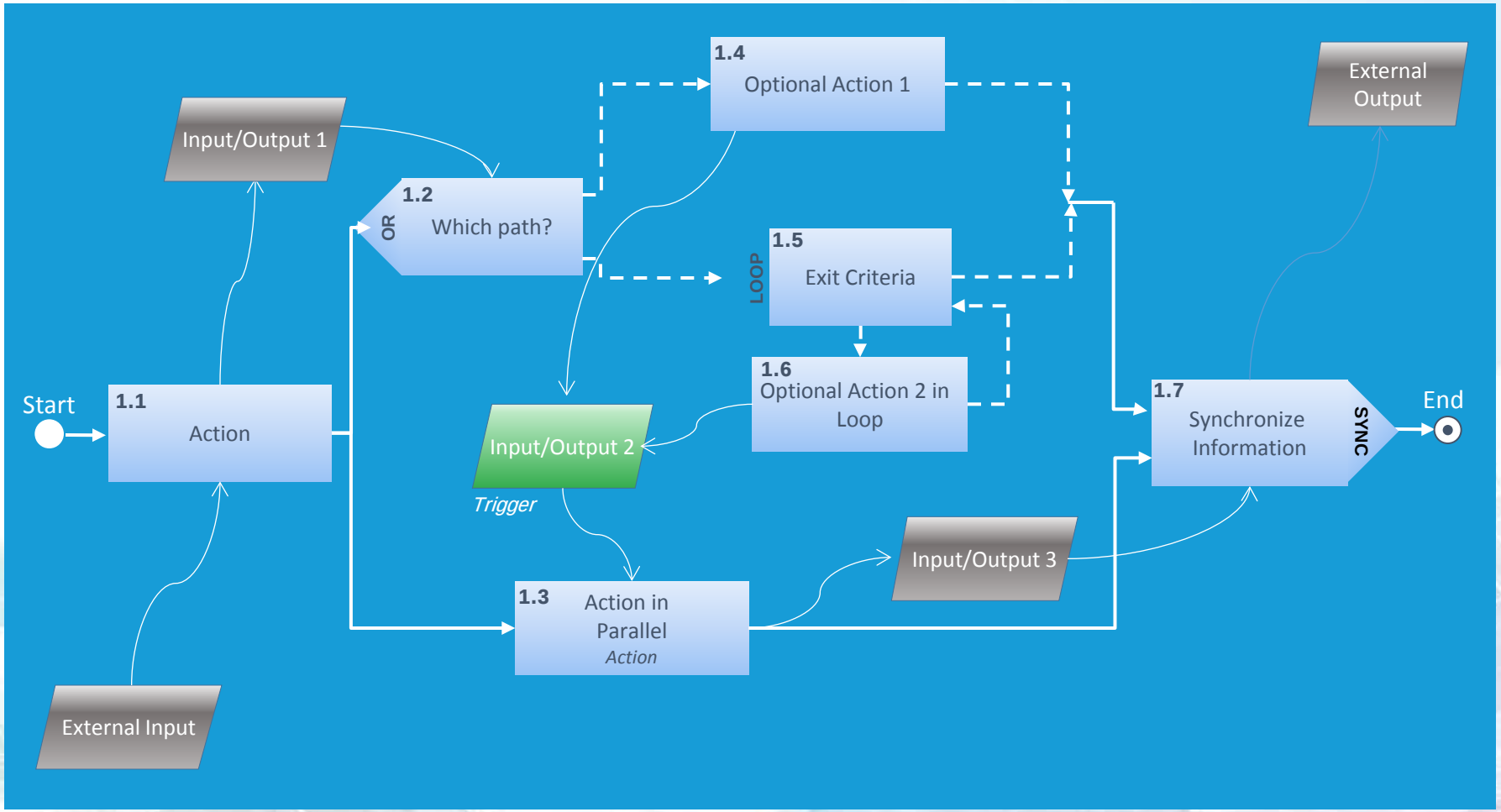


Figure 9. UML 2 Activity diagram corresponding to Figure 8.

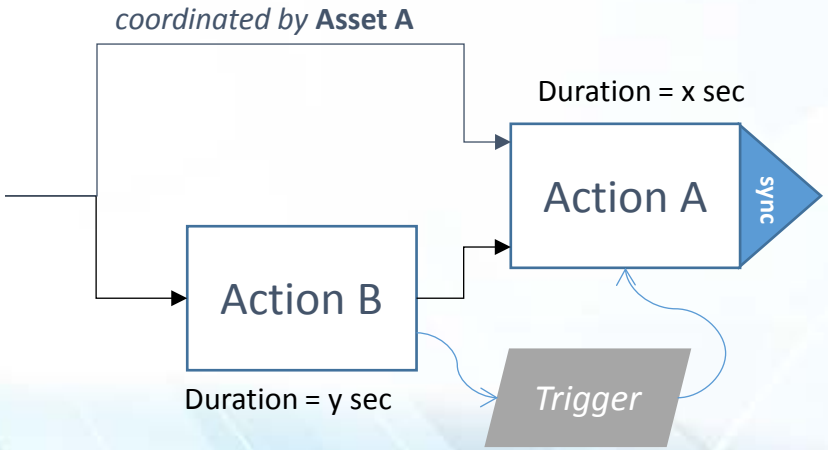
Source: http://www.mel.nist.gov/msidlibrary/doc/sysmlactivity.pdf

LML Action Diagram Captures Functional and Data Flow

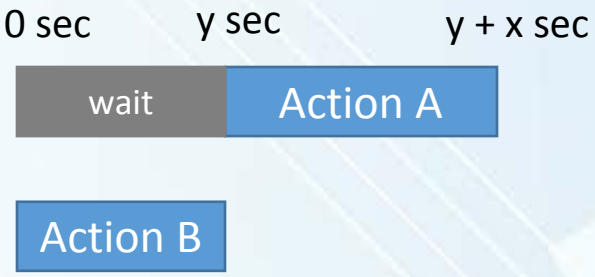


Execution Logic – Concurrency With Trigger; No Coordination Action

Action Diagram



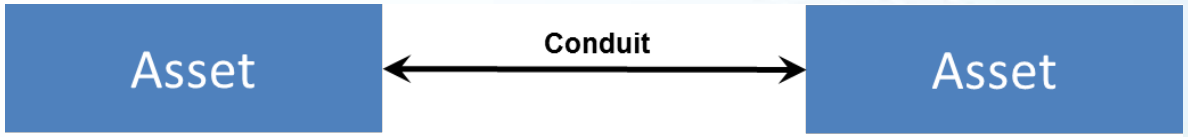
Timeline



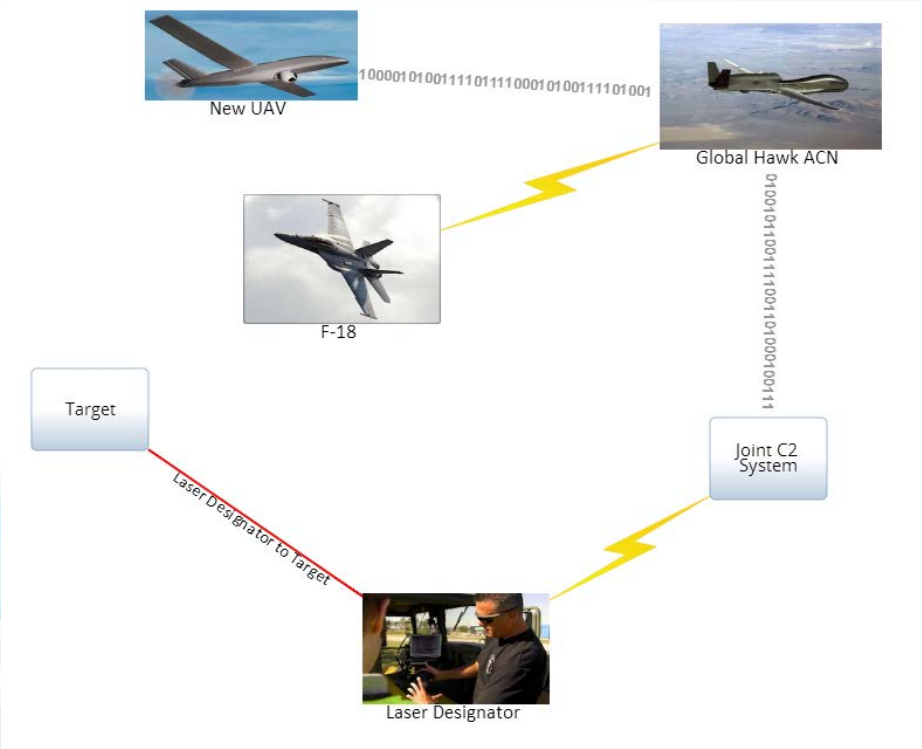
Trigger: Action A enabled, but must wait to execute; Asset A performs Action A

Finish to Start (FS) between B and A

Asset Diagram (mandatory)

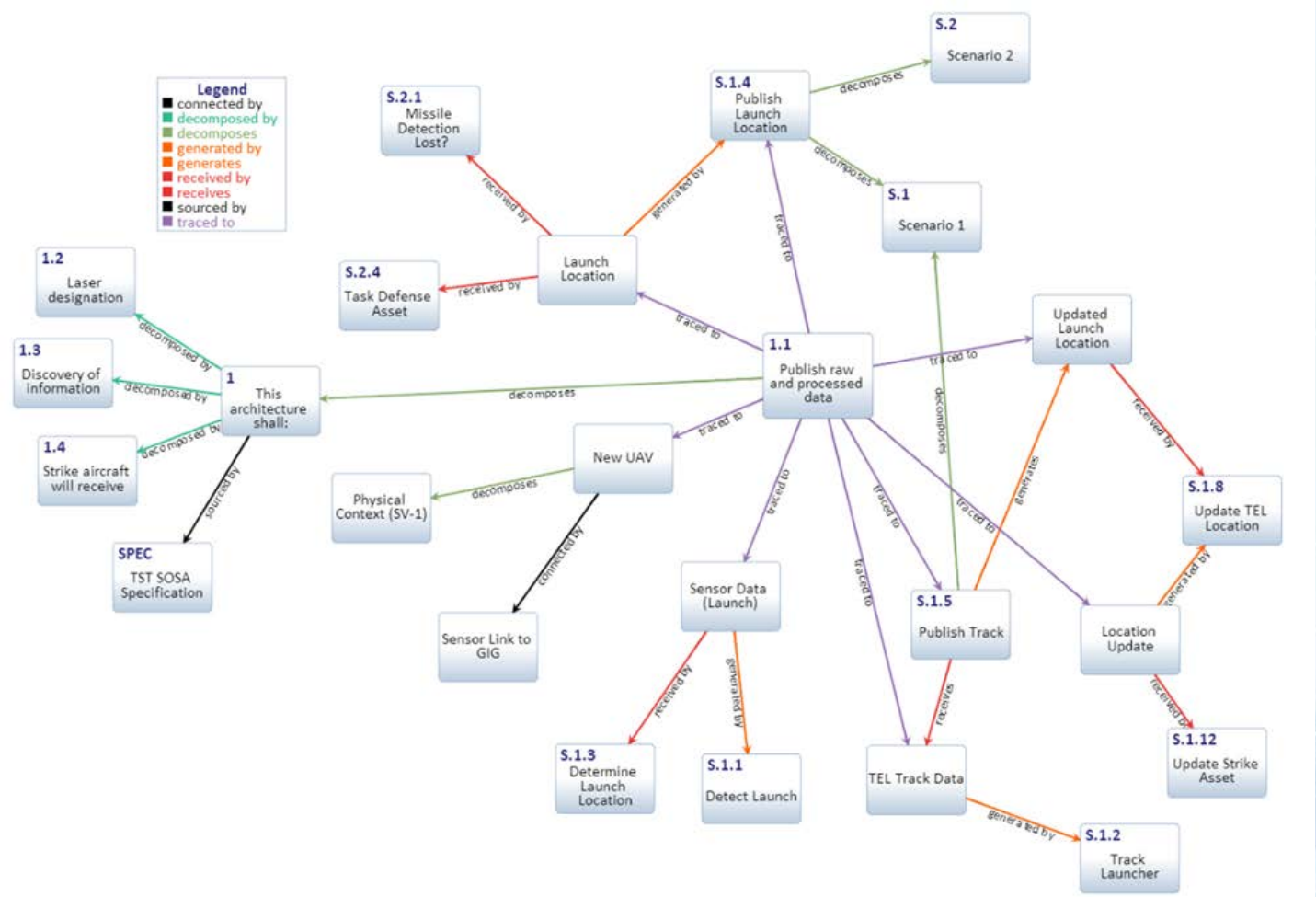


Block diagram general form



Block diagram using pictures

Spider Diagram (Mandatory for Traceability)



Shows entities and relationships in visual form

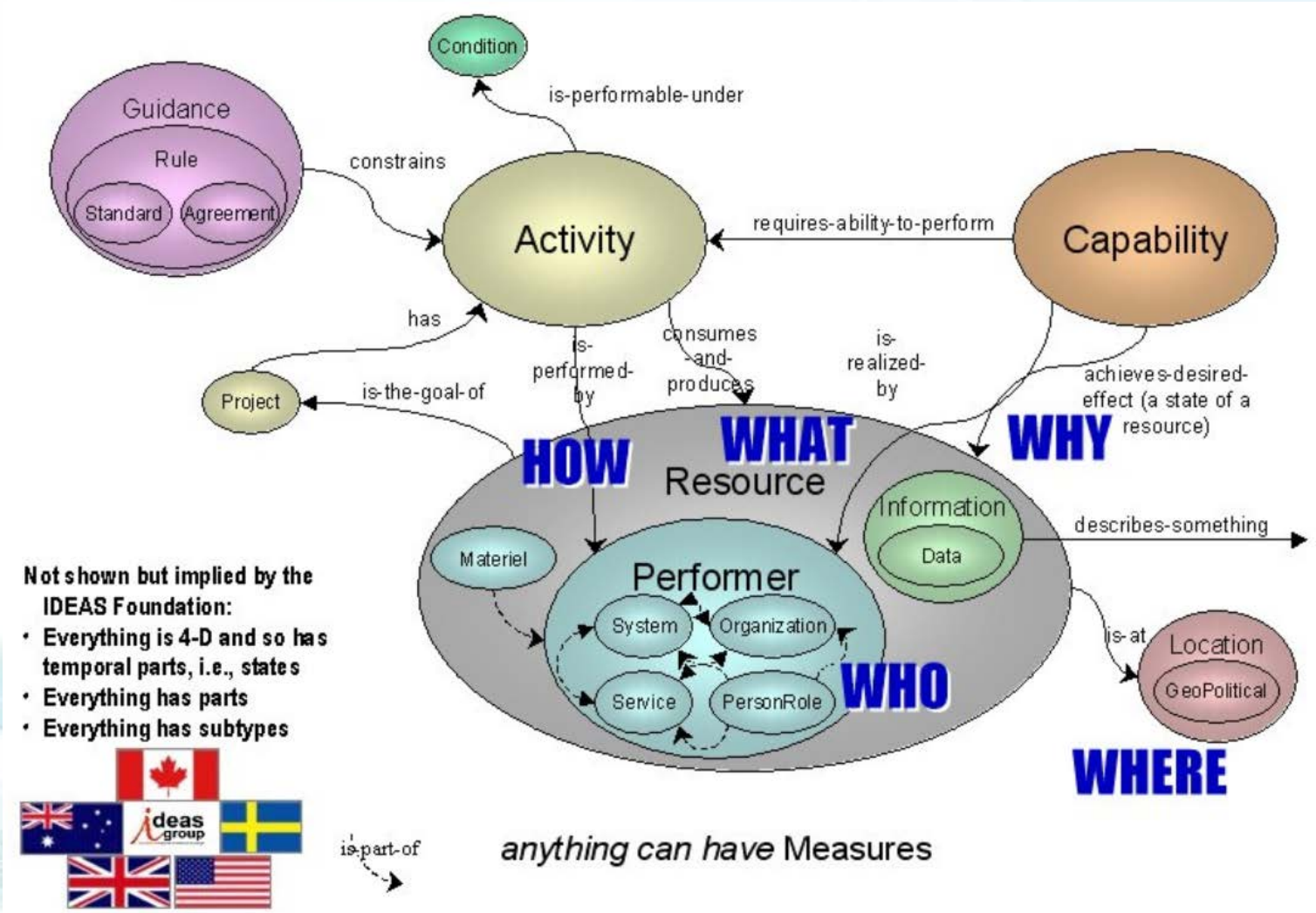
LML Translation

- 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)
  - Enable diagram translations (e.g., Action Diagram to IDEF 0)

LML Class	DM2	SysML	…
Action	Activity	Activity	
Asset	Performer	Actor	

LML Symbol	Electrical Engineering	BPMN	…
			

Example: Translation to DM2



DM2 Conceptual Model to LML Schema Mapping

DM2 Schema Element (Conceptual)	LML Equivalent
Activity	Action
Capability	Action with “Capability” type
Condition	Characteristic with “Condition” type
Information/Data	Input/Output
Desired Effect	Statement with “Desired Effect” type
Guidance	Statement with “Guidance” type
Measure	Measure
Measure Type	Measure Type
Location	Location
Project	Action with “Project” type
Resource	Asset with types for “Materiel,” “Organization,” etc.
Skill	Characteristic with “Skill” type
Vision	Statement with “Vision” type

How can LML support SoS?

Systems of Systems

- Definition*: “An SoS is defined as a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities”
- SoS’s can be
  - Virtual (lack central management and purpose)
  - Collaborative (voluntary interaction)
  - Acknowledged (independent, with higher level coordination)
  - Directed (integrated)
- The common denominator in all the SoS types: systems are dependent on other systems

*From Systems Engineering Guide for Systems of Systems

How do we capture and manage dependencies?

- First, we need to identify the relationship between the different systems
  - LML provides a set of relationships between Assets and Actions (Programs) that can capture this traceability
    - Assets are ***related to*** other Assets
    - Actions are ***related to*** other Actions
    - Assets ***perform*** Actions
  - Note the "***related to/relates***" relationships have an attribute for context
    - This attribute enables you to identify which Asset is dependent on the other
  - If the you want to add another relationship between Assets and/or Actions, LML encourages extensions for specific domains (e.g., DoDAF & SysML)

What Next?

- We want to establish the relationship between the different system schedules
  - The Time entity can be used to capture specific milestones, which can then also be related to one another using the ***related to/relates*** relationship
  - Visualization suggested is a timeline chart
    - Timeline charts can be created and compared
  - LML also provides another mechanism for time – the *duration* and *start* attributes for Action entities provide a means to capture tasks and milestones as part of the program process model
- Note that the language provides more than one way to capture and express the necessary information giving the analyst some flexibility to communicate to a broad audience

Capture other program information

- LML's ontology provides a means to capture other program information, such as Artifacts, Statement/Requirements, Input/Outputs (e.g., deliverables), Risks, Decisions, Location, and Costs
- This information can be related to each other within and between programs
- Critical information (dependencies) between the programs can also be related to each other

By capturing all the relevant program information in one place, it is easier to identify potential areas of concern and resolve them before they become problems

LML Summary

- LML provides a ontological foundation for supporting SoS SE
- 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)

For more information

- See the LML specification at www.lifecyclemodeling.org
- For implementation see www.innoslate.com
- Contact Steve Dam at www.specinnovations.com or sdam@specinno.com