Lifecycle Modeling Language (LML) Version 2.0

PDF Document URL: https://www.lifecyclemodeling.org/specification

Digital Version Document URL: https://cloud.innoslate.com/lmo/p/35/dashboard

April 23, 2025

Table of Contents

1. Specification Information *2*

1.1 Purpose of this Specification **3**

1.2 LML Steering Committee **3**

1.3 Documentation Conventions & Terminology **3**

1.4 Changes from Version 1.4 to Version 2.0 ..... **3**

2 ERA Fundamentals *4*

2.1 Entity Classes (noun) **4**

2.2 Entity Class Attribute (adjective) **4**

2.3 Entity Class Relationships (verb) **5**

2.4 Attributes on Relationships (adverb) **5**

2.5 Attribute Data Types **6**

2.5.1 Big_Text 6
2.5.2 Boolean 6
2.5.3 Computable 6
2.5.4 DateTime 6
2.5.5 Duration 6
2.5.6 Enumeration 6
2.5.7 Equation 7
2.5.8 File 7
2.5.9 GeoPoint 7
2.5.10 HTML 7
2.5.11 Multiplicity 7
2.5.12 Multiselect 7
2.5.13 Number 7
2.5.14 Percent 7
2.5.15 Quality 7
2.5.16 Text 7
2.5.17 URI 7
2.5.18 User_Team 8

2.6 Class Inheritance **8**

2.7 Extensions **8**

2.8 Implementation **8**

3 LML Semantic Ontology *9*

3.1 LML Classes **10**

3.2 LML Relationships **12**

3.3 Traceability **14**

3.4 Class Specifications 14
3.4.0.1 Common Attributes 14
3.4.0.2 Common Relationships 15
3.4.1 Action 17
3.4.2 Artifact 22
3.4.3 Asset 23
3.4.4 Characteristic 25
3.4.5 Connection (Abstract Class) 27
3.4.6 Cost 30
3.4.7 Decision 31
3.4.8 Input/Output 33
3.4.9 Location (Abstract Class) 34
3.4.10 Risk 38
3.4.11 Statement 39
3.4.12 Time 41

4 Visualizations 42

4.1 Action Diagram (Mandatory for Action entities with children) 43

4.2 Asset Diagram (Mandatory for Asset entities with children) 45

4.3 Spider Diagram (Mandatory for Traceability) 46

4.4 Interface Control Diagram (Mandatory) 47

4.5 Example Views for Other LML Entities 49
4.5.1 Class Diagram 49
4.5.2 ERAA Diagram 49
4.5.3 Timelines 50
4.5.4 Hierarchy Diagram 51
4.5.5 Risk Matrix 52
4.5.6 State Machine Diagram 53

Appendix A. SysML v1.X Mapping to LML 55

Appendix B. DoDAF MetaModel 2.0.1 (DM2) Mapping to LML 61

Appendix C. UAF Diagram Framework 63

Appendix D. Structuring Artifacts 64

Appendix E. Application Programming Interfaces 66

LML specification 2.0

The purpose of LML specification is to convey the foundational structure of the language and the specific entity classes, entity class relationships, and entity class and relationship attributes that are available to the language user. LML version 2.0 was developed to take advantage of the evolution of the language that has occurred during the version 1.4 releases of the specification. In particular, a number of extensions to the base language have been proposed and proven over time. The entity class extensions (trial specifications) for Verification Validation (VV), Program Management, and Interfaces have demonstrated value to enhance users Data-Driven Systems Engineering (DDSE) capability so those entity class extension specifications have been promoted from the LML Appendices into the main body of the LML Specification. These changes have resulted in three (3) new entity subclasses, a new type of Artifact (Test Suite), and a new diagram type (Interface Control Diagram).

1. Specification Information

Current Systems Engineering modeling languages tend to add complexity to already complex problems, thus making it more difficult to communicate. LML was designed as a simpler language, both in its semantic ontology and visual expressions. This feature makes it easy to understand by all stakeholders throughout the lifecycle. Such a simplified language may not include every “bin” of information needed for a particular domain, which is why LML is extendable by the practitioner.

As Information Technology continues to evolve, systems have become more complex and change more rapidly than ever before. When coupled with ever tightening budgets and schedules the future becomes even more challenging for the modern system engineer. Development of new systems requires:

- plans that are nimble and responsive to change,
- designs that are easy to understand for all stakeholders,
- architectures that are easy to modify,
- processes that support all stages of the system lifecycle,
- provides an actionable interface into artificial intelligence (AI).

Systems engineering approaches, methods and tools have evolved, but they have not kept pace with the rate of change in modern systems. Model Based Systems Engineering (MBSE) has made tremendous steps forward but does not address the entire challenge. Large portions of the development lifecycle are ignored by the languages in current use. A new approach to analyzing, planning, specifying, designing, building and maintaining modern systems is needed. The Lifecycle Modeling Language (LML) is that approach.

LML was designed with six major goals:

- To be easy to understand
- To be easy to extend
- To support both functional and object-oriented approaches within the same design
- To be a language that can be understood by most system stakeholders, not just Systems Engineers
- To support systems from cradle to grave
- To support both evolutionary and revolutionary changes to system plans and designs over a system’s lifecycle.

Most systems engineers recognize that MBSE’s ability to evolve, reuse and execute models is a significant improvement over the "document-based" static view of a system. Good models can bridge the gap between written requirements and bending metal or writing code, the thing that is desired versus the thing that is delivered.

LML takes the principles of MBSE beyond development and production and into the conceptual, utilization, support and retirement stages. It provides a robust, easy to understand ontology that allows you to model complex interrelationships between system components and programmatic artifacts, as well as express system information using easy to understand diagrams. “LML was designed to integrate all lifecycle disciplines, such as program management, systems engineering, testing, deployment and maintenance, into a single language. As a result, LML can be used throughout the lifecycle. LML's entity,

relationship, and attribute modeling language elements draw from words common to the SE and PM disciplines. Its primary modeling constructs are the box, a modeled element (entity instance), and the directed arrow which represents a relationship between modeled elements. This means that everyone from the least technical stakeholder to most highly skilled end users can model and understand systems using LML It enables easy communication between disparate disciplines across multiple industries.

1.1. Purpose of this Specification

The purpose of LML specification is to convey the foundational structure of the language and the specific entity classes, entity class relationships, and entity class and relationship attributes that are available to the language user.

1.2. LML Steering Committee

The direction and evolution of this standard is overseen by the LML Steering Committee, which is a committee under the Lifecycle Modeling Organization (LMO). It consists of expert systems engineers and program managers from industry, government, and academia. Their goal is to ensure LML evolves in such a way that it continues to meet the needs of its users. You may provide comments and suggestions on the LML website (www.lifecyclemodeling.org).

1.3. Documentation Conventions & Terminology

LML is a set of information based on the classic Entity-Relationship-Attribute (ERA) model. An entity is a distinct class of information. An entity instance is an element defined using an entity class. A relationship specifies a possible link between two or more classes of entities. An attribute is used to describe a property of an entity class or a relationship.

When referencing an entity class, outside of the LML specification heading, the class name is always in bold with the first letter capitalized, as in Action.

When referencing an attribute, outside of a LML specification heading, the attribute name is always in italics with the first letter lower case, as in control.

When referencing a relationship, outside of a LML specification heading, the relationship name is always in bold italics with the first letter lower case, as in ***traced to***.

When clarifying the context for an attribute, the entity class name should be appended to the end of the attribute name as follows: "(**Class Name**)"; where "**Class Name**" would be the name of the class of entities. For example: *units* (**Characteristic**) and *units* (**Cost**), which clarifies the difference between the *units* attribute of the **Characteristic** entity and the *units* attribute of the **Cost** Class.

When referencing an attribute on a relationship, outside of a LML specification heading, the attribute name is always underlined with the first letter lower case, as in trigger.

1.4. Changes from Version 1.4 to Version 2.0

LML version 2.0 was developed to take advantage of the evolution of the language that has occurred during the version 1.4 releases of the specification. In particular, a number of extensions to the base language have been proposed and proven over time. The entity class extensions (trial specifications) for Verification & Validation (V&V), Program Management, and Interfaces have demonstrated value to

enhance users Data-Driven Systems Engineering (DDSE) capability so those entity class extension specifications have been promoted from the LML Appendices into the main body of the LML Specification.

These changes have resulted in three (3) new entity subclasses, a new type of Artifact (Test Suite), and a new diagram type (Interface Control Diagram).

2. ERA Fundamentals

LML is an instance of the classic ERA metamodel, with the addition of including attributes on relationships. LML elements correspond to classes (class of entity type), relations (relationship), and properties (attribute) in other object-oriented languages.

This section defines the ERAs for LML, thus providing the basic definitions of the data types used to collect information about the system[1]. Furthermore, we describe how inheritance, extensions, limitations and instantiation can be used by tool developers to remain within the guidelines of this standard.

Entity, relationship and attribute have equivalent English language elements: noun, verb, and adjective. With the addition of attributes on the relationship, we also have the equivalent of the adverb. These equivalencies are provided to help explain the semantics of the language.

2.1. Entity Classes (noun)

LML defines 12 parent classes (**Action**, **Artifact**, **Asset**, **Characteristic**, **Connection**, **Cost**, **Decision**, **Input/Output**, **Location**, **Risk**, **Statement** and **Time**). The rationale for this set is presented in Section 3. Several child classes have also been defined as they have specific utility in capturing the information needed by the system lifecycle stakeholders. These child classes inherit the attributes and relationships from their parents, and possess additional attributes and/or relationships that make them unique. More on this "inheritance" of attributes and relationships is discussed in section 2.6.

Every LML entity class has *name, number, and description* attributes. Every LML entity instance must have a name or number populated to identify it. The *name* is a word or small collection of words to represent the entity instance. The *number* provides an alpha-numeric way to identify the entity for the users. The *description* provides more detail about that entity instance.

An optional vendor implementation of LML may choose to define a universally unique identifier (UUID) or other identifiers as a means to *uniquely* identify the entity instance.

2.2. Entity Class Attribute (adjective)

Every entity class possesses a set of attributes that represent its properties. These attributes further describe the class, enhancing its uniqueness. Every attribute must have a name to identify it uniquely within an class. The name is one word or a few words to succinctly identify the attribute. The attribute data type (see 2.5 below) specifies the type of data associated with the attribute.

[1] [1]INCOSE defines a system as a "combination of interacting elements organized to achieve one or more stated purposes."

Attribute names must be unique within a class, but may be used in other classes, such as the following example: *units* (**Characteristic**) and *units* (**Cost**).

2.3. Entity Class Relationships (verb)

A relationship connects entity classes to each other. In LML, all relations shall be defined in both directions and shall have unique names with the same verb. For example, the standard parent child relationship (used by all LML classes) is ***decomposed by*** and its inverse is ***decomposes***. The relationship names were selected to enable an English reading of the way entities connect. For example, when connecting an **Action** to a **Statement,** LML uses ***traced from*** as the relationship: an **Action** is ***traced from*** a **Statement**. The inverse relation of ***traced from*** is ***traced to***, and thus would be read as: a **Statement** is ***traced to*** an **Action**. Figure 2-1 shows an Entity-Relationship Diagram (ERD) that illustrates this example.

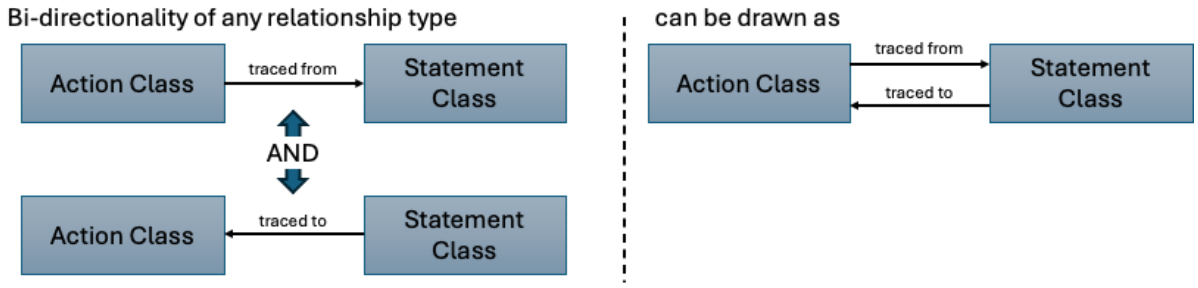


Figure 2-1. Entity-Relationship Diagram for the Relationship Between an Action and a Statement.

For relationships within the same entity, such as ***decomposed by*** and ***decomposes*** can be visualized as an ERD as well (see Figure 2-2).

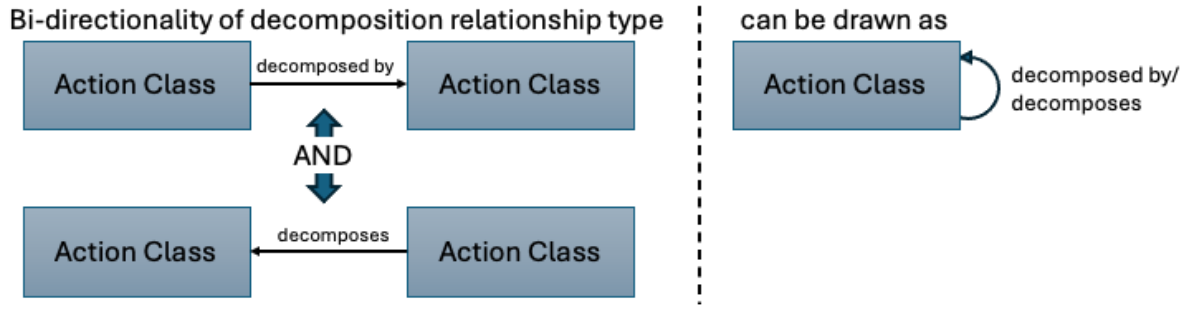


Figure 2-2. Entity-Relationship Diagram for the Relationship Between an Action and Itself.

2.4. Attributes on Relationships (adverb)

The classic ERA modeling does not include attributes on relationships. However, this addition is useful for LML. However, this addition is useful for LML to convey additional information about a relationship instance. It can be used to query certain relationships in their model as well. This addition of the "adverb" provides a more efficient way to express relational information.

Figure 2-3 shows an example of how the attribute on a relationship is depicted in an extended version of the ERD. The attribute on a relationship is shown with a dashed line to the relationship. The attribute on a relationship shall have a unique name for that relationship, but can be used in other relationships, if necessary to enhance communication.

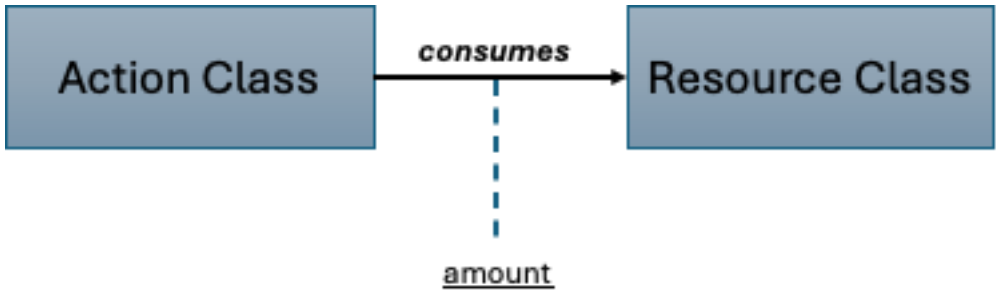


Figure 2-3. Extended Entity Relationship Diagram for the Attribute on the Relationship Between an Action and an Input/Output.

2.5. Attribute Data Types

For a complete specification, defining which data types are appropriate for the attributes is indispensable. This is because they can vary significantly, and because specification interoperability with other schemas (i.e., translations) would be exceptionally difficult without this. The following subsections present the current acceptable set of attribute data types for LML. Extensions (see Section 2.7) may also extend this list of data types.

2.5.1. Big_Text

A string of Rich Text characters up to 65,536 characters in length.

2.5.2. Boolean

Data that can have one of two values, including "true" and "false"

2.5.3. Computable

Data representing a mathematical formula, using a common language such as LaTeX, that can be used to visualize the equation and calculate values.

2.5.4. DateTime

Data representing a date and time value. Commonly stored as “YYYY-MM-dd hh:mm:ss”. Where “YYYY” is the four digit year, “MM” is the two digit month, “dd” is the two digit day, “hh” is the two digit hour (in twenty-four hours), “mm” is the two digit minute, and “ss” is the two digit second.

2.5.5. Duration

Data that is a special case of Number where the value will be assigned units of "seconds", "minutes", "hours", "days", "months" or "years".

2.5.6. Enumeration

Data representing a choice from set of defined options, where the selection of only one option is permitted. An enumeration must contain a minimum of two options. An Enumeration will be implemented as either a Text or a Number data type, depending on the character content of the options. All options within the set must be of the same data type (having a set containing both Text data type and Number data type is not permitted). Also, within a set, a repeated option is not permitted (A set cannot have repeated “Apple” as in: “Apple”, “Orange”, “Apple”, and “Plum”).

2.5.7. Equation

Data representing a mathematical formula, using a common language such as LaTeX, that can be used to visualize the equation in normal mathematical form.

2.5.8. File

Data representing an uploaded file that contains information in other formats, such as .pdf, .docx, .stl, etc.

2.5.9. GeoPoint

Data representing a longitude and latitude pair on the surface of a body.

2.5.10. HTML

A string of ASCII characters where the value may contain HTML tags. The text will be displayed according to the HTML standard. The string may be up to 65,536 characters in length.

2.5.11. Multiplicity

Data representing the potential range of the number of items that can be associated with a specific entity. Integers from 1 to n.

2.5.12. Multiselect

Data representing choices made from set of defined options, where the selection of one or more options is permitted. A multiselect must contain a minimum of two options. A Multiselect will be implemented as either a Text or a Number data type, depending on the character content of the options. All options within the set must be of the same data type (having a set containing both Text data type and Number data type is not permitted). Also, within a set, a repeated option is not permitted (A set cannot have repeated “Apple” as in: “Apple”, “Orange”, “Apple”, and “Plum”).

2.5.13. Number

Data representing any real number. This number can be represented by a distribution as well.

2.5.14. Percent

Data that is a special case of Number where the value is restricted to values between zero and one hundred.

2.5.15. Quality

Data representing the quality (the goodness) of an entity. For example, the quality of a requirement may have a number of factors, such as Appropriate, Complete, Correct, etc.

2.5.16. Text

A string of ASCII characters, such as a single character, a word, or multiple words up to 256 characters in length.

2.5.17. URI

A special case of Text where the value must be a Uniform Resource Identifier. Examples of URI: “C:/Program Files”, “http://www.google.com”, and “test@test.com”.

2.5.18. User_Team

Data representing the names (or usernames) of individuals assigned to a project.

2.6. Class Inheritance

With LML it is possible to create child classes. The child class inherits all the attributes and relationships from the parent class with the EXCEPTION of the class designation, which is overridden by the child class designation. Further, the child class will add additional attributes and/or possible relationship types.

A Resource class is an example of a child class to the Asset class. Providing a metaphor, consider that in the natural world there exists;a class of animal called a "Kangaroo" that inherits attributes from the Mammal class. The "Kangaroo" class is a child class to the Mammal class. The Kangaroo is a mammal with more attributes and added relationship types.

2.7. Extensions

There may be a need to capture other kinds of information than those defined in LML. There might be a need to add attributes and relationships to existing entity classes or you may want to add new entity classes or subclasses to LML. New entity classes or subclasses are recommended only when new attributes and/or relationships are needed to separate the new entity class or subclass from existing entity classes or subclasses. Extensions for domain-specific needs are allowed and encouraged when necessary.

It is strongly recommended that users avoid simply creating a new entity class, when it is really only a "*type*" of an existing entity class. For example, a type of Requirement might be a performance or safety requirement. If no new attributes or relationships are needed for the performance or safety requirement, they should be types of the Requirement. Tool developers may want to display the entity *type*, rather than the entity class *name*, to enhance the communication with these other languages. They may also implement types as labels, instead of attributes.

Note as mentioned in 2.6, *types* are not inherited by subclasses from the parent entity class.

The schema user can also apply a **Characteristic** entity class in many cases to provide "attributes" of an **Asset** or **Action**, thus making many other extensions unnecessary. It is recommended that users explore this option first, before creating an extension. For example, a **Characteristic** of "Color" (the Navy's "paint it haze gray" requirement) could be related to an **Asset**, instead of adding an attribute to the entity class.

If there are broadly valuable changes to the LML ontology, users are encouraged to submit them to the LML Steering Committee for adjudication.

2.8. Implementation

Implementation details of the LML specification in a tool will be up to vendors so long as they maintain compliance with this standard. If they find portions of the specification difficult to implement, they can contact the LML Steering Committee for guidance at info@lifecyclemodeling.org. LML can easily be accommodated in existing tools that have an open or tailorable schema. The unique diagram types: Action Diagram and Interface Control Diagram (see Section 4) could be added to existing tools.

3. LML Semantic Ontology

Systems engineers, enterprise architects and program managers have overlapping needs for information and LML provides a basic but comprehensive set of design elements that satisfy all of them. For example, the systems engineer is concerned with optimizing cost, schedule and performance. Performance includes form, fit, and function. Enterprise architects often use the "5WH" model (What, Why, When, Where, Who, and How) to capture their information. The Program Manager is primarily concerned with cost, schedule, tasks, resources, and risks. Table 3-1 shows these various information needs and how LML satisfies them. Each of these entities has unique and common attributes and relationships.

Table 3-1. Comparison Between LML and the Information Needs of Key System Lifecycle Stakeholders

Systems Engineering	Architecture	Program Management	Lifecycle Modeling Language
Cost	(How Much)	Cost	Cost
Schedule	When	Schedule	Time/Action
Performance			
Form	Who	Organization	Asset
	What	Resource	Resource
	Where	Location	Location
	Why	Goal, Objective & Decision	Decision & Statement/Requirement
Function	How	Task	Action
Metric (Fit)		Metric	Characteristic/Measure
Interface			Connection (Conduit) & Input/Output
Risk		Risk	Risk
		Artifact	Artifact

This section summarizes the LML Specification's entity classes in table format, entity class relationships in matrix format, and concludes with full entity class specification for each LML entity class.

In addition, this LML specification also defines common diagrams for visualization. Each class can use common visualizations or the unique diagrams defined by this standard. Other visualizations are allowed and encouraged as they aid in expressing the information, which is the real goal of any language visualizations. These ideas can and should be proposed as extensions to the language as well so that other practitioners can benefit from these visualizations.

Ontologies provide a set of defined terms and relationships between those terms to capture the physical, functional, performance, and programmatic aspects of the system. By system,1 we mean the entire set of processes, people and things which operate for the benefit of people. Common ways of describing such ontologies is entity, relationship, and attribute (ERA). ERA is often used to define database schemas. LML uses the ERA approach but extends it by adding attributes to relationships. The

extension reduces the number of relationships needed, just as attributes reduce the number of entities needed.

3.1. LML Classes

Table 3-2 summarizes the LML classes, their parent class, description and examples (where appropriate) of how they might be used. Note that these examples can be part of the *type* attributes and used as aliases for the class itself.

Table 3-2. Summary of the LML Classes

Class Name	Parent Class	Description	Examples
Action	None	An **Action** entity specifies an effort, operation, or a process by which inputs are transformed into outputs.	Activity, Capability, Event, Function, Process
Artifact	None	An **Artifact** entity specifies a set or collection of information that is referenced by or generated from the knowledgebase.	Document, E-mail, Procedure, Specification
Asset	None	An **Asset** entity specifies an object, person, or organization that performs Actions, such as a system, subsystem, component, or element.	Component, Entity, Service, Sub-system, System
Characteristic	None	A **Characteristic** entity specifies additional properties of an entity.	Attribute, Category, Power, Role, Size, Weight, Color
Conduit	Connection	A **Conduit** entity specifies the means for physically transporting **Input/Output** entities between **Asset** entities. It has limitations (attributes) of capability and latency.	Data Bus, Interface, Pipe
Connection	None	A **Connection** entity specifies the means for relating **Asset** entities to each other.	Abstract class
Cost	None	A **Cost** entity specifies the outlay or expenditure (as of effort or sacrifice) made to achieve an objective associated with another entity.	Earned Value, Work Breakdown Structure, Actual Cost, Planned Cost
Decision	None	A **Decision** entity specifies a choice, a resolution, a conclusion, or a selected option.	Major Decision, Challenge, Issue, Problem
Dependency	Connection	A Dependency entity specifies a connection between two Tasks in a Gantt Chart. It defines the relationship a Task depends on another Task in order to Start or Finish.	

Class Name	Parent Class	Description	Examples
Input/Output	None	An **Input/Output** entity specifies the information, data, or object input to, trigger, or output from an **Action**.	Item, Trigger, Information, Data, Energy
Location	None	A **Location** entity specifies where an entity resides.	Abstract entity
Logical	Connection	A **Logical** entity represents the abstraction of the relationship between two entities (e.g., **Asset** entities with the type “Entity”)	Has, “is a”, “relates to”
Measure	Characteristic	A **Measure** entity specifies properties of measurements and measuring methodologies, including metrics.	Key Performance Parameter (KPP), Measure of Effectiveness (MOE), Measure of Performance (MOP), Metric
Orbital	Location	An **Orbital** entity specifies a location along an orbit around a celestial body.	Orbit
Physical	Location	A **Physical** entity specifies a location on, above, or below the surface.	Latitude and Longitude, Street Address
Requirement	Statement	A **Requirement** entity identifies a capability, characteristic, or quality factor of a system that must exist for the system to have value and utility to the user.	Functional Requirement, Performance Requirement, Safety Requirement
Resource	Asset	A **Resource** entity specifies a consumable or producible **Asset**.	Fuel, Bullets, Missiles, People
Risk	None	A **Risk** entity specifies the combined probability and consequence in achieving objectives.	Cost Risk, Schedule Risk, Technical Risk
Statement	None	A **Statement** entity specifies text referenced by the knowledgebase and usually contained in an **Artifact**.	Need, Goal, Objective, Assumption
Task	Action	A Task entity specifies an Action that must be completed for a particular project. It serves as a "To-Do" for the Project.	

Class Name	Parent Class	Description	Examples
Test Case	Action	A Test Case entity specifies a verification or validation task, as well as its expected and actual results.	Verification Event, Validation Event, Demonstration
Time	None	A **Time** entity specifies a point or period when something occurs or during which an action, asset, process, or condition exists or continues.	Milestone, Phase
Verification Requirement	Statement	A Verification Requirement entity specifies what is required to confirm that a requirement is satisfied.	
Virtual	Location	A **Virtual** entity specifies a location within a digital network.	URL

3.2. LML Relationships

The relationships between the classes are provided in the Table 3-3. Note that the same verb is used for the inverse relationships.

*Implies the inverse relation is present, just not shown.

Note: This matrix does not include the extensions found in the Appendices.

Table 3-3. Summary Table of LML Relationships

	Action (Task, Test Case)	Artifact	Asset (Resource)	Characteristic (Measure)	Connection (Conduit, Dependency, Logical)	Cost	Decision	Input/Output	Location (Orbital, Physical, Virtual)	Risk	Statement (Requirement, Verification Requirement)	Time
Action (Task, Test Case)	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, Dependency, 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, Virtual)	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, Verification 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*

3.3. Traceability

Because LML was designed to simplify tracing requirements to their implementation mechanisms (**Asset** class entities), the primary path for traceability is in Figure 3-1. This diagram does not reflect all the relationships shown in Table 3-3.

Figure 3-1. Principal Entities and Relationships for Design LML Traceability

3.4. Class Specifications

The following subsections provide the detailed entity class, relationship, attribute, and attribute on relationship specifications for the LML ontology. These specifications will be used by language users to provide the basis for all LML-related extensions.

3.4.0.1. Common Attributes

Name, *Number* and *Description* attributes are common to all entities.

3.4.0.1.1. name

Definition: *name* designates the particular instance of a class (entity).

Data Type: Text

3.4.0.1.2. number

Definition: *number* provides the entity's place in a hierarchy.

Data Type: Text

3.4.0.1.3. description

Definition: *description* captures the text needed to describe this entity.

Data Type: Text

3.4.0.2. Common Relationships

Many relationships are common to all or almost all entity classes, including causes/caused by, decomposes/decomposed by, enables/enabled by, incurs/incurred by, locates/located at, mitigates/mitigated by, occurs/occurred by, references/referenced by, relates/related to, resolves/resolved by, results in/resulted by; specifies/specified by, and traced to/traced from. Exceptions will be noted in the specific classes.

3.4.0.2.1. causes/caused by

Definition: *causes* identifies the **Risk** resulting from this Class Entity.

Usage: Class Entity A causes Risk 1.

Inverse usage: Risk 1 is caused by Class Entity A.

3.4.0.2.2. decomposed by/decomposes

Definition: ***decomposed by*** identifies the children of this Class Entity.

Usage: Class Entity A is *decomposed by* Class Entity B.

Inverse usage: Class Entity B decomposes Class Entity A.

3.4.0.2.3. enables/enabled by

Definition: ***enables*** identifies the **Decision** that is empowered by this Class Entity.

Usage: Class Entity A *enables* Decision 1.

Inverse usage: Decision 1 is enabled by Class Entity A.

3.4.0.2.4. incurs/incurred by

Definition: ***incurs*** identifies the **Cost** value for this Class Entity.

Usage: Class Entity **A** *incurs* Cost 1.

Inverse usage: Cost 1 is incurred by Class Entity A.

3.4.0.2.5. located at/locates

Definition: ***located at*** identifies the **Location** where this Class Entity exists.

Usage: Class Entity A is *located at* **Location 1**.

Inverse usage: Location 1 *locates* Class Entity A.

3.4.0.2.6. mitigates/mitigated by

Definition: ***mitigates*** identifies the **Risk** that this Class Entity alleviates.

Usage: Class Entity A *mitigates* Risk 1.

Inverse usage: Risk 1 is mitigated by Class Entity A.

3.4.0.2.7. occurs/occurred by

Definition: ***occurs*** identifies the **Time** (or timespan) in which this Class Entity happens.

Usage: Class Entity A *occurs* at Time 1.

Inverse usage: Time 1 is occurred by Class Entity A.

3.4.0.2.8. references/referenced by

Definition: ***references*** identifies the **Artifact** that specifies and/or enhances the definition of this Class Entity.

Usage: Class Entity A *references* Artifact 1.

Inverse usage: Artifact 1 is referenced by Class Entity A.

3.4.0.2.9. related to/relates

Definition: ***related to*** identifies the entity that ties in a peer-to-peer way with this Class Entity.

Usage: Class Entity A is *related to* Class Entity B.

Inverse usage: Class Entity B *relates* Class Entity **A**.

3.4.0.2.9.1. context

Definition: context represents a description of this relation.

Data Type: Text

3.4.0.2.10. resolves/resolved by

Definition: ***resolves*** identifies the **Risk** that is closed by this Class Entity.

Usage: Class Entity A *resolves* Risk 1.

Inverse usage: Risk 1 is resolved by Class Entity A.

3.4.0.2.11. results in/result of

Definition: ***results in*** identifies the **Decision** that is caused by this Class Entity.

Usage: Class Entity A *results in* Decision 1.

Inverse usage: Decision 1 is the *result of* Class Entity A.

3.4.0.2.12. specified by/specifies

Definition: ***specified by*** identifies a **Characteristic** that provides further information about this Class Entity.

Usage: Class Entity A is *specified by* Characteristic 1.

Inverse usage: Characteristic 1 *specifies* Class Entity A.

3.4.0.2.13. traced from/traced to

Definition: ***traced from*** identifies a **Statement** that is accredited to this Class Entity.

Usage: Class Entity A is *traced from* Statement 1.

Inverse usage: Statement 1 is *traced to* Class Entity A.

3.4.1. Action

Definition: An **Action** entity generates effects and may have pre-conditions before it can be executed. This **Action** can include transforming inputs into outputs.

Parent: None

Subclasses: Task, Test Case

3.4.1.1. Action Attributes

Action class entities have attributes: *name*, *number*, *description*, *duration* and *type*. Note: LML v1.x had two other attributes (*start* and *status*). Those attributes have been moved to the **Task** subclass.

3.4.1.1.1. duration (Action)

Definition: *duration* represents the period of time this **Action** occurs. It can be represented as a floating number or distribution.

Data Type: Number

3.4.1.1.2. type (Action)

Definition: *type* provides aliases for the entities. For **Action** these can include: Activity, Capability, Event, Function, Mission, Operational Activity, Program, Service Orchestration, Simulation Workflow, Subprocess, System Function, Training, Use Case, Work Process, Workflow.

Data Type: Text

3.4.1.2. Action Relationships

The additional relationships for the Action class entities are provided below. The Action class uses all the common relationships provided in section 3.4.0.2.

3.4.1.2.1. consumes/consumed by

Definition: ***consumes*** identifies the **Resource** that this **Action** uses. After this **Action** is completed, the amount consumed is not returned to the **Resource**.

Usage: Action A *consumes* Resource 1.

Inverse usage: Resource 1 is consume by Action A.

3.4.1.2.1.1. amount

Definition: amount represents how much of the resource is ***consumed by*** the **Action**. Units are relative to the units selected for the **Resource**.

Data Type: Number

3.4.1.2.2. generates/generated by

Definition: ***generates*** identifies the **Input/Output** that this **Action** transforms.

Usage: **Action A** *generates* Input/Output 1.

Inverse usage: Input/Output 1 is generated by Action A.

3.4.1.2.3. performed by/performs

Definition: ***performed by*** identifies the **Asset** that executes this **Action**.

Usage: Action A is *performed by* Asset 1.

Inverse usage: Asset 1 *performs* Action A.

3.4.1.2.4. receives/received by

Definition: ***receives*** identifies the **Input/Output** that is taken in by this **Action**.

Usage: Action A *receives* Input/Output 1.

Inverse usage: Input/Output 1 is received by Action A.

3.4.1.2.4.1. trigger

Definition: trigger represents this relation as an enabling requirement for the **Action**. An **Action** begins execution when it has received control enablement, all of its triggers have arrived, and its necessary resources are available.

Data Type: Boolean

3.4.1.2.5. seizes/seized by

Definition: ***seizes*** identifies the **Resource** that this **Action** uses. After this **Action** has completed the **Resource** is released for use by other **Actions**.

Usage: Action A *seizes* Resource 1.

Inverse usage: Resource 1 is seized by Action A.

3.4.1.2.5.1. amount

Definition: amount represents how much of the resource is ***captured by*** the **Action**. Units are relative to the units selected for the **Resource**.

Data Type: Number

3.4.1.3. Task (Action)

Definition: A Task entity specifies an activity that must be completed for a particular project. It serves as a "To-Do" for the Project.

Parent Class: Action

3.4.1.3.1. Task Attributes

Task attributes include: name, number, description, duration, date due, estimated date of completion, finish, percent complete, planned start, priority, start, status, and type.

3.4.1.3.1.1. date due

Definition: Date and Time the Task is required to be completed.

Data Type: DateTime

3.4.1.3.1.2. estimated date of completion

Definition: Date and time the Task is expected to be completed.

Data Type: DateTime

3.4.1.3.1.3. finish

Definition: Date and time the Task is actually finished.

Data Type: DateTime

3.4.1.3.1.4. percent complete

Definition: *percent complete* represents the percentage this **Task** is complete.

Data Type: Percent

3.4.1.3.1.5. planned start

Definition: Date and Time the Task is (or was) planned to begin.

Data Type: DateTime

3.4.1.3.1.6. priority

Definition: *priority* identifies the level of urgency for Task completion. Values of: Low Priority, Medium Priority, High Priority.

Data Type: Enumeration

3.4.1.3.1.7. start

Definition: *start* represents the actual date and time when this **Task** begins.

Data Type: DateTime

3.4.1.3.1.8. status

Definition: Open; In Progress; in Review; Closed.

Data Type: Enumeration

3.4.1.3.1.9. type (Task)

Definition: Work Package; Step; Milestone; Schedule Activity; Baseline; Forecast; Kanban Board; Gantt Chart.

Data Type: Enumeration

3.4.1.3.2. Task Relationships

The additional relationships for the **Task** class entities are provided below. The **Task** class uses all the common relationships provided in section 3.4.0.2 and the additional **Action** relationships from section 3.4.1.2.

3.4.1.3.2.1. depends on/has dependent

Definition: ***depends on*** identifies the **Dependency** (a subclass of **Connection**) between two **Tasks.**

Usage: Task A depends on Task B.

Inverse usage: Task B has dependent Task A.

3.4.1.3.2.2. scheduled by/schedules

Definition: scheduled by identifies a Kanban Column (Time entity) that a Task is planned to take place within.

Usage: Task A is *scheduled by* Time 1.

Inverse usage: Time 1 *schedules* Action A.

3.4.1.3.2.3. tracked by/tracks

Definition: tracked by identifies Task entity associated with the Time entity that represent the Kanban Board Columns.

Usage: Task A*tracked by*Time 1

Inverse usage: Time 1 *tracks* Task A.

3.4.1.4. Test Case (Action)

Definition: A Test Case entity specifies a verification/validation task, as well as its expected and actual results.

Parent Class: Action

3.4.1.4.1. Test Case Attributes

Test Case attributes include: name, number, description, duration, actual result, expected result, event conditions, event constraints, status, setup, and type.

3.4.1.4.1.1. Actual Result

Definition: actual result summarizes the recorded results of the V&V task.

Data Type: Text

3.4.1.4.1.2. Expected Result

Definition: expected result summarizes the predicted results of the V&V task.

Data Type: Text

3.4.1.4.1.3. Event Conditions

Definition: *event conditions* specifies conditions and setup details to be used during the test. Conditions might specify parameters that simulate scenarios or physical analogs of processed materials.

Data Type: Text

3.4.1.4.1.4. Event Constraints

Definition: *event constraints* capture information that impose constraints on verification events, such as the need for independent verification, witness points, etc. Constraints might also indicate sampling requirements.

Data Type: Text

3.4.1.4.1.5. Status

Definition: status provides the state of the V&V task. Suggested values are: “Not Run,” “In Progress,” “Blocked,” “Failed,” and “Passed”

Data Type: Enumeration

3.4.1.4.1.6. Setup

Definition: setup details steps required prior to conducting the V&V task. setup details might require specific HWIL, SWIL, or surrogate input streams.

Data Type: Text

3.4.1.4.1.7. Type (Test Case)

3.4.1.4.2. Test Case Relationships

The additional relationships for the Test Case class entities are provided below. The Test Case class uses all the common relationships provided in section 3.4.0.2 and the additional **Action** relationships from section 3.4.1.2.

Special cases for ***references/referenced by*** and ***traced from/traced*** to are provided below.

3.4.1.4.2.1. evaluates/evaluated by

Definition: A ***verification event*** evaluates an **Asset** entity included in the event.

Usage: Test Case A *evaluates* Asset 1.

Inverse usage: Asset 1 is evaluated by Test Case A.

3.4.1.4.2.2. references/referenced by

Definition: ***references*** the Test Suite (a type of **Artifact**) that provides the container for the top level **Test Cases**.

Usage: Test Case A *references* Test Suite (type of **Artifact**) 1.

Inverse usage: Test Suite 1 is referenced by Test Case A.

3.4.1.4.2.3. traced from/traced to

Definition: A verification event (**Test Case**) is traced from a **Verification Requirement**

Usage: Test Case A is *traced from* **Verification Requirement 1**.

Inverse usage: Verification Requirement 1 is *traced to* Test Case A.

3.4.2. Artifact

Definition: An **Artifact** entity specifies a document or other source of information that is referenced by or generated in the knowledgebase. Example: Requirements Report.

Parent: None

3.4.2.1. Artifact Attributes

Artifact class entities have attributes: *name*, *number*, *description*, *date published, file,* and *type*.

3.4.2.1.1. date published

Definition: *date published* represents the date when this **Artifact** was accessed (webpage), published, or uploaded to the knowledgebase.

Data Type: Date/Time

3.4.2.1.2. file

Definition: The *file* that represents this **Artifact**.

Data Type: URI

3.4.2.1.3. type

Definition: *type* provides aliases for the entities. For **Artifact** these can include: Briefing, Change Notice, Change Request, Concept of Operations, Directive, Doctrine, Document, E-Mail, Guidance, Instruction, Interface Control Document, Interface Requirements Specification, Manual, Matrix, Meeting Minutes, Memorandum, Mitigation Plan, Model, Operational Concept, Policy, Procedure, Protocol, Proposal, Regulation, Requirements Document, Request for Proposals, Software Requirements Specification, Standard, System Requirements Document, System/Segment Design Document, System/Segment Specification, Test & Evaluation Plan, Test & Evaluation Report, Test Suite, Text Message, Trade Study, White Paper

Data Type: Text

3.4.2.2. Artifact Relationships

The additional relationships for the **Artifact** class entities are provided below. The **Artifact** class uses all the common relationships provided in section 3.4.0.2.

3.4.2.2.1. defines protocol for/defined protocol by

Definition: ***defines protocol for*** identifies the **Conduit** that uses the standard identified in this **Artifact**.

Usage: Artifact A defines protocol for Conduit 1.

Inverse Usage: Conduit 1 has a *defined protocol by* Artifact A.

3.4.2.2.2. source of/sourced by

Definition: ***source of*** identifies the **Statement** that is contained in this **Artifact**.

Usage: Artifact A is the *source of* Statement 1.

Inverse Usage: Statement 1 is *sourced by* Artifact A.

3.4.3. Asset

Definition: An **Asset** entity specifies an object, person, or organization that used to create value and perform **Actions**, such as a system, subsystem, component, or element.

Examples: Infrared Sensor, Accounting Department, Internal Revenue Service

Parent: None

Subclasses: Resource

3.4.3.1. Asset Attributes

Asset class entities have attributes: *name*, *number*, *description*, and *type*.

3.4.3.1.1. type

Definition: *type* provides aliases for the entities. For **Asset** these can include: Architecture, Assembly, Component, Context, CSC, CSCI, CSU, Element, Environment, External System, Facility, Hardware, Human, HW Element, HWCI, Infrastructure, LRU, Materiale, Operational Element, Organization, Part, Performer, Personnel, Segment, Service, Software, Subassembly, Subsystem, System, System Instantiation, Test Equipment, Test Software, Unit

Data Type: Text

3.4.3.2. Asset Relationships

The additional relationships for the **Asset** class entities are provided below. The **Asset** class uses all the common relationships provided in section 3.4.0.2.

3.4.3.2.1. connected by/connects to

Definition: ***connected by*** identifies the **Connection** that adjoins this **Asset**. Note: Connection is an abstract class. It has subclasses of Conduit and Logical.

Usage: Asset A is connected by Connection 1.

Inverse Usage: Connection 1 connects to Asset A.

3.4.3.2.2. orbited by/orbits

Definition: ***orbited by*** identifies the **Asset** entity that acts as a satellite to this **Asset** entity.

Usage: Asset A is *orbited by* Asset B

Inverse Usage: **Asset** B ***orbits*** **Asset** A.

3.4.3.3. Resource (Asset)

Definition: A **Resource** entity specifies a consumable or producible **Asset**.

Example: $5, 2 kilowatts, natural gas.

All attributes and relationships from **Asset** are inherited by the **Resource** entities.

3.4.3.3.1. Resource Attributes

Resource attributes include: name, number, description, initial amount, maximum amount, minimum amount, units, and type.

3.4.3.3.1.1. initial amount

Definition: initial amount represents this Resource's starting amount.

Data Type: Number

3.4.3.3.1.2. maximum amount

Definition: maximum amount represents this Resource's maximum amount allowed.

Data Type: Number

3.4.3.3.1.3. minimum amount

Definition: minimum amount represents this Resource's minimum amount allowed.

Data Type: Number

3.4.3.3.1.4. units

Definition: units represents this Resource's units used to measure the amounts, such as each or tons.

Data Type: Text

3.4.3.3.1.5. type

Definition: type provides aliases for the entities. For Resource these can include: Computer Resource, Human Resource, Fuel

Data Type: Text

3.4.3.3.2. Resource Relationships

The additional relationships for the Resource class entities are provided below. The Resource class uses all the common relationships provided in section 3.4.0.2 and the additional **Asset** relationships from section 3.4.3.2.

3.4.3.3.2.1. consumed by/consumes

Definition: ***consumed by*** identifies the **Action** that uses this **Resource**. After the **Action** is completed, the amount consumed is not returned to the **Resource**.

Usage: Resource A is consume by Action 1.

Inverse usage: **Action 1** *consumes* Resource A.

3.4.3.3.2.1.1. amount attribute

Definition: amount represents how much of the resource is ***consumed by*** the **Action**. Units are relative to the units selected for the **Resource**.

Data Type: Number

3.4.3.3.2.2. produced by/produces

Definition: ***produced by*** identifies the **Action** that creates this **Resource**. **Resources** are produced when the execution of the action completes.

Usage: Resource A is produced by Action 1.

Inverse usage: Action 1 produces Resource A.

3.4.3.3.2.2.1. amount

Definition: amount represents how much of the **Resource** is ***produced by*** the **Action**. Units are relative to the units selected for the **Resource**.

Data Type: Number

3.4.3.3.2.3. seizes/seized by

Definition: ***seized by*** identifies the **Action** that uses this **Resource**. After the **Action** has completed, this **Resource** is released for use by other **Actions**.

Usage: Resource A is seized by Action 1.

Inverse usage: Action 1 seizes Resource A.

3.4.3.3.2.3.1. amount

Definition: amount represents how much of the resource is ***captured by*** the **Action**. Units are relative to the units selected for the **Resource**.

Data Type: Number

3.4.4. Characteristic

Definition: A **Characteristic** entity specifies or captures properties of an entity.

Examples: Blue, no heavier than 2 oz, accurate to within 1%.

3.4.4.1. Characteristic Attributes

Characteristic attributes include: name, number, description, type, units, value, formatted equation, and computed value.

3.4.4.1.1. type

Definition: *type* provides aliases for the entities. For **Characteristic** these can include: Condition, Data Element, Environmental, Functional, Performance, Physical, Scenario, Security, Verification Category

Data Type: Text

3.4.4.1.2. units

Definition: *units* represents this **Characteristic**'s units used to measure the value, such as pounds or miles per hour.

Data Type: Text

3.4.4.1.3. value

Definition: *value* represents this **Characteristic**'s current value.

Data Type: Text

3.4.4.1.4. formatted equation (optional)

Definition: *formatted equation* provides a visual representation of a mathematical equation using LaTeX to specify the equation.

Data Type: Equation

3.4.4.1.5. computed value (optional)

Definition: *computed value* provides a calculation of a mathematical equation using LaTeX to specify the equation. Variables values are specified using numerical attributes of other classes.

Data Type: Computable

3.4.4.2. Characteristic Relationships

There are no additional relationships for the **Characteristic** class entities. The **Characteristic** class uses all the common relationships provided in section 3.4.0.2.

3.4.4.3. Measure (Characteristic)

Definition: A **Measure** entity specifies the set of measurements used to provide managers, system developers, and systems engineers with insight into the system definition, and the analysis of technical solutions with respect to performance, cost, and risk.

Examples: 19 inches, 22 grams, 1.21 gigawatts

All attributes and relationships from **Characteristic** are inherited by the **Measure** entities.

3.4.4.3.1. Measure Attributes

Measure attributes include: name, number, description, improvement direction, objective value, projected value, threshold value, tolerance, type, units, and value.

3.4.4.3.1.1. improvement direction

Definition: *improvement direction* represents the direction in which metric improvement occurs. It is the direction from the threshold value to the objective value.

Data Type: Enumeration [N/A, Positive, Negative]

3.4.4.3.1.2. objective value

Definition: *objective value* represents the goal for this **Measure**.

Data Type: Text

3.4.4.3.1.3. projected value

Definition: *projected value* represents this **Measure**'s expected value to be achieved with existing resources.

Data Type: Text

3.4.4.3.1.4. threshold value

Definition: *threshold value* represents the minimum acceptable value for this **Measure**.

Data Type: Text

3.4.4.3.1.5. tolerance

Definition: *tolerance* represents the percentage of the value that forms the positive and negative tolerance bands. tolerance is used when value represents the planned measure value at a given time.

Data Type: Text

3.4.4.3.1.6. type

Definition: *type* provides aliases for the entities. For Measure these can include: Critical Operational Issue (COI), Key Performance Parameter (KPP), Mean Time Between Failures (MTBF), Measure of Effectiveness (MOE), Measure of Performance (MOP), Metric, Technical Performance Measure (TPM).

Data Type: Text

3.4.4.3.2. Measure Relationships

No additional relationships from those inherited from the **Characteristic** parent class.

3.4.5. Connection (Abstract Class)

Definition: A **Connection** entity specifies the mechanism relating other classes to each other through a class entity. This is an abstract class.

3.4.5.1. Connection Attributes

No additional attributes are contained in the **Connection** abstract class.

3.4.5.2. Connection Relationships

The additional relationships for the **Connection** class entities are provided below. The **Connection** class uses all the common relationships provided in section 3.4.0.2, except for ***decomposes/decomposed***.

3.4.5.2.2. connects to/connected by

Definition: ***connects to*** identifies the **Asset** that this **Connection** adjoins.

Usage: Connection A connects to Asset 1.

Inverse Usage: Asset 1 and Asset 2 are connected by Connection A.

3.4.5.2.2.1. origin

Definition: origin represents if the **Asset** is the origin of the *Connection*. This attribute enables unidirectional (one or the other **Asset***'s* relationship origin is False), in additional to the default bi-directional flow (both origins are set to True by default).

Data Type: Boolean

3.4.5.2.2.2. multiplicity

Definition: multiplicity, also called cardinality, represents the number of relationships (one to many, many to one, etc.) that can occur.

Data Type: Text

3.4.5.2.5. joined by/joins

Definition: joined by identifies the Connection that connects to this Connection. joined by implies the end of the relationship.

Usage: Connection A is joined by Connection B.

Inverse Usage: Connection B joins Connection A.

3.4.5.2.6.1. bidirectional

Definition: bidirectional represents if the connection between two Connections is bidirectional. If the Connection is not bidirectional, the Connection that has the relation *joins* is the start and the Connection with *joined by* is the end.

Data Type: Boolean

3.4.5.3. Conduit (Connection)

Definition: A **Conduit** entity specifies the connection between Assets and has capacity and latency. A Conduit carries an Input/Output.

Examples: SpaceWire, Bluetooth, railway

All attributes and relationships from **Connection** are inherited by the **Conduit** entities.

3.4.5.3.1. Conduit Attributes

Conduit attributes include: name, number, description, capacity, latency, units, and type.

3.4.5.3.1.1. capacity

Definition: *capacity* represents the maximum rate supported by the Conduit. (Used in simulation as to add time delay by dividing the **Input/Output** *size* with the *capacity* value.)

Data Type: Number

3.4.5.3.1.2. latency

Definition: *latency* represents the time required to transmit the information or **Input/Output** entity over this **Conduit**. This value does not factor in any delays due to *capacity* limitations.

Data Type: Number

3.4.5.3.1.3. units (Conduit)

Definition: *units* represents units used to measure the *capacity* of ___

Data Type: Text

3.4.5.3.1.4. type (Conduit)

Definition: *type* provides aliases for the entities. For **Conduit** these can include: Cable, Downlink, Human-in-the-Loop, Human Machine Interface, Interface, Landline, Link, Needline, Network, Pipe, RF - Satcom, RF - Terrestrial, Roadway, Service Interface, Uplink, Wireless.

Data Type: Text

3.4.5.3.2. Conduit Relationships

The additional relationships for the **Conduit** class entities are provided below. The **Conduit** class uses all the common relationships provided in section 3.4.0.2 (including ***decomposes/decomposed by***) and the additional **Connection** relationships from section 3.4.5.2.

3.4.5.3.2.1. defined protocol by/defines protocol for

Definition: ***defined protocol by*** identifies the **Artifact** entity that contains the standard used by this **Conduit** entity.

Usage: Conduit A has a *defined protocol by* Artifact 1.

Inverse Usage: Artifact 1 *defines* (the) *protocol for* Conduit A.

3.4.5.3.2.2. transfers/transferred by

Definition: ***transfers*** identifies the Input/Output entity that is passed by this Conduit entity.

Usage: Conduit A ***transfers*** Input/Output 1.

Inverse Usage: Input/Output 1 is ***transferred by*** Conduit A.

3.4.5.4. Dependency (Connection)

Definition: A **Dependency** entity specifies a connection between two **Tasks** in a Gantt Chart. It defines the relationship a **Task** depends on another **Task** in order to Start or Finish.

3.4.5.4.1. Dependency Attributes

Dependency class entities have no additional attributes.

3.4.5.4.2. Dependency Relationships

The additional relationships for the Dependency class entities are provided below.
The Dependency class uses all the common relationships provided in section 3.4.0.2 (excluding *decomposes/decomposed by*) and the additional Connection relationships from section 3.4.5.2.

3.4.5.4.2.1. has dependent/depends on

Definition: ***has dependent*** identifies the **Task** entity that depends on this **Task** entity.

Usage: Task A ***has dependent*** Task B.

Inverse Usage: Task B ***depends on*** Task A.

3.4.5.4.2.1.1. origin

Definition: origin identifies which Task entity is depending on another Task entity for completion.

Data Type: Boolean

3.4.5.4.2.1.2. dependency

Definition: dependency identifies if the origin is at the start or finish of each **Task** entity. This capability enables the distinguishing between start-to-finish, finish-to-start, start-to-start, and finish-to-finish.

Data Type: Enumeration

3.4.5.5. Logical (Connection)

Definition: A **Logical** entity specifies relationship between **Assets**. It is primarily used in database schema development and entity-relationship diagrams.

All attributes and relationships from **Connection** are inherited by the **Logical** entities.

3.4.5.5.1. Logical Attributes

Logical class entities have no additional attributes.

3.4.5.5.2. Logical Relationships

The additional relationships for the **Logical** class entities are provided below. The **Logical** class uses all the common relationships provided in section 3.4.0.2 (including *decomposes/decomposed by*) and the additional **Connection** relationships from section 3.4.5.2.

3.4.5.5.2.1. specified by/specifies (Logical)

Definition: ***specified by*** identifies a **Characteristic** entity that provides further information about this **Logical** entity. This relationship is used to specify an attribute on a relationship in an extended Entity-Relationship-Attribute (ERA) schema.

Usage: Logical A is ***specified by*** Characteristic (attribute) 1.

Inverse Usage: Characteristic (attribute) 1 ***specifies*** Logical A.

3.4.6. Cost

Definition: A **Cost** entity specifies the outlay or expenditure (as of effort or sacrifice) made to achieve an objective associated with another entity.

Examples: $100, 6 man-hours

3.4.6.1. Cost Attributes

Cost attributes include: *name*, *number*, *description*, *amount*, *category*, *contract type*, *rate*, *units*, and *type*.

3.4.6.1.1. amount

Definition: *amount* represents this **Cost**'s value.

Data Type: Number

3.4.6.1.2. category

Definition: *category* represents the part of the lifecycle for which the money is used (commonly called Color of Money). This matches the US Federal Government (DoD) cost phases.

Data Type: Enumeration. Suggested Values: [Procurement, MILCON, MILPERS, O&M, RDT&E, SCN, N/A]

3.4.6.1.3. contract type

Definition: *contract type* represents this **Cost** way to identify the reimbursement structure (i.e., CPFF vs T&M).

Data Type: Enumeration. Suggested Values: [CPFF, CPAF, CPIF, FFP-Completion, FFP-LOE, T&M, N/A]

3.4.6.1.4. rate

Definition: *rate* represents how this **Cost** is billed.

Data Type: Enumeration. Minimum Values: _

3.4.6.1.5. units (Cost)

Definition: *units* represents the currency used for this **Cost**, such as $ or €.

Data Type: Text

3.4.6.1.6. type (Cost)

Definition: *type* provides aliases for the entities. For **Cost** these can include: Actual Cost, Earned Value, Estimated Cost, Cost Overrun, Planned Cost

Data Type: Text

3.4.6.2. Cost Relationships

No additional relationships are defined for the **Cost** class entities. The **Cost** class uses all the common relationships provided in section 3.4.0.2.

3.4.7. Decision

Definition: A **Decision** entity specifies an opportunity to make a choice.

Examples: slip schedule by two months, accept risk, hire ten new employees.

3.4.7.1. Decision Attributes

Decision attributes include: *name*, *number*, *description*, *assumptions*, *justification*, *status*, and *type*.

3.4.7.1.1. assumptions

Definition: *assumptions* represents facts that are used as a basis for this **Decision**.

Data Type: Text

3.4.7.1.2. justification

Definition: *justification* represents the reason and context for making this **Decision**. For additional justification, the user may want to apply the **enabled by** relationship to link it to a **Statement** or **Statement**s.

Data Type: Text

3.4.7.1.3. status

Definition: *status* represents the state of the **Decision** (open or closed).

Data Type: Enumeration. Mandatory Values: _

3.4.7.1.4. type (Decision)

Definition: *type* provides aliases for the entities. For **Decision** these can include: Challenge, Issue, Problem.

Data Type: Text

3.4.7.2. Decision Relationships

The additional relationships for the Decision class entities are provided below. The Decision class uses all the common relationships provided in section 3.4.0.2.

3.4.7.2.1. alternative/alternative of

Definition: ***alternative*** identifies the **Statement** entity that is a potential choice for this **Decision** entity.

Usage: Decision A ***alternative*** is Statement 1

Inverse Usage: Statement 1 is the ***alternative of*** Decision A.

3.4.7.2.1.1. choice

Definition: choice represents if this alternative was the choice selected. If the **Decision** entity is still open, none of these alternatives would be True.

Data Type: Boolean

3.4.7.2.2. date resolved by/date resolves

Definition: **date resolved by** identifies the **Time** entity when this **Decision** entity was closed. For open issues, this attribute can be left blank.

Usage: Decision A ***date*** (was) ***resolved by*** Time 1.

Inverse Usage: Time 1 ***date resolves*** Decision A.

3.4.7.2.3. decision due/decided by

Definition***: decision due*** identifies the **Time** entity when this **Decision** entity is scheduled to be closed.

Usage: **Decision** A ***decision*** (is) ***due*** on **Time** 1.

Inverse Usage: **Time** 1 ***decided by*** **Decision** A.

3.4.7.2.4. made by/made

Definition: made by identifies the Asset entity responsible for making this Decision entity.

Usage: **Decision** A is to be (or was) ***made by*** **Asset** 1.

Inverse Usage: **Asset** 1 ***made*** **Decision** A.

3.4.7.2.5. responded by/responds to

Definition: ***responded by*** identifies the **Asset** entity (usually a person or organization) that acts on this **Decision** entity.

Usage: Decision A is ***responded by*** Asset 1.

Inverse Usage: Asset 1 ***responds to*** Decision A.

3.4.7.2.5.1. responsibility

Definition: responsibility represents the **Asset** entity that has the responsibility for resolving the **Decision** entity.

Data Type: Enumeration. Suggested Values: [Primary, Secondary]

3.4.8. Input/Output

Definition: An **Input/Output** entity specifies the matter, energy, and/or information input to, triggers (controls), or output from an Action.

Examples: Nickel/gumball, gasoline/horsepower, investment/return.

3.4.8.1. Input/Output Attributes

Input/Output attributes include: *name*, *number*, *description*, *size*, *units*, and *type*.

3.4.8.1.1. size

Definition: *size* represents the amount or proportion of this **Input/Output**, such as 100 as 100 Gigabytes or number of entities (e.g., 10 as in 10 tokens).

Data Type: Number

3.4.8.1.2. units (Input/Output)

Definition: *units* represents the measure using in the *size* attribute of this **Input/Output** entity, such as Gigabytes or tokens.

Data Type: Text

3.4.8.1.3. type (Input/Output)

Definition: *type* provides aliases for the entities. For **Input/Output** these can include: Analog, Data, Digital, Event, Information, Item, Mixed, Physical, Product, Verbal.

Data Type: Text

3.4.8.2. Input/Output Relationships

The additional relationships for the **Input/Output** class entities are provided below.
The **Input/Output** class uses all the common relationships provided in section 3.4.0.2.

3.4.8.2.1 generated by/generates

Definition: ***generated by*** identifies the Action entity that transformed this Input/Output entity.

Usage: Input/Output A is ***generated by*** Action 1.

Inverse Usage: Action 1 ***generates*** Input/Output A.

3.4.8.2.2. received by/receives

Definition: ***received by*** identifies the **Action** entity that takes in this **Input/Output** entity.

Usage: Input/Output A is ***received by*** Action 1.

Inverse Usage: Action 1 ***receives*** Input/Output A.

3.4.8.2.2.1. trigger

Definition: trigger represents this relation as an enabling requirement for the **Action**. An **Action** begins execution when it has received control enablement, all of its triggers have arrived, and its necessary resources are available.

Data Type: Boolean

3.4.8.2.3. transferred by/transfers

Definition***: transferred by*** identifies the **Connection** (usually a **Conduit**) entity that passes this **Input/Output** entity.

Usage: Input/Output A is ***transferred by*** Connection 1

Inverse Usage: Connection 1 ***transfers*** Input/Output A.

3.4.9. Location (Abstract Class)

Definition: A **Location** entity specifies where an entity resides. Abstract Class.

3.4.9.1. Location Attributes

No additional attributes are contained in the Location abstract class.

3.4.9.2. Location Relationships

No additional relationships for the Location class entities are provided below. The Location class uses all the common relationships provided in section 3.4.0.2., except: ***causes/caused by***; ***enables/enabled by***; ***incurs/incurred by***; ***references/referenced by***; ***resolves/resolved by***; and ***results in/resulted by***.

3.4.9.3. Orbital (Location)

Definition: An **Orbital** entity specifies a location (ephemeris) along an orbit around a celestial body. Note that this includes transfer orbits as well. Examples: Mars orbit, transfer to lunar orbit

3.4.9.3.1. Orbital Attributes

Orbital attributes include: *name*, *number*, *description*, *inclination*, *semimajor axis*, *longitude of ascending node*, *reference plane*, *argument of periapsis*, *origin of longitude*, *mean anomaly*, *apoapsis*, *periapsis*, and *eccentricity*.

3.4.9.3.1.1. inclination

Definition: *inclination* represents the angle between the orbital plane and a *reference plane*, such as the equatorial plane for Earth. The *inclination* must be specified with the *longitude of ascending node* to characterize the orbital plane.

Data Type: Number

3.4.9.3.1.2. semimajor axis

Definition: *semimajor axis* represents the one half of the length of the longest diameter of the orbital ellipse. The *semimajor axis* must be specified with the *eccentricity* to characterize the orbital ellipse shape.

Data Type: Number

3.4.9.3.1.3. longitude of ascending node

Definition: *longitude of ascending node* represents the angle from the *origin of longitude* to the *ascending node*, measured in the *reference plane*. For the Earth, the *origin of longitude* is typically the Prime Meridian. The *longitude of ascending node* must be specified with the *inclination* to characterize the orbital plane.

Data Type: Number

3.4.9.3.1.4. reference plane

Definition: *reference plane* represents the *reference plane* from where the *inclination* angle will be calculated.

Data Type: Text

3.4.9.3.1.5. argument of periapsis

Definition: *argument of periapsis* represents the angle between the *periapsis* and the ascending node as measured in the orbital plane in the direction of motion.

Data Type: Number

3.4.9.3.1.6. origin of longitude

Definition: *origin of longitude* represents the reference meridian from where the *longitude of ascending node* will be calculated.

Data Type: Text

3.4.9.3.1.7. mean anomaly

Definition: *mean anomaly* represents the proportion of orbital area swept since the last *periapsis* at the specified time. It is used to define the position of the orbiting **Asset** along the orbital ellipse.

Data Type: Number

3.4.9.3.1.8. apoapsis

Definition: *apoapsis* represents the point of greatest distance from the celestial body being orbited. For Earth, the term is apogee. For the Sun the term commonly used is aphelion. The *apoapsis* must be specified with the *periapsis* to characterize the orbital ellipse shape.

Data Type: Number

3.4.9.3.1.9. periapsis

Definition: *periapsis* represents the point of closest distance from the celestial body being orbited. For Earth, the term is perigee. For the Sun the term commonly used is perihelion. The *periapsis* must be specified with the *apoapsis* to characterize the orbital ellipse shape.

Data Type: Number

3.4.9.3.1.10. eccentricity

Definition: *eccentricity* represents the amount the orbit deviates from a perfect circle (0 being perfectly circular and 1 is a parabola – no longer a closed orbit). The eccentricity must be specified with the *semimajor axis* to characterize the orbital ellipse shape.

Data Type: Number

3.4.9.3.2. Orbital Relationships

No additional relationships for the Orbital class entities are provided below. The **Orbital** class uses all the relationships provided in section 3.4.9.2.

3.4.9.4. Physical (Location)

Definition: A **Physical** entity specifies a location on, below, or above the surface of a celestial body.
Examples: North Pole, Camp Lejeune

Parent Class: **Location**

Note that this entity definition uses Cartesian Coordinates (x, y, z). It may be desirable to establish other coordinate systems (e.g., Cylindrical, Spherical, etc.) for other implementations.

3.4.9.4.1. Physical Attributes

Physical attributes include: *name*, *number*, *description*, *address*, *altitude/depth*, *coordinates*, *units*, and *type*.

3.4.9.4.1.1. address (Physical)

Definition: *address* represents this **Physical** location's complete address.

Data Type: Text

3.4.9.4.1.2. altitude/depth

Definition: *altitude/depth* represents the distance above (positive values) or below (negative values) the surface.

Data Type: Number

3.4.9.4.1.3. coordinates

Definition: *coordinates* represents the coordinate points for this **Physical** location (GPS or other system).

Data Type: GeoPoint

3.4.9.4.1.4. units (Physical)

Definition: *units* represents the units used to measure the *altitude/depth* of the **Physical** location.

Data Type: Text

3.4.9.4.1.5. type (Physical)

Definition: *type* provides aliases for the entities. For **Physical** these can include: Geospatial Location, Map Coordinates

Data Type: Text

3.4.9.4.2. Physical Relationships

No additional relationships for the Orbital class entities are provided below. The **Orbital** class uses all the relationships provided in section 3.4.9.2.

3.4.9.5. Virtual (Location)

Definition: A **Virtual** entity specifies a location within a digital network. Example: http://www.google.com

Parent Class: **Location**

3.4.9.5.1. Virtual Attributes

Virtual attributes include: *name*, *number*, *description*, and *address*.

3.4.9.5.1.1. address (Virtual)

Definition: *address* represents the identification address using the Uniform Resource Identifier (URI) protocols.

Data Type: URI

3.4.9.5.2. Virtual Relationships

No additional relationships for the **Virtual** class entities are provided below. The **Virtual** class uses all the relationships provided in section 3.4.9.2.

3.4.10. Risk

Definition: **Risk** entity specifies the combined probability and consequence in achieving objectives.
Example: the risk of a large meteorite hitting the earth in the next 100 years is low but it could cause the extinction of life as we know it.

3.4.10.1. Risk Attributes

Risk attributes include: *name*, *number*, *description*, *consequence*, *consequence description*, *mitigation status*, *probability*, *status*, *trend*, and *type*.

3.4.10.1.1. consequence

Definition: *consequence* represents the level of effect from this **Risk** entity occurring.

Data Type: Percent

3.4.10.1.2. consequence description

Definition: *consequence description* represents the result of this **Risk** entity occurring.

Data Type: Text

3.4.10.1.3. mitigation status

Definition: *mitigation status* represents the status of the mitigation technique for this **Risk** entity.

Data Type: Enumeration. Mandatory Values:

3.4.10.1.4. probability

Definition: *probability* or likelihood represents the chance that this **Risk** entity will occur.

Data Type: Percent

3.4.10.1.5. status

Definition: *status* represents the current state of this **Risk** entity.

Data Type: Enumeration. Suggested Values:

3.4.10.1.6. trend

Definition: *trend* indicates the change in the **Risk** entity over time as to whether it is increasing, decreasing, or staying the same.

Data Type: Enumeration. Suggested Values:

3.4.10.1.7. type (Risk)

Definition: *type* provides aliases for the entities. For **Risk** entities these can include: Cost Risk, Schedule Risk, Technical Risk

Data Type: Text

3.4.10.2. Risk Relationships

No additional relationships for the Risk class entities are provided below. The Risk class uses all the common relationships provided in section 3.4.0.2, except ***traced from/traced to***.

3.4.11. Statement

Definition: A **Statement** entity specifies text referenced by the knowledgebase and usually contained in an Artifact. Example: Elvis is king!, Our goal is to be the first on Mars

3.4.11.1. Statement Attributes

Statement attributes include: *name, number, description,* and *type*.

3.4.11.1.1. type (Statement)

Definition: *type* provides aliases for the entities. For **Statement** entities these can include: Acronym, Assumption, Constraint, Definition, Directive, Doctrine, Goal, Need, Objective, Plan, Policy, Question, Rule, Scope, Standard, Vision

Data Type: Text

3.4.11.2. Statement Relationships

The additional relationships for the **Statement** class entities are provided below. The **Statement** class uses all the common relationships provided in section 3.4.0.2.

3.4.11.2.1. alternative of/alternative

Definition: ***alternative of*** identifies the **Decision** entity that has this **Statement** entity as a potential choice.

Usage: Statement A is an *alternative of* Decision 1.

Inverse Usage: Decision 1 *alternative* is Statement A.

3.4.11.2.1.1. choice

Definition: choice represents if this ***alternative*** was the choice selected.

Data Type: Boolean

3.4.11.2.2. sourced by/source of

Definition: ***sourced by*** identifies the **Artifact** entity that contains this **Statement** entity.

Usage: Statement A is *sourced by* Artifact 1.

Inverse Usage: **Artifact** 1 is the ***source of*** **Statement** A.

3.4.11.3. Requirement (Statement)

Definition: A **Requirement** entity identifies a capability, characteristic, or quality factor of a system that must exist for the system to have value and utility to the user. Example: pump shall weigh no more than 1.2 kilograms.

Parent Class: **Statement**

3.4.11.3.1. Requirement Attributes

Requirement attributes include: *name*, *number*, *description*, *rationale*, and *type*.

Note the quality attributes below are optional. Other sets of quality attributes, such as the ones defined in the INCOSE Requirements Guide may be provided by the tool developer, or these may be user-definable. However, some form of quality attributes is recommended.

Attribute	Type	Description
clear	Boolean	*clear* represents if this **Requirement** entity is unambiguous and not confusing.
complete	Boolean	*complete* represents if this **Requirement** entity expresses a whole idea.
consistent	Boolean	*consistent* represents if this **Requirement** entity is not in conflict with other requirements.
correct	Boolean	*correct* represents if this **Requirement** entity describes the user's true intent and is legally possible.
design	Boolean	*design* represents if this **Requirement** entity does not impose a specific solution on design; says "what", not "how".
feasible	Boolean	*feasible* represents if this **Requirement** entity can be implemented with existing technology, within cost and schedule.
modular	Boolean	*modular* represents if this **Requirement** entity can be changed without excessive impact on other requirements.
traceable	Boolean	*traceable* represents if this **Requirement** entity is uniquely identified, and able to be tracked to predecessor and successor lifecycle items/objects.
verifiable	Boolean	*verifiable* represents if this **Requirement** entity is provable (within realistic cost and schedule) that the system meets the requirement.

3.4.11.3.1.1. rationale

Definition: *rationale* provides a place to capture the reason(s) behind this **Requirement** entity.

Data Type: Text

3.4.11.3.1.2. type (Statement)

Definition: *type* provides aliases for the entities. For **Requirement** entities these can include: Functional Requirement, Safety Requirement, Support Requirement, Verification Requirement

Data Type: Text

3.4.11.3.2. Requirement Relationships

No additional relationships are defined for the **Requirement** class entities. The Input/Output class uses all the common relationships provided in section 3.4.0.2 and the relationships found in section 3.4.11.2.

Missing satisfied by and verified by?

3.4.11.4. Verification Requirement (Statement)

Description: A **Verification Requirement** entity specifies what is required to confirm that a requirement is satisfied.

Parent Class: **Statement**

3.4.11.4.1. Verification Requirement Attributes

Verification Requirement attributes include: name, number, description, acceptance criteria, evidence, rationale, and verification method.

3.4.11.4.1.1. acceptance criteria

Definition: *acceptance criteria* state what results must be achieved to be considered satisfactory. These criteria should be apparent from the requirement statement and appropriate for the method of verification selected.

Data Type: Text

3.4.11.4.1.2. evidence

Definition: *evidence* specifies the type of documented or otherwise validated information and materials required to be obtained and delivered in order to confirm that a requirement was satisfied. For example, “Material test report with sample coupons”, “Inspection record recording actual geo-location measurements”, “Weld Radiographs”, “Analysis report based on results from a qualified model”, “written supplier warrantee.”

Data Type: Text

3.4.11.4.1.3. rationale

Definition: rationale captures the reasoning for the verification requirement properties.

Data Type: Text

3.4.11.4.1.4. verification method

Definition: The method used by the V&V specialist to verify that the system or its components meet requirements.

Data Type: Enumeration. Minimum values: “analysis”, “inspection”, “demonstration”, “test”. Other possibilities include “modeling and simulation”, “design review”, “supplier guarantee”. The user needs to be able to add to this list.

3.4.11.4.2. Verification Requirement Relationships

Name	Classes	Description
verifies	Requirement	Verifies a Requirement is satisfied by an Asset
traced to	Verification Event	Traced to a Verification Event to support planning activities
verifies	Asset	Verifies an Asset satisfies a requirement

3.4.12. Time

Definition: A **Time** class entity specifies a point or period when an action, asset, process, or condition exists, finishes, starts, or continues. Example: Milestone A, Phase 2

3.4.12.1. Time Attributes

Time attributes include: *name*, *number*, *description*, *duration*, *evidence*, *end*, *start*, and *type*.

3.4.12.1.1. duration

Definition: *duration* represents the period of time this **Time** entity occurs. A zero (0) duration indicates a milestone.

Data Type: Number

3.4.12.1.2. end

Definition: *end* represents the time when this **Time** entity finishes. It can be computed from the start and duration attributes.

Data Type: DateTime

3.4.12.1.3. start (Time)

Definition: *start* represents the time when this **Time** entity begins.

Data Type: DateTime

3.4.12.1.4. type (Time)

Definition: *type* provides aliases for the entities. For **Time** entities these can include: Duration, Milestone, Point In Time, Time Frame

Data Type: Text

3.4.12.2. Time Relationships

The additional relationships for the **Time** class entities are provided below. The Time class uses all the common relationships provided in section 3.4.0.2.

3.4.12.2.1. date resolves/date resolved by

Definition: ***date resolves*** identifies the **Decision** entity that is closed at this **Time** entity.

Usage: Time A ***date resolves*** Decision 1.

Inverse Usage: Decision 1 ***date resolved by*** Time A.

3.4.12.2.2. decided by/decision due

Definition: ***decided by*** identifies the **Decision** entity scheduled for closure at this **Time** entity.

Usage: Time A must be ***decided by*** Decision 1?

Inverse Usage: Decision 1 ***decision*** (is) ***due*** by Time A.

4. Visualizations

The following diagrams represent the common forms of visualizing information. They do not attempt to encompass every possible visualization. Only one is unique to LML: the Action Diagram. Many similar models have been developed over the years to express functional sequencing, such as the Flow Charts,

Activity Diagram in UML/SysML, Business Process Modeling Notation (BPMN), and others. Although these various notations are accurate, they use many different symbols, which often confuse the non-expert viewers/recipients of the visualization. As seen below, the visualization of this functional sequencing in LML is much simpler, but it appears to have all the necessary information for language execution. The usual constructs are replaced by special cases of Actions to denote decision points.

The other visualizations should be considered standard diagrams, used over many years by different techniques. We also provide suggested diagrams that LML users may want to consider as well. A few do not appear to require visualization beyond a hierarchy diagram. We denote which diagrams are appropriate for which classes.

4.1. Action Diagram (Mandatory for Action entities with children)

The Action Diagram (see Figure 4-1) represents the functional sequencing of **Actions** along with the data flow provided by the **Input/Output** entities. This combination of **Actions** with **Input/Outputs** is similar to the SREM "Behavior Diagram" or UML Activity Diagram. Without the **Input/Output** entities, the Action Diagram would be the equivalent of the classic Function Flow Block Diagram (FFBD) or IDEF 3. The main difference between LML and these other diagrams is the use of special kinds of **Actions** to replace the constructs used in these other languages. The construct set is shown below.

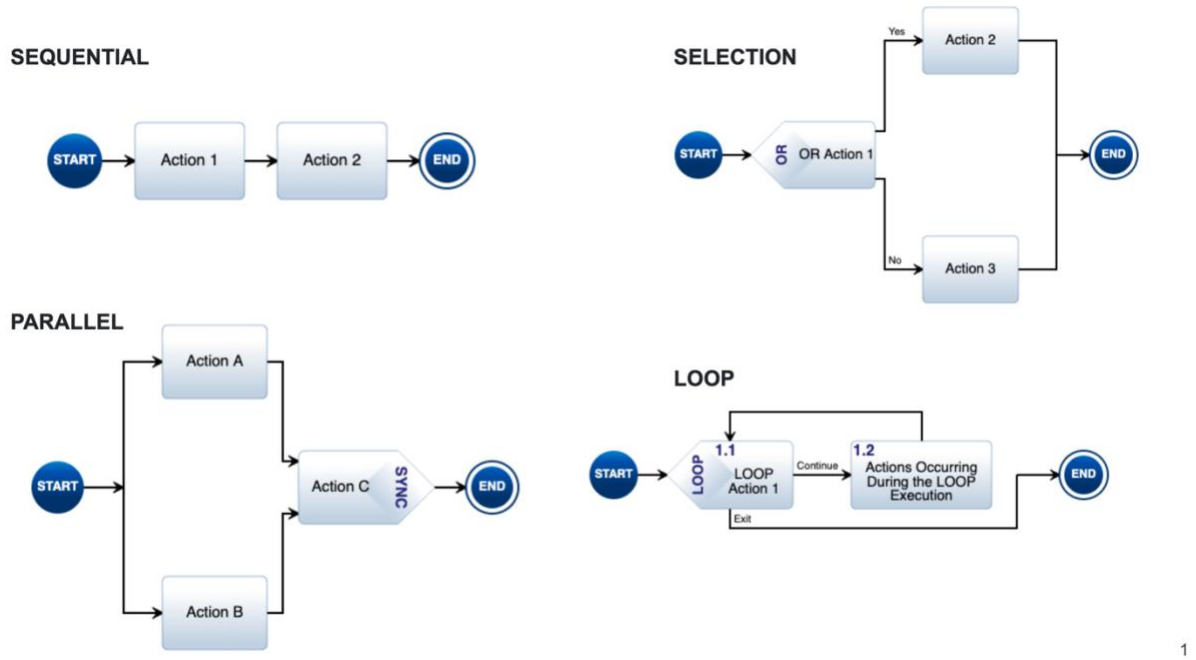


Figure 4-1. Special Actions for the Action Diagram to Capture Decision Points.

The special cases of **Actions**, denoted by the rectangle with a diamond embedded in it (showing ½ the diamond as a point on the rectangle), represent decision points. For example, in a loop the key decision is the exit criteria for the loop. This criterion can be as simple as the number of iterations of the loop or more complex logic that determines when the loop must stop.

The "OR" decision point represents an exclusive selection of one path or the other. The decision point in the case can be a probability or a specific criterion for path selection.

The final decision point type is the “SYNC.” The SYNC provides the functional rationale for ending parallel branches. Note that in the physical view, two separate entities can exist without any synchronization between them. However, in the functional view between two physical entities that are interacting, it is necessary decide how to terminate that interaction. We see this in software that when two parallel processes are spawned, these threads must be synchronized to complete the program.

The Action Diagram can include **Input/Output** entities as well. An example of this inclusion is shown in Figure 4-2.

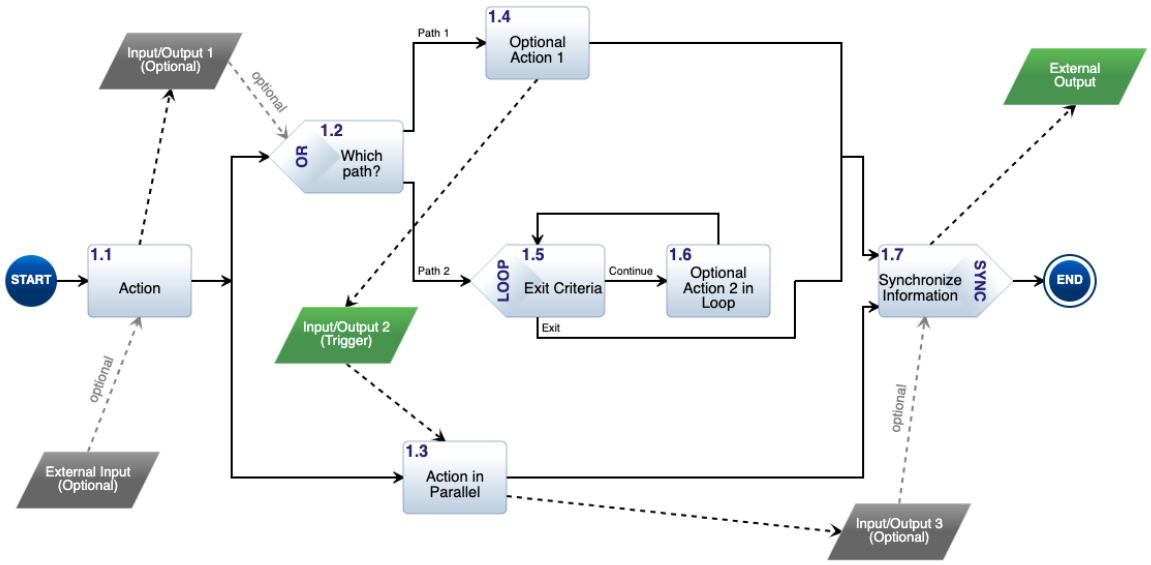


Figure 4-2. An Example Action Diagram with Inputs/Outputs.

The **Input/Output** entities are shown as parallelograms, reminiscent of the classical flow chart symbols used in the 1950s and 1960s. Two colors (or some other mechanism) are used to distinguish triggering **Input/Outputs** from optional **Input/Outputs**. It is recommended that a different type of line be used to show the **Input/Output** flow, thus making it easy to distinguish between the data flow and functional sequencing lines. The diagram should also contain a way to show the **Start** and **End** of the functional sequencing.

The Action Diagram can also explicitly show resources being consumed, produced, or seized. Figure 4-3 show the Resource entities as hexagrams.

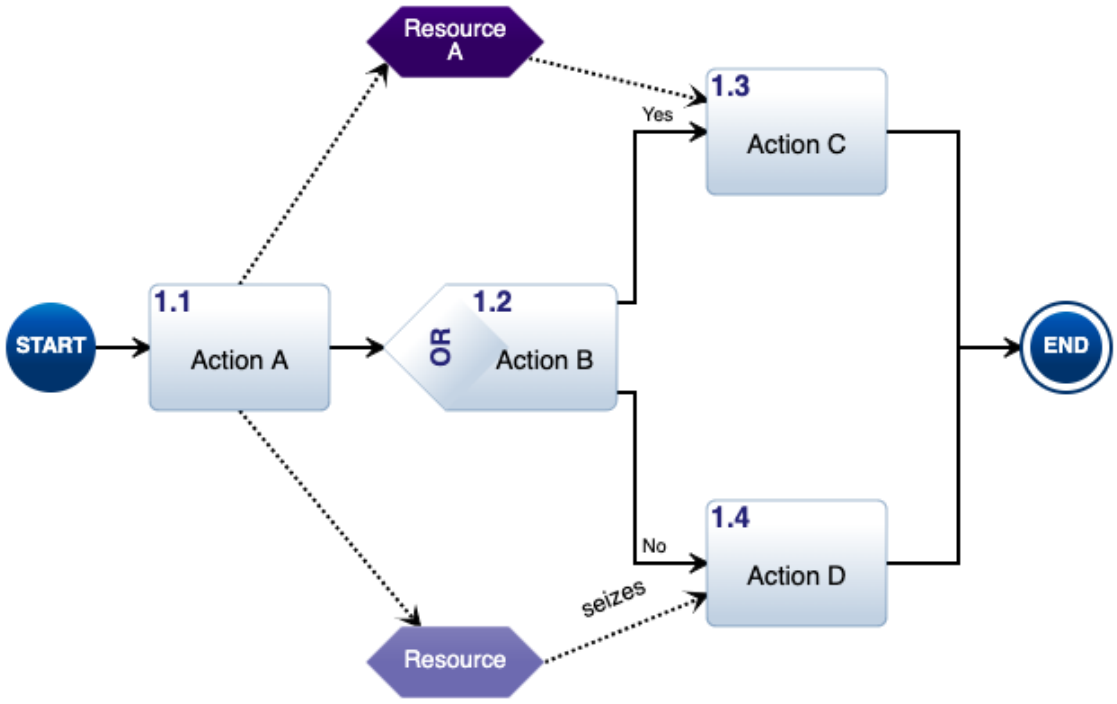


Figure 4-3. An Example Action Diagram with Resources.

No other constructs have been determined to be necessary. Other languages include a “Replicate,” however the research done by the LML developers indicated that it was a way to identify a physical instantiation of the functional entity in more than one physical entity at a time. This representation appears more appropriately in the physical diagram (see Asset Diagram in next section).

While this specification does not directly specify or standardize the Action Diagram, the constructs above are to be used as guidance for the overall look of an Action Diagram. The key takeaway is the inclusion of logic flow into the Action entities themselves, instead of relying on separate logic flow diagram elements.

4.2. Asset Diagram (Mandatory for Asset entities with children)

LML must have a physical representation of design elements as well as the functional one provided by the Action Diagram. Figure 4-4 shows one way of providing this information. The **Assets** are shown as rectangles, with **Conduits** displayed as lines connecting the **Assets**. Since **Resource** is a subclass of **Asset**, they could also be displayed by this diagram. Other information, such as the capacity and latency attributes of the **Conduit**, may be overlaid on this diagram as well. Also, since the **Asset** *type* may interest most users, it should be desirable to include that attribute on this diagram as well.

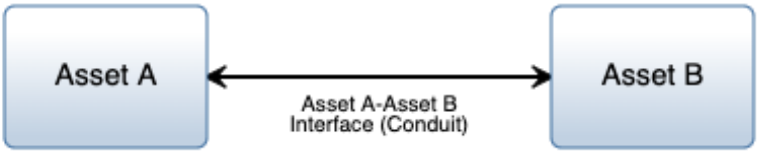


Figure 4-4. A Simple Asset Diagram.

Nodes and connections could be replaced with pictures as shown in Figure 4-5.

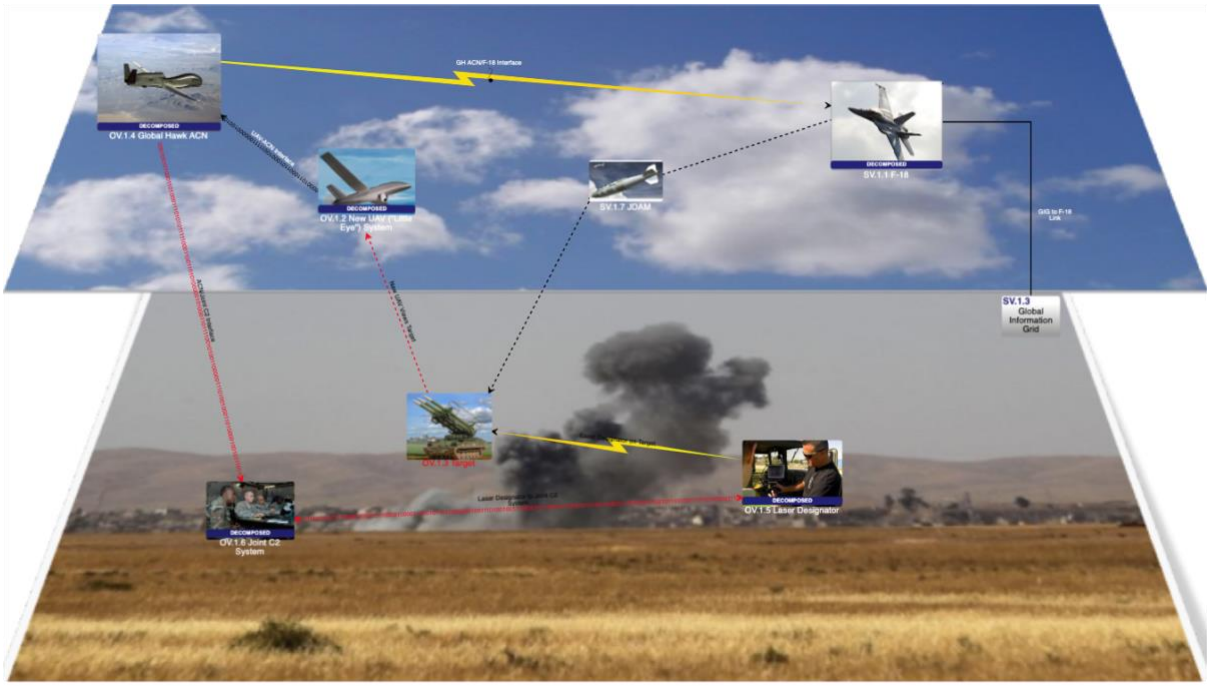


Figure 4-5. Asset Diagram with Pictures and Special Lines for Conduits.

The Asset Diagram is mandatory only for those Assets with children.

4.3. Spider Diagram (Mandatory for Traceability)

The spider diagram shows how entities are related to one another. This diagram is similar to the ERD shown above, but reflects LML's schema, not an abstract schema. This diagram (see Figure 4-6) shows traceability of LML entities with their relationships.

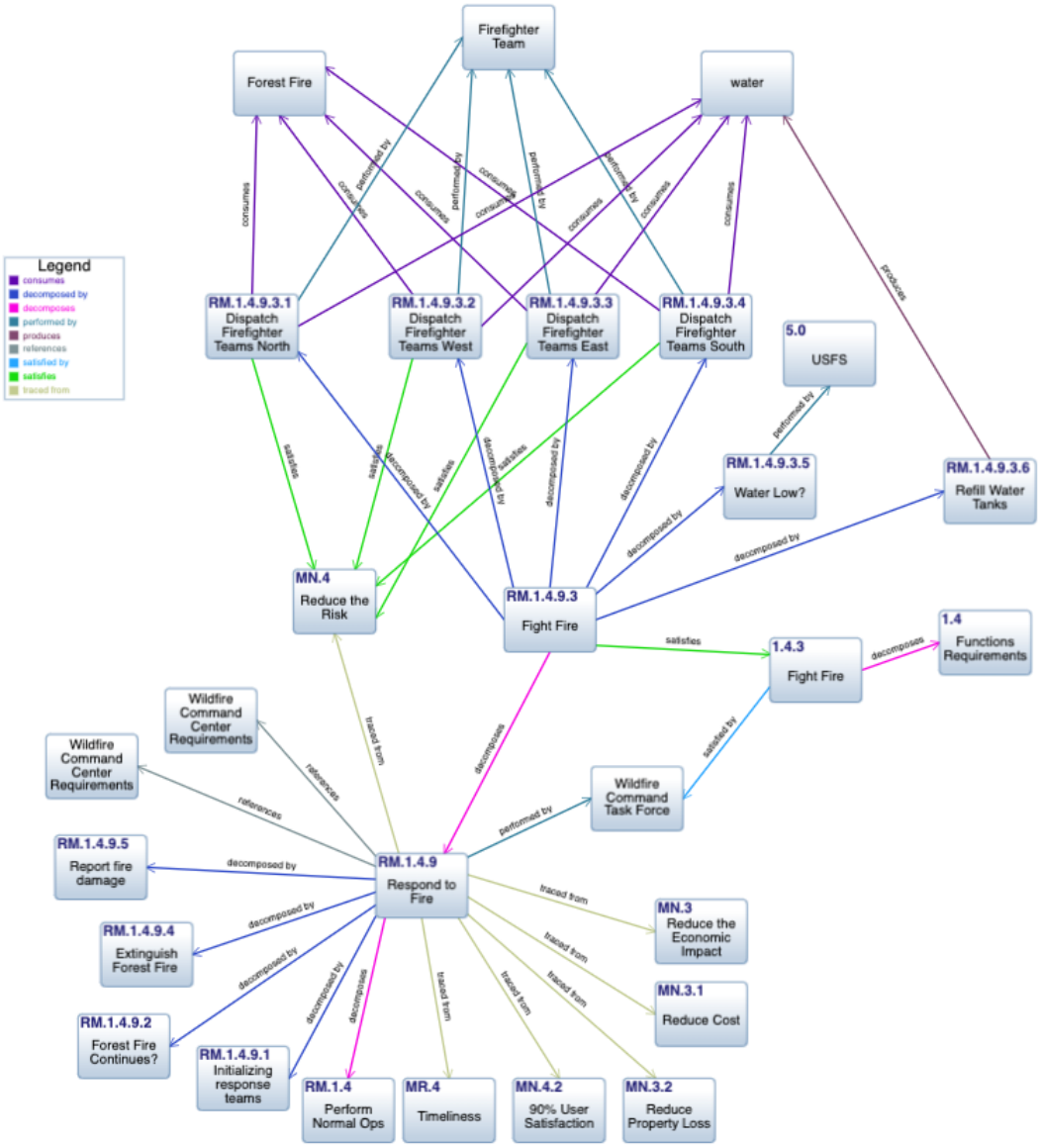


Figure 4-6. Example Spider Diagram to Show Traceability between Entities Using the LML Relationships.

4.4. Interface Control Diagram (Mandatory)

A number of interface diagram types have been suggested over the years. Most of these diagrams come from drawing approach that show different aspects of the interface. LML's Asset Diagram provides part of an interface definition by showing how **Assets** are connected by **Conduits**. But more information is needed to completely define an interface, including the *capacity* and *latency*, as well as the **Input/Output** entities that are transferred across the **Conduit**. **Conduits** can also be decomposed, so parent-child relationships need a means of being visualized too. Of course other information, such as protocols used, will more completely define the interface, but at some point the diagrams get to be too cluttered to be useful in helping the stakeholders understanding of the interface. Figure 4-7 shows a potential Interface Diagram that displays most of the information needed to define an Interface.

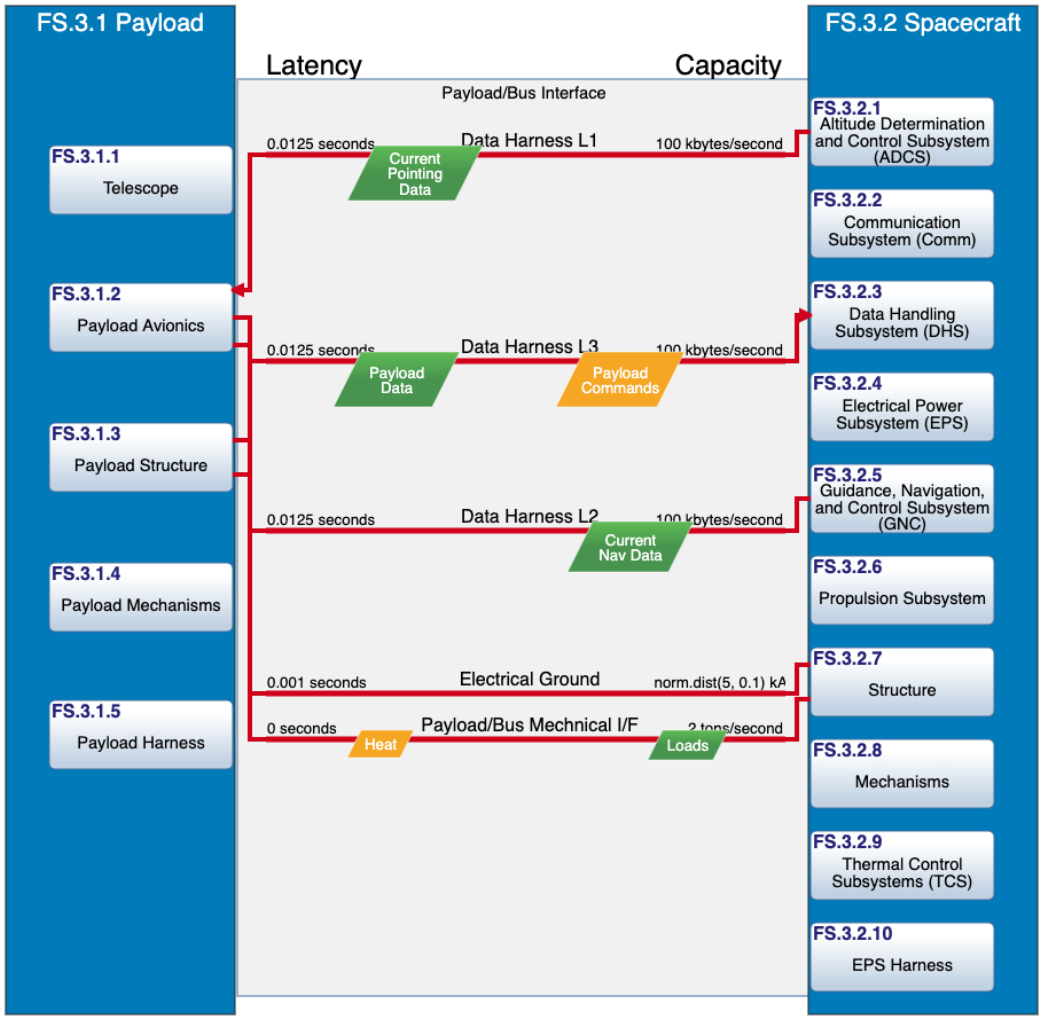


Figure 4-7. Interface Control Diagram

This figure shows the two **Assets** (Spacecraft and Payload) with the **Conduit** (Payload/Bus Interface) between them. The next level of decomposition for both the **Assets** and the **Conduits** are also shown. The child **Conduits** are attached to the child **Assets**. Note that an arrow can be used to depict the directionality of the child **Conduit**. The *latency* and *capacity* values of each child **Conduit** are shown as well. The green and gold items on the **Conduits** in this diagram depict the **Input/Output** entities transferred by the child **Conduits**. The difference between a green and gold **Input/Output** in this case indicate a potential error in the *size* units of the **Input/Output** entity. While error detection is not required for this diagram, such capabilities are desirable in all diagrams, where appropriate.

1 INCOSE defines a system as a *"*combination of interacting elements organized to achieve one or more stated purposes."

4.5 Example Views for Other LML Entities

LML was designed to support the full lifecycle. All of its entities can support the common visualizations that architects, systems engineers, software engineers, test engineers, operators, logisticians, and program managers use. The examples below are suggestions of how to implement these common visualizations using LML.

4.5.1 Class Diagram

The diagram below shows how to implement the UML Class Diagram using LML classes. Since the Class Diagram has become a standard for software development, it seems a good candidate for inclusion in LML's approach. In this diagram, the LML entities that match the diagram elements include: **Asset** with the *type* "object"; attributes are captured as **Characteristics**; methods are **Actions**; and relationships are **Logical** connections.

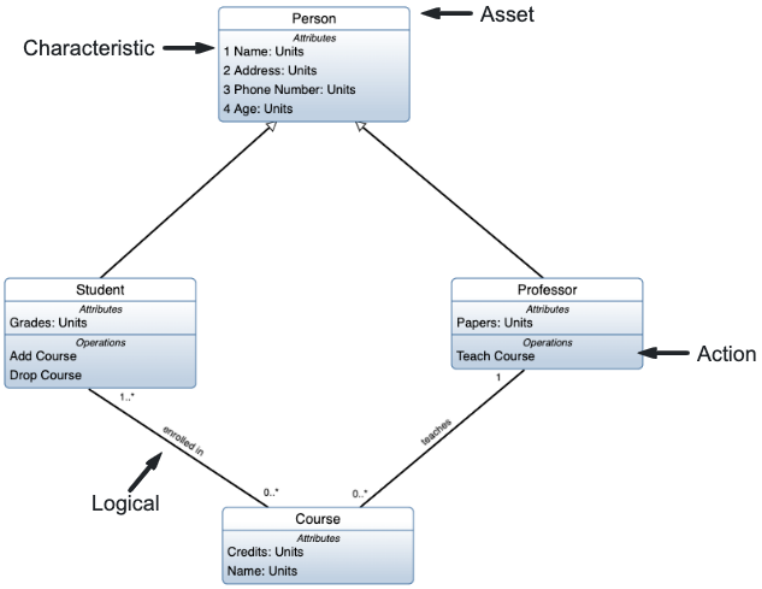


Figure 4-6. LML Encourages Use of the Class Diagram and Provides the Schema Entities to Support Its Creation.

4.5.2. ERAA Diagram

Another way to model information has been the classic entity-relationship diagram (ERD) or entity-relationship-attribute (ERA) diagram. These define the entities, relationships, and attributes of the entity only. Since LML has added attributes on the relationship, we have extended it to an ERAA diagram. An example of this is shown below using a class diagram. The relationships are expressed using the **Logical Connection**. Entities are represented by **Assets**. You can also capture attributes as **Characteristics**. The relationship attribute is shown under the relationship name in parentheses.

Figure 4-7. ERAA Diagram Developed Using the Class Diagram.

4.5.3 Timelines

The **Actions** and **Times** can be displayed using a classic Gantt Chart shows how functional elements execute over time. An example of this is shown below.

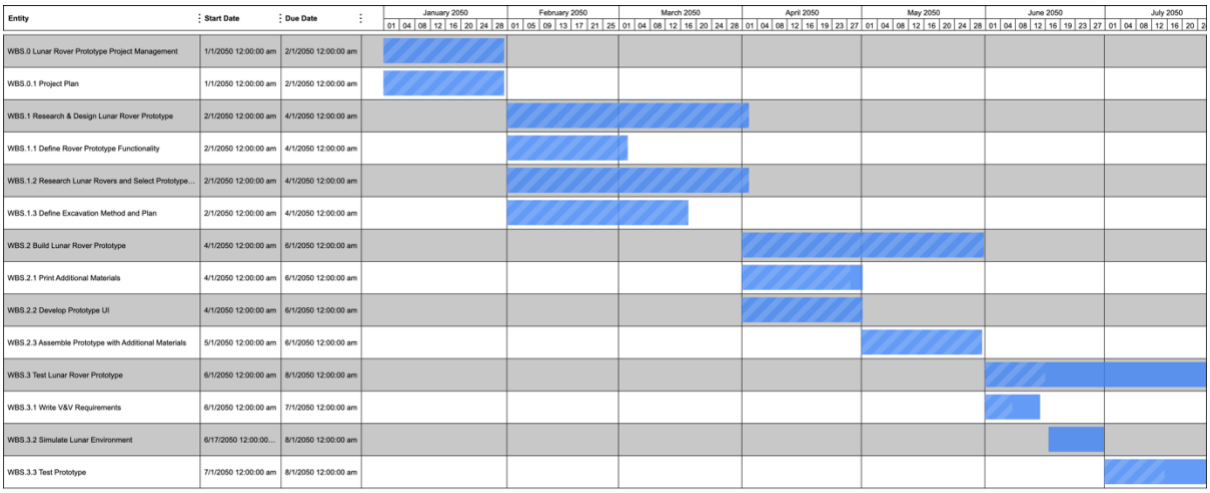


Figure 4-8. Use of Gantt Chart for Displaying Actions and Durations.

Time with **Actions** or **Assets** can also be visualized in many other ways. One of the most useful is shown below.

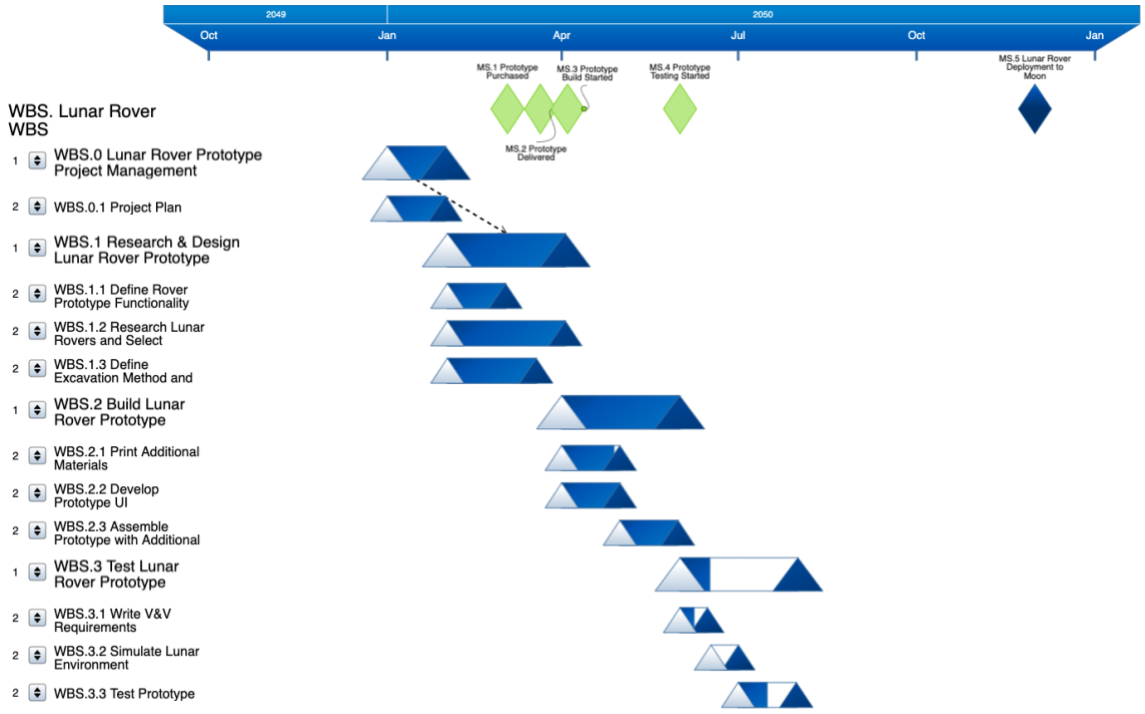


Figure 4-9. Timeline Diagram Showing Actions at Specific Times.

4.5.4. Hierarchy Diagram

A hierarchy chart is used in LML to show decomposition of elements. The figure below shows an example of requirements decomposition. This chart uses the ***decomposed by*** relationship.

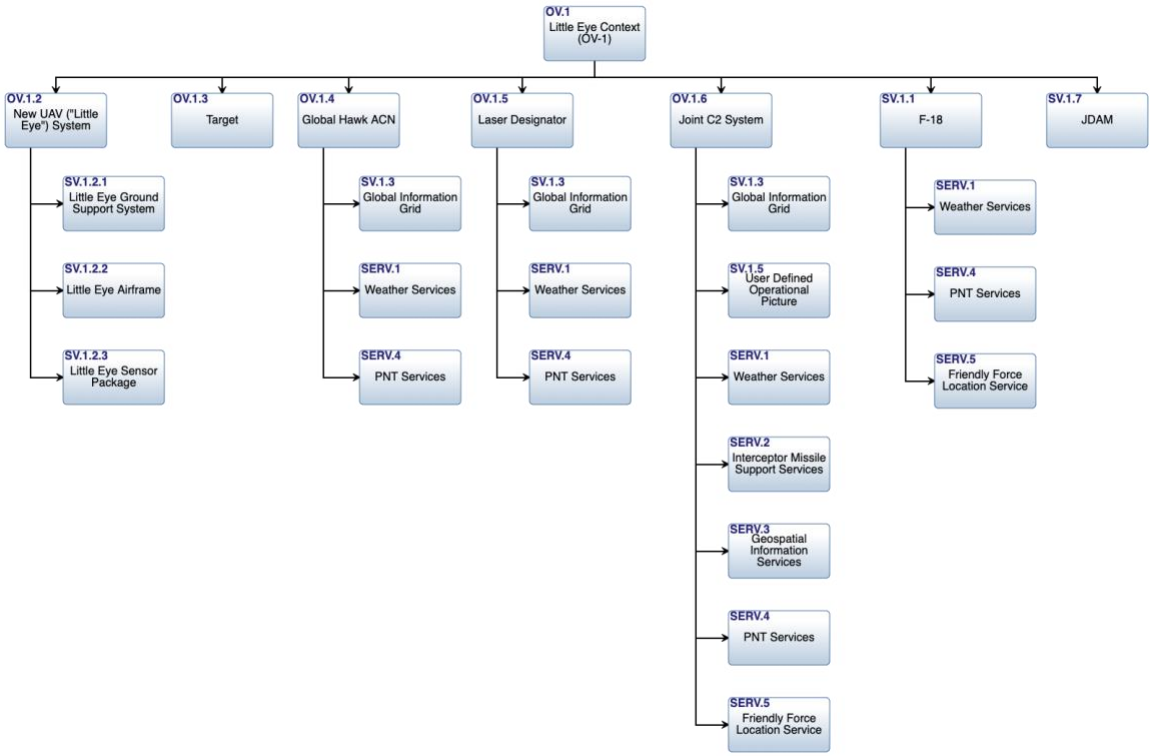


Figure 4-10. A Hierarchy Diagram Is a Good Way to Show Decomposition.

4.5.5. Risk Matrix

A standard DoD risk matrix (shown below) or other form can be used to display **Risk** entity information. Another type of risk analysis uses probabilities to create a fault-tree. A fault tree is often shown as a hierarchy diagram with the probabilities shown for each branch. An Action Diagram can also support fault tree analysis.

Figure 4-11. Typical Risk Matrix.

4.5.6. State Machine Diagram

The state machine diagram expresses how an **Asset** transitions from one state to another. In the diagram below, the state (or **Characteristic**) transition occurs when the **Action** entity event causes the transition to the other state.

Name	Class	Description
State	Characteristic	Means that it's a state of the system
Initial State	Characteristic	Means that it's the initial state
Final State	Characteristic	Means that it's the final state.

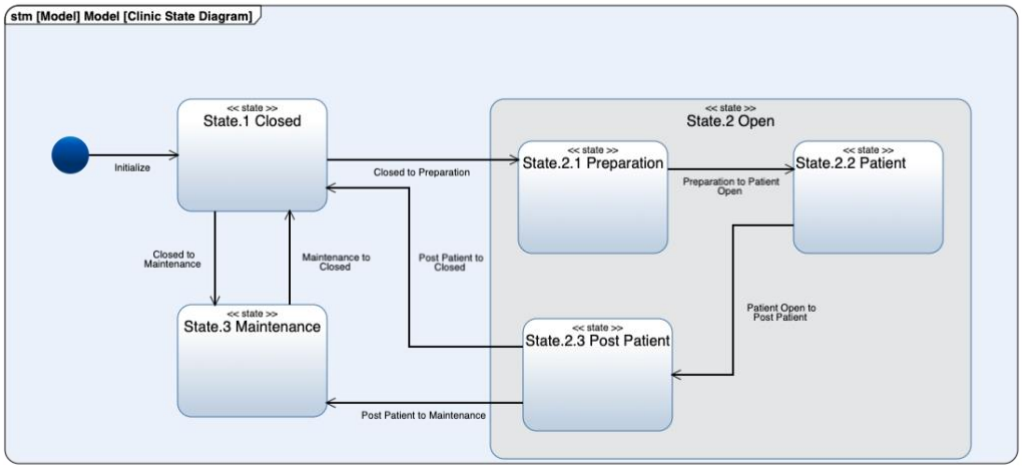


Figure 4-12. The State Machine Diagram Has Proven Useful and LML Supports It.

Appendix A. SysML v1.X Mapping to LML

SysML focuses mainly on diagrams, with an underlying ontology embedded in the XMI that each diagram represents. Currently (October 2013) a complete ontology is under development. The table below shows the various SysML diagrams and the LML equivalent diagram and associated classes.

Table A-1. SysML Diagram Mapping to LML Diagrams and Ontology

SysML Diagram	LML Diagram	LML Entities
Activity	Action Diagram	Action, Input/Output
Sequence	Sequence	Action, Asset
State Machine	State Machine	Characteristic (State), Action (Event)
Use Case	Asset Diagram	Asset, Connection
Block Definition	Class Diagram, Hierarchy Chart	**Input/Output** (Data Class), **Action** (Method), **Characteristic** (Property)
Internal Block	Asset Diagram	Asset, Connection
Package	Asset Diagram	Asset, Connection
Parametric	Hierarchy, Spider, Radar	Characteristic
Requirement	Hierarchy, Spider	**Requirement** and related entities

Although the Systems Modeling Language (SysML) does not have an official ontology many tool vendors have created these models from database schemas. The purpose of this appendix is to identify the entities, relationships and attributes necessary to completely visualize the SysML models in LML. The SysML 1.4 standard is available at http://www.omg.org/spec/SysML/... (as of 1 December 2015). This appendix will only address the changes to LML required to produce these models.

Note that significant changes between the Action Diagram and the Activity Diagram must occur if developers want to adhere to all the SysML requirements, since Action Diagrams do not contain the large number of constructs used in an Activity Diagram. This specification does not recommend such a large number of constructs as they impede understanding of the diagram. The same content is provided by the ontology.

SPEC Innovations' Innoslate® tool was used as the basis for this extension. Innoslate added one class (**Equation**) and one subclass to **Asset** (**Port**) to visualize the complete set of SysML models. Numerous relationships were added to accommodate the SysML visualizations. The changes to Innoslate's LML schema for SysML are shown in the tables below.

Table A-2. New Classes

Class	Parent	Description
Equation		An Equation entity specifies an equation (mathematical or logical) that can be used to describe a part of the model.
Port	Asset	An interaction point of a block, specifying the input and output flow.

Equation Class

An Equation entity specifies an equation (mathematical or logical) that can be used to describe a part of the model.

Table A-3. Equation Properties

Name	Type	Description
Value	Text	*Value* represents this Equation's text.

Table A-4. Equation Relations

Name	Classes	Description
decomposed by	Equation	*Decomposed by* identifies the children of this entity.
decomposes	Equation	*Decomposes* identifies the parent of this entity.
equation for	Cost	*Equation for* identifies the entity that this Equation represents.
equation for	Time	*Equation for* identifies the entity that this Equation represents.
equation for	Characteristic	*Equation for* identifies the entity that this Equation represents.
equation for	Statement	*Equation for* identifies the entity that this Equation represents.
equation for	Decision	*Equation for* identifies the entity that this Equation represents.
equation for	Action	*Equation for* identifies the entity that this Equation represents.
equation for	Risk	*Equation for* identifies the entity that this Equation represents.
equation for	Location	*Equation for* identifies the entity that this Equation represents.
equation for	Asset	*Equation for* identifies the entity that this Equation represents.
equation for	Artifact	*Equation for* identifies the entity that this Equation represents.
equation for	Connection	*Equation for* identifies the entity that this Equation represents.
equation for	Input/Output	*Equation for* identifies the entity that this Equation represents.
has variable	Characteristic	*Has variable* identifies the Characteristic that is represented in this Equation.
related to	Equation	*Related to* identifies the entity that ties in a peer-to-peer way with this entity.
relates	Equation	*Relates* identifies the peer-to-peer entity that is tied to this entity.

Port Class

An interaction point of a block, specifying the input and output flow.

Table A-5. Port Properties

Name	Type	Description
Direction	Enumeration	*Direction* represents the flow of data on this port.

Table A-6. Action Relations

Name	Classes	Description
depends on	Action	*Depends on* identifies the Action that this Action has a dependency on.
equation of	Equation	*Equation of* identifies the Equation that represents this entity.
extend	Action	*Extend* identifies the Action (use case) that is added to this Action (use case).
extended by	Action	*Extended by* identifies the Action (use case) that is added from this Action (use case).
fetches	Characteristic	*Fetches* identifies the State Characteristic that this Action receives.
has dependent	Action	*Has Dependent* identifies the Action that depends on this Action.
include	Action	*Include* identifies the Action (use case) that is added to this Action (use case).
included by	Action	*Included by* identifies the Action (use case) that is added from this Action (use case).
pushed by	Characteristic	*Pushed by* identifies the State Characteristic that this Action is generated from.
satisfies	Requirement	*Satisfies* identifies the Requirement that is fulfilled by this entity.
verifies	Requirement	*Verifies* identifies the Requirement that is supported by this entity.

Table A-7. Artifact Relations

Name	Classes	Description
equation of	Equation	*Equation of* identifies the Equation that represents this entity.
satisfies	Requirement	*Satisfies* identifies the Requirement that is fulfilled by this entity.
verifies	Requirement	*Verifies* identifies the Requirement that is supported by this entity.

Table A-8. Asset Relations

Name	Classes	Description
equation of	Equation	*Equation of* identifies the Equation that represents this entity.
extend	Asset	*Extend* identifies the Action (use case) that is added to this Action (use case).
extended by	Asset	*Extended by* identifies the Action (use case) that is added from this Action (use case).
instantiated by	Asset	*Instantiated by* identifies another entity that is an instance of this entity.
instantiates	Asset	*Instantiates* identifies another entity that has this entity as an instance.
represented in	Asset	*Represented in* identifies the Asset whose diagrams include this Asset.
represents	Asset	*Represents* identities the Asset that is included in this Asset's diagrams.
satisfies	Requirement	*Satisfies* identifies the Requirement that is fulfilled by this entity.
verifies	Requirement	*Verifies* identifies the Requirement that is supported by this entity.

Table A-9. Characteristic Relations

Name	Classes	Description
equation of	Equation	*Equation of* identifies the Equation that represents this entity.
fetched by	Action	*Fetched by* identifies the Action that this State Characteristic is received by.
generates	Input/Output	*Generates* identifies the Input/Output or Action that this Action or Characteristic transforms.
generates	Action	*Generates* identifies the Input/Output or Action that this Action or Characteristic transforms.
instantiated by	Characteristic	*Instantiated by* identifies another entity that is an instance of this entity.
instantiates	Characteristic	*Instantiates* identifies another entity that has this entity as an instance.
pushes	Action	*Pushes* identifies the Action that this State Characteristic generates.
satisfies	Requirement	*Satisfies* identifies the Requirement that is fulfilled by this entity.
variable of	Equation	*Variable of* identifies the Equation that this Characteristic is represented in.
verifies	Requirement	*Verifies* identifies the Requirement that is supported by this entity.

Table A-10. Conduit Relations

Name	Classes	Description
instantiated by	Conduit	*Instantiated by* identifies another entity that is an instance of this entity.
instantiates	Conduit	*Instantiates* identifies another entity that has this entity as an instance.

Table A-11. Connection Relations

Name	Classes	Description
equation of	Equation	*Equation of* identifies the Equation that represents this entity.
satisfies	Requirement	*Satisfies* identifies the Requirement that is fulfilled by this entity.
verifies	Requirement	*Verifies* identifies the Requirement that is supported by this entity.

Table A-12. Cost Relations

Name	Classes	Description
equation of	Equation	*Equation of* identifies the Equation that represents this entity.
satisfies	Requirement	*Satisfies* identifies the Requirement that is fulfilled by this entity.
verifies	Requirement	*Verifies* identifies the Requirement that is supported by this entity.

Table A-13. Decision Relations

Name	Classes	Description
equation of	Equation	*Equation of* identifies the Equation that represents this entity.

Table A-14. Input/Output Relations

Name	Classes	Description
equation of	Equation	*Equation of* identifies the Equation that represents this entity.
instantiated by	Input/Output	*Instantiated by* identifies another entity that is an instance of this entity.
instantiates	Input/Output	*Instantiates* identifies another entity that has this entity as an instance.
satisfies	Requirement	*Satisfies* identifies the Requirement that is fulfilled by this entity.
verifies	Requirement	*Verifies* identifies the Requirement that is supported by this entity.

Table A-15. Location Relations

Name	Classes	Description
satisfies	Requirement	*Satisfies* identifies the Requirement that is fulfilled by this entity.
verifies	Requirement	*Verifies* identifies the Requirement that is supported by this entity.

Table A-16. Requirement Relations

Name	Classes	Description
copied by	Requirement	*Copied by* identifies the Requirement that is a clone of this Requirement.
copies	Requirement	*Copies* identifies the Requirement that this Requirement clones.
derived by	Requirement	*Derived by* identifies the Requirement that is developed from this Requirement.
derives	Requirement	*Derives* identifies the Requirement that this Requirement develop from.
refined by	Requirement	*Refined by* identifies the Requirement that clarifies this Requirement.
refines	Requirement	*Refines* identifies the Requirement that this Requirement clarifies.
satisfied by	Cost	*Satisfied by* identifies the entity that fulfills this Requirement.
satisfied by	Time	*Satisfied by* identifies the entity that fulfills this Requirement.
satisfied by	Action	*Satisfied by* identifies the entity that fulfills this Requirement.
satisfied by	Artifact	*Satisfied by* identifies the entity that fulfills this Requirement.
satisfied by	Asset	*Satisfied by* identifies the entity that fulfills this Requirement.
satisfied by	Location	*Satisfied by* identifies the entity that fulfills this Requirement.
satisfied by	Input/Output	*Satisfied by* identifies the entity that fulfills this Requirement.
satisfied by	Characteristic	*Satisfied by* identifies the entity that fulfills this Requirement.
satisfied by	Connection	*Satisfied by* identifies the entity that fulfills this Requirement.
verified by	Input/Output	*Verified by* identifies the entity that supports this Requirement.
verified by	Characteristic	*Verified by* identifies the entity that supports this Requirement.
verified by	Asset	*Verified by* identifies the entity that supports this Requirement.
verified by	Artifact	*Verified by* identifies the entity that supports this Requirement.

Name	Classes	Description
verified by	Time	*Verified by* identifies the entity that supports this Requirement.
verified by	Action	*Verified by* identifies the entity that supports this Requirement.
verified by	Requirement	*Verified by* identifies the entity that supports this Requirement.
verified by	Cost	*Verified by* identifies the entity that supports this Requirement.
verified by	Connection	*Verified by* identifies the entity that supports this Requirement.
verified by	Location	*Verified by* identifies the entity that supports this Requirement.
verifies	Requirement	*Verifies* identifies the Requirement that is supported by this entity.

Table A-17. Risk Relations

Name	Classes	Description
equation of	Equation	*Equation of* identifies the Equation that represents this entity.

Table A-18. Statement Relations

Name	Classes	Description
equation of	Equation	*Equation of* identifies the Equation that represents this entity.

Table A-19. Time Relations

Name	Classes	Description
equation of	Equation	*Equation of* identifies the Equation that represents this entity.
satisfies	Requirement	*Satisfies* identifies the Requirement that is fulfilled by this entity.
verifies	Requirement	*Verifies* identifies the Requirement that is supported by this entity.

Appendix B. DoDAF MetaModel 2.0.1 (DM2) Mapping to LML

The Department of Defense has developed a schema called the DoDAF (DoD Architecture Framework) MetaModel 2.0.1 (DM2). The Conceptual Data Model is shown below. The physical data model contains over 500 entries. As shown below, this is a specialized model that focuses mainly on the DoD nomenclature and may be less useful in other domains. For example, since the DoD has developed their acquisition process around the concept of "Capability," that becomes a critical item in the top level of this schema, thus driving additional relationships. In LML we identified that capability could be a type of **Action**, **Asset**, **Characteristic** or even a **Statement**. However, LML does not preclude the user or tool vendor from adding "Capability" as an entity class. Schema extensions are actually encouraged for different domains. As such extensions are made and standardized; the LML Steering Committee will consider adding them as extensions to LML.

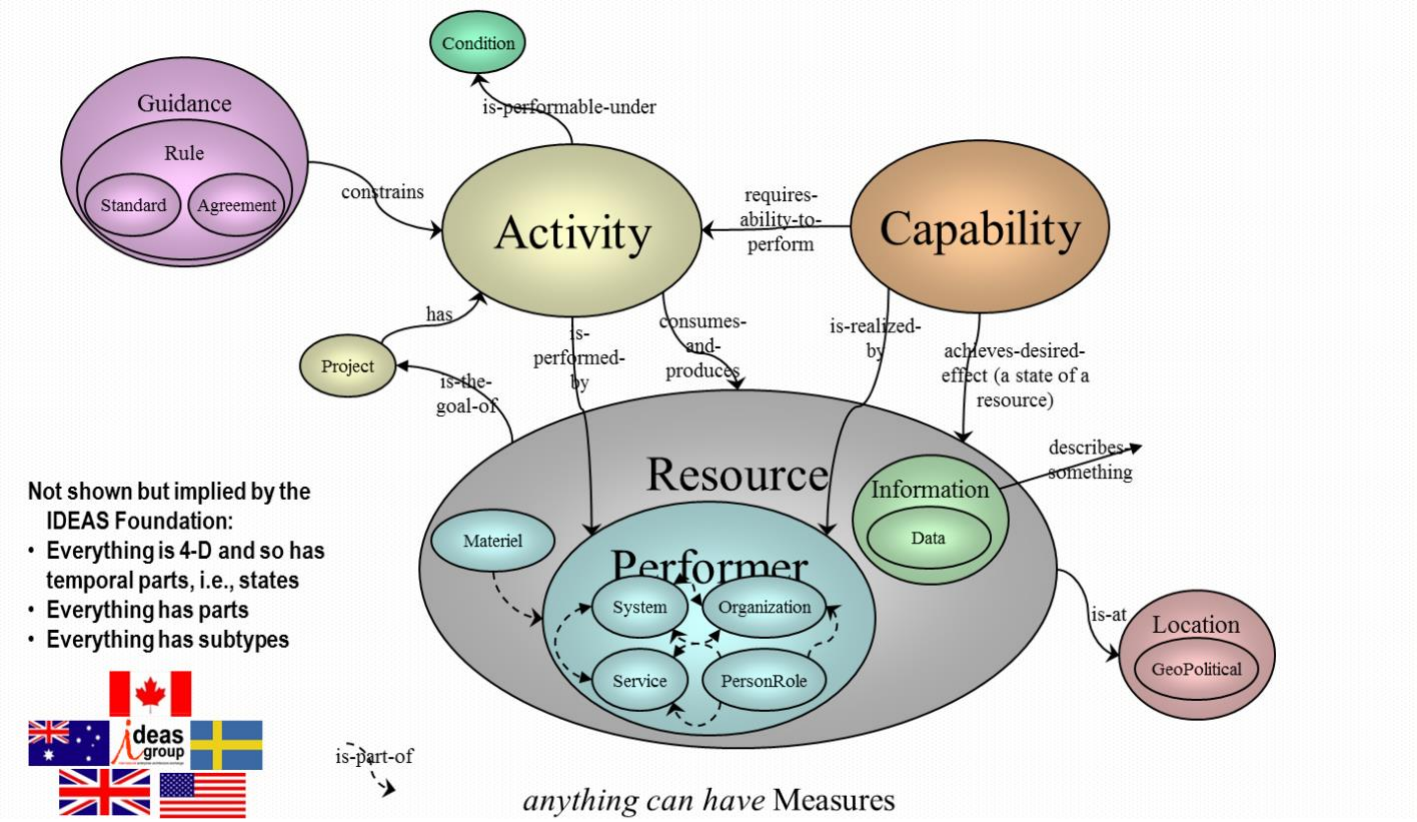


Figure B-1. DM2 Conceptual Data Model.

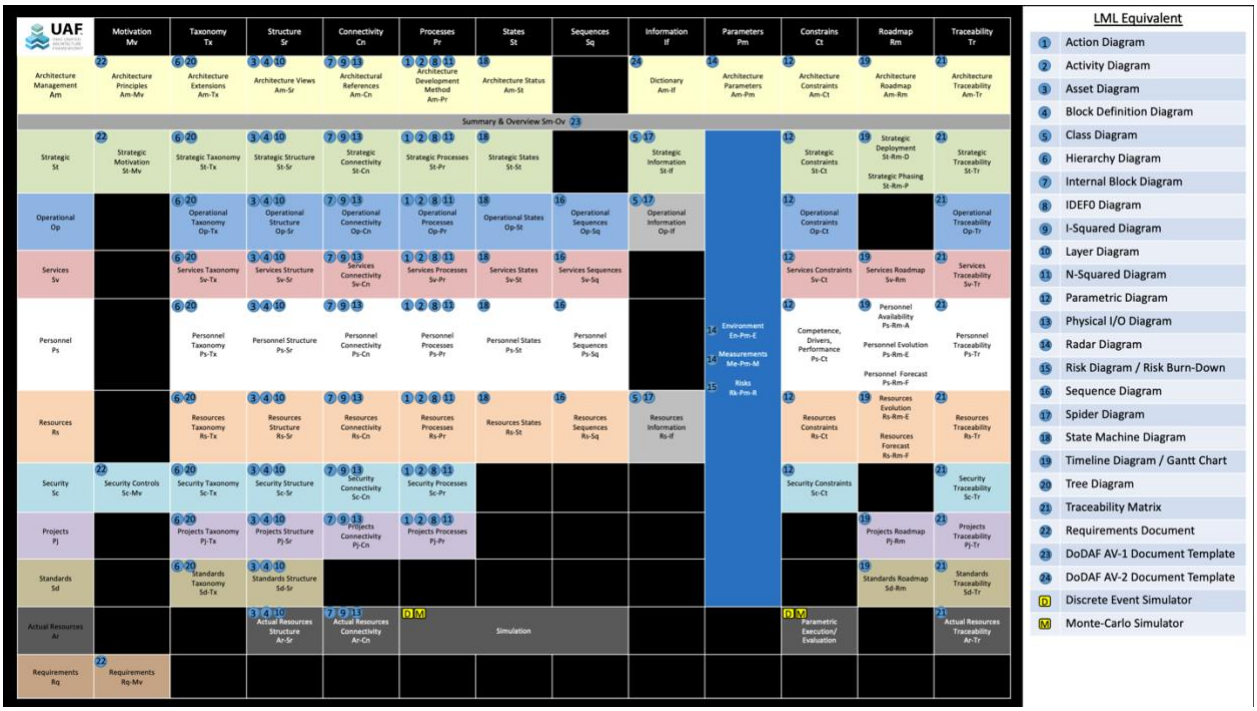
For a quick guide from the DM2 schema to LML, please see the table below.

Table B-1. DM2 Conceptual Data Model Mapping to LML

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

Appendix C. UAF Diagram Framework

The Unified Architecture Framework (UAF) represents another way to organize systems engineering products for architecture development. Figure C-1 below shows how LML, SysML and other diagram types fit into this framework. Also note the need for simulators to complete the matrix. LML provides the underlying ontology for creating discrete event and Monte Carlo simulations.



C-1. LML provides a more complete way to represent the Unified Architecture Framework.

Appendix D. Structuring Artifacts

The purpose of this appendix is to document an approach to structuring Artifacts, such as documents. This approach requires new metadata for the Artifact class, the addition of one new subclass (i.e. Heading) and one new relationship (originated by). This approach provides multiple advantages over the previous model, including:

- Eliminates numbering conflicts that arise when a requirement exists in multiple documents
- Enabling building content from any Entity class rather than being limited to Statement Class Entities
- Eliminates convoluted traceability resulting when Sections decompose to multiple levels of depth

The Artifact metadata, Heading class along with its attributes and relationships, and the "originated by" relationship are defined below. An entity relationship diagram illustrating the use of these items is also provided.

Table D-1. New Class

Class	Parent	Description
Heading	None	A Heading entity provides structure to an Artifact, representing Sections of a document

Heading

A **Heading** entity structures an **Artifact** and contains the content of the Section it represents.

Table D-2. Heading Relations (in addition to those that are standard for all entities)

Name	Classes	Description
sourced by	Artifact	A Heading is sourced by an Artifact when the Heading functions as a Level 1 Section
contains	Any	A Heading contains any other class. When viewed as a document, the Heading displays whatever attributes have been selected for the contained class

Table D-3. New Relationship (originated by)

Class	Name	Class	Description
Artifact	originates/ originated by	Requirement	Used to establish a direct link between a Requirement and the Artifact that functions as the originating authoritative source

The entity relationship diagram below (Figure D-1) shows how an Artifact might be structured. The Heading for Section 2.0 is directly linked to the **Artifact** using the ***source of*/*sourced by*** relationship since it represents a Level 1 section. That section is also decomposed to capture deeper document structure, including a child section (2.1) and a grandchild section (2.1.1). Use of the Heading object allows a section to contain multiple objects of any class, as can be seen in this example for Section 2.1 and 2.1.1. This enables constructing more complex **Artifacts** rather than being constrained to only the Statement class and it's subclasses. This supports common cases such as when a section contains both statement and requirement class objects, or such as when an artifact, representing an operating procedure built directly from an Action diagram, has sections that contain **Action** class objects. In a document view, the

Headings would show the structure with its section numbering, name and description attributes. The paragraph contents displayed underneath the Heading would include at least the description attribute of each contained object. The name and number of the “contained” object could optionally be displayed.

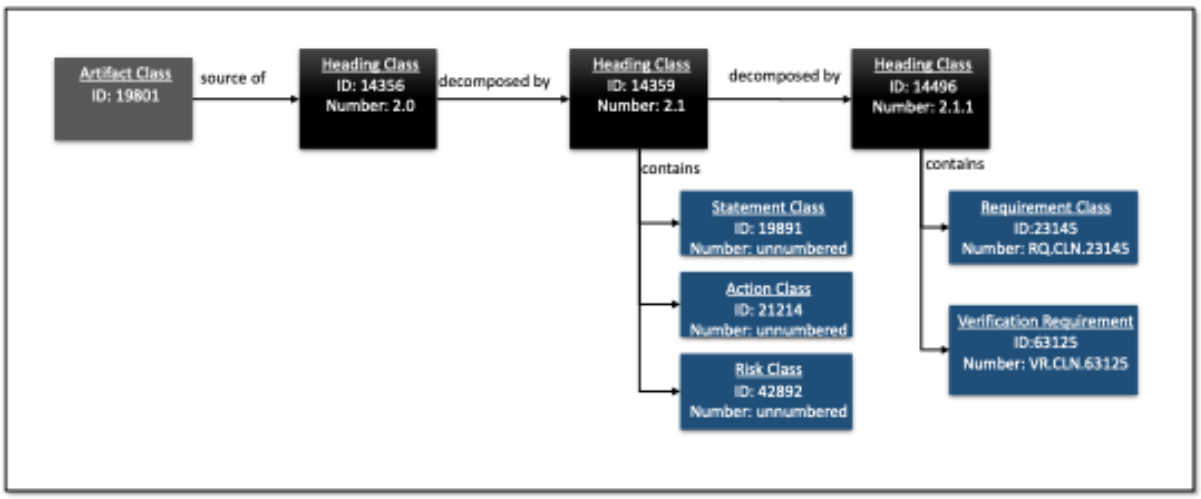


Figure D-1. Proposed New Classes for Structured Artifacts

Appendix E. Application Programming Interfaces

Application Programming Interfaces (APIs) allow multiple computer programs to talk to each other. They act like user interfaces, but instead of connecting computer programs to people, APIs connect computer programs to other computer programs. In this case, Applications refer to any software program with a distinct function, and Interfaces are like contracts of service between two of these Applications. Also, it is important to appreciate the distinction between client programs and server programs when talking about APIs.

The most popular kind of APIs are Representational State Transfer (REST) APIs. These work with a set of functions (e.g., GET, PUT, DELETE, etc.) that clients can use to access data on the server. The server responds to client requests in plain data, without graphical rendering. The server also does not preserve session information from the client (i.e., statelessness). The REST APIs in Innoslate 4.x for entities and schemas are listed below:

Entity

• Fetch entities from the search query

• Method: GET

• URL: api/v4/o/:organizationSlug/entities

• Parameters:

• query

• project Id

• Optional Parameters:

• limit

• offset

• Response: Array of Entity Objects

• Example: api/v4/o/demo/entities?query="class:Action"&projectId=64&limit=30&offset=0

• Update Entities

• Method: POST, PUT

• URL: api/v4/o/:organizationSlug/entities

• Payload: Array of Entity Objects

• Payload Example:
[{"number":"1","labelIds":[],"sortNumber":"000001.00000.0000.000.000.000.000.15639","classId":3,"cr
eatedIn":0,"modifiedIn":0,"linkedLabelId":0,"rels":[],"attrs":{},"controlStep":null,"controlType":"SERIAL",
"branches":[],"diagrams":{},"isArchived":false,"isLocked":false,"followers":["john.doe"],"projectId":64,"g
lobalId":"I_6168Y8QCZCK6S_AKAJ8HC1TC3ES","isRedacted":false,"id":15639,"name":"Asset","descriptio

n":"","created":1541771921633,"modified":1541771973213,"createdBy":"john.doe","modifiedBy":"joh
n.doe","version":2}]

- Response: Array of Entity Objects
- Fetch entities by Ids
- Method: GET
- URL: api/v4/o/:organizationSlug/entities/:ids
- Optional Parameters:
- includeChildren
- includeArchived
- includeRelations
- levels
- Response: Array of Entity Objects
- Update entities by Ids
- Method: POST, PUT
- URL: api/v4/o/:organizationSlug/entities/ids
- Payload: Array of Entity Object
- Payload Example: [234, 233]
- Response: Array of Entity Objects
- Delete entities by Ids
- Method: DELETE
- URL: api/v4/o/:organizationSlug/entities/ids
- Payload: Array of Entity Object
- Payload Example: [234, 233]
- Response: Array of Entity Objects
- Transform entities into another class
- Method: PUT
- URL: api/v4/o/:organizationSlug/entities/transform/:classId/:ids
- Response: Array of new transformed Entity Objects
- Example: api/v4/o/demo/entities/transform/3/9390,9450

- Revert entities to previous version
- Method: PUT
- URL: api/v4/o/:organizationSlug/entities/revert/:ids/:versionNumbers
- Response: Array of reverted Entity Objects
- Example: api/v4/o/demo/entities/transform/3/9390,9450
- Restore deleted entities
- Method: PUT
- URL: api/v4/o/:organizationSlug/entities/restore/:ids
- Response: Array of reverted Entity Objects
- Example:
- Auto number entities
- Method: PUT
- URL: api/v4/o/:organizationSlug/entities/autonumber/:id
- Optional Parameters:
- startNumber
- singleLevel
- useControlStep
- Response: Array of auto numbered Entity Objects
- Example:

Schema

- Fetch organization schema
- Method: GET
- URL: api/v4/o/:organizationSlug/schema
- Response: Schema Object
- Update organization schema
- Method: POST, PUT
- URL: api/v4/o/:organizationSlug/schema
- Payload: Schema Object
- Payload Example:

- Response: Schema Object
- Delete organization schema properties or labels
- Method: DELETE
- URL: api/v4/o/:organizationSlug/schema/:type/:ids
- Response: Schema Object
- Fetch project schema
- Method: GET
- URL: api/v4/o/:organizationSlug/p/:projectId/schema
- Response: Schema Object
- Update project schema
- Method: POST, PUT
- URL: api/v4/o/:organizationSlug/p/:projectId/schema
- Payload: Schema Object
- Payload Example:
- Response: Schema Object
- Delete project schema properties or labels
- Method: DELETE
- URL: api/v4/o/:organizationSlug/p/:projectId/schema/:type/:ids
- Response: Schema Object