OMG SysML™ Requirements Traceability

(informative)

This document has been published as OMG document ptc/07-03-09 so it can be referenced by Annex E of the OMG SysML™ specification.

This document describes the requirements tracability matrix (RTM) that shows how SysML satisfies the requirements in Sections 6.5 (Mandatory) and 6.6 (Optional) of the UML for SE RFP (OMG document ad/03-03-41). The matrix includes columns that correspond to those identified in the first paragraph of Section 6.5 of the RFP and are restated here. The text requirement statement is included in the RFP and was excluded from this document due to space limitations.

a) The UML for SE requirement number.

b) The UML for SE requirement name (or other letter designator). Note: The reader should refer to the UML for SE RFP for the specific text of the requirement, since there was inadequate room in the table to repeat it here.

c) Describes whether the proposed solution is a full or partial satisfaction of the requirement, or whether there is no solution provided. The section header rows that do not have a text requirement are marked N/A.

d) A description of how SysML addresses the requirement. Note: In some cases, there may be other SysML solutions to satisfy the requirement, but the intent was to describe at least one solution.

e) The specific UML and SysML metaclasses that address the requirement.

f) Reference to the applicable chapter in the SysML specification which addresses e) above. This diagram element tables in the chapter describe the concrete syntax (symbols) that show how the solution to the requirement is represented in diagrams. The usage examples in the chapters along with sample problem in "Annex B: Sample Problem" describe how the solution to the requirement is used in representative examples. Note: The reference to a chapter may require reference to a corresponding chapter in the UML specification. For example, when the blocks chapter is referenced, this may include a combination of the SysML blocks chapter and the UML classes and composite structure chapters.

Table E.1 - Requirement Traceability matrix

UML for SE Req't #	Requirement name	Compl (Y/N, Partial)	Requirement Satisfaction	Metaclass Extension	SysML Diagram Chapter	Ver #
6.5	*Mandatory Requirements*					
6.5.1	*Structure*	N/A	*Structure diagrams include block definition, internal block, and package diagrams*		Structural Constructs	
6.5.1.1	*System hierarchy*	Y	*Block composition (black or white diamond) in a block definition diagram and parts in internal block diagrams are the primary mechanisms for representing system hierarchy.*	*SysML::Block, UML::Association, SysML::Block Property*	*Blocks*	1.0

	a. Subsystem (logical or physical)	*Y*	*Typically represented by a set of logical or physical parts in an internal block diagram that realize one or more system operations. The corresponding sequence diagram and activity diagram with swim lanes can represent a hybrid of structure and behavior.*	SysML::Block, SysML::Block Property	Blocks	1.0
	b. Hardware (i.e., electrical, mechanical, optical)	*Y*	*Represented by a block or part.*	SysML::Block, SysML::Block Property	Blocks	1.0
	c. Software	*Y*	*Represented by a block or part or a UML component.*	SysML::Block, SysML::Block Property, UML::Component	Blocks	1.0
	d. Data	Y	*Represented by a block or part. Refer to input/output requirements in 6.5.2.1.1 and 6.5.2.5 for data flows.*	SysML::Block, SysML::Block Property, SysML::ValueType, UML::DataType	Blocks	1.0
	e. Manual procedure	*Y*	*Represented by a block or part. Can also be represented by the standard UML stereotype <<document>>.*	SysML::Block, SysML::Block Property, UML::Document	Blocks	1.0
	f. User/person	*Y*	*Represented by a block or part. External users are also represented as actors in a use case diagram.*	SysML::Block, SysML::Block Property	Blocks	1.0
	g. Facility	*Y*	*Represented by a block or part.*	SysML::Block, SysML::Block Property	Blocks	1.0
	h. Natural object	*Y*	*Represented by a block or part.*	SysML::Block, SysML::Block Property	Blocks	1.0
	i. Node	Y	*Represented by a block or part.*	*SysML::Block, SysML::Block Property*	Blocks	1.0
6.5.1.2	*Environment*	Y	*Environment is one or more entities that are external to the system of interest and can be represented as a block or part of a broader context. Also, represented as actors in use cases.*	*SysML::Block, SysML::Block Property*	*Blocks,* *Use Case*	1.0

6.5.1.3	*System inter-connection*		*Internal block diagram shows connections using parts, ports, and connectors.*	*SysML::Block, SysML::Block Property, UML Association, UML::Connector: SysML::Nested ConnectorEnd*	*Blocks*	1.0
6.5.1.3.1	*Port*	Y	*A port defines an interaction point on a block or part that enables the user to specify what can flow in/out of the block/part (flow port) or what services the block/part requires or provides (Standard Port). Ports are connected using connectors.*	*SysML::Standard Port,* *UML::Interface,* *SysML::FlowPort, SysML::Flow Specification, SysML::Flow Property*	*Ports and Flows*	1.0
6.5.1.3.2	*System boundary*	Y	*The enclosing block for an internal block diagram and its ports.*	*SysML::Block* *SysML::Standard Port,* *SysML::FlowPort*	*Blocks, Ports and Flows*	1.0
6.5.1.3.3	*Connection*	Y	*A connector binds two ports to support interconnection. A connector can be typed by an association. A logical connector can be allocated to a more complex physical path depicting a set of parts, ports, and connectors (refer to allocation). Note: A connector has limited decomposition capability at this time.*	*UML::Association, UML::Connector, SysML::Nested ConnectorEnd*	*Blocks*	1.0
6.5.1.4	*Deployment of components to nodes*	Y	*A structural allocation relationship enables the allocation (deployment) of one structural element to another.*	*SysML::Allocation, SysML::Allocated, UML::Named Element*	*Allocations*	1.0
	a.	Y	*Software part, block or component deployed to a hardware part or block (processor or storage device).*	SysML::Allocation, SysML::Allocated, SysML::Block, SysML::Block Property, UML::Component	Allocations	1.0
	b.	Y	*Generalized deployment relationship between a deployed element and its host.*	SysML::Allocation, SysML::Allocated, SysML::Block, SysML::Block Property	Allocations	1.0

	c	Y	*Deployed element and host can be decomposed using blocks and parts.*	SysML::Block, SysML::Block Property	Allocations	1.0
6.5.2	*Behavior*	N/A	*Behavior diagrams include activity, sequence, and state machine diagrams. Communication diagrams, interaction overview diagrams, and timing diagrams are interaction diagrams that are not included in SysML. Use case diagrams are also viewed as a behavior diagram in that they represent the functionality in terms of the usages of the system, but do not depict temporal relationships and associated control flow or input/output flow.*		Behavioral Constructs	
6.5.2.1	*Functional Transformation of Inputs to Outputs*		*A behavior is the generalized form of a function with inputs and output parameters. Activity is a subclass of behavior.*	*UML::Behavior*	Activities	
6.5.2.1.1	*Input/Output*	Y	*Inputs and outputs can be represented as parameters of activities, object nodes flowing between action nodes, and as item flows between parts in an internal block diagram. Note: Object nodes are more precisely represented by pins on action nodes.*	*UML::Parameter, UML::ObjectNode, SysML::ItemFlow*	*Activities, Ports and Flows*	1.0
	a	Y	*Parameters, object nodes, and item properties are typed by classifiers (blocks or value types) that can have properties.*	*SysML::Block, UML::Parameter, UML::ObjectNode, SysML::ItemFlow*	Activities, Ports and Flows	1.0
	b	Y	*The classifiers that represent the things that flow (type of parameter, object node, and item property) can be decomposed and specialized.*	*SysML::Block, UML::Parameter, UML::ObjectNode, SysML::ItemFlow*	Activities, Ports and Flows, Blocks	1.0
	c	Y	*"ItemFlows" associate the things that flow with the connectors that bind the ports. The parameters and object nodes are bound to the corresponding activities and actions.*	*SysML::Block, UML::Parameter, UML::ObjectNode, SysML::ItemFlow*	*Activities, Ports and Flows*	1.0

6.5.2.1.2	*System store*	Partial	*Stored items can be represented as parts of a block, and also represented in an activity diagram as object nodes or central buffer nodes.*	*SysML::Block, SysML::Block Property,* *UML::ObjectNode* *UML::Central BufferNode*	*Blocks, Activities*	1.0
	a	Partial	*Object nodes in an activity diagram can represent depletable stores, and a data store node can represent non-depletable stores.*	*UML::ObjectNode, UML::DataStore Node*	Activities	1.0
	b	Y	*A stored item can be the same type of classifier as an input or output in both an internal block diagram and an activity diagram. The classifier supports different roles (store vs. flow).*	*SysML::Block, SysML::Block Property, UML::ObjectNode, UML::DataStore Node*	Blocks, Activities	1.0
6.5.2.1.3	*Function*	Y	*Activity specifies a generic subclass of behavior that is used to represent a function definition in activity diagrams, sequence diagrams, and state-machine diagrams. Activities contain CallBehaviorActions that call (invoke) other activities to support execution of the generic behaviors.*	*UML::Activity*	Activities, Interactions, State Machines	1.0
	a	Y	*Behaviors and the associated parameters are named (i.e., name of activity and activity parameter node).*	*UML::Behavior*	*Activities, Interactions* *State Machines*	1.0
	b	Y	*The action semantics define different types of actions that include CreateObject, DestroyObject, ReadStructuralFeature (monitor), and WriteStructurealFeature (update). A CallBehavior action is a generalized action that can call any behavior (activity, interaction, state).*	*UML::CreateObject Action, UML::DeleteObject Action, the various object modification actions in UML, monitoring with UML::AcceptEvent Action*	*Activities, Interactions* *State Machines*	1.0
	c	Y	*The object nodes (pins) bind input and output parameters to actions.*	*UML::ObjectNode, UML::Pin*	*Activities*	1.0

	d	Y	*The queuing semantics for object nodes are specified. The default queuing is FIFO, but other forms of queuing including LIFO, ordered, and unordered as defined by the enumeration for ObjectNodeKind.*	*UML::Behavior, SysML::InputPin, SysML::ObjectNode*	*Activities*	1.0
	e	Partial	*Resource constraints to support an execution can be specified by Preconditions and PostConditions. The constraints can apply to resources that are generated, consumed, produced, and released, such as inputs and outputs, or the availability of memory or CPU. The constraints imposed on the resources can be further modeled using parametric diagrams.*	*UML::Constraint,* *SysML::Constraint Block*	*Activities, Constraint Blocks*	1.0
	f	Y	*Refer to c*	*UML::ObjectFlow, UML::Pin*	*Activities*	1.0
	g.	Y	*An activity can be decomposed into lower level actions that invoke other activities.*	*UML::Activity, UML::CallBehavior Action, UML::Activity ParameterNode, UML::ObjectFlow, UML:: Pin*	*Activities*	1.0
	h.	Y	*An action has control inputs that can enable the execution of a function, and a control value input from a control operator that can both enable or disable an execution of a function. An execution of a function can also be terminated when it is enclosed in an interruptible region. Alternatively, state machine diagrams can be used to enable or disable execution upon transition events.*	*UML::Action, UML::Interruptible ActivityRegion, SysML::ControlValue UML::State*	*Activities, State Machines*	1.0

	i	Y	*A computational expression can be used to specify the behavior (i.e. activity) that is invoked by an action or an action that represents a primitive function such as an arithmetic expression. Specific math expressions may be included in a math model library. The expressions should be represented in a formal mathematical language and specify the language if they are to be interpreted by a computational engine.*	*UML::Activity, UML::Action*	*Activities, Interactions* *State Machines*	1.0
	j	Y	*A continuous or discrete rate stereotype can be applied to inputs and outputs. Inputs and outputs are discrete by default. A time continuous input or output is an input or output whose value can change in infinitely small increments of time. An activity can accept the continuous inputs and provide continuous outputs while executing if the inputs and outputs are also streaming. An alternative approach is to continuously invoke an activity that does not have streaming inputs or outputs, in which case each execution of an activity accepts the inputs at the start of execution and produces the output at the completion of execution.*	*SysML::Rate* *SysML::Continous, SysML::Discrete* *UML::Parameter (isStream=Value)*	*Activities, State Machines*	1.0
	k	Partial	Different actions can invoke concurrent executions of the same generalized behavior. Actions can have multiplicity.	*UML::Behavior, UML::Action*	Activities	1.0
6.5.2.2	*Function activation/ deactivation*	*N/A*	*Actions can be activated and deactivated using multiple mechanisms within SysML as described below including control flows, control operators, and interruptible regions.*		*Activities, Interactions* *State Machines*	1.0
6.5.2.2.1	*Control input*	Y	*Control flows in activity diagrams provide the control input. Control flow is represented in state machine diagrams by a transitions which activate states and in sequence diagrams by the passing of messages.*	*UML::ActivityEdge,* *UML::ControlFlow, UML::Transition, UML::Message, SysML::ControlValue*	*Activities, Interactions* *State Machines*	1.0

	a	Y	*Multiple control flows in an activity diagram that are input to a single activity node (i.e., action) are assumed to be "anded" together.*	*SysML::ControlValue, SysML::InputPin.isControl=true for control queuing*	*Activities*	1.0
	b	Y	*Control inputs are discrete valued inputs that can enable or disable an activity node.*	*SysML::ControlValue*	*Activities*	1.0
	c	Y	*In activity diagrams, the activity is invoked (enabled) when a token is received by the calling action. This includes tokens from all mandatory inputs and control inputs.*	*UML::Action,* *UML::ControlFlow, UML::ActivityEdge*	*Activities*	1.0
	d	Y	*In activity diagrams, a control operator can produce an output control value to disable the execution of an activity. An action enclosed within an interruptible region also can disable the execution of an activity. In state machine diagrams, transition events can disable the actions in a state.*	*UML::Action, UML::Interruptible ActivityRegion, SysML::ControlValue, UML::State*	*Activities, State Machines*	1.0
	e	Y	*An executing activity with non-streaming inputs and outputs terminates when it completes its transformation and produces an output value. An executing activity with continuous streaming inputs will terminate when it receives a disable from a control value and/or a signal that terminates the actions within an interruptible region. A TimeExpression can be specified in a control operator or can signal a termination in an interruptible region. An activity can also be terminated based on events, including timeout events, on a transition in a state machine diagram. In state machine diagrams, completion events occur upon completion of an activity.*	*UML::Activity, UML::Interruptible ActivityRegion, SysML ControlValue, UML::Time Expression, UML::State*	*Activities, State Machines*	1.0
	f	Y	*The enabling of actions without explicit control flows as inputs are enabled based on the control associated with its inputs.*	*UML::Action, UML::ObjectNode*	*Activities*	1.0

	g	Y	*A control flow connects the control inputs from one activity node to another. The control input can also be the output control value of a control operator.*	*SysML::ControlValue, UML::Parameter, UML::ControlFlow*	*Activities*	1.0
6.5.2.2.2	*Control operator*	Y	*A control operator provides the mechanism apply control logic to enable and disable activity nodes.*	*SysML::Control Operator, SysML::ControlValue*	Activities	1.0
	a	Y	*Control Nodes such as joins, forks, etc. provide capability to activate activity nodes based on "and" and "or" logic. A SysML Control Operator provides the additional capability to disable an activity node.*	*UML::ControlNode, SysML::Control Operator, SysML::ControlValue, UML::Parameter*	*Activities*	1.0
	b	Y	*A join specification can be used to specify arbitrarily complex logic for enabling an activity node. A control operator can also be used to specify complex logic for enabling and disabling an activity node.*	*UML::JoinNode with join specification, UML::Parameter, SysML::Control Operator, SysML::ControlValue*	*Activities*	1.0
	c	Y	*The control nodes identified below provide the basic control logic for enabling activity nodes. Note: multi exit functions are supported by parameter sets. Also, Interaction Operators provide similar logic in Sequence Diagrams.*	*UML::ControlNode, UML::Interaction Operator*	Activities, Interactions	1.0
	c1	Y	*Decision nodes in activity diagrams support selection. The "alt" Interaction Operator supports selection in sequence diagrams.*	*UML::DecisionNode, UML::Interaction Operator.Alt*	Activities, Interactions	1.0
	c2	Y	Forks in activity diagrams support a single input flow generating multiple concurrent output flows. The “par” Interaction Operator supports concurrent message flow in Sequence Diagrams.	*UML::Fork, UML::Interaction Operator.par*	Activities, Interactions	1.0
	c3	Y	A join “and's” multiple input flows together resulting in a single output flow.	*UML::Join*	Activities	1.0
	c4	Y	A merge results a single output flow upon arrival of the first of multiple input flows.	*UML::Merge*	Activities	1.0

	c5	Y	*Decision and loop nodes support iteration and looping. The "loop" Interaction Operator supports loops in sequence diagrams.*	UML::Decision-Node, UML::Loop Node, Interaction-Operator.loop	Activities, Interactions	1.0
	c6	N				
6.5.2.2.3	*Events and conditions*	Partial	*Triggers and constraints as guards provide the mechanism for modeling events and conditions.*		Activities, Interactions, State Machines	1.0
	a	Partial	*A trigger can be used to specify an event. Events can be associated with control flows in activity diagrams, transitions in state machine diagrams, and sending and receiving of messages in sequence diagrams.*	*UML:: Trigger, UML::AcceptEvent Action including UML::TimeTrigger, UML::Event Occurence in Interactions.* *Note: Failure event can be result in various types of actions that terminate an Interruptible Region in Activities, etc.*	*Activity, Interactions* *State Machines*	1.0
	b	Y	*Refer to a) above*	*UML::ActivityEdge, UML::Trigger*	*Activity, Interactions* *State Machines*	1.0
	c	Y	*Conditions can be specified as constraints that define guards to control execution of behaviors.*	*UML::Constraint (guard)*	*Activity, Interactions* *State Machines*	1.0
6.5.2.3	*Function-based behavior*	Y	*Activity diagrams provide the capability to model function based behavior.*	*UML:: Activity*	*Activities*	1.0

6.5.2.4	*State-based behavior*		*State machine diagrams provide the capability to model state based behavior with the specific modeling constructs indicated. Note 2 response: Activities are common to each type of behavior including both function based and state based. Note 3 response: A state is defined based on some invariant being true. The invariant can include reference to certain property values.*	*UML::StateMachine*	*State Machines*	1.0
	a	Y	*State*	*UML::State*	State Machines	1.0
	b	Y	*Simple state*	*UML::State, isSimple=True*	State Machines	1.0
	c	Y	*Composite states can contain one region or two or more orthogonal (concurrent) regions, each with one or more mutually exclusive disjoint states*	*UML::State isComposite=True*	State Machines	1.0
	d	Y	*Transitions between states which are triggered by events with guard conditions.*	*UML::Transition, UML::Trigger*	State Machines	1.0
	e	Y	*Transition within a composite state*	*UML::Transition (TransitionKind= Internal)*	State Machines	1.0
	f	Y	*Pseudo states include joins, forks and choice*	*UML::PseudoState*	State Machines	1.0
	g	Y	*Transitions between states which are triggered by events with guard conditions.*	*UML::Activity*	State Machines	1.0
	h	Y	*Entry, exit, doActivities are performed upon entry or exit from a state or while in a state.*	*UML::Activity*	State Machines	1.0
	i	Y	*State machine semantics define the ordering of actions that are completed when exiting a composite state (refer to UML transition semantics). When a composite state is exited, the exit actions are executed beginning with the most nested state.*	*UML::State (Note: refer to semantics)*	State Machines	1.0

	j	Y	*Entry and exit actions must be completed prior to exiting a state. A doActivity does not need to be completed to execute.*	*UML::State (Note: refer to semantics)*	State Machines	1.0
	k	Y	*Send and receive signals can be sent via actions to interact with other objects.*	*UML::SendSignal Action*	State Machines	1.0
	l	Partial	The failure and/or exception states are user defined and have no uniquely defined representation. The use of exit points on states can be used to exit the state when a failure event occurs.	*UML::State*	State Machines	1.0
6.5.2.4.1	*Activation time*	Y	*The interval of time that an activity or state is active can be modeled by a UML Time Trigger or Time Interval and corresponding Time Expression (refer to UML trigger and interval notation). Note: A UML timing diagram is not included in SysML at this time, but could be used to model the time associated with the occurrence of events, such as state changes, or changes in property values.*	*UML::SimpleTime*	Activities, Interactions, State Machines	1.0
6.5.2.5	*Allocation of behavior to systems*	Y	An allocation relationship provides a generalized capability to allocate one model element to another.	*SysML::Allocation, SysML::Allocated, UML::NamedElement*	*Allocations*	1.0
	a	Y	In general, behaviors such as activities, interactions, and state machines are owned by a Behaviored Classifier which can correspond to an block. The SysML Allocation relationship can be used to explicitly allocate behaviors to blocks. Alternatively, activity partitions (swim lanes) can be used to allocate the action and/or activity to a part and/or block.	*UML::BehavioredClassifier and UML::Behavior (owned behavior) - Refer to UML Common Behaviors, SysML::Allocate, SysML::Allocate AcitivtyPartition*	*Allocations, Activities*	1.0

	b	Partial	*An object node in an activity diagram can be allocated to an item that flows in an internal block diagram using an allocation relationship. Note: the object node is typed by the same classifier as the item that flows. See req't 6.5.2.1.1.*	*SysML::Block (type of ObjectNode to type of ItemProperty), UML::ObjectNode, UML::Property*	*Allocations, Activities, Ports and Flows*	1.0
6.5.3	*Property*	N/A	Properties and their relationships are represented in SysML using properties of blocks in conjunction with constraint blocks to capture the relationships between them.		*Blocks, Constraint Blocks*	
6.5.3.1	*Property type*	Y	Primitive types, data types, and value types provide the capability to model the different types of quantitative properties.	*UML:: PrimitiveType, UML::DataType,* *SysML::Value Type*	*Blocks*	1.0
	a	Y	Primitive type.	UML::Integer		
	b	Y	Primitive type.	UML::Boolean		
	c	Y	Primitive type.	UML::Enumeration		
	d	Y	Primitive type.	UML::String		
	e	Y	Primitive type.	SysML::Real		
	f	Y	Data type.	SysML::Complex		
	g	Y	Composite data type made up of primitive types.	Refer to a-f		
	h	Y	Composite data type made up of primitive types.	Refer to a-f		
6.5.3.2	*Property value*	Y			*Auxiliary*	1.0
	a	Y	Value properties are typed by a value type or data type and have an associated value.	SysML::Block Property, SysML::ValueType, UML::DataType,	Blocks	1.0
	b	Y	A value type can include a dimension and units such as length and feet or meters.	SysML::ValueType (unit and dimension are defined as blocks in a model library)	Blocks	1.0
	c	Y	A value property is a block property that is typed by a value type that can have an associated probability distribution on its values.	*SysML::ValueType, SysML::Distribution Definition*	Blocks	1.0

	d	Y	Source data can be included in a comment attached to the property or a user defined stereotype could be applied.	UML::Comment	Model Elements	1.0
	e	Y	Reference data can be included in a comment attached to the property or a user defined stereotype could be applied.	UML::Comment	*Model Elements*	1.0
6.5.3.3	*Property association*		*A value property can be a feature of any classifier (.i.e., block)*	SysML::Block, SysML::Block Property	Blocks	1.0
	a	Y	Blocks, parts, or items that flow can have (or reference) properties.	*SysML::Block, SysML::Block Property*	Blocks	1.0
	b	Y	A function (activity) can have properties since it is a class	*UML::Activity*	Activities	1.0
	c	Partial	An event is specified by a trigger which is an element. The element does not have properties.A signal which is sent upon the occurrence of the event can have properties.	UML::Signal		1.0
	d	Y	A property can be related to other properties through a constraint property	SysML::Constraint-Block, SysML::Constraint-Property	Constraint Blocks	1.0
6.5.3.4	*Time property*	Y	Time can be treated as a property, typed by a Real that can represent either continuous or discrete time. Time ultimately derives from clocks which can be continuous or discrete. Clocks can be modeled as blocks which have a time property that can be bound to a parameter of a constraint property (e.g., equation). Time durations, start and stop times, etc. can be modeled using the UML time model for time triggers, time expressions, intervals, and durations. Note: More elaborate models of time and clocks can be found in the UML schedulability, performance, and time profile.	*SysML::Block,* *SysML::Block Property,* *SysML::ValueType,* *SysML::Constraint Property, SysML::Constraint Parameter, UML::SimpleTime Package*	*Blocks,* *Constraint Blocks, Interactions*	1.0

6.5.3.5	*Parametric model*	*Y*	*The parametric diagram supports modeling of constraints which bind parameters of the constraints to value properties.*	*SysML::Constraint Block,* *SysML::Constraint Property* *SysML::Constraint Parameter,* *SysML::Block Property,* *UML::Connector, SysML::Nested ConnectorEnd*	Constraint Blocks	1.0
	a	Y	Constraints blocks and their usages (constraint properties) specify the mathematical relationships/constraints between constraint parameters.	SysML::Constraint-Block, SysML::Constraint-Parameter	Constraint Blocks	1.0
	b	Partial	Mathematical and logical expressions can be defined in SysML in a reference language, but there is no interpreter built into SysML. The range of values can be specified via value properties and probability distributions per 6.5.3.2a-c.	SysML::Block Property, SysML::Distribution-Definition	Blocks	1.0
	c	Y	The reference language for interpreting the constraint can be included as part of the ConstraintBlock along with the compartment for the expression.	SysML::Constraint-Block	Constraint Blocks	1.0
6.5.3.6	*Probe*	N	*No specific mechanization has been provided. In the testing profile, there is a mechanism to capture data and create actions in response to the data. This will be investigated in a future version of SysML.*			N
6.5.4	*Requirement*	N/A	The requirements diagram provides the basic capability for relating text based requirements to other SysML models.		Requirements	1.0

6.5.4.1	*Requirement specification*	Y	A requirement is a stereotype of a class in SysML. The various subtypes of requirement are specified as subclasses of the the requirement stereotype and can include specific properties and constraints on what model elements can satisfy the subclass of requirement. A sample set of subclasses of requirements are included in the NonNormative Extensions Annex C.	*SysML::Requirement*	*Requirements,* *Non-Normative Extensions, Profiles & Model Libraries*	1.0
	Note 1	Y	Values and tolerances can be specified as part of the text property or via property values and distributions per 6.5.3.2a-c.	Requirement.text, SysML::Value Property	Requirements, Blocks	1.0
	Note 2	Y	There is no explicit subclass of requirement as a stakeholder need, but a requirement can be named or subclassed as "stakeholderNeed."	SysML::Requirement	*Requirements,* *Non-Normative Extensions*	1.0
	Note 3	Y	User defined requirements can be added via subclasses to specify any type of life cycle requirement of interest to the modeler.	SysML:: Requirement	*Requirements,* *Non-Normative Extensions, Profiles & Model Libraries*	1.0
	a	Y	Operational requirement	SysML:: Requirement	*Requirements,* *Non-Normative Extensions*	1.0
	b	Y	Functional requirement	SysML::functional-Requirement	*Requirements,* *Non-Normative Extensions*	1.0
	c	Y	Interface requirement	SysML::interface Requirement	*Requirements,* *Non-Normative Extensions*	1.0
	d	Y	Performance requirement	SysML::performanceRequirement	*Requirements,* *Non-Normative Extensions*	1.0

	e	Y	Activation/Deactivation (Control) requirement	SysML:: Requirement	*Requirements, Non-Normative Extensions*	1.0
	f	Y	Storage requirement	SysML:: Requirement	*Requirements, Non-Normative Extensions*	1.0
	g	Y	Physical requirement	SysML::physical Requirement	*Requirements, Non-Normative Extensions*	1.0
	h	Y	Design constraint	SysML:: Requirement	*Requirements, Non-Normative Extensions*	1.0
	i	Y	Specialized requirement	SysML:: Requirement	*Requirements, Non-Normative Extensions*	1.0
	j	Y	Measure of effectiveness	SysML::moe	*Requirements, NonNormative Extensions*	1.0
6.5.4.2	*Requirement properties*	Y	A requirement includes default properties for id and text. Other properties can be added via stereotype properties.	*SysML::Requirement*	*Requirements, Non-Normative Extensions, Profiles & Model Libraries*	1.0
6.5.4.3	*Requirement relationships*	Y	The requirement relationships include the relationships containment, trace, deriveReqt, satisfy, verify and refine relationships.		Requirements	1.0
	a	Y	A derive relationship relates a derived (target) requirement to a source requirement.	*SysML::deriveReqt*	*Requirements*	1.0
	b	Y	A satisfy relationship relates the model elements (i.e. the design) to the requirements that are satisfied.	*SysML::satisfy*	*Requirements*	1.0

	c	Y	Goals, capabilities, or usages of systems are often expressed using use cases. Subgoals can be represented using the include and extend relationships between use cases. Requirements can be related to use cases using the refine relationship. Requirements use the containment relationship to breakdown an existing requirement into its containing requirements.	*UML:UseCase, UML::Include,* *SysML::Requirement, UML:refine*	Requirements, Use Case	1.0
6.5.4.4	*Problem*	Y	A problem is an extension of a comment that can be attached to any model element. Note: This could also be used to represent issues.	*SysML::Problem*	*Model Elements*	2.0
6.5.4.5	*Problem association*	Y	Refer to 6.5.4.4	*SysML::Problem*	*Model Elements*	2.0
6.5.4.6	*Problem cause*	N				2.0
6.5.5	*Verification*	N/A	*The following responses to the Verification requirements will include references to the Testing Profile [OMG Adopted Specification* *ptc/03-08-03] which is not currently part of SysML but is intended to be evaluated for integration with version 1.1 of SysML [refer to white paper on integrating SysML with Testing Profile]*		Requirements	
6.5.5.1	*Verification Process*					

	a	Y	*The SysML verify relationship between one or more system requirements and one or more test cases represents the method for verifying that a system design satisfies its requirements. A verified system design implies that the system will satisfy its requirements if the component parts satisfy their allocated requirements. An alternative approach to capture the verify relationship is to associate a test case with a satisfy relationship using the rationale.*	*SysML::Verify, SysML::Rationale*	*Requirements, Model Elements*	1.0
	b	Y	*The SysML verify relationship between one or more requirement(s) and one or more test case(s) is used to verify that the implemented system design instances satisfy their requirements. Alternatively, a reference to a TestCase using SysML:Rationale may be attached to a satisfy relationship.*	*SysML::Verify SysML::Rationale*	Require-ments, Model Elements	1.0
	c	Y	*A derive relationship between the requirement being validated and the higher level requirement or need may have a Rationale attached that references the validation method(s).*	*SysML:deriveReqt* *SysML::Rationale*	Require-ments, Model Elements	1.0
	Note 1	Y	*Verification methods of analysis and similarity may be modeled as a Rationale with reference to the specific analysis report or other reference data. Verification methods including Test, Inspection, and Demonstration may be modeled as a TestCase.*	*SysML::Rationale, SysML::TestCase*	Require-ments	1.0
	Note 2	Partial	*Validation methods are user defined. A rationale can reference the user defined methods.*	*SysML::Rationale*	Model Elements	1.0
6.5.5.2	*Test case*	Partial	A test case refers to the method for verifying a require-ment. Note: The testing profile associates a test case with a behavior that can include the specific method and associ-ated input stimulus and response.	SysML::TestCase	*Requirements*	1.0

	Note 1	Partial	Refer to above note on the testing profile.			1.x
	Note 2	Partial	The test criteria can be established via the require-ment			1.x
	Note 3	Partial	Test cases can contain other test cases, like any other named element.	SysML::TestCase	Require-ments	1.0
6.5.5.3	*Verification result*	Partial	*The result of a SysML:TestCase may be expressed through its verdict attribute (Testing Profile)*	*SysML::TestCase, SysML::Verdict*	*Requirements*	1.0
6.5.5.4	*Requirement verification*	Partial	*A constraint may be used to relate the required value to the verification result.*	*SysML::Constraint Property;* *SysML::TestCase,* *SysML::Rationale*	*Requirements, Constraint Blocks*	1.0
6.5.5.5	*Verification procedure*	Partial	A rationale can be associated with the test case or the satisfy relationship between a require-ment and a design, and reference a verification procedure. Note: The testing profile will associate a behav-ior with a test case which can be implemented by a specific procedure.	*SysML::TestCase, SysML::Rationale*	*Requirements, Model Elements*	1.x
	Note					
6.5.5.6	*Verification system*	Partial	A verification system can be modeled as any other system (block) or it can be modeled as the system environment. However, the future integration with the testing profile is intended to provide explicit modeling of the verification system.	SysML::Block	Blocks	2.0
6.5.6	*Other*	N/A				
6.5.6.1	*General relationships*	Y	*SysML includes several standard UML relationships as described below.*			
	a	Y	An association relationship.	UML::Association	Blocks	1.0
	b	Y	A package contains package-able elements and can repre-sent collections of elements.	UML::Package, UML::Packageable Element; UML::owned Member	Class	1.0

	c	Partial	Blocks can be decomposed into parts that are typed by other blocks using composition (refer to Reqt 6.5.1.1). The completeness of the decomposition is not explicitly represented.	SysML::Block, SysML::Block Property, UML::Association (composition)	Blocks	1.0
	d	Y	A dependency relationship.	UML::Dependency	Model Elements	1.0
	e	Y	Generalization/specialization relationship. Generalization sets provide the means to partition specializations to support further categorization.	*UML::Generalization, UML::Generalization Set*	Blocks	1.0
	f	Y	Instantiation is modeled using Instance Specifications to uniquely identify a classifier. Instances are represented as a property specific value with a unique set of values.	UML::Instance Specification, UML::InstanceValue	Blocks	1.0
6.5.6.2	*Model views*	Partial	A view represents the model from a particular viewpoint. Both the view and the viewpoint are represented in SysML. The view is a stereotype of package that identifies the set of model elements that conform to the viewpoint, and the viewpoint specifies the stakeholders, their purpose, concerns and the construction rules (language and methods) to specify the view. Note: The model elements that depict the view are visually represented in diagrams, tables, and other notation. Integrity between model views is accomplished by creating a well formed model. This in part results from the constraints imposed by the language, and in part is defined by the specific methodology and tools that are employed. Navigation among views results from a tool vendor implementation.	SysML::View, SysML::Viewpoint SysML::Conform	*Model Elements*	1.0
6.5.6.3	*Diagram types*				Diagram Appendix	1.0

	a		The standard UML diagram types that are needed to support the requirements have been included in SysML. Some additional diagram types provide some redundant capabilities, but have been preserved to allow flexibility in representations and methodologies. For example, the sequence diagrams along with activity and state diagrams provide overlapping capability for representing behavior. A few diagram types have not been included explicitly in SysML, although they are not precluded from use along with SysML.	*N/A*	Diagram Appendix	1.0
	b		The requirements diagram and parametric diagram have been added to address the requirements of this RFP. In addition, an informal mechanism has been added to represent diagram usages. This enables renaming and constraining the usage of a particular diagram type for a particular usage.	SysML::Diagram Usage	Diagram Appendix	1.0
6.5.6.4	*System role*	Partial	A part in a block represents the role for a classifier in the context of the enclosing block. It defines the relationship between an instance of the classifier that types the part and an instance of the block that encloses the part. This is a primary mechanism for providing a unique context for a part of a whole (enclosing block). The part may use only a subset of the behavior and properties of the class that types the part. However, the specific mechanism for containing the subset has not been explicitly defined.	SysM::Block, SysML:Block Property	Blocks	1.0
6.6	*Optional Requirements*	N/A				
6.6.1	*Topology*	N				
	a	N				2.0

	b	N				2.0
6.6.2	*Documentation*	Y	A document (stereotype of artifact).	UML::Document	*Diagram Appendix*	1.0
	a	Y	The document stereotype can include stereotype properties to represent information about the document.	UML::Document	*Profiles & Model Libraries*	1.0
	b	Y	The trace relationship relates a document to other model elements.	UML::Trace	*Diagram Appendix*	1.0
	c	N	The ability to represent the text of the document in terms of the descriptions provided by the related (traced) model elements is accomplished by a tool implementation.			
6.6.3	*Trade-off studies and analysis*	Partial	*Parametric diagrams can depict the relationship between measures of effectiveness and various system properties (including probability distributions on their values) to evaluate the effectiveness of a particular system model. Specific constructs for criteria, weighting, and alternatives are planned for a future version of SysML to support modeling of trade studies.*	*SysML::moe, SysML::objective Function, SysML::Constraint Property*	Constraint Blocks, Non-Normative Extensions	1.0
	a	Y	Alternative models can be specified via organization of models/packages. Model libraries can be used to establish reusable portions of the model.	UML::Model, UML::Package	Model Elements, Profiles & Model Libraries	1.0
	b	Partial	Criteria can be modeled as properties typed by value types or as Requirements	SysML::Block Property, SysML::ValueType, SysML::Requirement	Blocks, Requirements	1.0, 2.0
	c	Y	Measures of effectiveness are modeled as a subclass of block property that represents a value property. A constraint can represent the objective function.	SysML::moe, SysML::Constraint-Property	Non-Normative Extensions, Constraint Blocks	1.0
6.6.4	*Spatial representation*	N				

6.6.4.1	*Spatial reference*	N				
6.6.4.2	*Geometric relationships*	N				
6.6.5	*Dynamic structure*	Partial				
	a	Y	The action semantics provide the capability for creating and destroying objects.	*UML::CreateObject Action, UML::DestroyObject Action*	Action (UML Spec)	1.0
	b	Partial	The capability is partially provided by 6.6.5a.			2.0
	c	N				2.0
	d	N				2.0
6.6.6	*Executable semantics*	*Partial*	*The action semantics are intended to provide execution semantics. There is no formal graphical syntax for this.*	*UML::Action*	Action in UML Spec	1.0
6.6.7	*Other behavior modeling paradigms*	Y	A UML behavior is a generalized behavior that can accommodate a wide range of behavior modeling paradigms. This include function based, state based, and message based behavior (sequence diagrams).	*UML::Behavior*	Activities, Interactions, State Machines	1.0
6.6.8	*Integration with domain-specific models*	Partial	SysML is a general purpose language that will integrate with other types of domain specific models. This is accomplished in part by mapping SysML via XMI to the AP233 data interchange standard. In addition, the parametric diagram is intended to provide a capability to integrate with domain specific engineering analysis models.		Model Interchange	1.x, 2.0
6.6.9	*Testing Model*	Partial	SysML is intended to be integrated with the UML Testing Profile. Refer to Response to Reqt 6.5.5 above.	SysML::TestCase	Requirement	2.0
6.6.10	*Management Model*	N				