# SysML v2 Requirements for Properties and Expressions

## Introduction

Status: Draft 5, work in progress, not yet reviewed by working group

Purpose: Preparation for insertion into clause 6.5.5 "System Modeling Concepts" of the SysML v2 RFP.

## Conventions

Throughout the requirements tables the following conventions are applied:

1. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, see <https://www.rfc-editor.org/info/rfc2119>. This is an established best practice for IETF, OMG, W3C, and other requirements specifications.
2. In the requirements table potential SysML v2 concepts are underlined.
3. The tables are structured with columns that correspond one-to-one with the attributes of the SysML v1 Requirement concept, so that the requirements can be easily imported into a SysML v1 model, e.g. the MagicDraw representation of the SECM.
4. The possible requirement types as defined in SysML v1 are: functional, interface, performance, physical, design constraint.

## Table ?? – Requirements concerning Properties and Expressions

| **Package** | **Req. Type** | **Id** | **Name** | **Text** | **Rationale** | **Comments** |
| --- | --- | --- | --- | --- | --- | --- |
|  | Functional | 1 | Core concept: value element | SysML v2 shall support the generic (abstract) concept of a value element that is typed by a value type. | A generic top level abstract type for all model elements typed by a value type is needed for a coherent and well organised data model. | Note 1: A value element is a named model element that has a value and is typed by a value type. |
|  | Functional | 2 | Core concept: value type | The value type concept shall be a 'first class citizen', i.e. it shall be handled the same way as other top level 'classifiers', and it shall support type specialization. | The typing of values is a core concept at the same level as other typing concepts such as system element. |  |
|  | Functional | 3 | Concrete value elements | SysML v2 shall support as a minimum the following concrete concepts that are typed by a value type, and therefore specializations of value element:  property, constant, variable, formal parameter, operation, operation call, operation call argument. | A property represents an attribute, a quality, or a characteristic of a system element. A constant is a value element with an invariant value. A variable represents a named symbol in a constraint or in a value expression. A formal parameter represents a named input symbol in the signature of an operation. An operation itself has a value type that represents its return type. By implication an operation call has the same value type as the operation it invokes. An operation call argument represents the runtime instance of a formal parameter at call time. | Note 1: Concepts signal and message are as defined in SysML v1. Details on how to integrate these with value type are TBD.  Question: Should we keep using the concept called 'operation' as in UML and SysML v1, or rather 'function' as in mathematics? |
|  | Functional | 4 | Scalar and compound value types | The value type shall comprise representation of both 'scalar' and 'compound' value types, where 'scalar' means with a single value, and 'compound' comprising value type components, where each component is typed in turn by a scalar value type or (another) compound value type. |  | Note 1: For reasons of implementation efficiency and verification, the number of nesting levels permitted in a compound value type may be constrained to a value that covers the majority of practical use cases, e.g. 2 or 3. |
|  | Functional | 5 | Concrete value types | The scalar value type shall include representation of numerical and non-numerical values, including boolean, text, enumeration, date, date-and-time, time-of-day and system element reference values. |  | Question: Should SysML v2 also support a specific rich text value type like XHTML in ReqIF? |
|  | Functional | 6 | Variable length collection value type | The value type shall support representation of variable length collections of values that are typed by a collection value type. |  |  |
|  | Functional | 7 | Collection item type | The collection value type shall define its item type that references another value type to type its permissible items. |  |  |
|  | Functional | 8 | Concrete collection value types | SysML v2 shall support the four basic kinds of collection value type, being: bag value type, set value type, sequence value type and ordered set value type. | The given kinds of collection correspond to the four different combinations of ordered / unordered and unique / non-unique items in the collection, similar to OCL and many programming / modeling languages. |  |
|  | Functional | 9 | Discretely sampled function value type | The value type concept shall include representation of a discretely sampled function definition, typed by a sampled function value type with an option to define an interpolation scheme for derivation of the function 'range' values for 'domain' values in between sampled values. | In many engineering applications there is a need and an established practice to represent observations or predictions as discretely sampled sets of independent and dependent values, both for univariate (scalar) and multivariate (vector space) sets. | Note 1: This is a very general type to represent discretely sampled tabular data, such as time series, frequency spectra, temperature-dependent material properties, pressure-and-enthalpy-dependent material properties, etc. Note 2: The interpolation scheme can be defined through an 'interpolation kind' being one of Polynomial, Linear-Logarithmic, Logarithmic-Linear, Logarithmic-Logarithmic, an 'interpolation degree' which is a positive integer that is only applicable if the 'interpolation kind' is Polynomial, and optionally a 'cyclic interpolation modulus' that defines that the dimension of a 'domain' value type is cyclic with the given modulus.  Note 3: In addition ordering may be defined for: strictly monotonic increasing, monotonic increasing, monotonic decreasing, strictly monotonic decreasing |
|  | Functional | 10 | Domain and range types of sampled function value type | The sampled function value type shall define an ordered set of 'domain' value type and an ordered set of 'range' value type. |  | Note 1: The set of 'domain' value type defines the type of the domain value tuples of the function and the set of 'range' value type defines the type of the range value tuples of the function. |
|  | Functional | 11 | Instantiation of sampled function value type | The sampled function value type shall have as values pairs of tuples, where the first tuple consists of values for the 'domain' value type and the second tuple of values for the 'range' value type. |  |  |
|  |  | 12 | System element reference | SysML v2 shall support a system element reference value type that allows to reference any system elements in value expressions. |  | Note 1: It should also be possible to reference (the value of) a property of a system element using the established object oriented dot notation: e.g. 'myObject.myProp'. |
|  | Functional | 13 | Concept: quantity kind | SysML v2 shall support an abstract specialization of value type named quantity kind to generically represent any numerical value type. | The term 'quantity kind' is in line with VIM and SysML v1 for value types with a numerical value. | Question: Is it better to rename quantity kind to quantity type, as 'type' is the more common term in data modeling, and UML / SysML use the postfix 'Kind' by convention for enumeration type identifiers? |
|  | Functional | 14 | Concept: quantity | SysML v2 shall support the concept quantity which is a value element that is typed by a quantity kind and has a numerical value as well as (a reference to) a measurement scale, that defines how to interpret the numerical value. |  | Question: There are two alternatives to associate a measurement scale with a quantity: (1) at the level of the quantity kind or (2) at the level of the individual quantity, where it shall be selected from the set of possible measurement scales defined for the relevant quantity kind. What is preferred? |
|  | Functional | 15 | Possible measurement scales | The definition of a quantity kind shall comprise a set of possible measurement scales, with at least one member, of which one is selected as the default (i.e. preferred) measurement scale. | Any numerical value (quantity) in SysML v2 must be unambiguously associated with a measurement scale.  The set of possible measurement scales permits an implementation to present a user with a list of scales to select from. Such a limited set of scales is needed because for each quantity kind a huge set of theoretically valid but in practice nonsensical measurement scales can be defined. Examples of nonsensical but valid measurement scales are a 'terametre per day' ratio scale for velocity, and a 'microwatt.second' ratio scale for energy. | Note 1: In order to get a complete and consistent approach, a non-dimensional quantity (e.g. a coefficient, ratio or dimensionless characteristic number) would be associated explicitly with the 'ratio measurement scale of dimension one'.  Note 2: A specialized quantity kind should inherit the possible measurement scales from its general super type quantity kind(s). |
|  | Functional | 16 | Concrete quantity kinds | SysML v2 shall support the following concrete specializations of quantity kind: (1) simple quantity kind, (2) derived quantity kind, (3) specialized quantity kind. | A simple quantity kind is a quantity kind that does not depend on any other quantity kind. A derived quantity kind is a quantity kind that is derived to be the product of powers of one or more other quantity kinds. A specialized quantity kind is a quantity kind that is a specialization (more narrow type) of another quantity kind. | Note 1: specialized quantity kind should be aligned with a generic capability to specialize a value type.  Example 1: Examples of simple quantity kinds are 'length', 'mass', 'time'. Example 2: Examples of derived quantity kinds are 'speed' ('length' to the power 1, 'time' to the power -1), 'power' ('energy' to the power 1, 'time' to the power -1). Example 3: Example of a specialized quantity kinds are 'radius' and 'altitude' (specializations of 'length'), 'maximum speed' (specialization of 'speed'), 'mass fraction' (specialization of 'fraction'), 'Reynolds number' (specialization of 'quantity of dimension one'). |
|  | Functional | 17 | Quantity conversion between measurement scales | SysML v2 shall capture all information necessary to perform automated conversion of the value of a quantity expressed on one measurement scale to the value expressed on another (compatible) measurement scale. | In a multi-user, multi-disciplinary, multi-organization modeling environment it is essential to be able to automatically align (sub-)models coming from different sources on a harmonized set of measurement scales. |  |
|  | Functional | 18 | Concept: system of quantities | SysML v2 shall support the definition of a system of quantities which consists of a core set of base quantity kinds that define the fundamental set of quantity dimensions of the system of quantities, and an additional set of derived quantity kinds and specialized quantity kinds that completes the system of quantities for a given application domain. | This permits to define any standard or customary system of quantities. |  |
|  | Functional | 19 | Acyclic value type graph | SysML v2 shall ensure that any value type definition forms an acyclic graph, i.e. there exist no circular references between value type definitions. |  |  |
|  | Functional | 20 | Support for ISO/IEC system of quantities | SysML v2 should support the "International System of Quantities (ISQ)", as defined in ISO/IEC 80000 and VIM. |  | Note 1: This is expected to take the form of a 'normative model library'.  Note 2: In addition to the seven base quantity kinds of ISO/IEC 80000: 'length', 'mass', 'time', 'electric current', 'thermodynamic temperature', 'amount of substance', 'luminous intensity', the concept of 'quantity of dimension one' should be modeled explicitly for reasons of consistency and to allow specializations, e.g. characteristic dimensionless numbers like 'Reynolds number'. |
|  | Functional | 21 | Concept: system of measurement units | SysML v2 shall support the definition of a system of units, containing as a main concept measurement unit, which consists of a core set of base measurement units that define the units for each of the base quantities of the associated system of quantities, and an additional set of derived units that completes the system of units for a given application domain. |  | Note 1: This permits to define any standard or customary system of units. |
|  | Functional | 22 | Concrete measurement units | SysML v2 shall support the following concrete specializations of measurement unit: (1) simple unit, (2) derived unit, (3) conversion based unit, (4) linear conversion unit, (5) prefixed unit. | The given specializations are defined in SysML v1.4 Annex E.5 (QUDV). | Note 1: SysML v1.4 E.5.2.1 AffineConversionUnit shall not be supported. Affine unit / scale conversion will be supported through interval scale. |
|  | Functional | 23 | Support for ISO/IEC system of units | SysML v2 should support the "International System of Units (SI)", as defined in ISO/IEC 80000 and VIM, as well as the associated system of measurement scales. |  | Note 1: This is expected to take the form of a 'normative model library'.  Note 2: In addition to the seven base measurement units of ISO/IEC 80000: 'metre', 'kilogram', 'second', 'ampere', 'kelvin', 'mol', 'candele', the concept of 'one' should be modeled explicitly for reasons of consistency and to allow its use in the defition of dimensionless measurement scales. |
|  | Functional | 24 | Concept: system of measurement scales | SysML v2 shall support the definition of a system of measurement scales, containing as a main concept measurement scale, in conjunction with a system of units. | The concept of measurement scale is needed in addition to measurement unit to support all possible automated conversions of quantities between compatible measurement scales, besides ratio scales also ordinal, interval, logarithmic and cyclic ratio scales. With exclusively a measurement unit it is only possible to (implicitly) define a ratio scale, i.e. a scale where zero is zero no matter what unit is selected and which supports the mathematical operations: addition, subtraction, multiplication and division. | Note 1: In most standards on quantities and units (including ISO/IEC 80000 and VIM) the main emphasis is on ratio scales which are applicable to the large majority of quantities. However in a semantically clean and complete data model the other relevant measurement scales must be supported robustly.  Note 2: The 'system of units' and the 'system of measurement scales' could potentially be merged into a single 'system of measurement units and scales'. |
|  | Functional | 25 | Concrete measurement scales | SysML v2 shall support the following concrete specializations of measurement scale: (1) ratio scale, (2) cyclic ratio scale, (3) logarithmic scale, (4) interval scale, (5) ordinal scale. |  | Note 1: The ratio scale is the most common type of measurement scale. In fact any ISO/IEC 80000 measurement unit constitutes a corresponding measurement scale. On a ratio scale the following mathematical operations are defined: subtraction, addition, multiplication and division. |
|  | Functional | 26 | Concept: constant | SysML v2 shall support the concept of constant, i.e. a named quantity with an invariant value, in a global model context, and optionally in lower level contexts. | Explicit modelling of constants is needed in order to support reproducible model execution. | Examples: mathematical constant 'π = 3.14159... ', physical constant 'speed of light in vacuum = 299,792,458 m/s', analysis case definition constant 'temperature convergence criterion = 0.01 K'. |
|  | Functional | 27 | Concept: numerical precision | SysML v2 shall support the definition of the numerical precision for values of a quantity kind. |  | Note 1: The precision can be defined in terms of the minimum number of decimal digits. |
|  | Functional | 28 | Support for undefined value | SysML v2 shall support a special 'undefined' value for any value element. |  | Note 1: 'undefined' means no value has been assigned to the value element. In many data modeling / programming languages this is equivalent to a 'null' value. |
|  | Functional | 29 | Support for not a number, infinity | SysML v2 shall support the following special values for any quantity: 'not a number' (NaN), 'positive infinity', 'negative infinity'. |  | Note 1: The value 'not a number' (NaN) for a quantity can be regarded as a specialization of 'undefined' for the value of a value element in general. |
|  | Functional | 30 | Support for mathematical number set | The measurement scale concept shall comprise the mathematical number set that applies to quantity values expressed on that measurement scale. |  | Note 1: Possible 'mathematical number sets' are: set of natural numbers (N); set of integer numbers (Z); set of rational numbers (C); set of real numbers (R). |
|  | Functional | 31 | Support for computer data type | The measurement scale concept shall allow to define the computer data type that applies to quantity values expressed on that measurement scale. |  | Note 1: The 'computer data type' should include definition of number of bits, bit ordering, encoding, ...  Examples: Possible 'computer data types' are: integer, big endian int32, int64, IEEE 754 binary32, IEEE 754 binary64, IEEE 754 binary128, double (usually same as IEEE 754 binary64), unsigned int16, etc.  Question: Should a generic data model to define a 'computer data type' be added? It may be better to refer to other existing standards. |
|  | Functional | 32 | Support for minimum and / or maximum value on scale | The measurement scale concept shall include the possibility to define a permissible value range through inclusive or exclusive minimum and/or maximum values. |  | Note 1: As a default, for any newly defined measurement scale, the minimum value should be negative infinity and the maximum value should be positive infinity, both inclusive. |
|  | Functional | 33 | Value assigment | SysML v2 shall support assignment of a value to a value element by a literal value or a value expression. |  |  |
|  | Functional | 34 | Concept: value expression | SysML v2 shall support the definition of a value expression, in which (references to) value elements can be used as 'variables', together with all standard mathematical operators and functions, as well as boolean and string operators and functions. |  | Note 1: It is probably wise and practical to select an existing modeling language with strong expression capabilities like Modelica verbatim or as guidance, rather than to develop a full expression language from scratch. Question: Should SysML v2 also support evaluation of such expressions? |
|  | Functional | 35 | Support for expression operators | SysML v2 shall support a normative minimum set of mathematical, boolean and string operators and functions. |  | Note 1: The normative minimum set of operators and functions should probably be implemented as a normative model library.  Note 2: The initial set of functions could be taken from the set of functions defined in MathML and Modelica. |
|  | Functional | 36 | Concept: parametric constraint | SysML v2 shall support the formulation of a declarative constraint, defined as a mathematical equality or inequality between two value expressions, in which the value elements that are in scope can be used as variables. | The definition of SysML constraints must be aligned with the value elements and value types. | Note 1: This corresponds to SysML v1 parametric constraint blocks. |
|  | Functional | 37 | Concept: operation | SysML v2 shall support the formulation of operations (as defined in UML and SysML v1), where value types are used to define the formal parameters and return type for the signature of the operation. |  |  |
|  | Functional | 38 | Algorithmic body of operation | SysML v2 shall support the formulation of the algorithmic body of an operation in 'imperative', 'declarative' or 'functional' style. |  | Note 1: It is probably wise and practical to select an existing modeling language with strong expression capabilities like Modelica verbatim or as guidance, rather than to develop a full expression language from scratch. Question: Should SysML v2 also support the execution of such operations? |
|  | Design Constraint | 39 | ValueType migration from SysML v1 | SysML v2 shall enable automated import and transformation of ValueType definitions as well as QUDV model libraries from SysML v1.x models into the new SysML v2 formulation. | Capability to migrate SysML v1 to v2 models |  |
|  | Functional | 40 | State dependent property values | SysML v2 shall support making a property state dependent, which means associating a property with a (possibly nested) set of finite states, and enabling assignment of a distinct value for each defined finite state. |  | Note 1: In case of nested sets of finite states, some combinations of finite states could be marked "not used" or "impossible", therefore have an undefined value. Example 1: For a spacecraft design, assume that the two finite state sets are defined: (1) "mission phase" with 7 states {"pre-launch", "launch-ascent", "transfer-orbit", "commisioning", "operations", "de-commisioning", "disposal"}, and (2) "system mode" with 5 states {"hibernation", "safe", "stand-by", "basic operation", "science operation"}, then a 'property' "consumed power" can be made 'state dependent' on the combination of "mission phase" and "system mode" with 35 (potentially different) values. In this example e.g. the combination of "launch-ascent" and "science-operation" would be marked "not used". |
|  | Functional | 41 | Variant dependent property values | SysML v2 shall enable making a property variant dependent, i.e. making a 'property' dependent on a set of variants, and enabling assignment of a distinct value for each variant. |  |  |
|  | Functional | 42 | Property inheritance and value override in element usage | SysML v2 shall support the possibility to to override the value of a property of an element usage, where the usage's property and default value are inherited from its defining element definition. | An element usage (SysML v1 'Part' or 'Port') inherits the properties of its typing element definition (SysML v1 'Block'). For some kinds of property there is a need to override the value of a property at the element usage level. | Example 1: Suppose a physical element (Block) has 'location' and 'orientation' properties, that define its placement in 3D space with respect to the coordinate frame of another physical element. Then when such a physical element (Block) would be used as a usage (Part) contained by another physical element it must be possible to give the 'location' and 'orientation' 'properties' different values for each usage. The typical example for this is a 'Car' that contains 4 usages of a 'Wheel': for 'front\_left', 'front\_right', 'rear\_left', 'rear\_right' wheels, each with different values for 'location' and 'orientation'. |
|  | Functional | 43 | Value type name | Every value type shall have a name that is a human-readable natural language character string that uniquely identifies the value type within the context of a name space. |  | Note 1: As a general principle it is proposed to identify all SysML v2 model elements consistently with a UUID, a name and a short name, where the UUID is the authoritative invariant identifier. |
|  | Functional | 44 | Value type short name | Every value type shall have a short name that is a character string that uniquely identifies the value type within the context of a name space. | In addition to name it is very useful to be able to identify model elements by a user-defined short name that can be used where limited space is available, for example in the column header of a table or as a variable (identifier) in a value expression. | Note 1: A short name may be an acronym or an abbreviation. |
|  | Design Constraint | 45 | Lexical constraint on short name | The short name of a value type shall begin with an ASCII letter and further only contain ASCII letter, underscore and digit characters. | The permitted value for short name is constrained because it needs to be compatible with use as an identifier (of a variable, parameter or object-property) in the majority of programming and modeling languages in order to facilitate mapping value types and value elements to other non-SysML models. | Note 1: Upon first read – now that Unicode is supported in many modeling and programming languages – it may seem backward to only support ASCII characters in a short name, but in a highly multi-disciplinary system modeling environment in which many different modeling tools need to be connected, it is a major interoperability enabler to use the least common denominator so that the short name identifiers of model elements are virtually always one-to-one mappable on the identifiers used in non-SysML modeling environments / tools. |
|  | Functional | 46 | Value type symbol | Every value type shall have a symbol that is a character string that identifies the value type. | The symbol of a value type allows capturing standard and conventional nomenclature for value types, in particular quantity kinds. | Note 1: Any Unicode characters should be permitted for symbols. |
|  | Functional | 47 | Value type description | Every value type shall have a description that allows for a human readable textual description of its definition and intended use. |  |  |
|  | Functional | 48 | Property name derivation | Every property shall derive its name, short name and description from the value type by which it is typed. | A natural way to ensure consistent use and reuse of properties of system elements is to define subtle variations of properties at the value type level with a meaningful name and short name and then use them throughout a SysML model. | Note 1: This is in line with the approach taken in RDF/OWL where data and object properties are defined in the global name space. It resolves a lot of mapping and interpretation problems including the diamond name-clash pattern in multiple inheritance.  Note 2: This is a deviation from UML where the name of a property is user-definable in the local namespace of the classifier by which it is owned.  Note 3: For reasons of one-to-one mapping to SysML v1 models or other particular use cases this may need to be an optional feature. |
|  | Functional | 49 | Measurement scale compatibility | SysML v2 shall be capable of determining the compatibility of any pair of measurement scales, where compatibility means that a quantity expressed on the first measurement scale can be expressed on the second measurement scale (typically with a different numerical value) without changing the 'essential value' of the quantity. | In order to perform automated measurement scale conversion of quantity values an algorithm to establish measurement scale compatibility is necessary. |  |
|  | Functional | 50 | Quantity value conversion between measurement scales | SysML v2 shall be capable of converting any quantity in a system model from a given measurement scale to another compatible measurement scale. |  | Note 1: The rationale is as follows: (1) When integrating incoming system models from different sources (e.g. from other organizations or disciplines) into a larger system model, one needs to unify the set of used 'measurement scales' across multiple constituent models in order to get a well-defined and manageable new system model. (2) When exporting a system model to an external engineering analysis environment it is often necessary to convert the 'quantities' in a model to a given set of 'measurement scales'. |
|  | Functional | 51 | Support for probabilistic values | SysML v2 shall support representation of the value of a value element with a probabilistic value, including a probability distribution. | There are many use cases in which non-nominal values need to be represented to include uncertainties, inaccuries, errors / error bars, margins, safety factors, etc. | Note 1: The details of what probabilistic value definitions to support need to be elaborated. Typically it will be a mean value with some variance and a probability distribution. |
|  | Functional | 52 | Support for value interval | SysML v2 shall support representation of the value of a value element as a value interval that is defined by inclusive or exclusive minimum and maximum values. |  | Note 1: A value interval is a specialization of a probabilistic value with a uniform probability distribution of one. |
|  | Functional | 53 | Support for probability distribution | SysML v2 shall support representation of a probability distribution, including probability density functions (continuous) and probability mass functions (discrete). |  | Note 1: The minimum set should cover the distributions defined in SysML v1 Annex E.7: interval, uniform, normal, and in addition: |

ADD GLOSSARY OF TERMS (LEVERAGE MDA GUIDE)

TBD