Conceptual Behavior Ontology v1.1

Summary

This page defines the concepts and relations necessary to capture state-based behavior of elements (named `BehavingElements`) and the interactions among them.

Here, state-based means that the behavior is the state evolution of the system, not that it is necessarily captured using a state-space representation. Elements can be components (e.g., sensors, actuators) or environments, characterized by `StateVariables` that vary with time. This ontology makes a distinction between the internal behavior of a `BehavingElement` and the interaction behavior among `BehavingElements`. Both are captured using constraints on appropriate `StateVariables`. These constraints are not limited to a system of first-order differential equations as it would be in an exclusive state-space representation.

The ontology is *Conceptual* because it captures these concepts & relationships without regard to an implementation (in IMCE's case: via the SysML language). Having a Behavior Ontology that is independent from its implementation language has two distinct benefits:

1. it makes it easier to understand since the concepts and relationships are not muddied by implementation-specific concepts;
2. it allows for the possibility of easily changing/adapting the implementation should languages other than SysML be used.

Note that the concepts of scenario and trajectory are included in the behavior ontology, but will be moved to an upcoming standalone ontology.

This page gives a detailed description of both the concepts and relations within the ***Conceptual*** Behavior Ontology, as well as simple examples to help clarify the semantics. Fully-fledged examples (simple flashlight example and spacecraft power and data example) are provided her e and specific links to that page for each concepts are included in the narrative. This page also has a section listing the validation & well-formedness assertions that apply to the ***Conceptual*** B ehavior Ontology. These assertions could be used by validation tools to explicitly check that a particular model instance conforms to a proper behavior specification.

<u>Note</u>: This ***Conceptual*** version of the Behavior Ontology cannot be embedded directly into SysML due to some current restrictions (see SysML-Embeddable Ontology & Implementation p age, Section "SysML-Embeddable Behavior Ontology - Description" for details). Because of this, a related ***SysML-Embeddable*** Behavior Ontology was developed to enable actual behavior modeling in SysML and is described here. [**Note that this ontology is only a temporary solution until infrastructure enhancements are made to handle the construction of a SysML profile based directly on the Conceptual ontology.**]

Page Table of Contents:

- Summary
- Key Features of the Ontology
- Ontology Segments
    - Ontology Representation Conventions
        - General Ontology Conventions
        - Convention for Diagram Representations
        - Conventions for Table Information
        - Segment 1: BehavingElement Taxonomy
        - Segment 2: Properties (State Variables and Parameters)
        - Segment 3: Internal Behavior (Intra-Element)

        - Segment 4: External Behavior (Inter-Element Interactions)
        - Segment 5: Scenarios and Trajectories
    - Combined Behavior Ontology
        - Abstract classes
        - Complete ontology diagram
- Conceptual Validation/Well-Formedness Assertions
- Conceptual Examples
- Discussion about InteractionTerminal
    - Additional validation rules

Key Features of the Ontology

There are a few key features of the ontology introduced here to provide some overall context for the specific details of the ontology diagrams.

1. The ontology supports both the modeling of behavior at the system-level and at the component-level: in the latter case, the behavior of the system is the sum of the components' behaviors and their interactions.
2. The ontology builds on the concepts of internal (intra-element) and external (inter- or between- element) behaviors.
3. In this ontology, behavior properties are not directly included with the system elements whose behavior is being described. Instead, the IMCE characterization pattern is used to maintain the properties in separate but explicitly linked entities. It allows for example for multiple engineering authorities (e.g. mechanical, thermal, fault protection, electrical, etc) to apply distinct descriptions of the behaviors that are most pertinent to each domain. In this ontology, both internal and external behavior specifications are considered to be characterizations.

Ontology Segments

Page Top

The following section gives a detailed description of the ***Conceptual*** Behavior Ontology by breaking the contents into 5 segments for clarity. For each segment, ontological diagrams shows concepts and the relations between concepts, and each of the concepts and relations (including their inverse) are also explained with a short prose description; examples to illustrate specific semantics are given as needed.

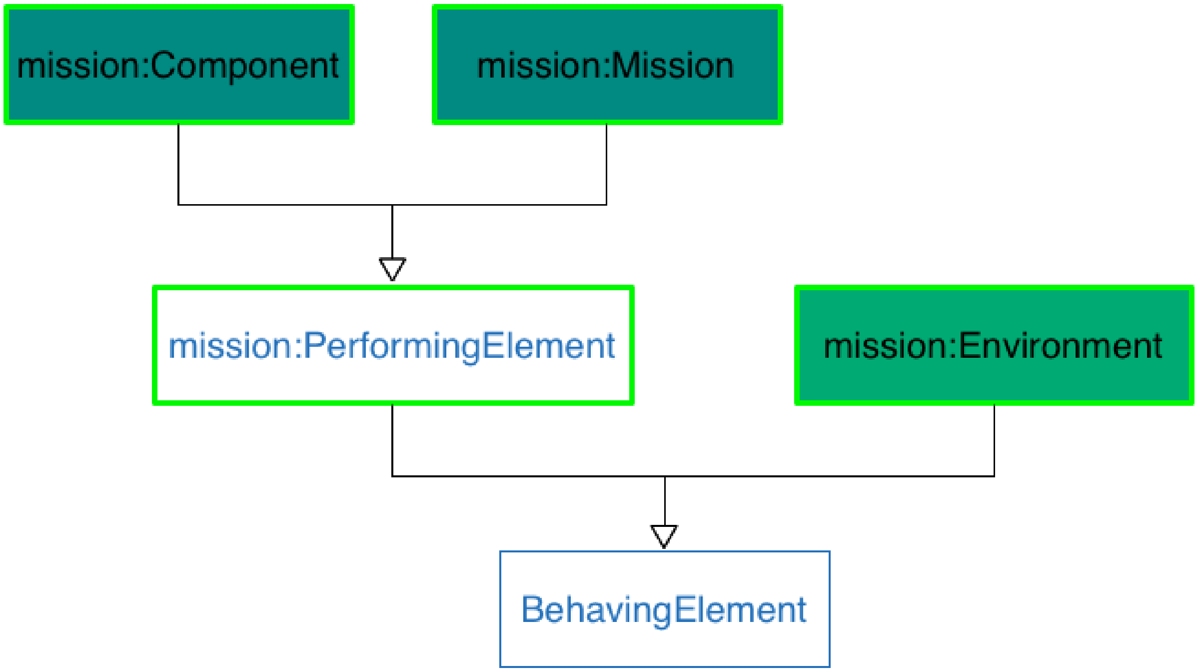
Ontology Representation Conventions

Expand to show the conventions used for describing the ontology on this page:

Segment 1: BehavingElement Taxonomy

`BehavingElement` is the central concept in this ontology - it is an abstract classifier that specifies which system elements can have some behavior characterization specified about them. Specifically, it indicates which system elements have notion of state and a prescribed way of evolving that state over time.

The diagram below shows the taxonomic structure of `BehavingElement`. `m:PerformingElement` (with `m:Component` and `m:Mission` as subclasses) and `m:Environment` (already existing IMCE Foundation concepts) specializing the abstract `BehavingElement`, meaning both of these elements can have some behavior characterization specified about them. The IMCE mission ontology has been modified accordingly.



Concept Descriptions:

Page Top | Segment 1 Top

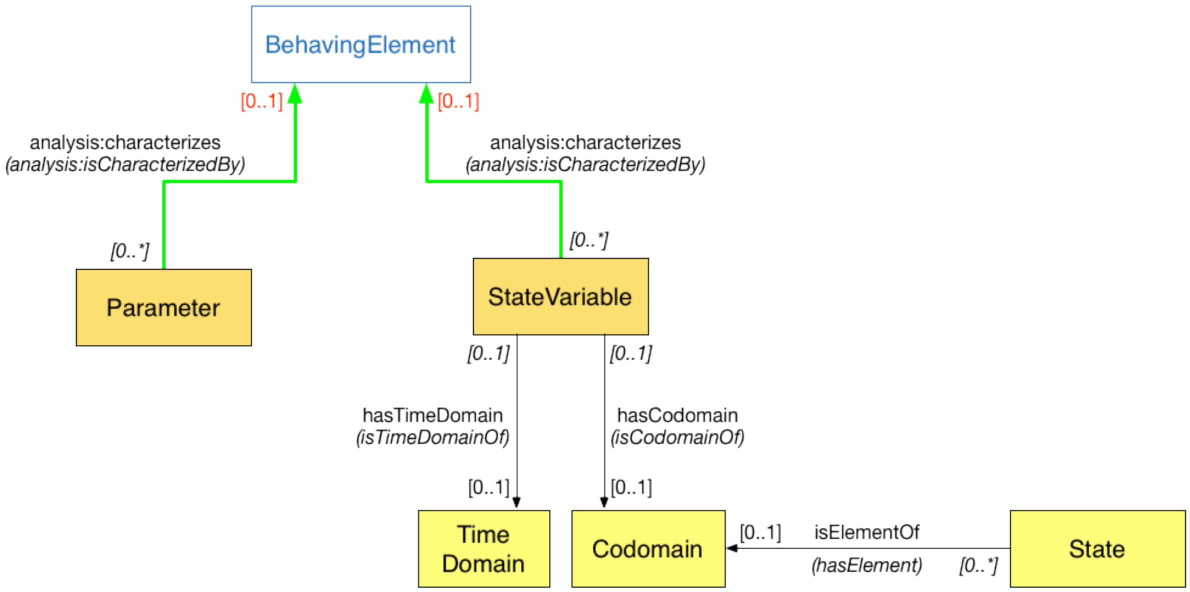
Concept	Description	Notes
BehavingElement	An abstract classifier that specifies which system elements have some behavior characterization.	Abstract class that generalizes `m:Component` and `m:Environment`
m:PerformingElement	Link to IMCE documentation: "object that performs one or more Functions".	Disjoint with the concept of `m:Environment`. Abstract class specialized by `m:Component` and `m:Mission`
m:Component	Link to IMCE documentation. A `m:Component` is any physical or logical entity/element of a system that is designed/specified through an engineering process. Naturally occurring items, such as the Jovian moon Europa or the Jovian radiation environment, are not considered `m:Components` (instead they are `m:Environments`).	Disjoint with the concept of `m:Environment`. A `m:Component` representation is valid at any level within a system hierarchy (part, assembly, subsystem, or whole system). It can also represent logical entities such as software modules or human entities such as operations teams.
m:Mission	Link to IMCE documentation. `m:Components` are deployed by a `m:Mission`.	Disjoint with the concept of `m:Environment`.
m:Environment	Link to IMCE documentation. A `m:Environment` corresponds to a set of conditions in which a `m:Component` must perform its `m:Functions`. Example `m:Environments` include low earth orbit, trans-Jupiter cruise, and Martian north polar surface.	Disjoint with the concept of `m:PerformingElement`.

<u>Note about `m:Functions`</u>: One difference between `m:PerformingElement` and `m:Environment` is that the former performs functions ("`m:performs m:Functions`") while the latter does not. The relation between `m:Functions` and `BehavingElements` is not tackled in this pattern, but should be the topic of a future pattern.

Segment 2: Properties (State Variables and Parameters)

Page Top

This segment introduces the concepts of `StateVariables` and `Parameters`, that are used in the expression of the behaviors of `BehavingElements` or their interaction. The structure of a `StateVariable` is further elaborated with the concepts of `TimeDomain`, `Codomain`, and `State`. Some of these concepts are formally defined from a set theoretic point of view.

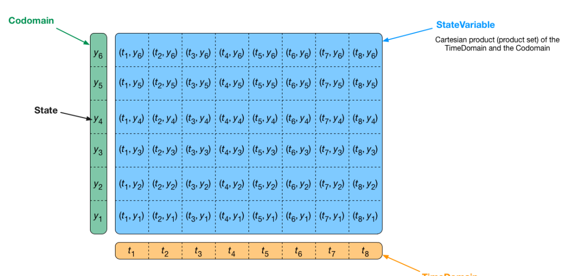


Concept	Description	Notes

StateVariable	A variable that represents a particular quantity (as per ISO-80000) of a `BehavingElement` that changes with time. Using set theory representation, we define a `StateVariable` as the Cartesian product of its `TimeDomain` *X* and its `Codomain` *Y*: $X \times Y \equiv \{(x,y) \mid x \in X \wedge y \in Y\}$ See figures below this table for visualization aid.	***def***. <u>quantity</u> (as per ISO-80000): "property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed by means of a number and a reference". E.g, voltage, current, temperature. <u>For a given scope</u>, the set of all `StateVariables` for a particular `BehavingElement` will completely describe how the state of the `BehavingElement` changes in time within that scope. For example, in the context of an electrical current loop analysis, component voltage, current, and impedance may be the only `StateVariables` that are required to adequately describe the state evolution of that component. It is the task of the modeler to determine the appropriate modeling scope for a given design need, and what the set of `StateVariables` are that cover that given scope. Note that `StateVariables` can be constrained to be constant in time. For example: Power(t) = 10 Watts. `StateVariables` can also represent multidimensional quantities (e.g. vector quantities or quaternions) through the definition of a multi-dimensional `Codomain`. A `Trajectory` is defined as the "value" of a `StateVariable` (see `Trajectory` for details).
TimeDomain	The set of possible times that belong to a particular `StateVariable`. See figures below this table for visualization aid.	`TimeDomains` can represent continuous time or discrete time. A `TimeDomain` that is continuous is always uncountably infinite (even though it may be bounded by maximum and minimum values), but a discrete `TimeDomain` may either be countably finite (bounded) or countably infinite (unbounded).
Codomain	The set of possible values of a quantity represented by a `StateVariable`. For practical usage, `Codomains` are usually assigned a type (quantity kind) and possibly a unit. However, this structure will not be detailed here. See the SysML-Embeddable Behavior Ontology for more details on that particular specification. See figures below this table for visualization aid.	`Codomains` can either be continuous or discrete, including enumerated lists. For example, temperature in Celsius may be represented as the following continuous `Codomain`: {T | T -273.15 °C} As another example, the operational mode of a radio may be represented as the following discrete enumerated list `Codomain`: {"OFF","IDLE","LOW_POWER","HIGH_POWER"} `Codomains` can also represent multi-dimensional values. For example, the 3-D position of a spacecraft in a particular navigation frame may be represented with the following `Codomain`: {(x, y, z) | x, y, z } ^{3}

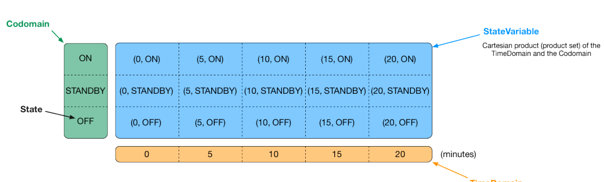
State	A possible value of the quantity of a `BehavingElement`, and in effect an element of the `Codomain` of a `StateVariable`. (See figures below this table for visualization aid.) `States` need not be singular atomic values; they can be structured to represent more than one dimension (for example, a vector quantity).	Given the examples in `Codomain` Notes section above, here are example `States` of the presented `Codomains`: * for temperature: T_1 = 10.37 °C * for radio mode: operational mode = "LOW_POWER" * for spacecraft position: (x_1, y_1, z_1) = (3.5 m, 10.2 m, 8.45 m) With the exception of discrete enumerated lists ("ON", "OFF", "IDLE", etc…), complete explicit definition of `States` for a given `Codomain` will most likely not exist in models (for reasons of practicality since there can be many or even infinite `States` belonging to a given `Codomain`). Instead, an explicit `Codomain` will be defined that restricts the `States` it contains to a certain set. An example of `States` explicitly defined would be in a behavior model specified by a state machine; another case could the threshold levels for thermal sensors or fault monitors. Also, it is an implied semantic that all `States` within a particular `Codomain` are unique (it would not make sense to have two values of "ON" for a mode `StateVariable` or two values of 20.0V for a voltage `StateVariable`). For now, this uniqueness check is left to the modeler. Note that in other contexts, the word "state" is used to refer to the element of a state variable set. This is not the case here: a `State` is an element of the `Codomain` of a `StateVariable`, and the element of the `StateVariable` (as a Cartesian product) is a pair (time point, `State`). That element can also be conceived as a point on a `Trajectory` (see below).

Parameter	A quantity of a `BehavingElement` that does not dynamically change in time. A `Parameter` is similar to a `Codomain` in that a type (quantity kind) and unit are usually assigned for practical usage.	`Parameters` can be thought of as `StateVariables` that do not change in time, and hence they have no associated `TimeDomain`. Both `Parameters` and `StateVariables` are generalized by the abstract `Property` concept (see Segment 4 for details). Even though the values of `Parameters` are fixed in time, they can have their value changed between different analyses or simulation runs. For example, if you are evaluating a spacecraft attitude control system, you may have a `Parameter` in your model for the inertia of the spacecraft. You could vary the inertia values between analyses/simulation runs to see how well a particular control system (with its associated gains and other tuning parameters) handles variation in spacecraft inertia for a given set of control inputs. *<u>Usage of `Parameters` vs. `StateVariables`</u>*: The specific selection of a `Parameter` or `StateVariable` is left to the intent of the modeler: when s/he wants to capture the time dynamics of that quantity, then a `StateVariable` is the logical choice; when the dynamical aspect in time of the quantity is of no interest to the modeler, then a `Parameter` could be selected. However, nothing prevents the modeler to use exclusively `StateVariables` for example, but that choice could be at the expense of complexity or computation, as it is less expensive to track `Parameters` (single value) compared to `StateVariables` (time history to resolve).



The figure on the left represent the relationship between the `TimeDomain`, the `Codomain` and the `StateVariable` as their Cartesian product, and `State`. For clarity purposes, a discrete case was chosen, but these concepts are easily extended to the continuous case.

Example for the `StateVariable` "Operating mode" of a Television:

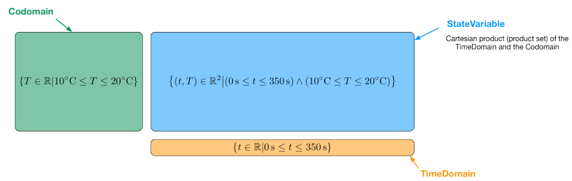


This is an example of a discrete `StateVariable`:

- its `Codomain` is made of 3 `States`: "OFF", "STANDBY", and "ON";
- its `TimeDomain` has been discretized and 5 time points have been selected for this example: 0, 5, 10, 15 and 20 minutes that are of interest;
- the `StateVariable` is defined as the Cartesian product of these `TimeDomain` and `Codomain` and

represents all possible (time point, state) combinations.

Example of the `StateVariable` "Temperature" of an entity:



This is an example of a continuous `StateVariable`:

- continuous `TimeDomain` where time can take values between 0 and 350 seconds;
- continuous `Codomain` where the `States` can be between 10 and 20 degrees Celsius. For example, 15 degrees Celsius is a `State`, 17.26879 degrees Celsius is another one;
- the `StateVariable` is the set of all (infinite) possible combinations of times and temperature states.

StateVariable Structure Relation Descriptions:

Page Top

NOTE: this table reads by columns

	hasTimeDomain (inverse)	**isTimeDomainOf** (inverse)	**hasCodomain** (inverse)	**isCodomainOf** (inverse)	**isElementOf** (inverse)	**hasElement** (inverse)
Subject	**StateVariable**	**TimeDomain**	**StateVariable**	**Codomain**	**State**	**Codomain**
Verb	**hasTimeDomain**	**isTimeDomainOf**	**hasCodomain**	**isCodomainOf**	**isElementOf**	**hasElement**
Multiplicity	**[0..1]**	**[0..1]**	**[0..1]**	**[0..1]**	**[0..1]**	**[0..*]**
Object	**TimeDomain**	**StateVariable**	**Codomain**	**StateVariable**	**Codomain**	**State**

Multiplicity Rationale	A `TimeDomain` is one of the two ***axial sets*** required for the complete definition of a `StateVariable` [the term ***axial set*** is used because when values of `StateVariables` (as time-based functions) are plotted, values from the `TimeDomain` set are on one axis and values from the `Codomain` set are on the other axis]. Therefore, a `hasTimeDomain` relationship between a `StateVariable` and exactly one `TimeDomain` is a necessary, but not sufficient, condition for the definition of that `StateVariable` to be semantically complete.	The current Behavior Ontology does not support the assignment of a particular `TimeDomain` to many `StateVariables` - separate `TimeDomains` need to be defined for each `StateVariable`. For example: Current_Power (t) for continuous times between 0 and infinity and Current_Power (t) for discrete times between 0 and infinity in 0.1 sec increments would be separate `StateVariables`	A `Codomain` is one of the two ***axial sets*** required for the complete definition of a `StateVariable`. Therefore, a `hasCodomain` relationship between a `StateVariable` and exactly one `Codomain` is a necessary, but not sufficient, condition for the definition of that `StateVariable` to be semantically complete.	Given that the `Codomain` is the set of possible values for the aspect represented by the `StateVariable`, it would be a conflict to have multiple `Codomains` associated with the same `StateVariable`.	It is semantically explicit that a `State` is defined in the context of a <u>specific</u> `StateVariable`. Therefore, a `State` can be an element of at most one `Codomain`.	A `Codomain` can contain as many `States` as necessary to define the unique dynamic values of the aspect captured by its associated `StateVariable`. `Codomain` sets may truly have an infinite number of `States` as elements (for example: in the cases of continuous-time or unbounded discrete-time `StateVariables`).
Description	Ownership relation indicating that a given `StateVariable` is directly associated with a particular `TimeDomain`.	Inverse of `hasTimeDomain`; indicates that a given `TimeDomain` is associated with a particular `StateVariable`.	Ownership relation indicating that a given `StateVariable` is directly associated with a particular `Codomain`.	Inverse of `hasCodomain`; indicates that a given `Codomain` is associated with a particular `StateVariable`.	Membership relation indicating that a given `State` belongs to the set of elements in a particular `Codomain`.	Inverse of `isElementOf`; indicates that a given `Codomain` has the related `State` as one of its element members.
Notes	none	none	none	none	none	none

<u>BehavingElement Characterization Relation Descriptions:</u>

Page Top | Segment 2 Top

NOTE: this table reads by columns

(note: Columns in this table are only partially green; the relations described here are special usages of other IMCE Foundation Ontology relations. The names are the same, but the implied semantics are more specific to the particular usage of the relations. Also, the multiplicity restrictions are different in the case of `a:characterizes`).

	a:characterizes (inverse)	**a:isCharacterizedBy** (inverse)	**a:characterizes** (inverse)	**a:isCharacterizedBy** (inverse)
Subject	**Parameter**	**BehavingElement**	**StateVariable**	**BehavingElement**
Verb	**a:characterizes**	**a:isCharacterizedBy**	**a:characterizes**	**a:isCharacterizedBy**
Multiplicity	**[0..1]**	**[0..*]**	**[0..1]**	**[0..*]**

Object	BehavingElement	Parameter	BehavingElement	StateVariable
Multiplicity Rationale	Having a `Parameter` `a:characterize` more than one `BehavingElement` would be ambiguous. The aspect/condition/property represented by the `Parameter` is about a specific `BehavingElement`, and this specific context would be lost if associated with multiple `BehavingElements`. This ontology requires that separate `Parameters` be defined for similiar aspects of different `BehavingElements` (for example, a thruster and a reaction wheel might both have a max power consumption, but these need to be modeled as two separate `Parameters` - one for each element).	A `BehavingElement` can be `a:characterizedBy` as many `Parameters` as necessary to define appropriate behavioral constraints.	Having a `StateVariable` `a:characterize` more than one `BehavingElement` would be ambiguous. The aspect represented by the `StateVariable` is about a specific `BehavingElement`, and this specific context would be lost if associated with multiple `BehavingElements`. This ontology requires that separate `StateVariables` be defined for similiar aspects of different `BehavingElements` (for example, a thruster and a reaction wheel might both have a current power consumption, but these need to be modeled as two separate `StateVariables` - one for each element).	A `BehavingElement` can be `a:characterizedBy` as many `StateVariables` as necessary to define appropriate behavioral constraints.
Description	Link to IMCE documentation For this specific usage of `a:characterizes`, it is being used in the sense of a description (the `Parameter` is describing an aspect present in the `BehavingElement`).	Inverse of `a:characterizes`; indicates that there is a `Parameter` describing some static aspect of the `BehavingElement`.	Link to IMCE documentation For this specific usage of `a:characterizes`, it is being used in the sense of a description (the `StateVariable` is describing a dynamic aspect of the `BehavingElement`).	Inverse of `a:characterizes`; indicates that there is a `StateVariable` describing some dynamic aspect of the `BehavingElement`.
Notes	Use of the `a:characterizes` relation here implies that `Parameter` is a specialization of `a:Characterization` and `BehavingElement` is a specialization of `a:CharacterizedElement`. *** In this specific usage of `a:characterizes`, the multiplicity restriction is tighter than in the overall definition of `a:characterizes`, being [0..*].	The current Behavior Ontology does not enforce maintenance of consistency/uniqueness between `Parameter` definitions (there could be two separate `Parameters` called ***max_power*** for the same `BehavingElement`). It is up to the modeler to check for logical violations in this area (either manually or through the use of specialized analysis scripts).	Use of the `a:characterizes` relation here implies that `StateVariable` is a specialization of `a:Characterization` and `BehavingElement` is a specialization of `a:CharacterizedElement`. *** In this specific usage of `a:characterizes`, the multiplicity restriction is tighter than in the overall definition of `a:characterizes`, being [0..*].	The current Behavior Ontology does not enforce maintenance of consistency/uniqueness between `StateVariable` definitions (there could be two separate `StateVariables` called ***X_position*** for the same `BehavingElement`). It is up to the modeler to check for logical violations in this area (either manually or through the use of specialized analysis scripts).

Segment 3: Internal Behavior (Intra-Element)

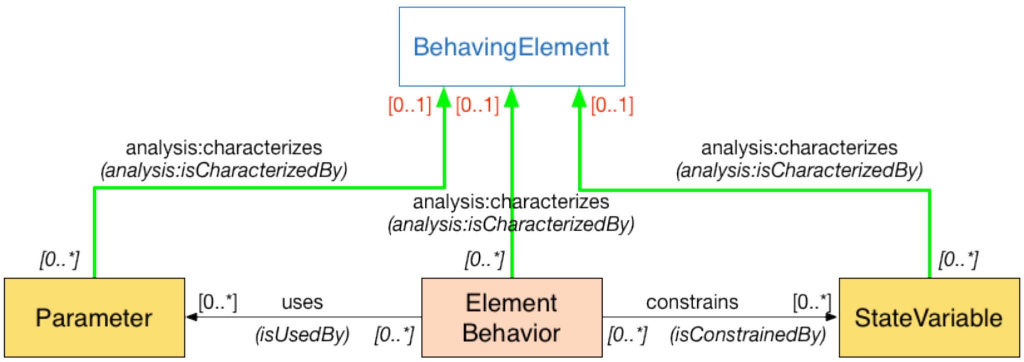
Page Top

Segment 3 of the Behavior Ontology includes the concepts necessary for describing the internal behavior for a particular `Behaving`

Element.

Internal behavior, captured by the ElementBehavior concept, is behavior that exists directly as a result of the intrinsic nature of the BehavingElement; in other words, it is endogenous. Internal behavior can include specification for how the BehavingElement responds to stimuli from or provides influence to external elements, but the internal behavior is not modified by these external stimuli.

In addition, the abstract concepts of StateVariableConstraint and UsingElement are included here are generalizations of ElementBehavior as well as other concepts introduced in Segment 4.



Concrete Concept Descriptions:

Page Top | Segment 3 Top

Concept	Description	Notes

ElementBehavior	This is a specification of how the dynamic state of a `BehavingElement` is allowed to evolve in time (but is not a specific trace of state evolution). `ElementBehavior` is conceptually internal or intrinsic in nature. However, this does not preclude `ElementBehavior` specifications from accepting external happenings (signals, commands, etc) as inputs, or providing outputs (further commands, data, etc) to external elements. For example, in the case of a variable resistor with adjustment knob, both Ohm's Law (V=I*R) and a calibration relation between knob position and resistance R would be the `ElementBehavior`. This representation would accept a knob position as input from some external element (maybe a human operator), but both Ohm's Law and the calibration relationship are not dependent on the position of the knob (see discussion on input in the Notes column). `ElementBehavior` is represented through constraints on `StateVariables`. `Parameters` can also appear in these constraints. **The specification of constraint expressions has not yet been finalized and is the object of an upcoming pattern**. It will most likely involve mathematical constraints, state machine formalisms, temporal-logic constraint, predicate logic statements, etc. In the examples presented in subsequent pages, mathematical expressions and state machines formalisms are employed, as they were the most logical and compact representations of the constraints in that context. *Note that the behavior pattern does not specify how to specify the constraints (this is left to the modelers and project internal agreement), only that the behavior is represented as a constraint on `StateVariables`.* It is implicit that the definition of any behavior is only valid within a scope, however large it may be (see Notes for further discussion). Currently, the ontology does not support the specification of the scope (see Open Questions).	Behavior specifications covering protocols/interactions between different elements of a system are covered in a separate concept called `InteractionBehavior`. `ElementBehavior` is semantically interpretable to be true under all possible operating conditions even though some of these conditions are most likely not realistic (e.g. 100MV across a standard carbon 1 ohm resistor, which would result in overheating and burnout - under these conditions, Ohm's Law no longer applies). Therefore, it is the responsibility of the modeler to ensure that a particular `ElementBehavior` specification is appropriate for the operating conditions the `BehavingElement` will likely see, and to re-evaluate the `ElementBehavior` specification if it is determined (through other analysis or simulation) that the range of reasonable operating conditions is different than originally considered. **Inputs** are not explicitly supported in the current version of the behavior pattern, and will be investigated in future work. It is of particular interest to drive the behavior of a system for example, or in linking triggers of state machines to a ontological concept of the behavior pattern.

BehavingElement Characterization Relation Descriptions:

Page Top | Segment 3 Top

NOTE: this table reads by columns

(note: Columns in this table are only partially green; the relations described here are special usages of other IMCE Foundation Ontology relations. The names are the same, but the implied semantics are more specific to the particular usage of the relations. Also, the multiplicity restrictions are different in the case of a:characterizes).

	a:characterizes (inverse)	a:isCharacterizedBy (inverse)
Subject	**ElementBehavior**	**BehavingElement**
Verb	**a:characterizes**	**a:isCharacterizedBy**
Multiplicity	**[0..1]**	**[0..*]**
Object	**BehavingElement**	**ElementBehavior**
Multiplicity Rationale	`ElementBehaviors` use `Parameters` and constrain `StateVariables` of only the single specific `BehavingElement` they characterize.	A `BehavingElement` can be `a:characterizedBy` as many `ElementBehaviors` as necessary to capture the dynamics of the `BehavingElement`.
Description	<<IMCE definition link>> For this specific usage of `a:characterizes`, it is being used in the sense of a restriction (the `ElementBehavior` is restricting the evolution of aspects of the `BehavingElement`).	Inverse of `a:characterizes`; indicates that there is an `ElementBehavior` constraining the `StateVariables` and/or using the `Parameters` of the `BehavingElement`.
Notes	Use of the `a:characterizes` relation here implies that `ElementBehavior` is a specialization of `a:Characterization` and `BehavingElement` is a specialization of `a:CharacterizedElement`. *** In this specific usage of `a:characterizes`, the multiplicity restriction is tighter than in the overall definition of `a:characterizes`, being [0..*].	The current ontology does not enforce maintenance of consistency/uniqueness between `ElementBehavior` definitions (there could be two separate `ElementBehaviors` with inconsistent constraints for same `BehavingElement`). It is left to the modeler to check for logical violations (either manually or through the use of specialized analysis scripts).

__Abstract Concept Relation Descriptions:__ (also applicable to Interaction and Execution Realm Segments)

Page Top | Segment 3 Top

NOTE: this table reads by columns

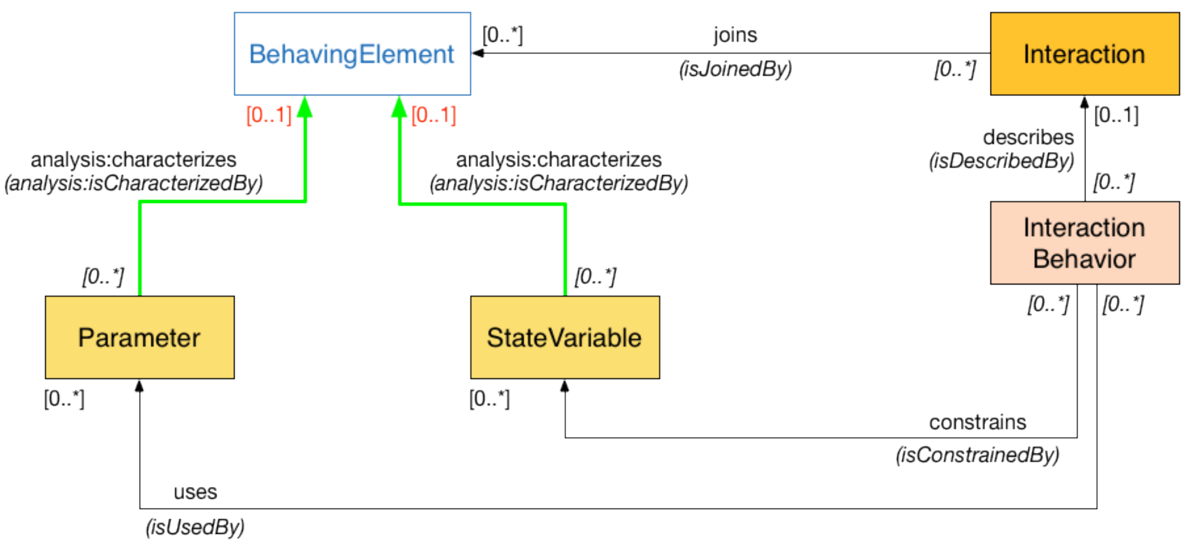
	constrains (inverse)	isConstrainedBy (inverse)	uses (inverse)	isUsedBy (inverse)
Subject	**ElementBehavior**	**StateVariable**	**ElementBehavior**	**Parameter**
Verb	**constrains**	**isConstrainedBy**	**uses**	**isUsedBy**
Multiplicity	**[0..*]**	**[0..*]**	**[0..*]**	**[0..*]**
Object	**StateVariable**	**ElementBehavior**	**Parameter**	**ElementBehavior**
	see generalizations for multiplicity rationale and descriptions			

Further Notes	Example of constraining multiple `StateVariables`: Ohm's Law (V(t) = I(t) * R) for an Ohmic resistor, where V(t) and I(t) are `StateVariables` and R is a `Parameter`. In this case a `ElementBehavior` `constrains` two `StateVariables`.		For example, one might have an exponential decay model for the power generated by an RTG power source. This model could include an equation with initial power and decay constant terms defined as an `ElementBehavior` (a type of `UsingElement`) - this would look like: Power(t) = ***Power_init*** * e ^ (-t). In this case, both ***Power_init*** and are `Parameters`.	Since the inverse `uses` relationship is essentially just a value reference for a `Parameter`, there is no limitation on the number of `UsingElements` that can reference its value.

Segment 4: External Behavior (Inter-Element Interactions)

Page Top

Segment 4 of the Behavior Ontology includes the concepts necessary for describing behavioral interactions among `BehavingElements`. This segment introduces the concepts of `Interaction` (to indicate which `BehavingElements` participate in the interaction) and `InteractionBehavior` to capture the constraints of the interaction.



Concrete Concept Descriptions:

Page Top | Segment 4 Top

NOTE: this table reads by columns

Concept	Description	Notes

Interaction	A link that joins `BehavingElements`, representing the context of the interaction among the associated `BehavingElements`. An `Interaction` does not represent any physical or logical entity of the system - it is merely a connection point that defines a context space where the exposed `StateVariables` and `Parameters` of the joined `BehavingElements` are allowed to affect one another through the definitions of `InteractionBehaviors`. `Interactions` represent N-ary interactions among `BehavingElements` (contrary to `m:Junction` that are strictly binary). This allows the modeler to choose potentially more efficient representations where the interaction expressions cannot be decomposed into binary relationships between `BehavingElements` (see the flashlight example for such a case as Kirchhoff's voltage law in a loop).	The defined usage for an `Interaction` is that it must join at least two different `BehavingElements`. Otherwise, the associated `InteractionBehaviors` could just be represented as `ElementBehaviors` instead. It allows for the specification that an interaction takes place without specifying exactly what the interaction among `BehavingElements`: this allows for example for multiple or competing descriptions of the interaction (e.g., different levels of fidelity) or for collaboration among different domain engineers.
InteractionBehavior	A specification describing how different `BehavingElements` interact with one another (by expressing the effects of `Property`(ies) (see below) of a `BehavingElement` on other `Property`(ies) of the `BehavingElements` it interacts with). `InteractionBehaviors` are represented through constraints on `StateVariables`. `Parameters` can also appear in these constraints. `InteractionBehaviors` are external in nature: they dependent on a particular deployment of `BehavingElements` in relation to each other. It is implicit that the definition of any interaction is only valid within a scope, however large it may be (as for `ElementBehavior`). Currently, the ontology does not support the specification of the scope (see Open Questions).	A difference between `ElementBehavior` and `InteractionBehavior` is that `ElementBehavior constrains/uses StateVariables/Parameters` of only one `BehavingElement`, while an `InteractionBehavior constrains/uses StateVariables/Parameters` of at least two different `BehavingElements`.

Interaction Relation Descriptions:

Page Top | Segment 4 Top

NOTE: this table reads by columns

	joins (inverse)	**isJoinedBy** (inverse)	**describes** (inverse)	**isDescribedBy** (inverse)
Subject	**Interaction**	**BehavingElement**	**InteractionBehavior**	**Interaction**
Verb	**joins**	**isJoinedBy**	**describes**	**isDescribedBy**
Multiplicity	**[0..*]**	**[0..*]**	**[0..1]**	**[0..*]**
Object	**BehavingElement**	**Interaction**	**Interaction**	**InteractionBehavior**

Multiplicity Rationale	This is a direct consequence of the N-ary nature of the `Interaction`.	The `StateVariables` of a given `BehavingElement` may be affected by many external interactions that are modeled through different `Interactions`.	`InteractionBehaviors` use `Parameters` and `constrain` `StateVariables` in the context of a single specific `Interaction` they `describe`.	A `Interaction` can be `describedBy` as many `InteractionBehaviors` as necessary to capture the dynamics of the interaction.
Description	Relation indicating that the `Property`(ies) of the `BehavingElements` are now in scope of the interaction represented by the `joining` `Interaction`.	Inverse of `joins`; indicates that there is an `Interaction` that `joins` the `BehavingElement`.	The constraints defined in the `InteractionBehavior` apply within the context of the `described` `Interaction`.	Inverse of `describes`; indicates that there is an `InteractionBehavior` that `describes` a given `Interaction`.
Notes	Also indicates that the `exposed` `Property`(ies) can now be referenced by the `InteractionBehaviors describing the Interaction`.	All of the associated constraints of all related interactions are levied on the `Property`(ies) of the `BehavingElements`.	none	none

InteractionBehavior Relation Descriptions:

Page Top | Segment 4 Top

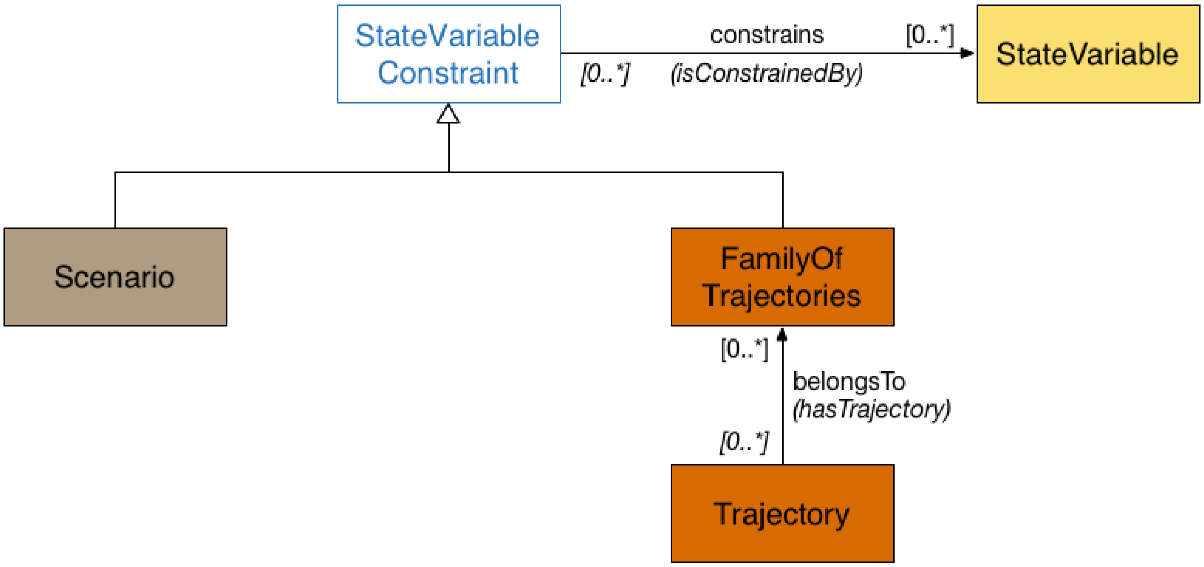
NOTE: this table reads by columns

	constrains (inverse)	**isConstrainedBy** (inverse)	**uses** (inverse)	**isUsedBy** (inverse)
Subject	**InteractionBehavior**	**StateVariable**	**InteractionBehavior**	**Parameter**
Verb	**constrains**	**isConstrainedBy**	**uses**	**isUsedBy**
Multiplicity	**[0..*]**	**[0..*]**	**[0..*]**	**[0..*]**
Object	**StateVariable**	**InteractionBehavior**	**Parameter**	**InteractionBehavior**
	see generalizations for multiplicity rationale and descriptions			

Segment 5: Scenarios and Trajectories

Page Top

Segment 5 of the Behavior Ontology introduces concepts related to the execution of system behavior: `Scenario`, `Trajectory`, and `FamilyOfTrajectories`. **Their full specification is currently in work and part of an upcoming separate pattern, but they are included in this pattern initially to describes how they relate to behavior**.

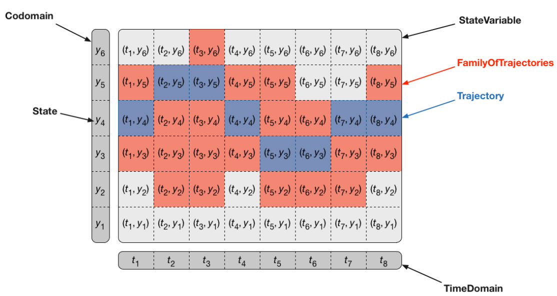


Concrete Concept Descriptions:

Page Top | Segment 5 Top

Concept	Description	Notes
Scenario	A set of constraints on `StateVariable` s and/or other specifications (such as initial conditions, system inputs or sequence of commands) without any guarantee that these will actually happen in any particular execution case. **Formal specification is currently in work and part of an upcoming pattern** . For one take on scenario modeling, see K. Donahue work on modeling Scenarios using SysML Activity Diagrams and the Timeline ontology (reference here).	For example: • In the flashlight example, the scenario indicates that the flashlight should be turned on and turned off at certain times • a sequence of planned file uploads to a spacecraft
Trajectory	*Functional subset* of a `StateVariable` that spans the entire `TimeDomain` (see figure for clarification). The set of constraints that define a `Trajectory` may result from the mathematical reduction of the set of `ElementBehaviors` and `InteractionBehaviors` within a particular system with the set of constraints from a `Scenario` under which that same system is executed. **Relation to Behavior and Scenario is being further developed and is part of an upcoming pattern**.	The *functional subset* in this context is a set of ordered pairs of times and `State` s in which no time values are repeated. Also, a `Trajectory` is defined to be a `StateVariable` "value". A Trajectory implies a fully-constrained behavioral execution of a given system.

FamilyOfTrajectories	Subset of a `StateVariable` that spans the entire `TimeDomain` (see figure for clarification), which can include many ordered pairs of times and `States` where the time values are repeated. This is also equivalent to a subset of a `StateVariable` defined by a set of `Trajectories`. The set of constraints that define a `FamilyofTrajectories` may result from the mathematical reduction or solving of the set of `ElementBehaviors` and `InteractionBehaviors` within a particular system with the set of constraints from a `Scenario` under which that same system is executed. **Relation to Behavior and Scenario is being further developed and is part of an upcoming pattern**.	none



This figure is an extension of the figure presented in Segment 2.

Concrete Concept Relation Descriptions:

Page Top | Segment 5 Top

	belongs **(inverse)**	**hasTrajectory** **(inverse)**
Subject	**Trajectory**	**FamilyOfTrajectories**
Verb	**belongsTo**	**hasTrajectory**
Multiplicity	**[0..*]**	**[0..*]**
Object	**FamilyOfTrajectories**	**Trajectory**
Multiplicity Rationale	Mathematically, a `Trajectory` can be a subset of many different `FamilyOfTrajectories`.	As per definition, a `FamilyOfTrajectories` is also equivalent to a subset of a `StateVariable` defined by a set of `Trajectories`.
Description	If the `Trajectory` is a subset of the `FamilyOfTrajectories`, then the `Trajectory belongsTo` it.	Inverse of `hasTrajectory`
Notes	none	none

Combined Behavior Ontology

Abstract classes

The `constrains` and `uses` relationships have each one target concept (`StateVariable` and `Parameter` respectively), but several source concepts:

- `ElementBehavior`, `InteractionBehavior`, `Scenario` and `FamilyOfTrajectory` for `StateVariable`;
- `ElementBehavior`, `InteractionBehavior` for `Parameter`.

To respect IMCE's "simple range class expression" rules (relationships have only one source concept and one target concept), we introduce the `StateVariableConstraint` and the `UsingElement` classes.

Concept	Description	Notes
UsingElement	Abstract concept representing a behavioral constraint element (either `ElementBehavior` or `InteractionBehavior`) that `uses` a particular `Parameter` value in its constraint specification(s). See the `uses` relation definition for an example of the usage of a `Parameter`.	Generalizes `ElementBehavior` and `InteractionBehavior`.
StateVariableConstraint	Abstract concept representing a behavioral constraint element (either `ElementBehavior`, `InteractionBehavior`, `Scenario`, or `Trajectory`) that `constrains` a particular `StateVariable` through its constraint specification(s). See the `constrains` relation definition for an example of the constraint of a `StateVariable`.	Generalizes `ElementBehavior`, `InteractionBehavior`, `Scenario`, and `Trajectory`. Each of the 4 specializations of `StateVariableConstraint` `constrain` `StateVariables`, but the semantic for each constraint is slightly different in each case: 1) `ElementBehavior` - constraints that pertain to a single `BehavingElement` that are physically true for all time in all execution cases; 2) `InteractionBehavior` - constraints that relate at least two different `BehavingElements` that are physically true for all time in all execution cases; 3) `Trajectory` - constraints that give the (time-) complete specification of `StateVariable`'s `States` for a given execution of the system; the set of constraints that define a `Trajectory` may result from the resolution/simplification of the set of constraints from `ElementBehaviors` and `InteractionBehaviors` within a particular system with the set of constraints from a `Scenario` under which that same system is executed. 4) `Scenario` - (*still under consideration*: constraints on `StateVariable` without any guarantee that these will actually happen in any particular execution case).

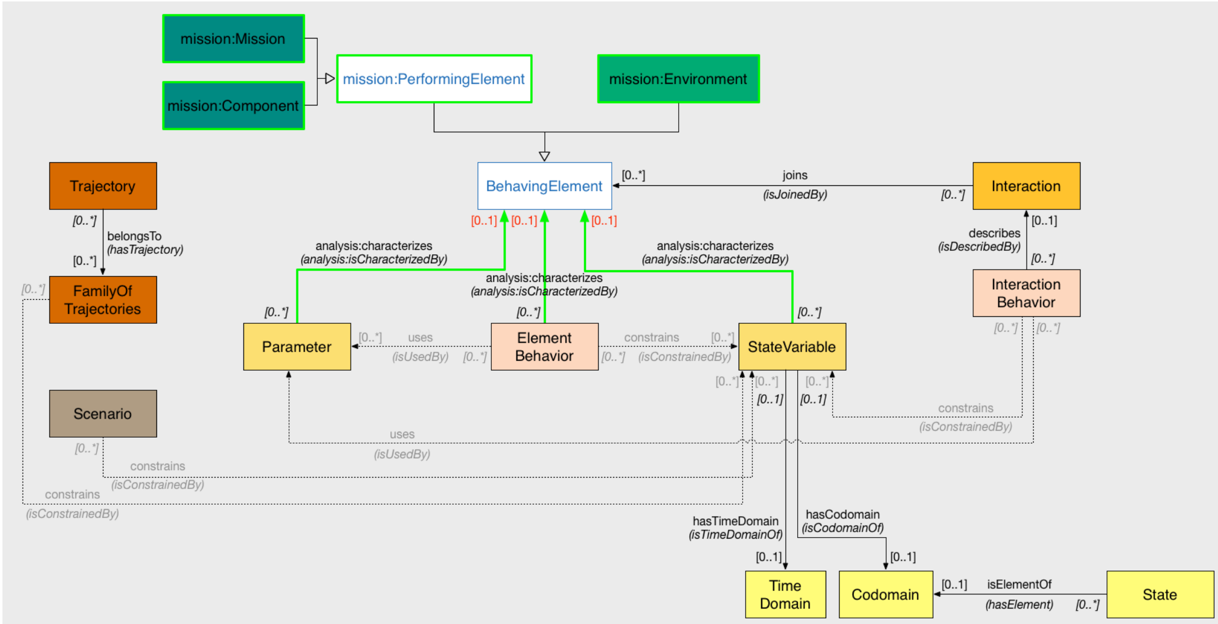
NOTE: this table reads by columns

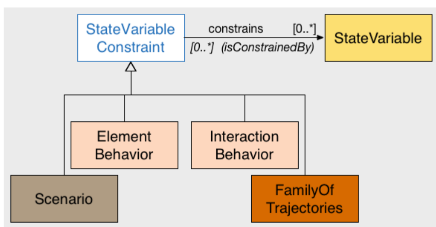
	constrains (inverse)	**isConstrainedBy** (inverse)	**uses** (inverse)	**isUsedBy** (inverse)
Subject	**StateVariableConstraint**	**StateVariable**	**UsingElement**	**Parameter**
Verb	**constrains**	**isConstrainedBy**	**uses**	**isUsedBy**

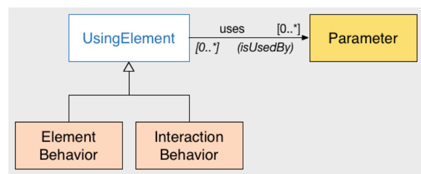
Multiplicity	[0..*]	[0..*]	[0..*]	[0..*]
Object	**StateVariable**	**StateVariableConstraint**	**Parameter**	**UsingElement**
Multiplicity Rationale	A `StateVariableConstraint` may constrain many `StateVariables` to convey the dynamic effects within the system.	Since `StateVariableConstraints` can take many different forms (which have different semantics), there is no limitation on the number of `StateVariableConstraints` that can constrain any particular `StateVariable`. For example: a voltage `StateVariable` in a resistor may be constrained by both an `ElementBehavior` describing the internal dynamics of the resistor (via Ohm's Law), and an `InteractionBehavior` describing the dynamics of the external interactions of that resistor with other elements in the same circuit (via Kirchoff's Voltage Law).	A `UsingElement` may `use` many `Parameters` to convey the dynamic effects within the system.	Since the inverse `uses` relationship is essentially just a value reference for a `Parameter`, there is no limitation on the number of `UsingElements` that can reference its value.
Description	Levying a constraint on a `StateVariable` will result in limiting the generated `Trajectory`(ies)'s range(s) to a subset of the original `Codomain`.	Inverse of `constrains`; indicates that there is a `StateVariableConstraint` applying a limitation to a given `StateVariable`.	Using a `Parameter` (in the specific context of this ontology) is equivalent to referencing its value in a behavioral constraint without having the constraint affect its value. Therefore, a `Parameter` is causally independent within the constraint.	Inverse of `uses`; indicates that there is a `UsingElement` that references the value of the `Parameter`.

Complete ontology diagram

Below is the diagram showing all concepts and relations defined within the Behavior Ontology:







Conceptual Validation/Well-Formedness Assertions

Related Segment	Assertion
Segment 2	`a:characterizes` with `StateVariable` as domain must have a class that specializes `BehavingElement` as range
Segment 2	`a:characterizes` with `Parameter` as domain must have a class that specializes `BehavingElement` as range
Segment 2	A `StateVariable` must `a:characterize` exactly one `BehavingElement`
Segment 2	A `Parameter` must `a:characterize` exactly one `BehavingElement`
Segment 2	`StateVariable` must have exactly one `TimeDomain`
Segment 2	`StateVariable` must have exactly one `Codomain`
Segment 2	A `State` must be an element of exactly one `Codomain`
Segment 3	An `ElementBehavior` must `a:characterize` exactly one `BehavingElement`
Segment 3	An `ElementBehavior` must only `constrain` `StateVariable`(s) that `a:characterizes` the same `BehavingElement` as the `ElementBehavior`
Segment 3	An `ElementBehavior` must only `use` `Parameter`(s) that `a:characterizes` the same `BehavingElement` as the `ElementBehavior`

Segment 4	An `Interaction` must `join` at least two different `BehavingElements`
Segment 4	An `InteractionBehavior` must only `constrain` `StateVariable`(s) that `a:characterizes` a `BehavingElement` that `isJoinedBy` the `Interaction` that the `InteractionBehavior` `describes`
Segment 4	An `InteractionBehavior` must only `use` `Parameter` (s) that `a:characterizes` a `BehavingElement` that `isJoinedBy` the `Interaction` that the `InteractionBehavior` `describes`
Segment 4	An `InteractionBehavior` must `describe` exactly one `Interaction`

Validation rules for Segment 5 will not be elaborated until the Scenario pattern is fully formulated.

Conceptual Examples

Two conceptual examples are provided here to illustrate the concepts introduced in this pattern and their usage: a simple flashlight example and a more advanced spacecraft power and data example.

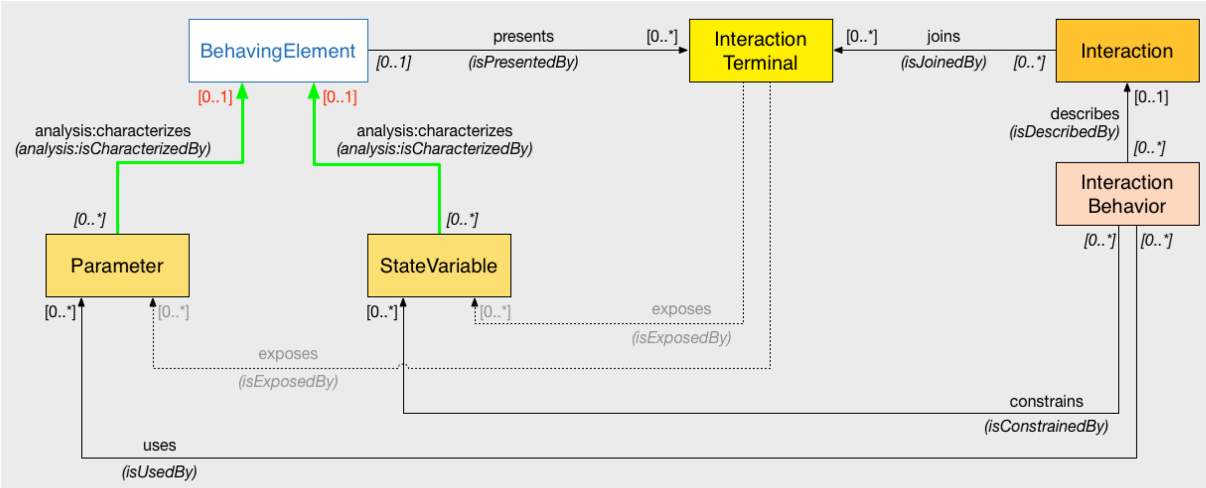
Discussion about InteractionTerminal

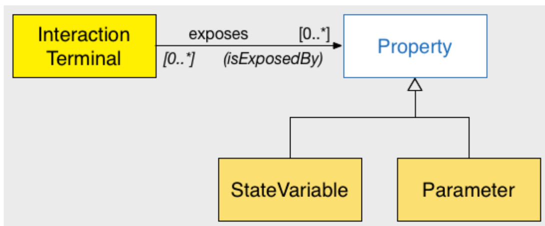
Page Top

An additional concept that was discussed during the elaboration of the behavior ontology is the concept of *InteractionTerminal*. An `InteractionTerminal` is defined as a construct that acts as a filter for selecting the `StateVariables` and `Parameters` of one `BehavingElement` involved in an interaction with other `BehavingElements`. Their use would be similar to declaring a variable public or private in a programming language. They would also be a potential hook to reconcile structural interface definition and behavior specification. The added complexity of introducing `InteractionTerminal` compared to their potential, but unproven, usage benefits, led us to propose them as ***optional*** to the modeler for now (both approaches, with or without `InteractionTerminal` are seen as valid semantics). The recommended approach does not make use of them (see complete ontology above), but the modeler can experiment with them if s/he wishes to do so.

They would fit between `BehavingElements` and `Interaction` as shown in the figure below. The joins relationship would point to the `InteractionTerminal` instead of the `BehavingElement`. The `InteractionTerminal` and the newly introduced relationships are described in tables below.

Additional validation rules are proposed below as well.





Concrete Concept Descriptions:

Page Top | Discussion Top

NOTE: this table reads by columns

Concept	Description	Notes
InteractionTerminal	A construct that acts as a filter for selecting the `StateVariables` and `Parameters` of one `BehavingElement` involved in an interaction with other `BehavingElements`, for the purposes of being `constrained` and `used`, respectively, by the `InteractionBehaviors` describing the behavioral nature of the interaction. `InteractionTerminals` do not represent anything other than a "window" into the `BehavingElement` so that `InteractionBehaviors` describing interaction constraints have access to them. Currently, there is no explicit semantic connection with any physically or logically defined `m:Interfaces`.	The manner of associating an `InteractionTerminal` with any particular physically or logically defined `m:Interface` is still under consideration. This type of connection could answer questions related to the consistency of the behavioral model with the physical and logical models of the system (e.g. whether the architecture/topolgy of the physical/logical system can actually support the specified behavior). An `InteractionTerminal` can only expose `Property`(ies) of the `BehavingElement` presenting that `InteractionTerminal`. This chain rule is currently not explicitly enforced by the ontology, and its validation is left to the modeler.

Interaction	A link that joins InteractionTerminals, representing the context of the interaction among the associated BehavingElements. An Interaction does not represent any physical or logical entity of the system - it is merely a connection point that defines a context space where the exposed StateVariables and Parameters of the joined BehavingElement InteractionTerminals are allowed to affect one another through the definitions of InteractionBehaviors. Interactions represent N-ary interactions among BehavingElements (contrary to m:Junction that are strictly binary). This allows the modeler to choose potentially more efficient representations where the interaction expressions cannot be decomposed into binary relationships between BehavingElements (see the flashlight example for such a case as Kirchhoff's voltage law in a loop).	The defined usage for an Interaction is that it must join at least two different InteractionTerminals and at least two of those InteractionTerminals must be presented by different BehavingElements. Otherwise, the associated InteractionBehaviors could just be represented as ElementBehaviors instead. Another perspective on this usage: it is allowed for an Interaction to join InteractionTerminals of the same BehavingElement, as long as it also joins at least one other InteractionTerminal from a different BehavingElement. It allows for the specification that an interaction takes place without specifying exactly what the interaction among BehavingElements: this allows for example for multiple or competing descriptions of the interaction (e.g., different levels of fidelity) or for collaboration among different domain engineers.
InteractionBehavior	A specification describing how different BehavingElements interact with one another (by expressing the effects of Property(ies) (see below) of a BehavingElement on other Property(ies) of the BehavingElements it interacts with). InteractionBehaviors are represented through constraints on StateVariables. Parameters can also appear in these constraints. InteractionBehaviors are external in nature: they dependent on a particular deployment of BehavingElements in relation to each other. It is implicit that the definition of any interaction is only valid within a scope, however large it may be (as for ElementBehavior). Currently, the ontology does not support the specification of the scope (see Open Questions).	A difference between ElementBehavior and InteractionBehavior is that ElementBehavior constrains/uses StateVariables/Parameters of only one BehavingElement, while an InteractionBehavior constrains/uses StateVariables/Parameters of at least two different BehavingElements. It is implied that InteractionBehavior constrains/uses StateVariables/Parameters that have been exposed by InteractionTerminals that are joined by the Interaction that the InteractionBehavior describes. This chain rule is currently not explicitly enforced by the ontology, and its validation is left to the modeler.
Property	Abstract concept representing some aspect of a BehavingElement. A Property can either be static (Parameter) or dynamic (StateVariable) in time. Property(ies) can be exposed through InteractionTerminals to be used or constrained by InteractionBehaviors, which represent behaviors of interactions between system elements.	Generalizes StateVariable and Parameter.

InteractionTerminal Relation Descriptions:

Page Top | Discussion Top

NOTE: this table reads by columns

	presents (inverse)	isPresentedBy (inverse)	exposes (inverse)	isExposedBy (inverse)
Subject	**BehavingElement**	**InteractionTerminal**	**InteractionTerminal**	**Property**
Verb	**presents**	**isPresentedBy**	**exposes**	**isExposedBy**
Multiplicity	**[0..*]**	**[0..1]**	**[0..*]**	**[0..*]**
Object	**InteractionTerminal**	**BehavingElement**	**Property**	**InteractionTerminal**
Multiplicity Rationale	A `BehavingElement` may `present` many `InteractionTerminals` to filter and organize access to `Property`(ies)	The purpose of an `InteractionTerminal` is to provide access to `Property`(ies) of a single `BehavingElement`.	An interaction can affect several `Property`(ies) of a single `BehavingElement`. The modeler may choose to `expose` them into a single `InteractionTerminal`.	The modeler can choose to model separate interactions that affect the same `Property`. For example, the temperature `StateVariable` of a component could be affected by thermal and electrical interactions. In that case, the temperature can be exposed by both the "thermal `InteractionTerminal`" and the "electrical `InteractionTerminal`".
Description	A `BehavingElement` `presents` an `InteractionTerminal` if its `Property`(ies) are affected by `Property`(ies) of other `BehavingElements`.	Inverse of `presents`; indicates that there is a `BehavingElement` that `presents` this `InteractionTerminal`.	It indicates that the exposed `Property`(ies) can now be affected by influences outside of the `BehavingElement`.	Inverse of `exposes`; indicates that there is a `InteractionTerminal` that `exposes` this `Property`.
Notes	none	none	The `InteractionTerminal` can `expose` only the `Property`(ies) of the `BehavingElement` presenting that `InteractionTerminal`.	none

Interaction Relation Descriptions:

Page Top | Discussion Top

NOTE: this table reads by columns

	joins (inverse)	isJoinedBy (inverse)	describes (inverse)	isDescribedBy (inverse)
Subject	**Interaction**	**InteractionTerminal**	**InteractionBehavior**	**Interaction**
Verb	**joins**	**isJoinedBy**	**describes**	**isDescribedBy**
Multiplicity	**[0..*]**	**[0..*]**	**[0..1]**	**[0..*]**
Object	**InteractionTerminal**	**Interaction**	**Interaction**	**InteractionBehavior**

Multiplicity Rationale	This is a direct consequence of the N-ary nature of the `Interaction`.	The `StateVariables` of a given `BehavingElement` may be affected by many external interactions that are modeled through different `Interactions`. Continuing from the example in the `isExposedBy` multiplicity rationale - the temperature of a spacecraft component may be affected by two separate thermal interactions: 1) incoming solar radiation, and 2) physical conduction to spacecraft structure. In this case, the modeler may choose to have one thermal `InteractionTerminal` `exposing` the temperature `StateVariable` that `isJoinedBy` `Interactions` representing both of these thermal effects.	`InteractionBehaviors` use `Parameters` and `constrain` `StateVariables` in the context of a single specific `Interaction` they `describe`.	A `Interaction` can be `describedBy` as many `InteractionBehaviors` as necessary to capture the dynamics of the interaction.
Description	Relation indicating that the `Property`(ies) `exposedBy` the `joined` `InteractionTerminals` are now in scope of the interaction represented by the `joining` `Interaction`.	Inverse of `joins`; indicates that there is an `Interaction` that `joins` the `InteractionTerminal`.	The constraints defined in the `InteractionBehavior` apply within the context of the `described` `Interaction`.	Inverse of `describes`; indicates that there is an `InteractionBehavior` that `describes` a given `Interaction`.
Notes	Also indicates that the `exposed` `Property`(ies) can now be referenced by the `InteractionBehaviors` `describing the Interaction`.	All of the associated constraints of all related interactions are levied on the `Property`(ies) exposed by the `InteractionTerminals`.	none	none

Additional validation rules

Assertion
An `InteractionTerminal` can only `expose` `Property`(ies) that `a:characterizes` the `BehavingElement` that `presents` the `InteractionTerminal`
An `InteractionTerminal` must `expose` at least one `Property`
An `Interaction` must `join` at least two different `InteractionTerminals`
At least two of the `InteractionTerminals` `joined` by the same `Interaction` must be `presented` by different `BehavingElements`

(if only using `InteractionTerminal`) An `InteractionBehavior` must only `constrain` `StateVariable`(s) that `a:characterizes` a `BehavingElement` which `presents` an `InteractionTerminal` that `isJoinedBy` the `Interaction` that the `InteractionBehavior` `describes`
(if only using `InteractionTerminal`) An `InteractionBehavior` must only `use` `Parameter` (s) that `a:characterizes` a `BehavingElement` which `presents` an `InteractionTerminal` that `isJoinedBy` the `Interaction` that the `InteractionBehavior` `describes`

Behavior Page Navigation - continue reading:

(0) Community Page

(1) Main Behavior Pattern Page

(2) Conceptual Behavior Ontology v1.1 (3) Behavior Pattern: Conceptual Examples v1.1

(4) SysML-Embeddable Ontolgy & Implementation v1.1 (5) Behavior Pattern: SysML Example v1.1