Ontology and Modeling Patterns for State-Based Behavior Representation

Jean-Francois Castet[1], Matthew L. Rozek[2], Michel D. Ingham[3], Nicolas F. Rouquette[4], Seung H. Chung[5], Aleksandr A. Kerzhner[6], Kenneth M. Donahue[7], J. Steven Jenkins[8], David A. Wagner[9], Daniel L. Dvorak[10], Robert Karban[11]
Jet Propulsion Laboratory – California Institute of Technology, Pasadena, CA

This paper provides an approach to capture state-based behavior of elements, that is, the specification of their state evolution in time, and the interactions amongst them. Elements can be components (e.g., sensors, actuators) or environments, and are characterized by state variables that vary with time. The behaviors of these elements, as well as interactions among them are represented through constraints on state variables. This paper discusses the concepts and relationships introduced in this behavior ontology, and the modeling patterns associated with it. Two example cases are provided to illustrate their usage, as well as to demonstrate the flexibility and scalability of the behavior ontology: a simple flashlight electrical model and a more complex spacecraft model involving instruments, power and data behaviors. Finally, an implementation in a SysML profile is provided.

I. Introduction

OVER the past few years, the Jet Propulsion Laboratory (JPL) has been developing and implementing a model-driven systems engineering process, known generally in industry and academia as Model-Based Systems Engineering (MBSE). To support this process, a number of tools and infrastructure elements are currently in development under the JPL's Integrated Model-Centric Engineering (IMCE) initiative[1]. One of these infrastructure elements is a series of ontologies[2] that support standard modeling for JPL space missions and other projects. As a subset of these ontologies, we present in this paper a novel ontology that enables a consistent means to formally describe and specify state-based behavior of systems, as well as associated modeling patterns that capture its usage.
In our ontological representation, system state is captured through the use of state variables, i.e. key time-varying properties that represent features of the system necessary to describe its behavior and perform various engineering analyses. For example, component position and velocity are state variables usually found in the mechanical domain, and component voltage and current are state variables usually found in the electrical domain. *State-based behavior* of a system is then asserted to be the *set of constraints on these state variables*, constraints that define how the dynamic state of the system can evolve in time (the term "behavior" refers to "state-based behavior" in the remainder of this paper).

The development of the behavior ontology was motivated by the difficulty of maintaining a consistent behavior specification within the existing systems engineering process. This made it problematic for engineers to systematically evaluate interactions between behaviors in various elements of the system, and how behaviors of individual elements affect the system as a whole. In addition, it was possible to introduce discrepancies between specified behavior (in the design phase) and its implementation due to a lack of common vocabulary between the engineers involved in each phase. These discrepancies could result in unexpected or even hazardous situations

[1] Systems Engineer, Autonomy and Fault Protection Group, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 321-560.
[2] Systems Engineer, Flight System Systems Engineering Group, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 301-490.
[3] Technical Group Supervisor, System Architectures and Behaviors Group, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 301-490, AIAA Associate Fellow.
[4] Principal Computer Scientist, System Architectures and Behaviors Group, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 301-490.
[5] Technical Group Supervisor, Modeling and Verification Group, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 301-270, and AIAA Senior Member.
[6] Software Systems Engineer, System Architectures and Behaviors Group, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 321-560.
[7] Software Systems Engineer, System Architectures and Behaviors Group, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 321-560.
[8] Principal Engineer, Engineering Development Office, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 301-237.
[9] System Architect, System Architectures and Behaviors Group, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 301-490, AIAA Member.
[10] Principal Engineer, Engineering Development Office, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 301-237.
[11] Senior Systems Architect, System Architectures and Behaviors Group, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 301-490.

occurring during operation of the system. Using a common behavioral semantic allows for improving communication in a collaborative environment – hence reducing the associated risks mentioned above –, for performing efficient design trades and optimization, and for performing formal querying, verification and validation of the behavioral model using powerful reasoning engines (e.g., SPARQL[3]). In addition, it increases productivity by allowing for re-use of model elements and behavior patterns from project to project. Finally, a common vocabulary allows for the exchange of behavior information between different tools or languages.

As a basis for this work, technical literature related to behavior modeling was investigated. This included reviews of function modeling[4], other work on behavioral descriptions of systems[5,6], existing modeling language paradigms[7], and previous work conducted at JPL, such as behavior engineering[8] approaches, the State-Analysis methodology[9],[10] and the Timeline representation (with focus on operations and scenario modeling)[11]. While this body of work has strongly influenced the scope, definition, and elaboration of this behavior ontology, it did not provide us with modeling languages that have either the breadth or the depth to capture the semantics of behavior to a desired level of specification.

For example, modeling languages such as the Object Management Group (OMG) UML[12]/SysML[13] provide semantics for capturing only a subset of behaviors, and do not integrate them in their base form without further extension[5]. Another standard modeling language, Modelica[14] (with large offering of open-source and commercial solvers), captures flow-based multi-domain physics models, and supports differential algebraic equations (i.e. ordinary differential equations with algebraic constraints) with both continuous and discrete variables. One of the strengths of Modelica is that the models can be defined using composable and reusable building blocks. On the other hand, there are a number of domains that Modelica is not designed to model, for example logical existential or universal assertions. Also, the composability in Modelica is focused on creating binary connections between pre-defined interfaces using Kirchhoff semantics[15]. Although this is well suited for many physical systems, it can also lead to complex models. The same effect applies to the behavioral approach proposed by Willems[6]. This is illustrated in section III.B, and the behavior pattern proposed in this work aims to offer flexible behavioral representations to the modeler. Previous efforts have attempted to integrate SysML and Modelica[16]; SysML would contain systems engineering knowledge such as requirements and test plans and Modelica would contain behavioral equations. These previous approaches have developed a Modelica concrete syntax for SysML, hence restricting the behavior equations to those captured by Modelica. In the approach in this paper, a goal is to provide a flexible representation that can then be transformed into any number of languages for analysis (for example Mathematica[17], Maple[18], Modelica, or JPL tools such as APGEN[19]), and save users from learning multiple analysis languages.

Finally, State Analysis[9,10] provides a methodology for modeling behavior of a system and using these models to specify the control design and operations for this system, but does not impose any particular modeling formalism. As such, our work developing the behavior ontology is complementary to the State Analysis methodology, and can be used to capture the results of applying this methodology.

This paper describes several features of the behavior ontology, such as a carefully designed relationship between behavior specification and structural definition, allowing for flexible (and potentially multiple) characterizations to be applied to the same structural elements based on varying degrees of fidelity (analysis usage or customer presentation).

A note regarding the scope of this paper and its implementation: while there is a difference between the actual behavior of a system and modeled behavior (which is an engineering approximation of the actual behavior), only modeled behavior will be discussed for the purpose of this paper.

The remainder of this paper is organized as follows: Section II describes the key concepts and relationships of the proposed behavior ontology. Section III presents two example applications to illustrate the behavioral concepts and demonstrate the flexibility and scalability of the ontology. Finally, Section IV discusses a SysML implementation of the behavior ontology.

II. Behavior ontology

A. Conventions

Before describing the ontology, a brief discussion about the conventions used in the figures of this paper to present concepts and relationships is provided. Figure 1 is a notional figure in the style of subsequent figures of the paper and is used to introduce conventions.

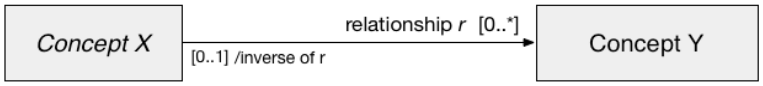

Figure 1. Diagram conventions for concepts, relationships and multiplicities.

Concepts are represented using rectangles; abstract concepts are written in italics. For example in Figure 1, "Concept X" and "Concept Y" are concepts. In addition, "Concept X" is an abstract concept, i.e., concrete concepts will implement it.

Relationships are represented using directed arrows, pointing from the source to the target of the relationship. The relationship name is written next to the target of the relationship. For example in Figure 1, "*r*" is the relationship from Concept X and Concept Y, X being the source of *r* and Y being the target of *r*. We also provide the inverse relationship: it is written close to the source of *r*, and preceded by "/" to indicate its derived nature (it is not independent from *r*). For example in Figure 1, the inverse relationship of *r* relates Concept Y to Concept X.

Finally, we indicate the multiplicities associated with relationships between brackets: for example in Figure 1, a Concept X instance (i.e., a realization of Concept X) can be related through the relationship *r* to zero to many Concept Y instances. Conversely, a Concept Y instance can be related to at most one Concept X instance.

B. BehavingElement

The behavior ontology is centered around the concept of a *BehavingElement*, that is an element that exhibits a behavior (inherent physics or intended specification). BehavingElements can be engineered components (physical or logical entities of a system designed through an engineering process), environments (e.g., the radiation environment around the Jovian moon Europa) or any element that may exhibit behavior, such as a team, a person, etc. As such, BehavingElement is an abstract concept in this ontology, and is implemented by the classes of objects described (examples of this implementation are shown in section III.A).

The ontology can be described in relation to BehavingElement through three distinct but related segments: "Properties", "Internal Behavior" (Intra-BehavingElement), and "External Behavior" (Inter-BehavingElements). We also describe in general terms its relation with "Scenarios" and "Trajectories". Examples of application of these concepts are described in section III.

C. Properties of BehavingElements

Two types of properties of BehavingElements are defined as part of the behavior ontology: *StateVariable* and *Parameter*.

A *StateVariable* is a variable that represents a particular quantity[*] or nominal property[†] (as per ISO-80000-1[20]) of a BehavingElement that changes with time. Two important concepts are introduced to formally define a StateVariable: a *TimeDomain* and a *Codomain*. A TimeDomain represents the set of possible times encompassed by the StateVariable; a Codomain represents the set of possible *States*, i.e., values of the quantity represented by the StateVariable. Using set theory representation, a StateVariable is then defined in Eq. (1) as the Cartesian product of its TimeDomain *X* and its Codomain *Y*:

$$X \times Y = \left\{ (x, y) \middle| x \in X \wedge y \in Y \right\} \tag{1}$$

TimeDomains and Codomains can both be either discrete or continuous. For example, the discrete StateVariable representing the power status of a television (ON, OFF or STANDBY) for five times in a 20-minute time interval is shown in Figure 2. The StateVariable then represents all possible combinations (pairs) of times and States from the defined TimeDomain and Codomain.

In the case of a continuous StateVariable such as temperature shown in Figure 3, the TimeDomain (continuous time interval of 350 seconds) and the Codomain (temperatures in a continuous interval between 10 and 20°C) cannot be represented graphically as easily. The StateVariable is then the infinite set of possible pairs (time, State).

[*] "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.

[†] "Property of a phenomenon, body, or substance, where the property has no magnitude." E.g., color, switch position

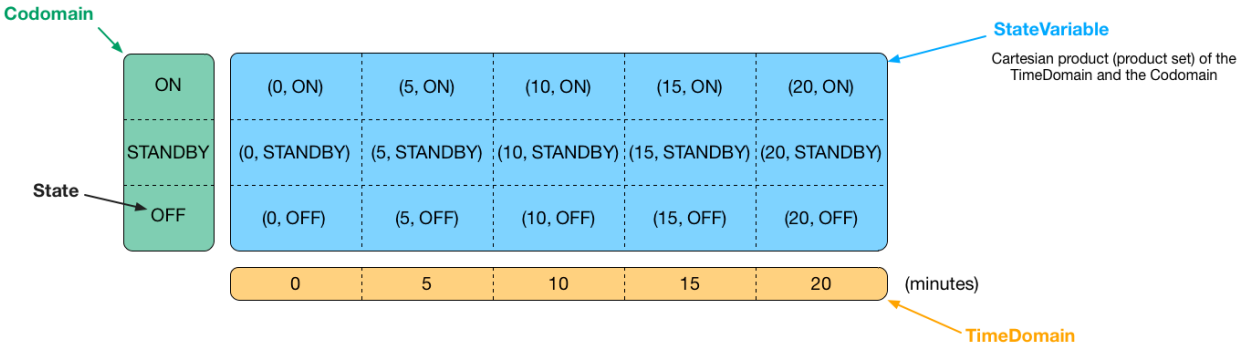

Figure 2. Example of a discrete StateVariable: power status of a television.

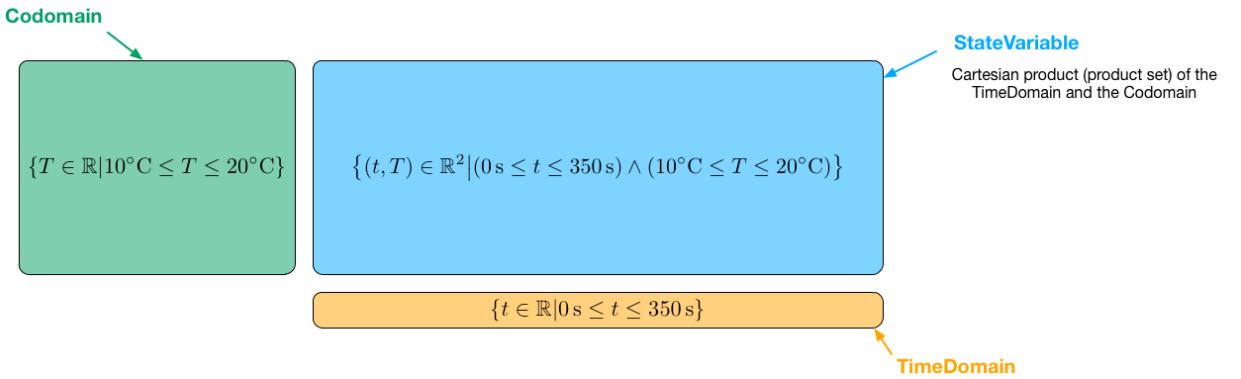

Figure 3. Example of a continuous StateVariable: temperature.

A *Parameter* is a quantity or nominal property[20] of a BehavingElement that does not dynamically change in time. Hence it does not have a TimeDomain (nor a Codomain). An example of a Parameter could be the resistance of a resistor that obeys Ohm's law.

Figure 4 shows the concepts and relationships related to properties. Parameters and StateVariables "characterize" BehavingElements. The "analysis" namespace (or prefix) indicates that this relationship has been defined in another IMCE ontology, named Analysis[2]. This relationship has been defined to indicate that the source of the relationship describes the target of the relationship. The concept of characterization is an important aspect of the IMCE ontological approach that allows engineers to "describe components in ways that are not intrinsic to the definition of components but are necessary to work within their problem space."[21] Characterizations allow clear separation of the structural definition of these components from their characteristics of interest for the engineer. The behavior ontology takes advantage of this mechanism, as properties (Parameters and StateVariables) of a BehavingElement can vary across engineering discipline domains, or even across behavioral model levels of detail. One set of StateVariables might be of interest to one engineer, but not to another. This mechanism also handles different and/or conflicting StateVariable definitions caused by differences in fidelity of modeled behavior, as properties are not part of the intrinsic definition of the component.

Finally, a word on the usage of StateVariable versus Parameter. 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 from exclusively using StateVariables, but that choice could be for example at the expense of complexity or computation, as it takes more memory or more computation to store or resolve StateVariables (time history to resolve) compared to Parameters (single value).

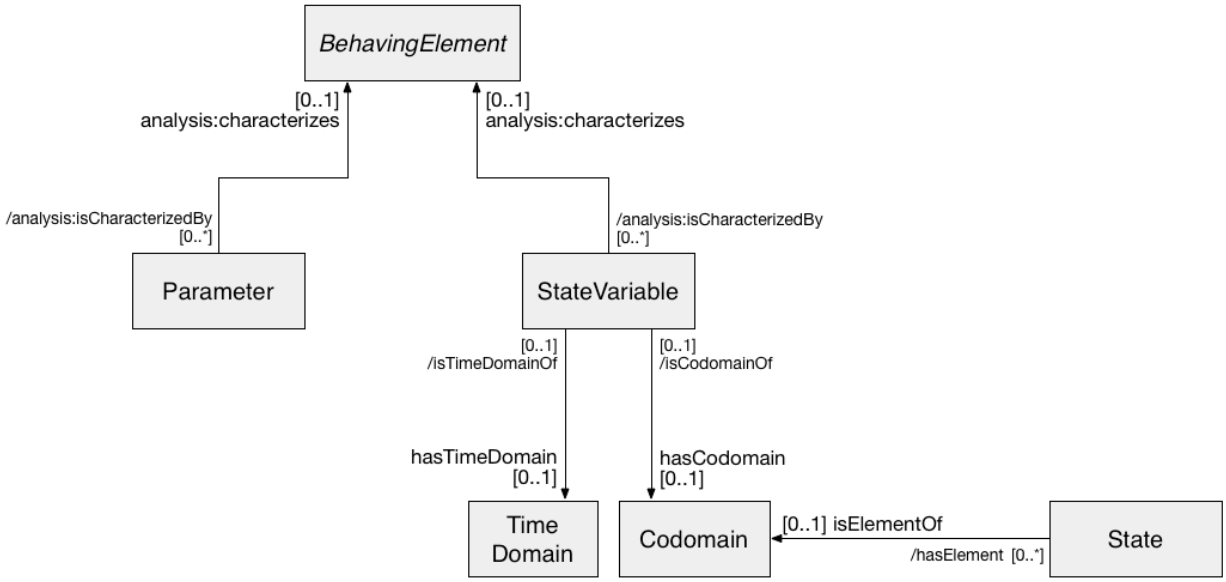

Figure 4. Properties concepts and relationships. *Conventions on how to read this figure are provided in section II.A.*

D. Internal Behavior

Internal behavior is described by the concept of *ElementBehavior*, i.e., a specification of how the dynamic state of a BehavingElement is allowed to evolve in time. An ElementBehavior is conceptually intrinsic in nature. However, this does not preclude ElementBehavior specifications from using external information (signals, commands, etc.), or providing some to external elements. For example, in the case of a variable Ohmic resistor, both Ohm's law (voltage across the resistor is equal to the resistance times the current through it) and the calibration relation between the knob position and the resistance value would be the ElementBehavior. This specification accepts the knob position as a driver of the behavior, but both Ohm's law and the calibration relationship are independent of a specific value of the knob position.

An ElementBehavior is represented through constraints on StateVariables (with Parameters potentially appearing in such constraints), such as mathematical expressions, state-machine formalisms, temporal-logic constraints, and predicate logic statements. In this paper, examples of mathematical expressions and state machines are given, as they were the most logical and compact representation of the ElementBehavior for these application cases. Note that the definition of any behavior is only valid within a scope. For example, Ohm's law would break if a massive current were to be applied to the resistor (the resistor would blow instead). However, this paper does not generally discuss the scope specification.

To summarize, an ElementBehavior *characterizes* a BehavingElement, *constrains* StateVariables (of that BehavingElement) and *uses* Parameters. Figure 5 shows these concepts and their relationships.

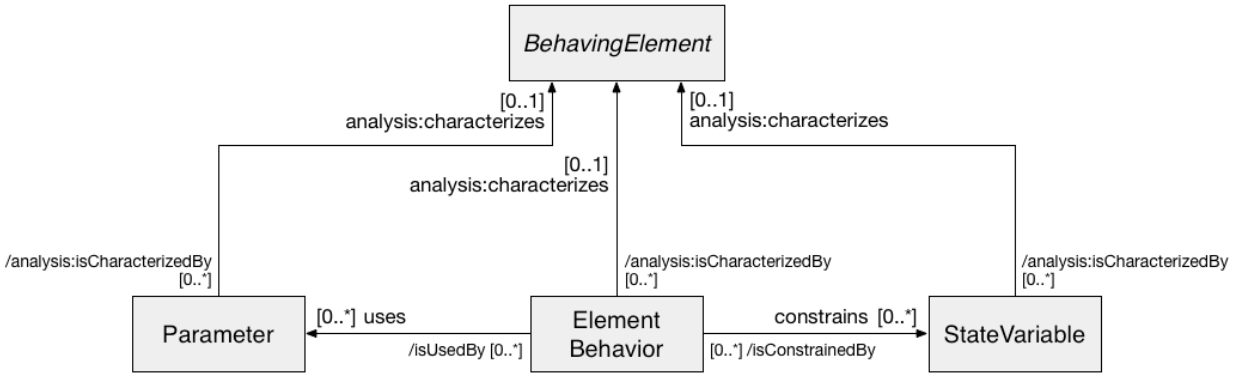

Figure 5. ElementBehavior concepts and relationships.

E. External Behavior

This section presents the concepts necessary for describing behavioral interactions among BehavingElements: *Interaction* and *InteractionBehavior*.

An Interaction is a link that joins at least two different BehavingElements in the context of their interaction. It does not represent any physical or logical entity of the system: it is merely a connection point that defines a context space where the StateVariables and Parameters of joined BehavingElements are allowed to affect one another through the definitions of InteractionBehaviors. Interactions represent n-ary interactions among BehavingElements (contrary to the "connect" in Modelica, or the "affects" relationship in State Analysis, for example, that are strictly binary). This allows the modeler to choose potentially more efficient representations of interaction behaviors, as demonstrated in later examples.

InteractionBehavior is the counterpart of ElementBehavior in the context of an Interaction: it is the specification *describing* how different BehavingElements interact with one another, by *constraining* the StateVariables of the BehavingElements involved in that Interaction. InteractionBehaviors are external in nature: they depend on a particular deployment of BehavingElements in relation to each other. Once again, the definition of any interaction is only valid within a scope, as for ElementBehavior. A key difference between ElementBehavior and InteractionBehavior is that ElementBehavior constrains StateVariables of only one BehavingElement, while an InteractionBehavior constrains StateVariables of at least two different BehavingElements.

The concepts and relationships introduced in this section are summarized in Figure 6.

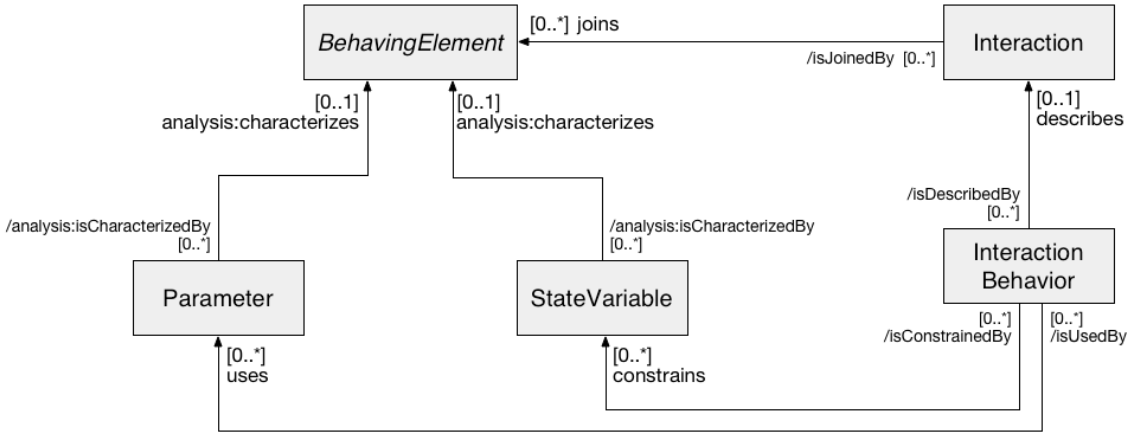

Figure 6. Interaction concepts and relationships.

F. Scenarios and Trajectories

Scenarios and *Trajectories* are the object of a current investigation that will lead to the definition of their own ontology, but are briefly discussed in this paper to describe how they relate to behavior.

A Scenario is defined as a set of constraints on StateVariables and other specifications (such as initial conditions) without any guarantee that these will actually happen in any particular execution case. Scenarios orchestrate the execution of behaviors: for example in the case of the television presented in Figure 2, a scenario can indicate at which times the television will be turned on or off. Examples of scenarios are given in Section III.

A *Trajectory* of a StateVariable results from a fully constrained combination of behaviors and scenarios: it represents the particular single path for that StateVariable (one State per time of the TimeDomain). It is formally defined as a "functional subset[‡]"[22] of a StateVariable that spans the entire TimeDomain.

The *FamilyOfTrajectories* generalizes the concept of Trajectory by combining a set of Trajectories. It is formally defined as a subset of a StateVariable that spans the entire TimeDomain. Figure 7 provides a visual representation of Trajectory and FamilyOfTrajectories.

[‡] In the behavior context, the functional subset is a set of ordered pairs of times and States in which no time values are repeated.

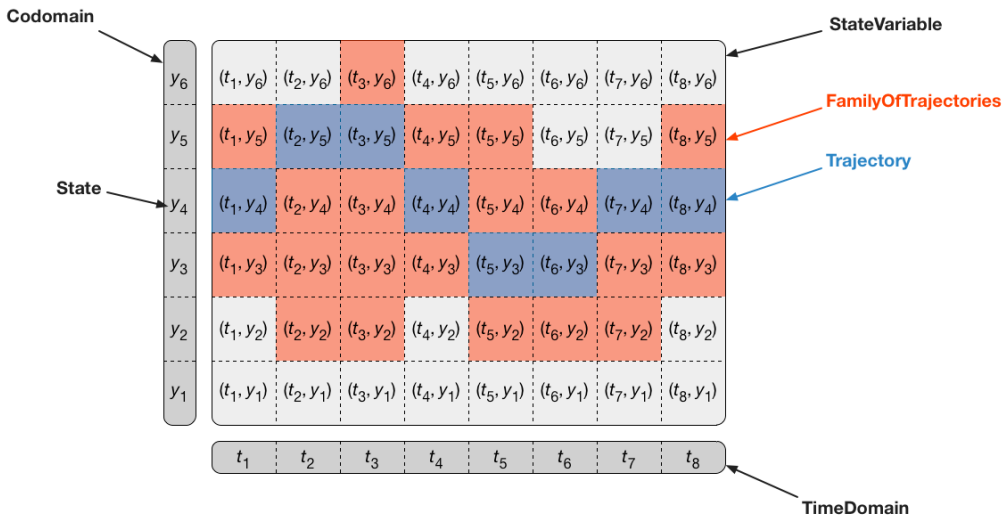

Figure 7. Example of Trajectory and FamilyOfTrajectories of a StateVariable.

G. Complete Behavior Ontology

The complete behavior ontology introduced in this paper is given in Figure 8.

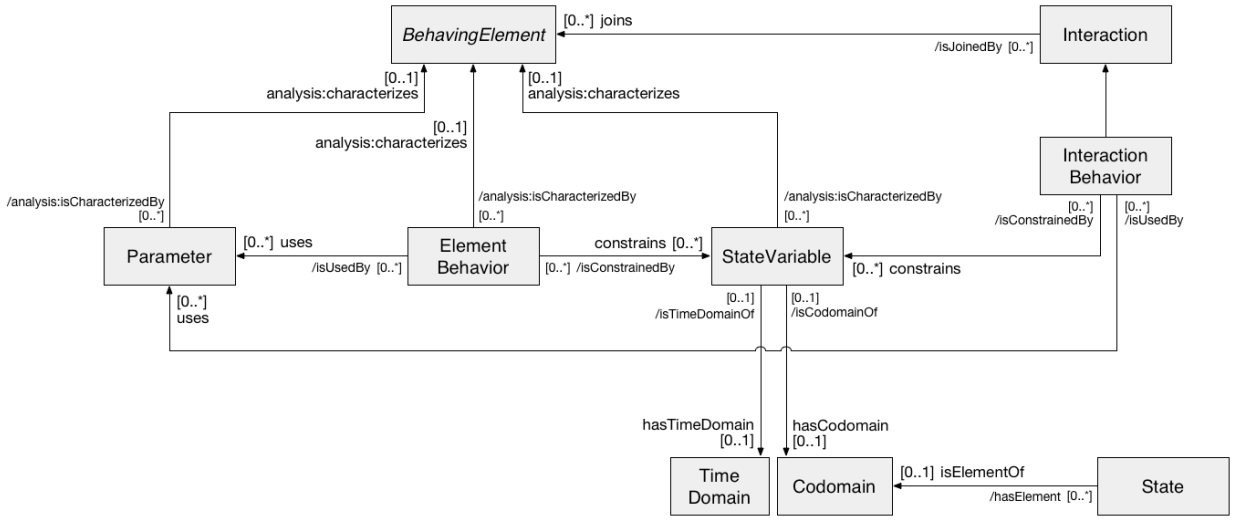

Figure 8. Complete behavior ontology.

H. Discussion about InteractionTerminals

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. Figure 9 shows how the InteractionTerminal relates to other concepts introduced in this ontology. The added complexity of introducing InteractionTerminal compared to their potential, but unproven, usage benefits, led us to propose them as optional to the modeler.

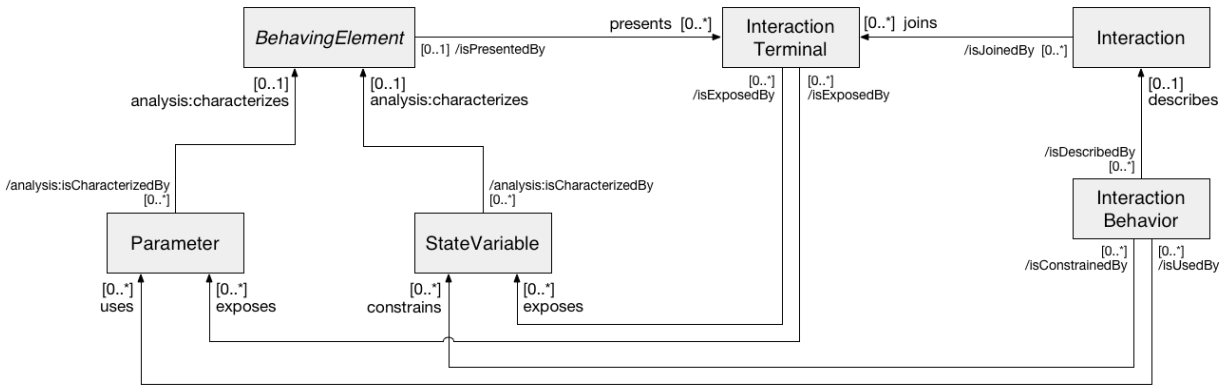

Figure 9. InteractionTerminal concepts and relationships.

III. Examples

This section discusses two application cases. The first case was selected to present a straightforward application of the behavior ontology and illustrate each of the concepts and relationships introduced. It shows the electrical behavior model of a simple flashlight design, consisting of three components connected in series, namely a battery, a switch, and a lamp (light bulb). The second case was selected to demonstrate the capability of the behavior ontology and its application patterns to handle complex component behaviors and interactions. This example presents the behavior model of two representative spacecraft instruments (a camera and a magnetometer) and two related spacecraft subsystems that are tightly coupled: the power subsystem and the data subsystem (data handling and data transmission).

A. Flashlight Example

Three BehavingElements are of interest in this example: a battery, a switch and a lamp, connected in series. As mentioned earlier, BehavingElement is an abstract class; it is realized in this example using a concept from another IMCE ontology, named mission:Component[2] (physical or logical entity/element of a system that is designed/specified through an engineering process). StateVariables of interest are the voltages and currents associated with each component, as well as the switch position and the lamp luminous flux.

Simple idealized internal behaviors for each component were built: the battery provides a constant nominal voltage across its electrical terminals; the switch can be in two States—open (with zero current through the switch) or closed (with zero voltage across the switch terminals); and the incandescent lamp is idealized as a Ohmic resistor with Joule heating that produces light due to the resulting thermal radiation. Note that the Ohmic resistance was chosen to remain constant in this simplified model, and as such, a Parameter is the logical choice for capturing it. A more detailed model could have represented it as a StateVariable varying with the temperature of the filament for example.

Figure 10 depicts the ElementBehavior of the lamp. The ElementBehavior models of the battery and the switch can be seen in Figure 12. Note that the ElementBehavior of the switch was represented using state-machine formalisms (the choice of the formalism, such as Harel statechart or UML state machine, is left to the agreement between modelers).

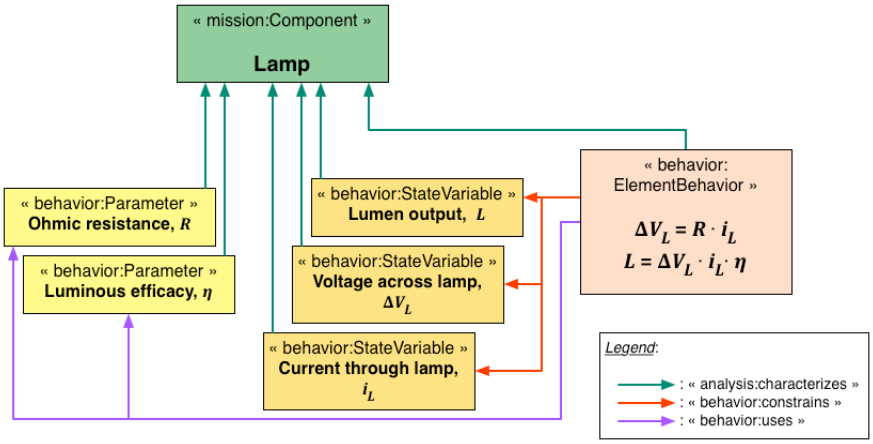

Figure 10. Example of Properties and ElementBehavior for the lamp.

The interaction behavior between the components connected in series is based on a mesh (or loop) analysis of the electrical circuit and captures the conservation of current flow through the electrical circuit loop, and Kirchhoff's voltage law (the directed sum of voltages in a closed loop is equal to zero). Figure 11 shows this interaction model.

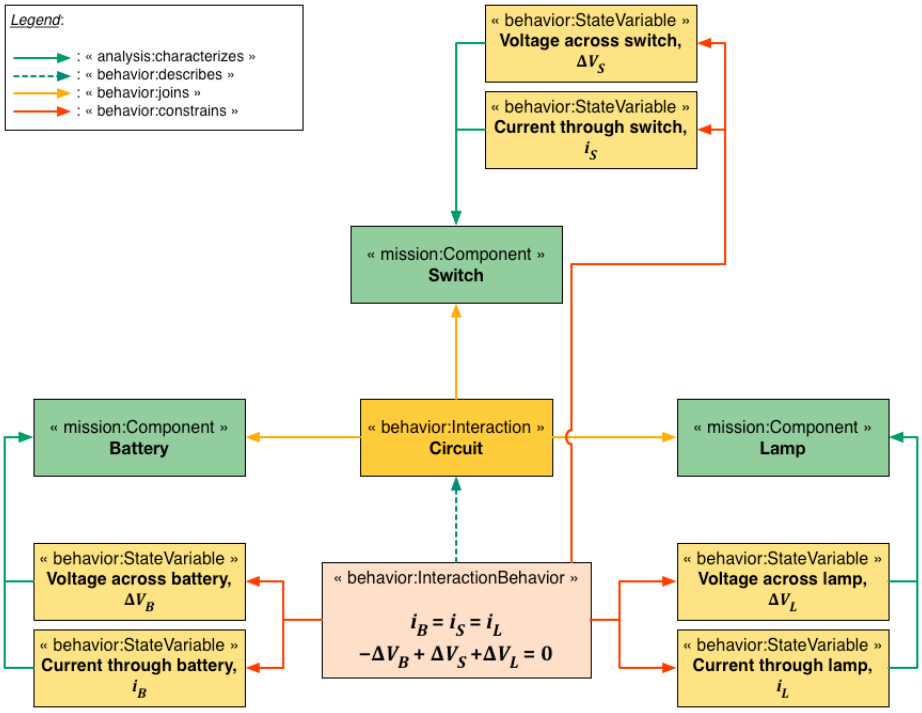

Figure 11. Example of an Interaction for the flashlight components.

The complete behavior model of the flashlight is given in Figure 12. Note that the grey boxes do not represent an ontological concept; their only purpose is for readability and to graphically highlight concepts related to each component.

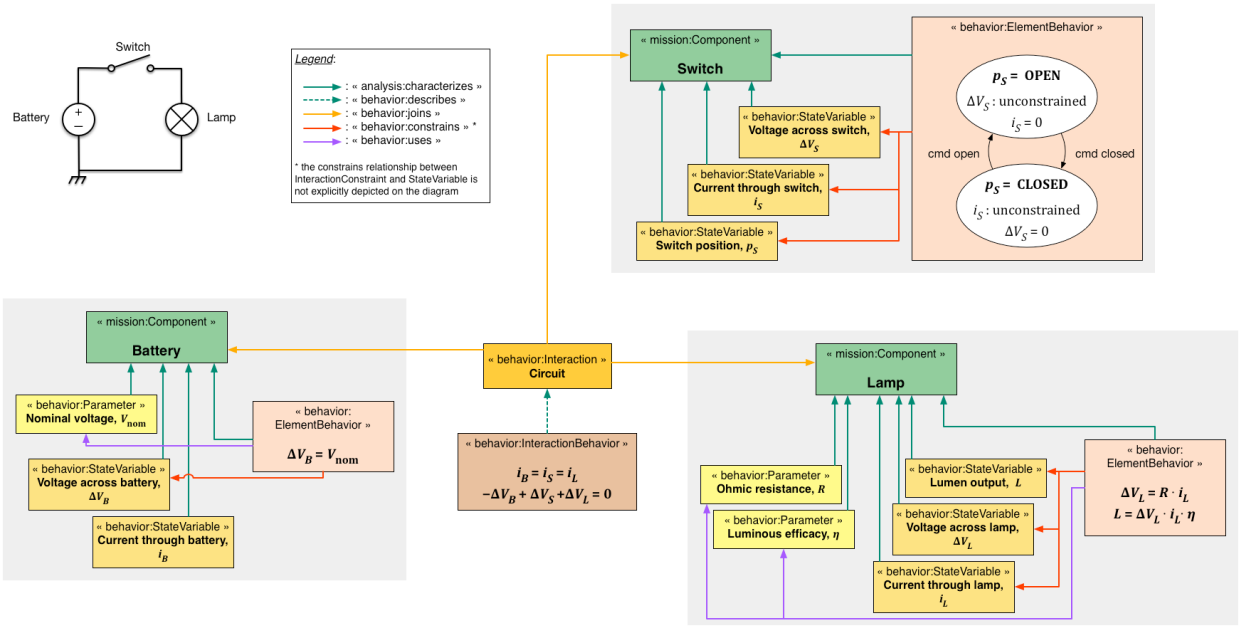

Figure 12. Complete behavioral flashlight model.

B. Alternative Flashlight Example

Note that the Interaction in the previous section was joining three BehavingElements at the same time, allowing for a compact representation of the InteractionBehavior constraints. Should Interaction be restricted to binary connections (i.e., connect exactly two BehavingElements), it would not have been possible to capture the flashlight model as in Figure 12. Instead, the flashlight model would be captured as in Figure 13.

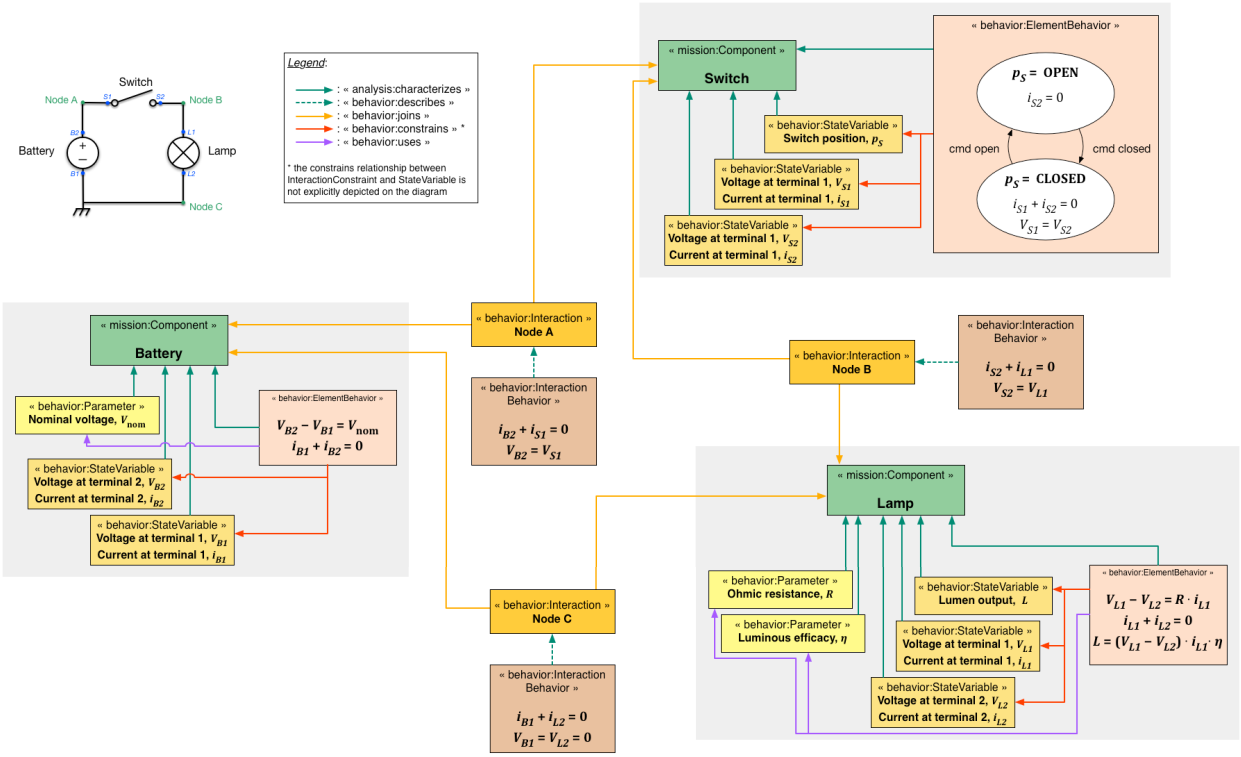

Figure 13. Alternate model for the behavioral flashlight model. *Incoming currents are positive and outgoing currents are negative.*

The representation shown in Figure 13 is less compact than the representation in Figure 12, but it is important to note that it still conforms to the behavior ontology introduced in this paper. Here, the Interactions are captured in a nodal analysis point-of-view, focusing on the physical electrical connections between components. Three Interaction blocks are now necessary to capture the series circuit: they represent the conservation of the current across the nodes and the equality of the electrical potentials at the nodes (this is similar to the "connect" function in Modelica[14]). As a consequence, the number of StateVariables necessary close to double in this model; the advantage is that this provides for modularity, such that the components and behaviors can be stored in and drawn from a library, without having to know in what circuits they will be used. ElementBehaviors were written to handle the new StateVariables, and this model can of course be reduced to the model presented in the previous section.

As demonstrated with the flashlight models, the behavior ontology introduced in this paper was designed to allow flexibility in how modelers capture behavior specification.

C. Spacecraft Instruments, Power and Data Subsystems

The example presented in this section demonstrates the capability of the behavior ontology to handle more complex behaviors and interactions. It presents the behavior model of two spacecraft instruments (a camera and a magnetometer) and two related spacecraft subsystems: the power subsystem and the data subsystem (command and data handling, and telecommunications). The model is presented in Figure 14 and described below.

In this model, seven components are considered, and shown in green in Figure 14:

- a magnetometer instrument,
- a camera instrument,
- a Multi-Mission Radioisotope Thermal Generator (MMRTG), acting as the primary electric power source,
- a battery, used when power demand exceeds the capability of the MMRTG,
- a power manager that acts as the medium between the MMRTG power source, the battery, and the power loads (instruments, C&DH and Telecom subsystems),
- a Command and Data Handling (C&DH) subsystem that manages the storage of the data generated by the two instruments,
- a Telecom subsystem that transmits the data stored by the C&DH.

In this example, the behaviors of some components capture more complex dynamics than in the flashlight example (such as for the battery model). The ElementBehaviors of the two instruments are briefly described below:

- Magnetometer: the magnetometer behavior captures four states of the magnetometer: OFF, SURVIVAL or ON at a low sampling rate, and ON at a high sampling rate (the survival state indicates that the survival heaters of the magnetometer are turned on). For each state, an estimated power load has been captured as a characterization of the magnetometer component (i.e., the magnetometer consumes 0 W when turned off, and 5 W when turned on). A similar approach for data generation has been taken: for each state, an associated generated data rate has been determined. For example, when on, the magnetometer data output rate is a function of the number of sensors, the sampling rate associated with the current sampling rate mode (low or high) and the sampling size. The transitions between states have been captured as commands, but have not been elaborated further.
- Camera: the camera behavior captures three states of the camera: OFF, SURVIVAL or TAKING PICTURES (the survival state indicates that the survival heaters of the camera are turned on; the taking pictures state indicates that data is being captured and/or transmitted out of the camera). For each state, an estimated power load has been captured (i.e., the camera consumes 0 W when turned off, and 15 W when actively taking pictures). A similar approach was taken for the data generation capture: for each state, a data output rate has been determined. In particular, a choice was made to model the behavior of the camera using a data rate based on a data volume determined by the number of pictures taken. This is particularly of interest for transitioning out of the taking pictures state: this transition is based on the time necessary to transmit the generated data volume based on the given data rate output. Other models for the camera can be envisioned.

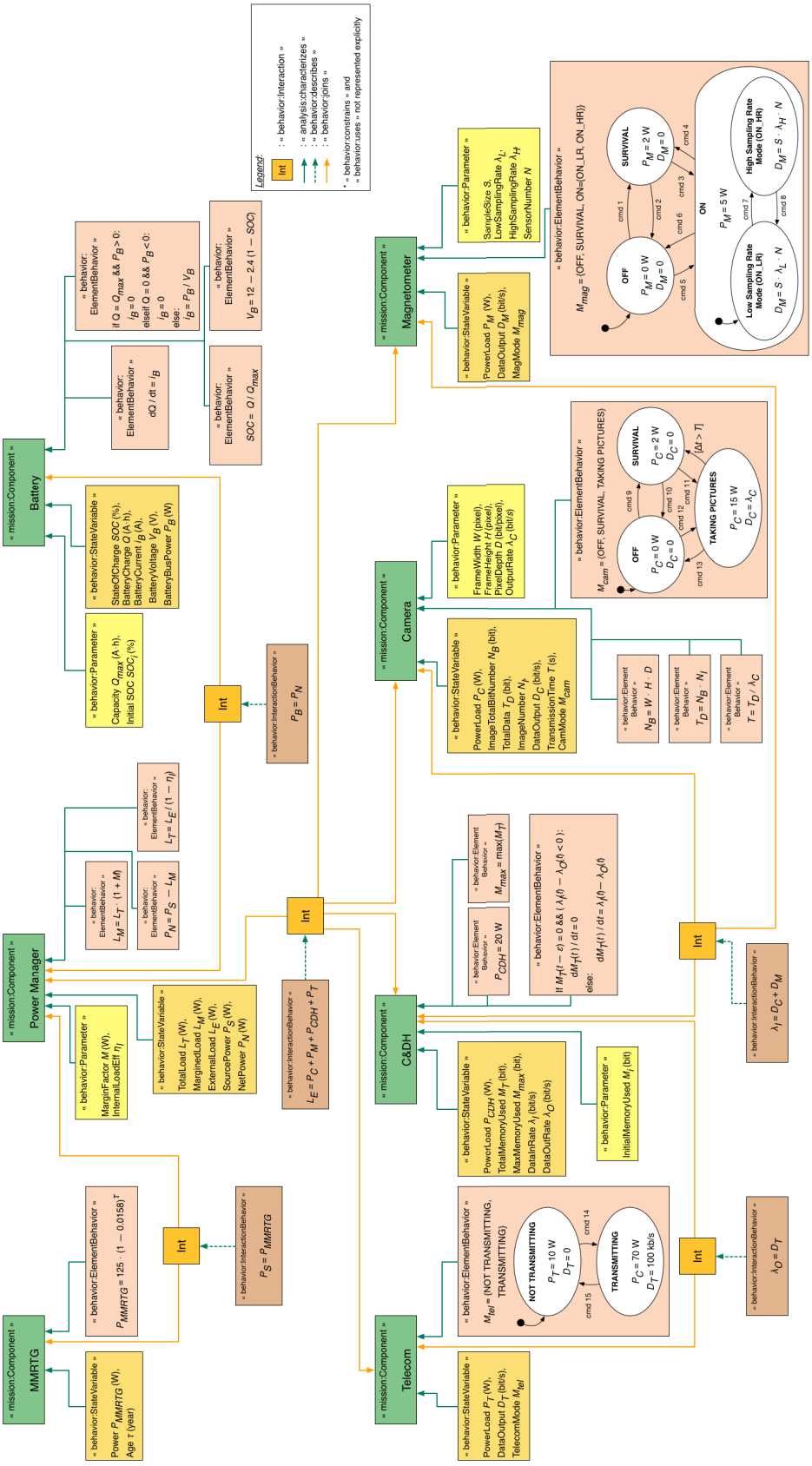

Figure 14. Spacecraft power and data behavioral model.

The power aspect of this system is described as follows:

- The C&DH is assumed to be always on and consuming 20 W of power. More detailed power load behavior could be supported by the behavior pattern.
- The Telecom has two power states: transmitting or not transmitting with associated power loads.
- The MMRTG power behavior is modeled using an exponentially decaying model. It produces 125 W of power at the beginning of life and 100 W after 14 years. This model is based on published work by Misra[23].
- The power manager behavior captures the power margin and the internal losses on top of the total power load of the spacecraft. The net power is then captured by subtracting the total load to the available power.
- The battery behavior model is more complex than the one presented in the flashlight, as it takes into consideration the state of charge of the battery, and its influence on the voltage and current of the battery. The battery itself is idealized with zero internal resistance, resulting in the linear variation of voltage with state of charge[24]. The battery can be charging or discharging based on the battery bus power sign.

The information above captured the ElementBehavior of each of the components. The following describes how these components interact with one another from a power perspective. Three interactions are modeled:

- The interaction between the MMRTG and the power manager: this interaction indicates that the power produced by the MMRTG is equal to the source power available to the power manager.
- The interaction between the power manager and the four power loads (Telecom, C&DH, camera and magnetometer): the total power load is the sum of each of the power load of the four components.
- The interaction between the power manager and the battery: the net power (available power minus total load) is equal to the power that the battery experiences: if the net power is negative (the MMRTG is not providing enough power for the load current demand), then the battery provides that extra power (within the limits of its capacity) and is thus discharging; if the net power is positive, then the battery uses this extra power to recharge if necessary.

Finally, the data aspect of this system is described as follows:

- The C&DH commits to memory the data it receives and removes from memory the data it outputs for transmission. This model does not capture the practice of keeping data in storage until the ground has acknowledged reception.
- As described above, the Telecom has two states: a non-transmitting state, and a transmitting one with a fixed data output rate.

The information above is captured in the ElementBehavior of each of the components. The following describes how these components interact with one another from a data perspective. Two interactions are modeled:

- The interaction between the C&DH and the two instruments: the data the C&DH receives is the sum of the data generated by the two instruments.
- The interaction between the C&DH and the Telecom: the data the Telecom transmits is the data the C&DH outputs.

One of the benefits of constructing such models is the ability to generate Trajectories for StateVariables given some Scenario. This also allows modelers to conduct trade studies and optimizations (for example sizing the battery or on-board memory storage) by modifying the model or Scenario and observing the resulting effect in the Trajectories. An example Scenario is shown in Figure 15: the camera and the magnetometer are turned on and back to survival mode according to a specified schedule, and the Telecom transmitting all of the data generated. The scenario is captured using a goal network representation form State Analysis: the green rectangles represent constraints on the StateVariables of the relevant components, the grey circles are timepoints (dotted lines indicates timepoints that occur at the same time), and the arcs between timepoints indicate duration intervals or conditions. The reader is referred to Ingham et al.[9] for more details on this representation.

Assuming some values for all Parameters of the model, the Trajectories of several StateVariables can be determined by solving the constraint equations of the ElementBehavior and InteractionBehavior blocks. The StateVariables of interest in this analysis are the power of the MMRTG, the total load with margin, the individual power loads, the battery state of charge, and the total memory used by the C&DH. The results of the analysis are shown in Figure 16.

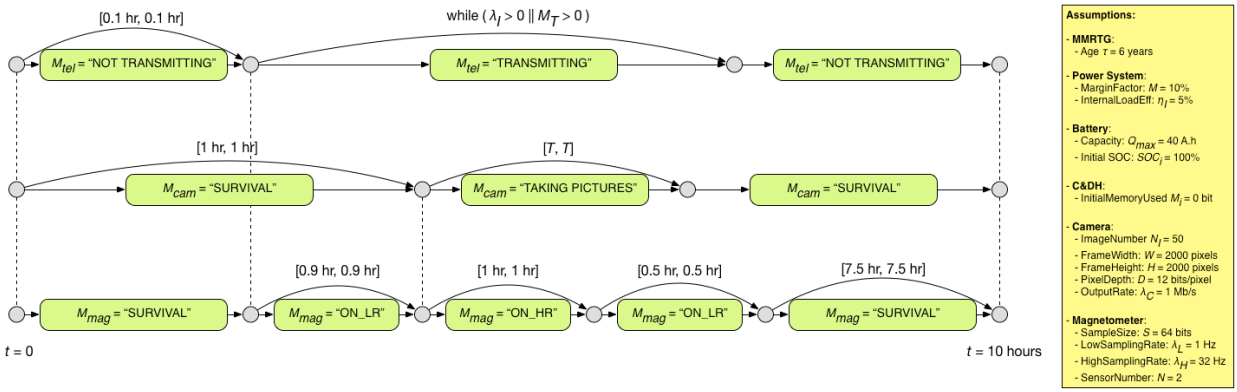

Figure 15. Example scenario for the spacecraft power and data behavior model.

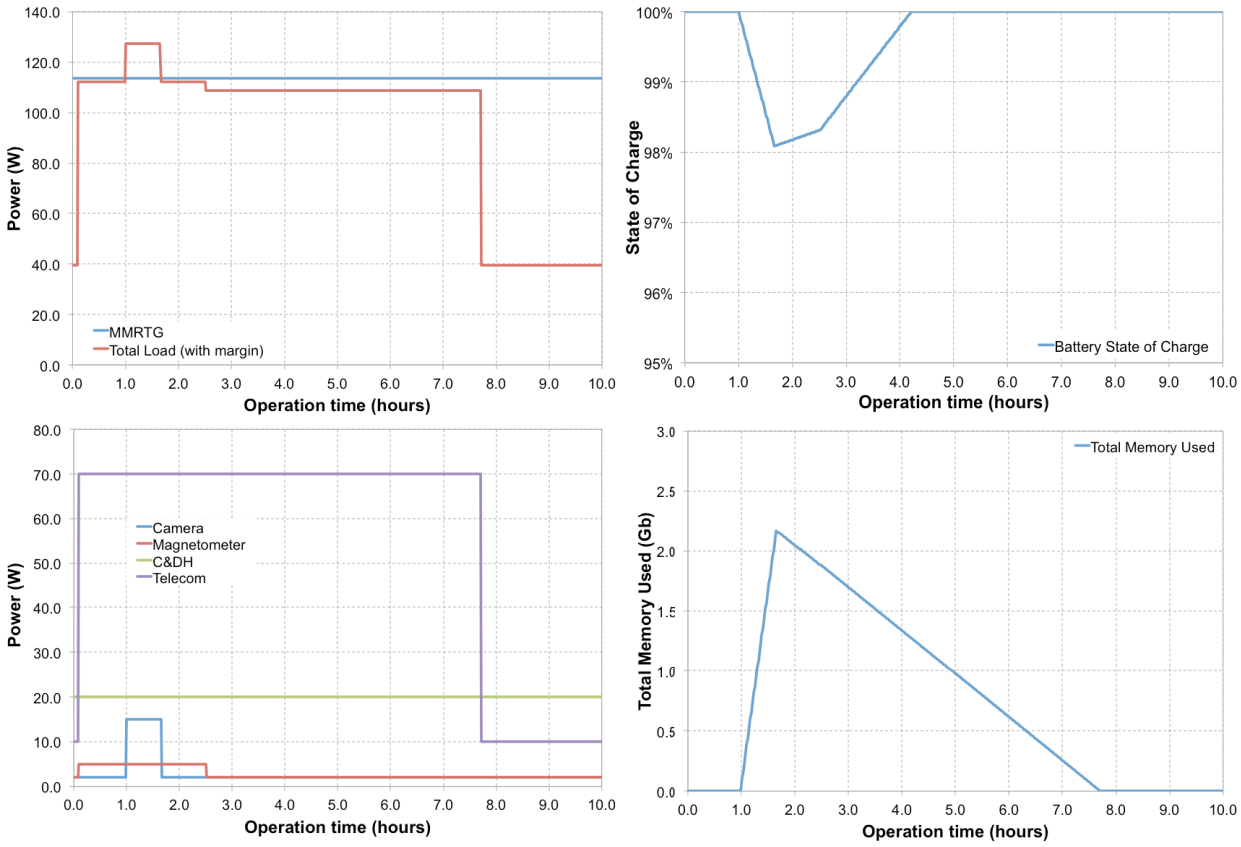

Figure 16. StateVariable Trajectories given the behavior model in Figure 14 and the scenario in Figure 15.

IV. SysML Implementation

In order to practically utilize the behavior ontology, a SysML extension (from SysML 1.3) was produced in the form of a stereotype profile. SysML is "a general-purpose graphical modeling language for representing systems"[25] and this profile allows for the projection of the behavior concepts and relationships identified in this work onto SysML model elements. This implementation was developed to provide engineers with the ability to use the behavior ontology in a compatible fashion with related models already captured in SysML. This section heavily uses UML/SysML concepts, and the reader interested in the definition of these concepts and their usage is referred to the OMG specifications[12,13], or to the book by Friedenthal et al.[25].

Table 1 presents the SysML embedding that was chosen for each of the ontology concept and relationship from Figure 8.

Table 1. SysML embedding of behavior ontology concepts and relationships.

Concept	**SysML embedding**
BehavingElement	Abstract concept, concrete classes embedded instead
Codomain	Value Property
ElementBehavior	Component ConstraintBlock or StateMachine
InteractionBehavior	Component ConstraintBlock
Interaction	Component Block
Parameter	Value property
State	State
StateVariable	Value property
TimeDomain	Value property
Relationship	**SysML embedding**
analysis:characterizes	Dependency
constrains	No direct embedding – see examples
hasCodomain	No direct embedding, using Value Type (see Figure 17)
hasTimeDomain	No direct embedding, using Value Type (see Figure 17)
isDescribedBy	Composite association
isElementOf	No direct embedding – see examples
joins	Shared association
uses	No direct embedding – see examples

BehavingElement is an abstract concept that is realized using concrete classes, such as mission:Component or mission:Environment from the IMCE mission ontology. Both of these classes are embedded in SysML as Components.

In addition to the concepts listed in Table 1, value types have been used to handle the definition of TimeDomain and Codomain of StateVariables: specifically, two value type stereotypes have been created:

- StateVariableValueType: this value type is used to type StateVariables and possesses two value properties: one for the TimeDomain and one for the Codomain;
- ParameterValueType: this value type is used to type Parameters.

Both these specialized value types takes advantage of the ISO 80000 Model Library from OMG[13], and examples of StateVariables from the flashlight example are shown in Figure 17. Using the flashlight example, the StateVariable CurrentThroughBattery is typed by the StateVariableValueType "currentSV" shown in Figure 17: this value type has two properties: (i) a domain (representing the TimeDomain of the StateVariable) typed by the value type "time" from the ISO library (associated with the time quantity kind); and (ii) a codomain (representing the Codomain of the StateVariable) typed by the value type "electric current" from the ISO library (associated with the electric current quantity kind). More specialized and customized value types and quantity kinds can also be created as necessary. Also, the handling of discrete StateVariables is described later in this paper.

Finally, a word on the SysML implementation of the optional InteractionTerminal: an InteractionTerminal is embedded as InterfaceBlock, and the "presents" relationship is embedded as a full port, owned by components grouping properties and typed by the InterfaceBlock. The properties are then exposed by binding them (using binding connectors) to associated UML ports nested in the full port. InteractionTerminals are not used in the examples presented in the remainder of this paper.

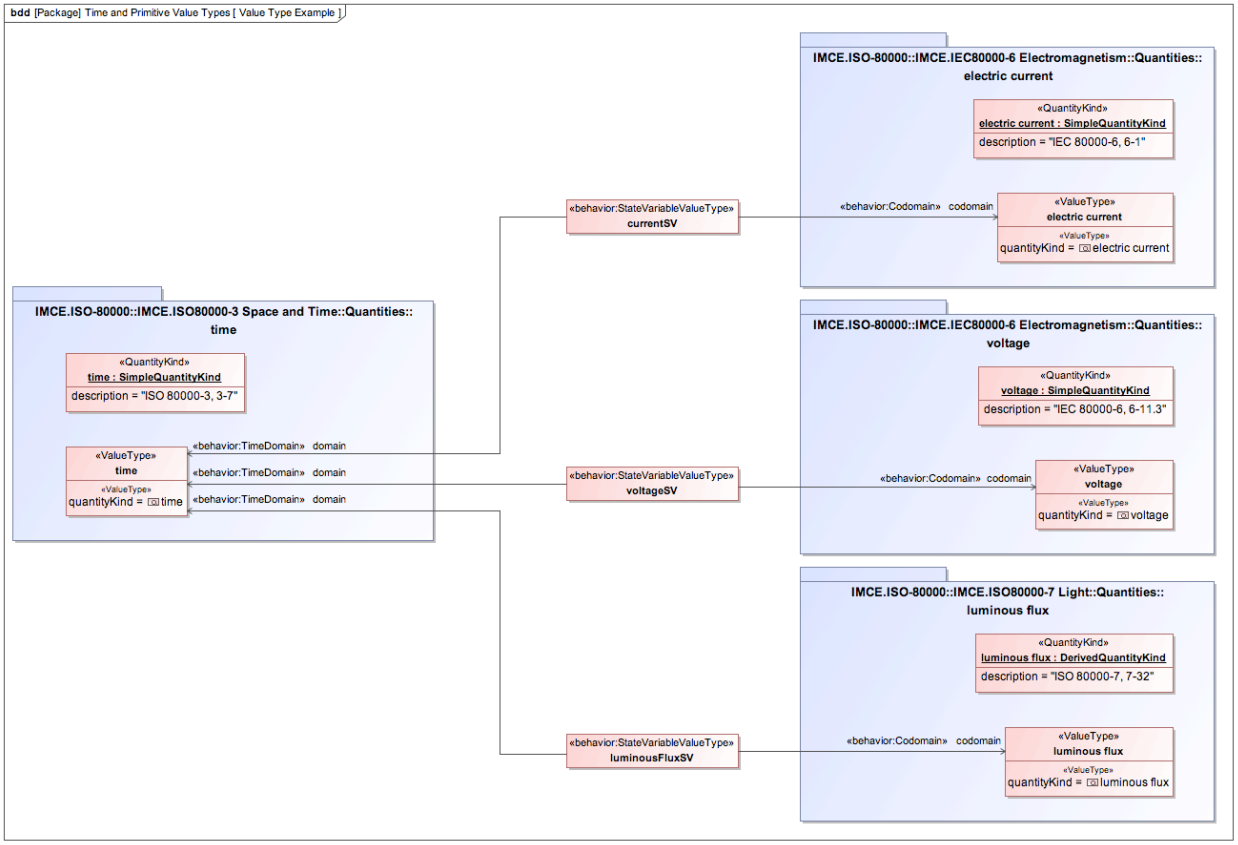

Figure 17. Value type example for three StateVariables from the flashlight example: current, voltage, and luminous flux.

Figure 18 shows a SysML model of the properties defined for the battery from the flashlight example, as well as the associated ElementBehavior. This behavior was described conceptually in Figure 12.

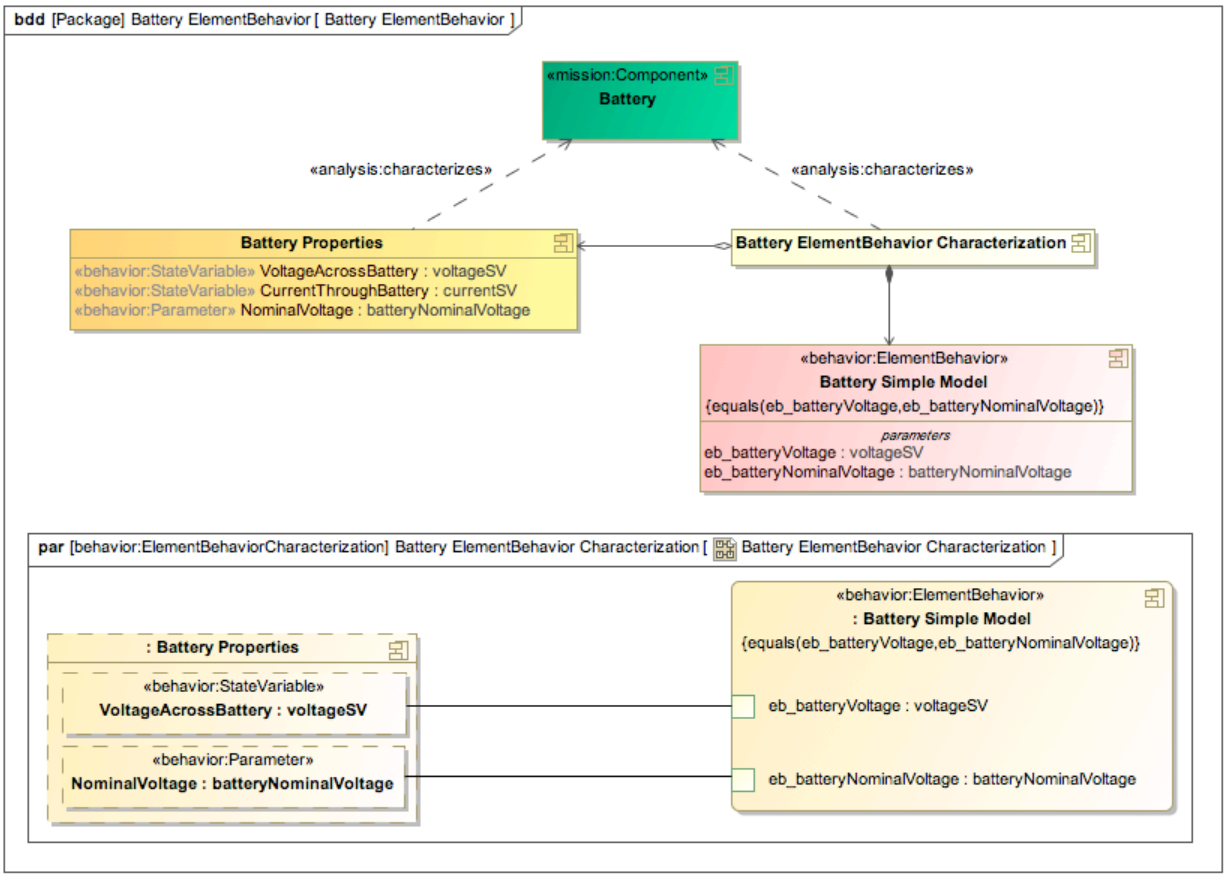

Figure 18. ElementBehavior model in SysML for the battery in the flashlight example.

As discussed earlier, the use of IMCE characterizations results in the separation of the properties used in this behavior model from the battery component itself. As a consequence, the StateVariables and Parameters are grouped into Component Blocks that characterize the component. The meaning of the grouping is left to the modeler: in Figure 18, all properties of interest are gathered into one characterization, while in Figure 20, the properties are split between StateVariables and Parameters. Note that the StateVariables are typed by the value types shown in Figure 17.

Regarding the embedding of ElementBehavior, Figure 18 shows the presence of additional elements. For example, the Battery ElementBehaviorCharacterization has been added in SysML to act as an grouping mechanism of several ConstraintBlocks (stereotyped by «behavior:ElementBehavior»). This mechanism is used for different reasons, such as: (i) for example in Figure 19 to indicate that the OPEN constraint and the CLOSE constraint are part of the same ElementBehavior specification; (ii) to define in the same model alternative characterizations for a given BehavingElement: e.g., one Battery ElementBehaviorCharacterization can be used to represent a battery model with constant voltage output, while another ElementBehaviorCharacterization of the Battery could be used to define a more realistic model with usage-dependent voltage level; or (iii) to separate different types of behavior of the same BehavingElement: e.g., all the constraints related to its power aspect would be grouped with one ElementBehaviorCharacterization, and all the constraints related to its data aspect would be grouped with another one.

In addition, constraint parameters are used in the ConstraintBlocks (such as “eb_batteryVoltage”) in an effort to make the ConstraintBlocks reusable. One can envision the creation of library of behavior specifications (such as Ohm’s law in Figure 20), and the constraint parameters are then bound using binding connectors to the appropriate properties of the BehavingElement in the parametric diagram of the ElementBehavior Characterization block, referencing the appropriate property characterization. In Figure 18,, the Battery ElementBehavior Characterization references the Battery Properties, and the constraint parameters eb_batteryVoltage and eb_batteryNominalVoltage are bound respectively to the StateVariable VoltageAcrossBattery and the Parameter NominalVoltage.

Finally, the ConstraintBlock "Battery Simple Model" is constrained by a constraint that represents the mathematical expression equating the battery voltage to the battery nominal voltage. It is currently embedded in SysML using expression trees, operations and Element Values, but the specification of the constraint language is out of the purview of the behavior ontology and is part of an agreement amongst modelers.

Figure 19 shows the SysML model of the switch ElementBehavior, and it introduces State Machines and the handling of discrete StateVariables.

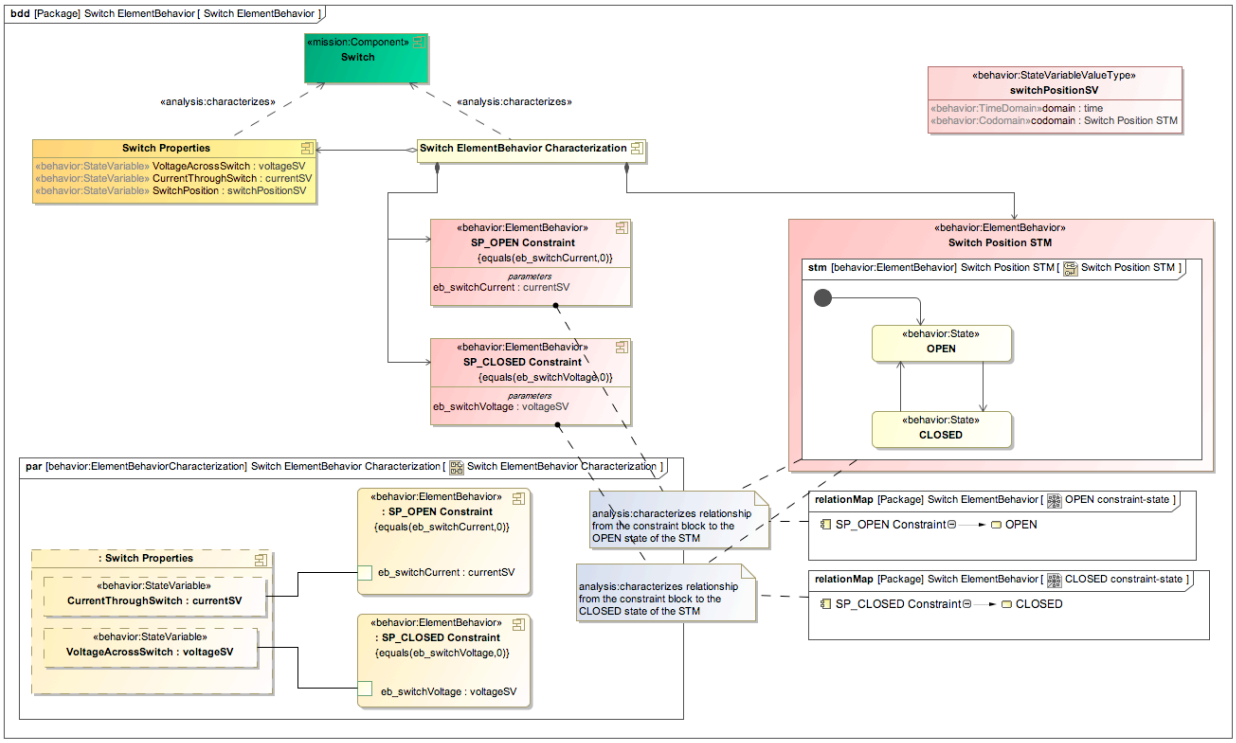

Figure 19. ElementBehavior model in SysML for the switch in the flashlight example.

As described in Figure 12, the behavior of the switch is captured using a state machine formalism, and is shown in Figure 19 using UML State Machines. Note that the value type of the StateVariable "SwitchPosition" is shown in the top right corner of Figure 19, and indicates that the Codomain of this particular discrete StateVariable is typed by the State Machine "Switch Position STM", represented as one of the ElementBehaviors of the switch component in Figure 19. The flexibility of SysML to use State Machines as classifiers allows us to avoid the duplication of the specification of the Codomain of the discrete StateVariable with the specification of the relevant behavior using a State Machine. In this example, the State Machine serves both roles: the states of the State Machine enumerate the States (OPEN and CLOSED , elements of the Codomain of the StateVariable SwitchPosition), and the State Machine is also used to specify the ElementBehavior constraints associated with each state. For example, if the switch is open, then the current across the switch is zero. This constraint is captured by the ConstraintBlock named "SP_OPEN Constraint", and is linked to the OPEN state in the state machine by a dependency stereotyped by «analysis:characterizes», as indicated by the notes and the relationMap diagram in Figure 19. This is a convention to indicate the domain of applicability of the constraint: the constraint is applicable only for the states it characterizes (using the dependency). Alternatively, if a constraint does not characterize any state in the presence of a state machine, then it is assumed that the constraint applies to all states of the state machine. Note that this convention could also be handled by using conditional logic statements instead in the Constraint Blocks:

IF (SwitchPosition = OPEN)
    CurrentThroughSwitch = 0
ELSEIF (SwitchPosition = CLOSED)
    VoltageAcrossSwitch = 0
ENDIF

Figure 20 shows the ElementBehavior model of the lamp, capturing the Ohmic behavior of the lamp and the light generation behavior. These behaviors were described conceptually in Figure 10.

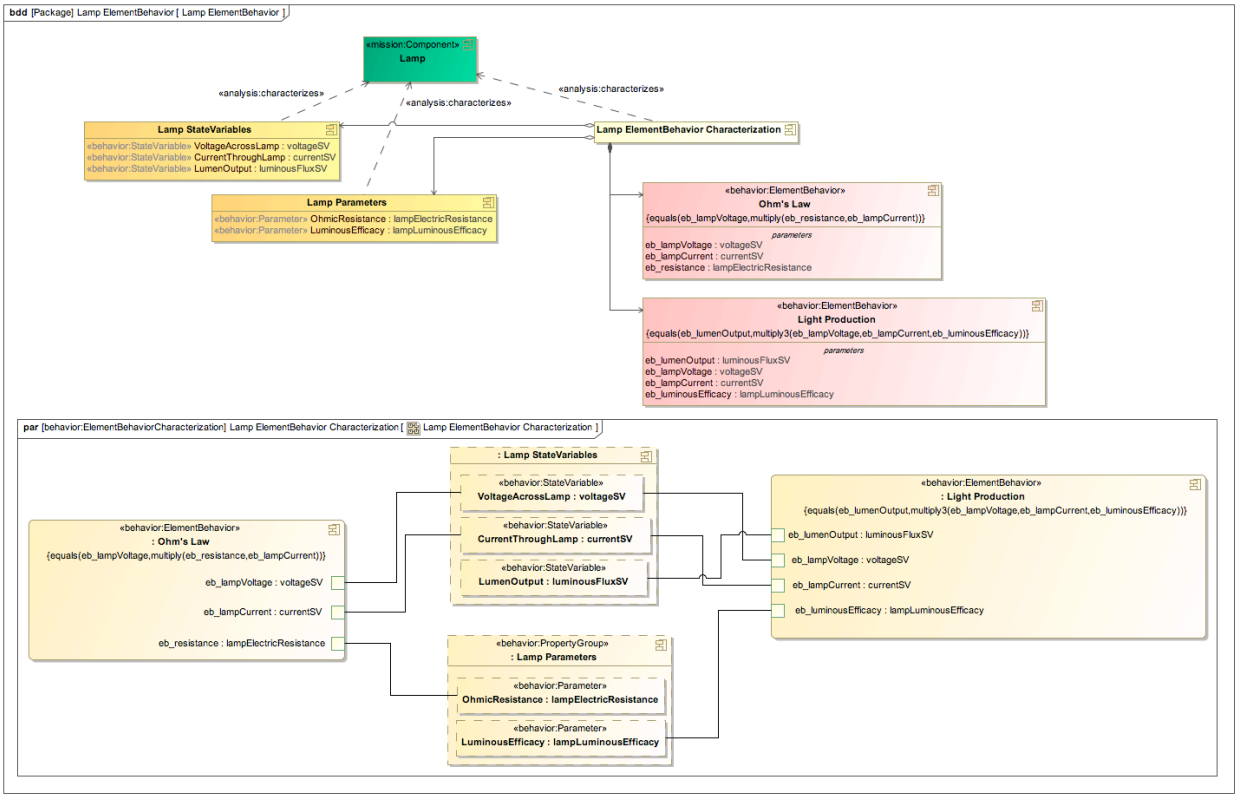

Figure 20. ElementBehavior model in SysML for the lamp in the flashlight example.

Finally, the interaction between the three components from a mesh analysis's perspective is shown in Figure 21, and was described conceptually in Figure 11.

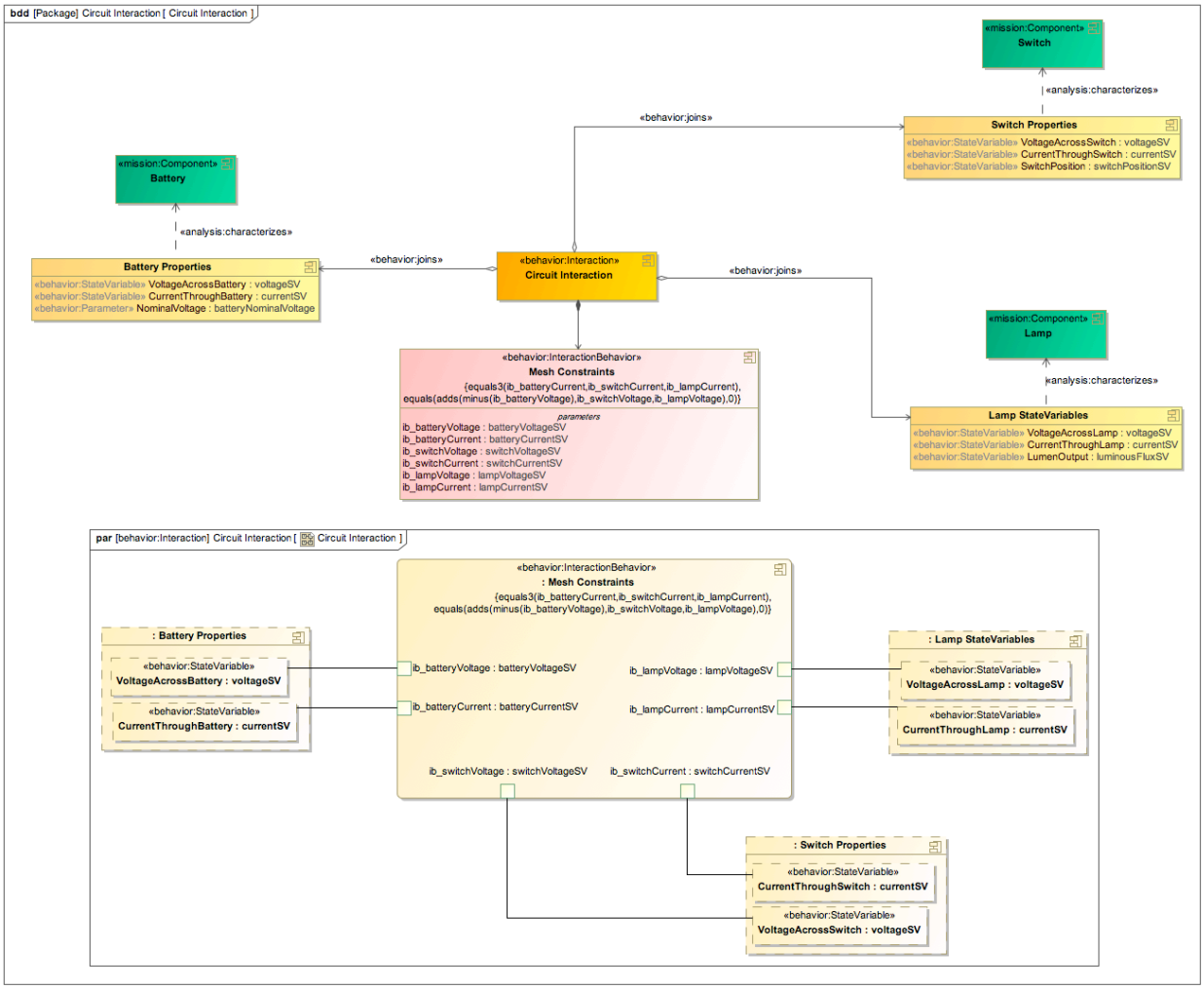

Figure 21. Interaction model in SysML for the flashlight example using a mesh analysis.

The embedding of the InteractionBehavior is captured similarly as for the ElementBehavior, using ConstraintBlocks, constraint parameters, constraints, and binding connectors. Note that in the ontology, the Interaction joins BehavingElements; however in our SysML implementation, due to the separation of the BehavingElement and the property characterizations, the «behavior:joins» shared associations relate the Interaction to the property groupings directly. Also the "isDescribedBy" relationship (inverse of "describes" relationship between an InteractionBehavior and Interaction) is captured using a composite association (i.e., a "black diamond" arrow).

V. Conclusion

The behavior ontology presented in this paper provides a consistent behavior specification within the existing systems engineering process, and improves upon current best practices to allow for efficient design trades and optimization, re-use of model elements from project to project, more rigorous and formal behavioral analyses, and exchange of behavior information between different tools or languages.

The behavior ontology provides an approach to capture the state-based behavior of elements and the interactions amongst them. These elements are characterized by state variables that vary with time, and the dynamic evolution of the state of the system is represented using constraints on these state variables. This paper discussed the various concepts and relationships introduced to capture behavioral specifications: the properties (StateVariables and Parameters) necessary to characterize BehavingElements, ElementBehaviors to capture the internal behavior of these elements, and Interactions and InteractionBehaviors to capture the dynamics among them. In addition, Scenarios and Trajectories of StateVariables were introduced to capture a driving mechanism for behavioral

analyses that consider the resulting evolution in time of StateVariables. Two examples were discussed in this paper: (i) a simple flashlight electrical model to illustrate the concepts and relationships introduced by the ontology and their flexible usage to the modeler; and (ii) a more complex spacecraft model involving instruments, power and data behaviors to demonstrate the ability of the behavior ontology to handle real-world engineering problems. Finally, an implementation in a SysML profile was provided to enable engineers to use the behavior ontology in a compatible fashion with related models already captured in SysML.

Wider-scale application of the behavior ontology to flight projects is currently under way at JPL, as is further work on scenario specification. This will allow performance of end-to-end behavioral analyses and identification of areas for further development of the approach proposed in this paper. Another area of investigation is to extend the behavior ontology to describe off-nominal behaviors and provide capabilities for fault management activities, such as automatic generation of reliability artifacts (FMECA, fault trees) or fault coverage analyses.

Acknowledgments

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

We wish to acknowledge Robert Rassmussen, John Day, Maddalena Jackson, Chris Delp, Elyse Fosse, Bjorn Cole, Johannes Gross and Sandy Friedenthal for their participation in the maturation of the behavior ontology and the SysML embedding.

References

[1]Bayer, T. J., Bennett, M., Delp, C. L., Dvorak, D., Jenkins, J. S., and Mandutianu, S., "Update – Concept of Operations for Integrated Model-Centric Engineering at JPL," *IEEE Aerospace Conference*, Paper AC1122, March 2011. doi: 10.1109/AERO.2011.5747538.

[2]Jenkins, J. S., and Rouquette, N. F., "Semantically-Rigorous Systems Engineering Modeling using SysML and OWL," *5th International Workshop on Systems & Concurrent Engineering for Space Applications*, Lisbon, Portugal, October 17–19, 2012.

[3]W3C, *SPARQL 1.1 Query Language*, 21 March 2013, http://www.w3.org/TR/sparql11-query [retrieved 11/18/2014].

[4]Erden, M. S., et al., "A Review of Function Modeling: Approaches and Applications," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 22, pp. 147–169, 2008.

[5]Bock, C., and Odell, J., "Ontological Behavior Modeling," Journal of Object Technology, Vol. 10, 2011.

[6]Willems, J. C., "The Behavioral Approach to Open and Interconnected Systems – Modeling by Tearing, Zooming, and Linking," *IEEE Control Systems Magazine*, December 2007.

[7]Mattsson, S. E., and Elmqvist, H., "Modelica – An International Effort to Design the Next Generation Modeling Language," *7th IFAC Symposium on Computer Aided Control Systems Design, CACSD '97*, Gent, Belgium, April 28–30 1997.

[8]Ingham, M. D., et al., "A Model-Based Approach to Engineering Behavior of Complex Aerospace Systems," Infotech@Aerospace 2012 Conference, Santa Ana, CA, June 19–21 2012.

[9]Ingham, M. D., et al., "Engineering Complex Embedded Systems with State Analysis and the Mission Data System" *Journal of Aerospace Computing, Information, and Communication*, Vol. 2, 2005.

[10]Wagner, D. A., et al., "An Ontology for State Analysis: Formalizing the Mapping to SysML," *IEEE Aerospace Conference*, March 3–10, 2012.

[11]Chung, S. H., and Bindschadler, D. L., "Timeline-Based Mission Operations Architecture: An Overview," *Proceedings of the 12th International Conference on Space Operations (SpaceOps 2012)*, Stockholm, Sweden, June 11–15 2012

[12]Object Management Group, *Unified Modeling Language (UML)*, Version 2.5, http://www.omg.org/spec/UML/2.5/Beta2 [retrieved 11/18/2014].

[13]Object Management Group, *Systems Modeling Language (SysML)*, Version 1.3, http://www.omg.org/spec/SysML/1.3 [retrieved 11/18/2014].

[14] Modelica Association, *Modelica Language Specification*, Version 3.3 Rev. 1, July 11, 2014, https://modelica.org/documents/ModelicaSpec33Revision1.pdf [retrieved 11/18/2014].

[15]Paredis, C., et al., "An Overview of the SysML-Modelica Transformation Specification," *INCOSE International Symposium*, 2010.

[16]Johnson, T., Kerzhner, A., Paredis, C. J., and Burkhart, R., "Integrating Models and Simulations of Continuous Dynamics into SysML," *Journal of Computing and Information Science in Engineering*, Vol. 12, No. 1, 2012.

[17]Wolfram Research, *Mathematica*, http://www.wolfram.com/mathematica [retrieved 11/18/2014]

[18]Maplesoft, *Maple*, http://www.maplesoft.com/products/maple [retrieved 11/18/2014]

[19]Wissler, S., Maldague, P., Rocca, J. and Seybold, C., "Deep Impact Sequence Planning Using Multi-Mission Adaptable Planning Tools with Integrated Spacecraft Models," *SpaceOps 2006 Conference*, Rome, Italy, June 19–23, 2006.

[20]International Organization for Standardization, *Quantity and Units – Part 1: General*, 2009.

[21]Characterization Pattern Description, JPL internal document URS 247599, available upon request.

[22]Bourbaki, N., *Elements de Mathematique,Theorie des Ensembles*, Hermann & Cie, 1954.

[23]Misra, A. K., "Overview of NASA Program on Development of Radioisotope Power Systems with High Specific Power," *4th International Energy Conversion Engineering Conference and Exhibit (IECEC)*, San Diego, CA, 2006.

[24]Pang, S., Farrell, J., Du, J., and Barth, M., "Battery State-of-Charge Estimation," in *Proceedings of the American Control Conference*, Vol. 2, pp. 1644–1649, June 2001.

[25]Friedenthal, S., Moore, A., and Steiner, R., *A Practical Guide to SysML: The Systems Modeling Language*, 2nd ed., Elsevier, 2012.