Protocol Activity Markup Language (PAML) Version 1.0.0-alpha

Bryan Bartley	*Raytheon BBN Technologies, USA*
Jacob Beal	*Raytheon BBN Technologies, USA*
Daniel Bryce	*SIFT, USA*
Robert P. Goldman	*SIFT, USA*
Benjamin Keller	*University of Washington, USA*
Peter Lee	*Strateos, USA*
Joshua Nowak	*Strateos, USA*
Miles Rogers	*Raytheon BBN Technologies, USA*
Mark Weston	*Netrias, USA*
Additional authors	from contributing organizations

Version 1.0-alpha

TBD, 2021

Contents

1 Purpose

Laboratory protocols are critical to biological research and development, but can be difficult to communicate or reproduce due to the differences in context, skills, and resources between different projects, investigators, and organizations. The Protocol Activity Markup Language (PAML) aims to address this problem by providing a data model for description of laboratory protocols that is unambiguous enough for precise interpretation and automation, yet simultaneously abstract enough to support reuse and adaptation.

1.1 Requirements

In order to better understand the requirements on PAML as an artifact, here is an enumeration unpacking some of the key computational tasks that PAML needs to be able to support in order to further the goals above:

Instantiation An *abstract* protocol specifies a recipe to be followed in the laboratory. A concrete *instance* of a protocol is an execution of such a recipe at a specific time and place, on a specific set of equipment, etc. This implies at least two representational requirements:

Execution Record We must be able to create some persistent data structure that records an instance in which a protocol is executed.

Derivation We must be able to record the relationship between the execution record for a protocol instance and the protocol recipe from which it is derived.

Metadata markup Closely related to instantiation, a PAML protocol description should support automatic metadata tagging of data that are collected in the course of protocol execution, *to the extent lab automation at the executing lab supports this*. For example, a protocol that collects flow cytometry data on samples of different strains under varying growth conditions, should support automatic association of each FCS file produced by the flow cytometer with the strain, growth conditions, time of data collection, calibration, etc. of the sample from which the FCS file was produced.

Mapping a protocol to a laboratory PAML aims to support replication in part by providing automated support for mapping protocols developed at one lab onto another. Note, however, that PAML intentionally will not attempt to provide explicit guarantees about transfer or replicability of protocol executions. Such guarantees cannot be made for a poorly understood or overly delicate protocol, no matter how it is described. Rather, the PAML specification must allow a protocol to say how to *predict* if a mapping will product a correct execution and how to *check* if an execution should be considered correct, i.e., what a protocol specification truly requires, what aspects can safely be varied, which must be honored, and what reasonable tolerances are for inputs and outputs.

Execution To enable replication, PAML must specify what it means to execute a protocol correctly. This definition must support a wide range of degrees of lab automation, from entirely manual, to manual supported by some automated workstation to entirely robotic. Specifically this involves giving a *compositional* account of the execution semantics in which the meaning of the protocol is a function of its component actions and the relationships between them. This also involves supporting *translation* from PAML to alternative representations for different uses. For example, an initial PAML proof-of-concept translates PAML to Markdown[1] for labs operated by technicians, and an initial translation to Autoprotocol[2] for wholly robotic laboratories using that control standard.

Modification We must be able to record modifications made to an original protocol. For example, a protocol may be modified in order to enable the protocol to be executed at a lab with different equipment from the lab at which it was originated, to enable the protocol to be scaled up or scaled down, etc.

[1] https://github.com/SD2E/paml for the translator, https://commonmark.org/ gives the Markdown spec.
[2] autoprotocol.org

A use pattern that may end up being common is to have a protocol be “too strict" (too specific) to be instantiated in a lab, and thus need modification. Rather than the modification being a patch, as one might do in programming, however, the user would generalize the original protocol *into a new protocol* that is a generalization allowing it to be instantiated in both the labs where it was originated *and* in new labs. This new protocol would be released as an updated version of the existing protocol.

In this context, the term “modification” is intended to refer to *tracking modifications*: what is required is a combination of *versioning* and *provenance tracking*. A method for computing understandable version differences would also be part of this high level task, which encompasses ongoing development and versioning of the protocol.

A key requirement here will be maintaining information about the *specific version* of the protocol used in an execution record, and underlying a particular dataset, in order to ensure that new versions released later do not cause incorrect data interpretation.

Authoring derived works As with most complex formal artifacts (programs, spreadsheets, etc.), we confidently expect that few users will create new protocols from an entirely blank slate. Instead, many will take an existing protocol, copy it, and edit the copy, possibly by copying and editing material from yet other protocols. PAML must support this mode of operation. Key issues here are determining which component structures can be incorporated by reference, and which must be copied instead. Tracking the relationship between an original protocol, and protocols that have branched from it would also be desirable, in order to support, for example, propagating fixes from a root protocol to others.

Protocol maintenance Closely related to the tasks of “Modification,” and “Derived works”, PAML also needs to be able to support the ongoing development, repair, and versioning of a protocol.

Verification Authoring a formal protocol, in PAML or otherwise, requires substantial care and effort, and the usefulness of the protocol can be compromised if its specification is ill-formed or erroneous. Supporting protocol authors in achieving correctness is an important goal. Some of this will fall onto the shoulders of those building PAML support tools, however the specification itself must provide guidance as to what it means for a protocol to be complete, consistent, etc.

Planning and scheduling Laboratory resources are valuable, and some organizations will want to be able to optimize their use. To do so, PAML should support (at least) extraction of resource requirements from activities in the protocol, and estimated durations. Note that *what* resource requirements and duration estimates are required will likely be a function of both the protocol and the lab. Which resources are limited, and must be considered in a planner or scheduler, versus those that can be effectively treated as unlimited, will vary by lab. Management styles will also vary between organizations.

1.2 Implementation Approach and Scope

Where possible, PAML builds on other existing standards. In particular, PAML uses the Unified Modeling Language (UML) version 2.5.1 (Object Management Group, 2017) to describe the organization of actions in a protocol, the Synthetic Biology Open Language (SBOL) version 3 (Baig et al., 2020) to describe biological materials in terms of combinations of strains, media, etc., and uses the Ontology of Units of Measure (OM) (Rijgersberg et al., 2021) to describe parameters with physical units. As a foundation, PAML uses existing Semantic Web practices and resources, such as *Uniform Resource Identifiers* (`URI`s) and ontologies, to unambiguously identify and define biological system elements. and to provide serialization formats for encoding this information in electronic data files, as well as the SBOL approach to closure in reasoning about knowledge. This approach also allows PAML to be extended with additional custom information for particular uses and deployments.

Note that PAML is intentionally agnostic to any details of computer networking used to discover, share, or otherwise operate with protocols, in order to maximize flexibility in available options for implementation of such services.

2 Conventions

This section provides some preliminary information to aid in the understanding of the specification. The PAML data model is specified using Unified Modeling Language (UML) 2.5 diagrams (Object Management Group, 2017). This section reviews terminology conventions, the basics of UML diagrams, and our naming conventions.

2.1 Terminology Conventions

This document indicates requirement levels using the controlled vocabulary specified in IETF RFC 2119. In particular, the key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

- The words “MUST”, “REQUIRED”, or “SHALL” mean that the item is an absolute requirement.
- The phrases “MUST NOT” or “SHALL NOT” mean that the item is an absolute prohibition.
- The word “SHOULD” or the adjective “RECOMMENDED” mean that there might exist valid reasons in particular circumstances to ignore a particular item, but the full implications need to be understood and carefully weighed before choosing a different course.
- The phrases “SHOULD NOT” or “NOT RECOMMENDED” mean that there might exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications needs to be understood and the case carefully weighed before implementing any behavior described with this label.
- The word “MAY” or the adjective “OPTIONAL” mean that an item is truly optional.

2.2 UML Diagram Conventions

The types of data modeled by PAML are commonly referred to as *classes*, especially when discussing the details of software implementation. Each PAML class can be instantiated by many PAML objects. These objects MAY contain data that differ in content, but they MUST agree on the type and form of their data as dictated by their common class. Classes are represented in UML diagrams as rectangles labeled at the top with class names (see Figure 1 for examples).

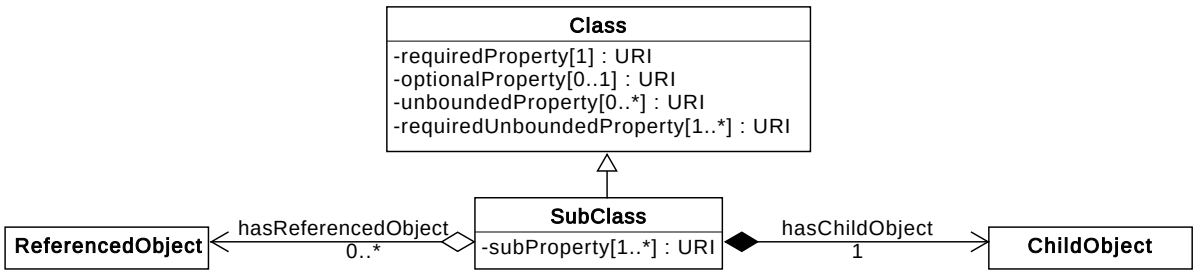

Figure 1: Examples of UML diagram conventions used in this document

Classes can be connected to other classes by association properties, which are represented in UML diagrams as arrows. These arrows are labeled with data cardinalities in order to indicate how many values a given association

property can possess (see below). The remaining (non-association) properties of a class are listed below its name. Each of the latter properties is labeled with its data type and cardinality.

In the case of an association property, the class from which the arrow originates is the owner of the association property. A diamond at the origin of the arrow indicates the type of association. Open-faced diamonds indicate shared aggregation, also known as a reference, in which the owner of the association property exists independently of its value.

By contrast, filled diamonds indicate composite aggregation, also known as a part-whole relationship, in which the value of the association property MUST NOT exist independently of its owner. In addition, in the PAML data model, it is REQUIRED that the value of each composite aggregation property is a unique PAML object (that is, not the value for more than one such property). Note that in all cases, composite aggregation is used in such a way that there SHOULD NOT be duplication of such objects. Such objects are also commonly referred to as "child" objects, and their owning objects as "parent" objects.

All PAML properties are labeled with one of several restrictions on data cardinality. These are:

- 1 - REQUIRED: there MUST be exactly one value for this property.
- 0…1 - OPTIONAL: there MAY be a single value for this property, or it MAY be absent.
- 0…∗ - zero or more: there MAY be any number of values for this property, including none.
- 1…∗ - REQUIRED, one or more: there MAY be any number of values for this property, as long as there is at least one.
- n…∗ - at least: there MUST be at least n values for this property.

Finally, classes can inherit the properties of other classes. Inheritance relationships are represented in UML diagrams as open-faced, triangular arrows that point from the inheriting class to the inherited class. Some classes in the PAML data model cannot be instantiated as objects and exist only to group common properties for inheritance. These classes have italicized names and are known as abstract classes.

2.3 Naming and Typographic Conventions

PAML classes are named using upper "camel case," meaning that each word is capitalized and all words are run together without spaces, e.g., `Protocol`. Properties, on the other hand, are named using lower camel case, meaning that they begin lowercase but if they consist of multiple words, all words after the first begin with an uppercase letter (e.g., `hasActivity`).

PAML properties are always given singular names irrespective of their cardinality, e.g., `hasActivity` is used rather than `hasParameters` even though a `Protocol` can have multiple parameters. This is because each relation can potentially stand on its own, irrespective of the existence of others in the set.

Two conventions are used for property names, `name` and `hasName`. When a property is pointing to a class using the same name, it uses the `hasName` convention (e.g., the `Protocol` class uses `hasActivity` to point to a `Activity` object). When the property uses a different name than the class of the object it points to, it uses the `name` convention instead (e.g., the `Protocol` class uses `inputActivity` to point to an input `om:Activity` object).

3 SBOL Imports: Identifiers, Primitive Types, and Classes

PAML builds on the Synthetic Biology Open Language (SBOL) version 3 (Baig et al., 2020) in several ways:

- PAML uses the same conventions as SBOL for URIs and types.
- PAML uses the SBOL base classes `sbol:Identified` and `sbol:TopLevel` as parents for all PAML classes.
- PAML uses SBOL classes to describe biological samples in terms of combinations of strains, media, etc.
- PAML makes use of the same external measurement ontology as SBOL, the Ontology of Units of Measure (OM) (Rijgersberg et al., 2021).

In order to make this document more self-contained, this section repeats the material from the SBOL specification on conventions and SBOL base classes. A summary is also provided for key SBOL and OM classes; for complete documentation, see the SBOL specification. In case of conflict between any material in this section and the SBOL specification, the SBOL specification should be held correct.

3.1 Uniform Resource Identifiers

As PAML is built upon the Resource Description Framework (RDF), all class instances are identified by a Uniform Resource Identifier (URI). In the PAML data model, the value of an association property MUST be a `URI` or set of `URI`s that refer to PAML objects belonging to the class at the tip of the arrow. Every `sbol:Identified` object's URI MUST be globally unique among all other `sbol:Identified` object URIs. It is also highly RECOMMENDED that the `URI` structure follows the recommended best practices for compliant `URI`s specified in Section 3.1.1.

Whenever a `sbol:TopLevel` object's URI is a URL (e.g., following the conventions of HTTP(S) rather than a UUID), its structure MUST comply with the following rules:

- A `sbol:TopLevel` URL MUST use the following pattern: `[namespace]/[local]/[displayId]`, where `namespace` and `sbol:displayId` are required fragments, and the `local` fragment is an optional relative path.

  For example, a `Protocol` might have the URL `https://igem.org/protocols/OD/calibration_2018`, where `namespace` is `https://igem.org`, `local` is `protocols/OD`, and `sbol:displayId` is `calibration_2018`.

- A `sbol:TopLevel` object's URL MUST NOT be included as prefix for any other `sbol:TopLevel` object.

  For example, the `run102` `ProtocolExecution` object cannot have a URL of `https://igem.org/protocols/OD/calibration_2018/run102`, since the `https://igem.org/protocols/OD/calibration_2018` prefix is already used as a URL for a `Protocol` object.

- The URL of any child or nested object MUST use the following pattern:`[parent]/[displayId]`, where `parent` is the URL of its parent object. Multiple layers of child objects are allowed using the same `[parent]/[displayId]` pattern recursively.

  For example, a `uml:CallBehaviorAction` object owned by the `calibration_2018` `Protocol` and having a `sbol:displayId` of `CallBehaviorAction1` will have a URL of `https://igem.org/protocols/OD/calibration_2018/CallBehaviorAction1`. Similarly, if the `CallBehaviorAction1` object has a `uml:InputPin` child object with a `sbol:displayId` of `InputPin1`, then that object will have the URL `https://igem.org/protocols/OD/calibration_2018/CallBehaviorAction1/InputPin1`.

3.1.1 Compliant URIs

Maintaining unique URIs for objects can be challenging. Compliant URIs follow a set of rules that simplify this challenge.

Compliant URIs for `sbol:TopLevel` objects MUST conform to the following pattern:

⟨namespace⟩/⟨collection_structure⟩/⟨displayId⟩

The ⟨namespace⟩ token MAY further decompose into ⟨domain⟩/⟨root⟩ tokens. The ⟨root⟩ and ⟨collection_structure⟩ tokens may optionally be omitted; alternatively, they may consist of an arbitrary number of delimiter-separated layers. Note that this pattern means that SBOL-compliant `URI`s can be automatically decomposed with the aid of a `sbol:TopLevel` object's `sbol:hasNamespace` property. SBOL-compliant objects can be easily remapped into new namespaces by changing only the ⟨namespace⟩.

Consider, for example, the SBOL-compliant `URI`:

"https://igem.org/engineering/protocols/platereader/OD/calibration_2018"

for a `sbol:Component` with a `sbol:hasNamespace` value "https://igem.org/engineering/protocols". This `URI` can be decomposed as follows:

namespace:	"https://igem.org/engineering/protocols"
domain:	"https://igem.org"
root:	"engineering/protocols"
collection:	"platereader/OD"
displayId:	"calibration_2018"

SBOL-compliant URIs also facilitate auto-construction of child objects with unique `URI`s. Child objects of `sbol:TopLevel` objects with compliant `URI`s MUST conform to the following pattern: "⟨parent_uri⟩/⟨child_type⟩⟨child_type_counter⟩" where the ⟨parent_uri⟩ refers to the URI of the parent object, the ⟨child_type⟩ refers to the SBOL class of the child object, and ⟨child_type_counter⟩ is a unique index for the child object. The ⟨child_type_counter⟩ of a new object SHOULD be calculated at time of object creation as 1 + the maximum ⟨child_type_counter⟩ for each ⟨child_type⟩ object in the parent (e.g., "⟨parent_uri⟩/Parameter7"). Note that numbering is independent for each type, so a `Protocol` can have children "CallBehaviorAction7" and "ControlFlow7".

3.2 PAML URIs

The actual ontology implementations aren't currently versioned correctly, per https://github.com/SD2E/paml/issues/59

The PAML namespace, which is `http://bioprotocols.org/paml/v1#`, is used to indicate which entities and properties in an PAML document are defined by PAML. For example, the URI of the type `Protocol` is `http://bioprotocols.org/paml/v1#Protocol`. This convention is assumed throughout the specification. The PAML namespace MUST NOT be used for any entities or properties not defined in this specification.

Likewise, because no suitable ontology previously existed for UML 2.5.1 (Object Management Group, 2017), the subset used by PAML is defined within a sibling namespace, `http://bioprotocols.org/uml/v251#`

Other namespaces are also used by PAML, however, notably the SBOL 3 namespace `http://sbols.org/v3#`, as well as other namespaces used by SBOL including Dublin Core (DCMI Usage Board, 2012)), the PROV-O ontology (`https://www.w3.org/ns/prov#`), Ontology of Units of Measure (OM), and various biological ontologies.

3.3 Primitive Data Types

When PAML uses simple "primitive" data types such as `string`s or `integer`s, these are defined as the following specific formal types:

- `string`: `http://www.w3.org/2001/XMLSchema#string`

Example: "LacI coding sequence"

- `integer`: http://www.w3.org/2001/XMLSchema#integer
  Example: 3
- `long`: http://www.w3.org/2001/XMLSchema#long
  Example: 9223372036854775806
- `float`: http://www.w3.org/2001/XMLSchema#float
  Example: 3.14159
- `boolean`: http://www.w3.org/2001/XMLSchema#boolean
  Example: `true`

The term `literal` is used to denote an object that can be any of the five types listed above.

In addition to the simple types listed above, PAML also uses objects with types *Uniform Resource Identifier* (`URI`). It is important to realize that in RDF, a `URI` might or might not be a resolvable URL (web address). A `URI` is always a globally unique identifier within a structured namespace. In some cases, that name is also a reference to (or within) a document, and in some cases that document can also be retrieved (e.g., using a web browser).

3.4 PAML Types

All PAML objects are given the most specific `rdfType` in the PAML namespace ("`http://bioprotocols.org/paml/v1#`") that defines the type of the object. Likewise, properties in the PAML namespace should only be used by objects with a PAML `rdfType`.

We should consider whether we want to maintain this, given UML's love for mix-in classes

PAML does not use multiple inheritance: all PAML classes are disjoint except with respect to their abstract parent classes. Accordingly, an object MUST NOT be given two `rdfType` properties referring to disjoint classes in the PAML namespace. An object MAY have redundant `rdfType` properties for its parent types, but this is NOT RECOMMENDED.

For example, an object cannot have both the `rdfType` of `Protocol` and `SampleCollection`. Also, a `Protocol` would have this `rdfType` and not also include `rdfType`s for classes that it inherits from, such as `uml:Activity` and `sbol:Identified`.

The same rules apply to the UML namespace, the SBOL namespace, and all unions thereof.

3.5 Object Closure and Document Composition

In RDF, there is no requirement that all of the information about an object be stored in one location. Instead, there is a "open world" assumption that additional triples describing the object may be acquired at any time. Documents are allowed to be fragmented and composed in an arbitrary manner, down to their underlying atomic triples, with no consideration for object structure.

This limits the ability to reason about properties of objects and validate the correctness of a model. For example, it would not be possible to validate that an `sbol:Identified` object has no more than one value for its `sbol:displayId` property, because it would not be possible to determine whether some other document somewhere in the world holds a second value for the property.

SBOL addresses this by adding an object closure assumption that allows stronger reasoning about individual objects and their children, and PAML adopts this convention. For any given PAML document, if the document contains an `rdfType` statement regarding an `sbol:Identified` object X, then it is assumed that the document also contains all other property statements about object X as well. This enables strong validation rules, since any statement of the form "X *predicate* Y" that is not present can be assumed to be false. For example, if a document has one value for

an object's sbol:displayId, then it is valid to conclude that there are no other sbol:displayId values, and thus its "zero or one" cardinality requirement is satisfied.

We further assume that any document containing an object also contains all of its child objects. In other words, the fundamental unit of PAML documents is the sbol:TopLevel object, and any document containing a sbol:TopLevel also contains the complete set of information necessary to describe that sbol:TopLevel—but not necessarily any other sbol:TopLevel objects that it refers to. For example, a document containing an ProtocolExecution describing the execution of a protocol is guaranteed to contain every ActivityEdgeFlow for the execution, but the document might not contain the Protocol that was executed.

A PAML document thus cleaves naturally along the boundaries of sbol:TopLevel objects, implying the following set of rules of fragmentation and composition of documents:

- Any subset of sbol:TopLevel objects in a valid PAML document is also a valid PAML document.
- Any disjoint set of sbol:TopLevel objects from different PAML documents MAY be composed to form a new PAML document. The result is not guaranteed to be valid, however, since the composition may expose problems due to the relationships between sbol:TopLevel objects from different documents.
- If two sbol:TopLevel objects in different PAML documents have the same identity and and both they and their child objects contain equivalent sets of property assertions, then they MAY be treated as identical and freely merged.
- If two sbol:TopLevel objects in different PAML documents have the same identity but different property values, then they MUST be considered different (possibly conflicting) versions, and any merger managed through some version control process.

3.6 SBOL Classes

PAML classes use sbol:Identified and sbol:TopLevel as their root classes. PAML also uses sbol:Component to describe biological materials such as strains, reagents, genetic constructs, and media. This subsection summarizes the minimum information required to use these SBOL classes in PAML; for full details, see the Synthetic Biology Open Language (SBOL) version 3 specification (Baig et al., 2020)

3.6.1 sbol:Identified

All PAML- and SBOL-defined classes are directly or indirectly derived from the sbol:Identified abstract class. This inheritance means that all PAML and SBOL objects are uniquely identified using URIs that uniquely refer to these objects within an SBOL document or at locations on the World Wide Web.

As shown in Figure 2, the sbol:Identified class includes the following properties: sbol:displayId, sbol:name, sbol:description, prov:wasDerivedFrom, and prov:wasGeneratedBy.

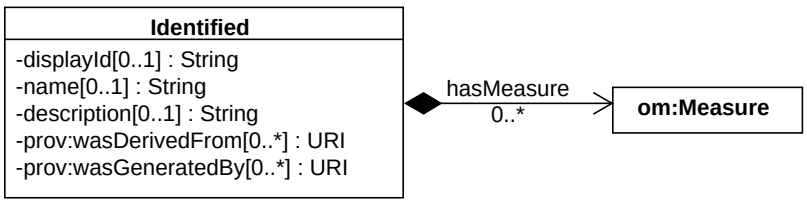

Figure 2: Diagram of the sbol:Identified abstract class and its associated properties

- The sbol:displayId property is an OPTIONAL identifier with a data type of string (and REQUIRED for objects with URL identifiers). This property is intended to be an intermediate between a URI and the sbol:name property that is machine-readable, but more human-readable than the full URI of an object. If set, its string value MUST be composed of only alphanumeric or underscore characters and MUST NOT begin with a digit.
- The sbol:name property is OPTIONAL and has a data type of string. This property is intended to be displayed to a human when visualizing an sbol:Identified object. If an sbol:Identified object lacks a name, then software tools SHOULD instead display the object's sbol:displayId or URI.
- The sbol:description property is OPTIONAL and has a data type of string. This property is intended to contain a more thorough text description of an sbol:Identified object.
- The prov:wasDerivedFrom property MAY contain any number of URIs. This property is defined by the PROV-O ontology and is located in the https://www.w3.org/ns/prov# namespace.
- The prov:wasGeneratedBy property MAY contain any number of URIs. This property is defined by the PROV-O ontology and is located in the https://www.w3.org/ns/prov# namespace.
- The sbol:hasMeasure property MAY contain any number of URIs, each of which refers to a om:Measure object that describes a measured parameter for this object.

3.6.2 sbol:TopLevel

sbol:TopLevel is an abstract class that is extended by any sbol:Identified class that can be found at the top level of a PAML or SBOL document or file. In other words, sbol:TopLevel objects are never nested inside of any other object as a child object. The sbol:TopLevel classes defined in PAML are Protocol and Primitive.

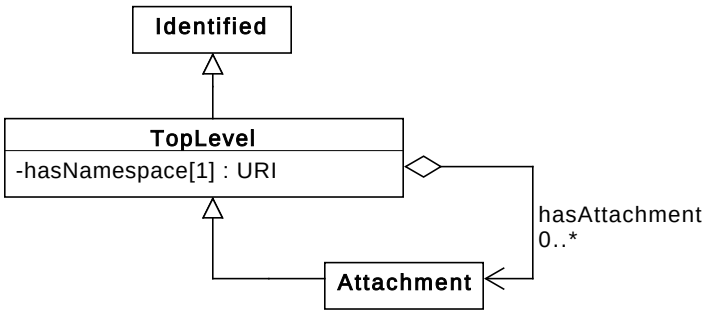

Figure 3: Classes that inherit from the sbol:TopLevel abstract class.

- The sbol:hasNamespace property is REQUIRED and MUST contain a single URI that defines the namespace portion of URLs for this object and any child objects. If the URI for the sbol:TopLevel object is a URL, then the URI of the sbol:hasNamespace property MUST prefix match that URL.
- The sbol:hasAttachment property MAY have any number of URIs, each referring to an sbol:Attachment object.

3.6.3 sbol:Component

The sbol:Component class represents the structural and/or functional entities of a biological design. In PAML, this is primarily used to represent the design of experimental samples as combinations of entities such as strains, genetic constructs, media, inducers, etc.

As shown in Figure 4, the `sbol:Component` class describes a design entity using a number of different properties. In many PAML usages, however, a `sbol:Component` will simply be used as a pointer to an external description of a material to be manipulated, and the only property required for interpreting PAML will be `sbol:type`.

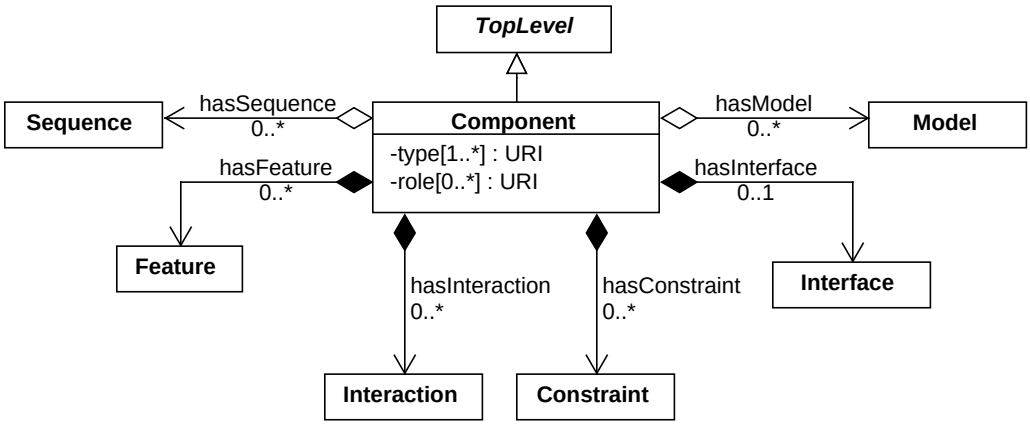

Figure 4: Diagram of the `sbol:Component` class and its associated properties.

- The `sbol:type` property MUST have one or more `URI`s specifying the category of biochemical or physical entity (for example DNA, protein, or simple chemical) that a `sbol:Component` object represents.
- The `sbol:hasFeature` property MAY have any number of `URI`s, each referencing a `sbol:Feature` object. Each `sbol:Feature` represents a specific occurrence of a part, subsystem, or other notable aspect within that design, such as an ingredient in the composition of a growth medium.
  This is not typically required for specifying protocols in PAML.
- The `sbol:role` property MAY have any number of `URI`s, which MUST identify terms from ontologies that are consistent with the `sbol:type` property of the `sbol:Component`.
  This is not typically required for specifying protocols in PAML.
- The `sbol:hasSequence` property MAY have any number of `URI`s, each referencing a `sbol:Sequence` object. These objects define the primary structure or structures of the `sbol:Component`.
  This is not typically required for specifying protocols in PAML.
- The `sbol:hasConstraint` property MAY have any number of `URI`s, each referencing a `sbol:Constraint` object. These objects describe, among other things, any restrictions on the relative, sequence-based positions and/or orientations of the `sbol:Feature` objects contained by the `sbol:Component`, as well as spatial relations such as containment and identity relations.
  This is not typically required for specifying protocols in PAML.
- The `sbol:hasInteraction` property MAY have any number of `URI`s, each referencing an `sbol:Interaction` object describing a behavioral relationship between `sbol:Feature`s in the `sbol:Component`.
  This is not typically required for specifying protocols in PAML.
- The `sbol:hasInterface` property is OPTIONAL and MAY have a `URI` referencing an `sbol:Interface` object that indicates the inputs, outputs, and non-directional points of connection to a `sbol:Component`.
  This is not typically required for specifying protocols in PAML.
- The `sbol:hasModel` property MAY have any number of `URI`s, each referencing a `sbol:Model` object that links the `sbol:Component` to a computational model in any format.
  This is not typically required for specifying protocols in PAML.

3.7 PROV-O

The PROV-O ontology (https://www.w3.org/ns/prov#) defines a complementary data model that is leveraged by PAML to describe provenance. PAML builds on PROV-O to represent the execution of Protocol workflows and the Primitive behaviors that they comprise. A subset of PROV-O, already adapted for use with SBOL, is used for this purpose by PAML as well.

The key class used is prov:Activity, which serves as a parent class for BehaviorExecution records, along with prov:Association and prov:Agent, which are used to record the entity carrying out an execution. We repeat here only the key portions referenced in this specification.

3.7.1 prov:Activity

An prov:Activity is used to describe how different prov:Agents and other entities were used. An prov:Activity is linked through a prov:qualifiedAssociation to prov:Associations, to describe the role of agents. Each prov:Activity includes optional prov:startedAtTime and prov:endedAtTime properties.

- An prov:Activity MAY have one or more type properties, each of type URI that explicitly specifies the type of the provenance prov:Activity in more detail.
- The prov:startedAtTime property is OPTIONAL and contains a DateTime value, indicating when the activity started. If this property is present, then the prov:endedAtTime property is REQUIRED.
- The prov:endedAtTime property is OPTIONAL and contains a DateTime value, indicating when the activity ended.
- An prov:Activity MAY have one or more prov:qualifiedAssociation properties, each of type sbol:URI that refers to an prov:Association object.

3.7.2 prov:Association

An prov:Association is linked to an prov:Agent through the prov:agent relationship. The prov:Association includes the prov:hadRole property to qualify the role of the prov:Agent in the prov:Activity.

- The prov:agent property is REQUIRED and MUST contain a URI that refers to an prov:Agent object.
- An prov:Association MAY have one or more prov:hadRole properties, each of type URI that refers to particular term(s) that describes the role of the prov:Agent in the parent prov:Activity.

3.7.3 prov:Agent

Examples of agents are a person, organization, or software tool. These agents should be annotated with additional information, such as software version, needed to be able to run the same prov:Activity again.

3.8 Ontology of Units of Measure

In most cases where a number is needed in PAML, that number is a measure with units associated with it. The Ontology of Units of Measure (OM) (http://www.ontology-of-units-of-measure.org/resource/om-2) already defines a data model for representing measures and their associated units. A subset of OM, already adapted for use with SBOL, is used for this purpose by PAML as well.

The key class used is om:Measure, which associates a number with a unit and a biology-related property. In most cases, it should be possible to use one of the om:Unit instances already defined by OM; when this is not possible, an appropriate unit can be defined using om:Unit and om:Prefix classes.

3.8.1 om:Measure

The purpose of the om:Measure class is to link a numerical value to a om:Unit.

- The om:hasNumericalValue property is REQUIRED and MUST contain a single float.
- The om:hasUnit property is REQUIRED and MUST contain a URI that refers to a om:Unit.
- The sbol:type property MAY contain any number of URIs. It is RECOMMENDED that one of these URIs identify a term from the Systems Description Parameter branch of the Systems Biology Ontology (SBO) (http://www.ebi.ac.uk/sbo/main/). This sbol:type property was added by SBOL to describe different types of parameters (for example, rate of reaction is identified by the SBO term http://identifiers.org/SBO:0000612).

3.9 Recommended Ontologies for External Terms

External ontologies and controlled vocabularies are an integral part of SBOL and thus used by PAML as well. SBOL uses URIs to access existing biological information through these resources. Although RECOMMENDED ontologies have been indicated in relevant sections where possible, other resources providing similar terms can also be used. A summary of these external sources can be found in Table 1. The URIs for ontological terms SHOULD come from identifiers.org. However, it is acceptable to use terms from purl.org as an alternative, for example when RDF tooling requires URIs to be represented as compliant QNames, and software may convert between these forms as required.

SBOL Entity	Property	Preferred External Resource	More Information
Component	type	SBO (physical entity branch)	http://www.ebi.ac.uk/sbo/main/
	type	SO (nucleic acid topology)	http://www.sequenceontology.org
	role	SO (*DNA* or *RNA*)	http://www.sequenceontology.org
	role	CHEBI (*small molecule*)	https://www.ebi.ac.uk/chebi/
	role	PubChem (*small molecule*)	https://pubchem.ncbi.nlm.nih.gov/
	role	UniProt (*protein*)	https://www.uniprot.org/
	role	NCIT (*samples*)	https://ncithesaurus.nci.nih.gov/
om:Measure	type	SBO (systems description parameters)	http://www.ebi.ac.uk/sbo/main/

Table 1: Preferred external resources from which to draw values for various SBOL properties.

4 Imported UML Classes

PAML builds on the Unified Modeling Language (UML) version 2.5.1 (Object Management Group, 2017) to describe the organization of actions in a protocol. In order to make this document more self-contained, this section describes the classes from UML that have been imported for use with PAML. In particular, each such class has been adapted to be an SBOL subclass, assigned to either `sbol:Identified` or `sbol:TopLevel`, in order to be able to be used with SBOL closure assumptions in an RDF environment.

4.1 ValueSpecification

A `uml:ValueSpecification` is the specification of a (possibly empty) set of values. See UML 2.5.1 specification section 8.

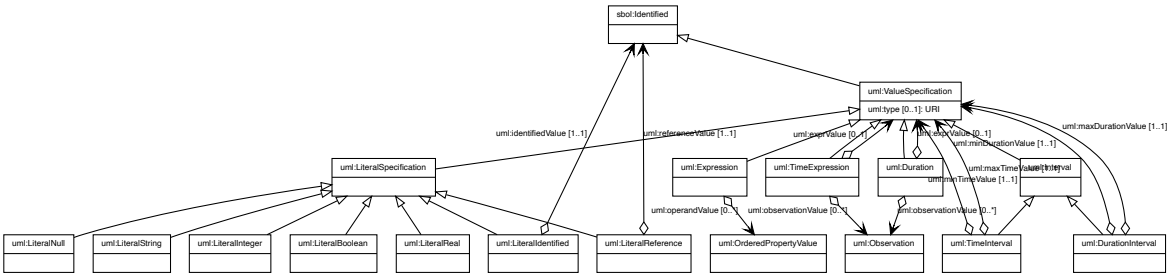

Figure 5: ValueSpecification

The `uml:ValueSpecification` class is shown in Figure 5. It is derived from `sbol:Identified` and includes the following specializations: `uml:LiteralSpecification`, `uml:Expression`, `uml:TimeExpression`, `uml:Duration`, `uml:Interval`. This class includes the following properties: `uml:type`.

- The `uml:type` property is OPTIONAL and has a singleton value of type URISpecifies a set of Type instances constraining the allowed values. See UML 2.5.1 specification section 7.5.

4.1.1 LiteralSpecification

A `uml:LiteralSpecification` identifies a literal constant being modeled. See UML 2.5.1 specification section 8.2.

The `uml:LiteralSpecification` class is shown in Figure 5. It is derived from `uml:ValueSpecification` and includes the following specializations: `uml:LiteralNull`, `uml:LiteralString`, `uml:LiteralInteger`, `uml:LiteralBoolean`, `uml:LiteralReal`, `uml:LiteralIdentified`, `uml:LiteralReference`.

4.1.1.1 LiteralNull

A `uml:LiteralNull` specifies the lack of a value. See UML 2.5.1 specification section 8.2.

The `uml:LiteralNull` class is shown in Figure 5. It is derived from `uml:LiteralSpecification`.

4.1.1.2 LiteralString

A uml:LiteralSpecification identifies a literal constant being modeled. See UML 2.5.1 specification section 8.2.

The uml:LiteralString class is shown in Figure 5. It is derived from uml:LiteralSpecification.This class includes the following properties: uml:stringValue.

- The uml:stringValue property is REQUIRED and has a singleton value of type stringThe specified String value.

4.1.1.3 LiteralInteger

A uml:LiteralInteger is a specification of an Integer value. See UML 2.5.1 specification section 8.2.

The uml:LiteralInteger class is shown in Figure 5. It is derived from uml:LiteralSpecification.This class includes the following properties: uml:integerValue.

- The uml:integerValue property is REQUIRED and has a singleton value of type integerThe specified Integer value.

4.1.1.4 LiteralBoolean

A uml:LiteralBoolean is a specification of a Boolean value. See UML 2.5.1 specification section 8.2.

The uml:LiteralBoolean class is shown in Figure 5. It is derived from uml:LiteralSpecification.This class includes the following properties: uml:booleanValue.

- The uml:booleanValue property is REQUIRED and has a singleton value of type booleanThe specified Boolean value.

4.1.1.5 LiteralReal

A uml:LiteralReal is a specification of a Real value. See UML 2.5.1 specification section 8.2.

The uml:LiteralReal class is shown in Figure 5. It is derived from uml:LiteralSpecification.This class includes the following properties: uml:realValue.

- The uml:realValue property is REQUIRED and has a singleton value of type floatThe specified Real value.

4.1.1.6 LiteralIdentified

A uml:LiteralIdentified is used for linking SBOL objects as a child object to UML objects.

The uml:LiteralIdentified class is shown in Figure 5. It is derived from uml:LiteralSpecification.This class includes the following properties: uml:identifiedValue.

- The uml:identifiedValue property is REQUIRED and contains a URI reference to an associated object of type IdentifiedThe embedded SBOL object

4.1.1.7 LiteralReference

A uml:LiteralReference is used for embedding SBOL objects as a reference.

The uml:LiteralReference class is shown in Figure 5. It is derived from uml:LiteralSpecification.This class includes the following properties: uml:referenceValue.

- The `uml:referenceValue` property is REQUIRED and contains a URI reference to an associated object of type IdentifiedThe referenced SBOL object.

4.1.2 Expression

An `uml:Expression` represents a node in an expression tree, which may be non-terminal or terminal. It defines a symbol, and has a possibly empty sequence of operands that are ValueSpecifications. See UML 2.5.1 specification section 8.6.5.1.

The `uml:Expression` class is shown in Figure 5. It is derived from `uml:ValueSpecification`.This class includes the following properties: `uml:isOrdered`, `uml:symbolValue`, `uml:operandValue`.

- The `uml:operandValue` property is OPTIONAL and contains URI references to associated objects of type OrderedPropertyValueSpecifies a sequence of operand ValueSpecifications.
- The `uml:isOrdered` property is REQUIRED and has a singleton value of type booleanFor MultiplicityElement abstract class; UML 2.5.1 specification section 7.5
- The `uml:symbolValue` property is OPTIONAL and has a singleton value of type stringThe symbol associated with this node in the expression tree.

4.1.3 TimeExpression

A `uml:TimeExpression` is a `uml:ValueSpecification` that represents a time value.

The `uml:TimeExpression` class is shown in Figure 5. It is derived from `uml:ValueSpecification`.This class includes the following properties: `uml:observationValue`, `uml:exprValue`.

- The `uml:observationValue` property is OPTIONAL and contains URI references to associated objects of type ObservationRefers to the Observations that are involved in the computation of the Duration value.
- The `uml:exprValue` property is OPTIONAL and contains a URI reference to an associated object of type ValueSpecificationA ValueSpecification that evaluates to the value of the Duration.

4.1.4 Duration

A `uml:Duration` is a `uml:ValueSpecification` that specifies the temporal distance between two time instants.

The `uml:Duration` class is shown in Figure 5. It is derived from `uml:ValueSpecification`.This class includes the following properties: `uml:observationValue`, `uml:exprValue`.

- The `uml:observationValue` property is OPTIONAL and contains URI references to associated objects of type ObservationRefers to the Observations that are involved in the computation of the Duration value.
- The `uml:exprValue` property is OPTIONAL and contains a URI reference to an associated object of type ValueSpecificationA ValueSpecification that evaluates to the value of the Duration.

4.1.5 Interval

An `uml:Interval` defines the range between two ValueSpecifications.

The `uml:Interval` class is shown in Figure 5. It is derived from `uml:ValueSpecification` and includes the following specializations: `uml:TimeInterval`, `uml:DurationInterval`.

4.1.5.1 TimeInterval

A uml:TimeInterval defines the range between two TimeExpressions.

The uml:TimeInterval class is shown in Figure 5. It is derived from uml:Interval.This class includes the following properties: uml:minTimeValue, uml:maxTimeValue.

- The uml:minTimeValue property is REQUIRED and contains a URI reference to an associated object of type ValueSpecificationRefers to the TimeExpression denoting the minimum value of the range.
- The uml:maxTimeValue property is REQUIRED and contains a URI reference to an associated object of type ValueSpecificationRefers to the TimeExpression denoting the maximum value of the range.

4.1.5.2 DurationInterval

A uml:DurationInterval defines the range between two Durations.

The uml:DurationInterval class is shown in Figure 5. It is derived from uml:Interval.This class includes the following properties: uml:minDurationValue, uml:maxDurationValue.

- The uml:minDurationValue property is REQUIRED and contains a URI reference to an associated object of type ValueSpecificationRefers to the Duration denoting the minimum value of the range.
- The uml:maxDurationValue property is REQUIRED and contains a URI reference to an associated object of type ValueSpecificationRefers to the Duration denoting the maximum value of the range.

4.2 Observation

Observation specifies a value determined by observing an event or events that occur relative to other model entities.

The uml:Observation class is shown in Figure 6. It is derived from sbol:Identified and includes the following specializations: uml:TimeObservation, uml:DurationObservation.

4.2.1 TimeObservation

A uml:TimeObservation is a reference to a time instant during an execution. It points out which entity in the model to observe and whether the observation is when this entity is entered or when it is exited.

The uml:TimeObservation class is shown in Figure 6. It is derived from uml:Observation.This class includes the following properties: uml:timeObservationValue, uml:firstEventValue.

- The uml:timeObservationValue property is REQUIRED and contains a URI reference to an associated object of type IdentifiedThe TimeObservation is determined by the entering or exiting of the event during execution.
- The uml:firstEventValue property is REQUIRED and contains a URI reference to an associated object of type OrderedPropertyValueThe value of firstEvent[i] is related to event[i] (where i is 1 or 2). If firstEvent[i] is true, then the correspondingobservation event is the first time instant the execution enters event[i]. If firstEvent[i] is false, then the corresponding observation event is the time instant the execution exits event[i].

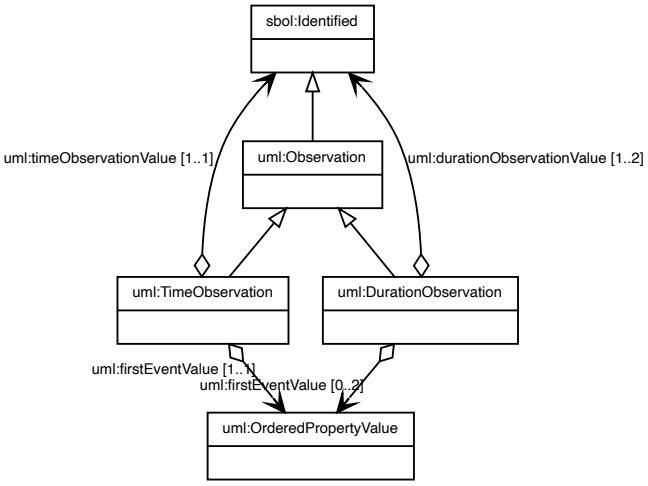

Figure 6: Observation

4.2.2 DurationObservation

A `uml:DurationObservation` is a reference to a duration during an execution. It points out the entities in the model to observe and whether the observations are when these entities are entered or exited.

The `uml:DurationObservation` class is shown in Figure 6. It is derived from `uml:Observation`.This class includes the following properties: `uml:durationObservationValue`, `uml:firstEventValue`.

- The `uml:durationObservationValue` property is REQUIRED and contains URI references to associated objects of type IdentifiedThe DurationObservation is determined as the duration between the entering or exiting of a single event during execution, or the entering/exiting of one event and the entering/exiting of a second.
- The `uml:firstEventValue` property is OPTIONAL and contains URI references to associated objects of type OrderedPropertyValueThe value of firstEvent[i] is related to event[i] (where i is 1 or 2). If firstEvent[i] is true, then the correspondingobservation event is the first time instant the execution enters event[i]. If firstEvent[i] is false, then the corresponding observation event is the time instant the execution exits event[i].

4.3 Constraint

A `sbol:Constraint` is a condition or restriction expressed in natural language text or in a machine readable language. See UML 2.5.1 specification section 7.6.

The `uml:Constraint` class is shown in Figure 7. It is derived from `sbol:Identified` and includes the following specializations: `uml:IntervalConstraint`. This class includes the following properties: `uml:constrainedElement`.

- The `uml:constrainedElement` property is OPTIONAL and contains URI references to associated objects of type OrderedPropertyValueThe OrderedPropertyValue referenced by this Constraint.

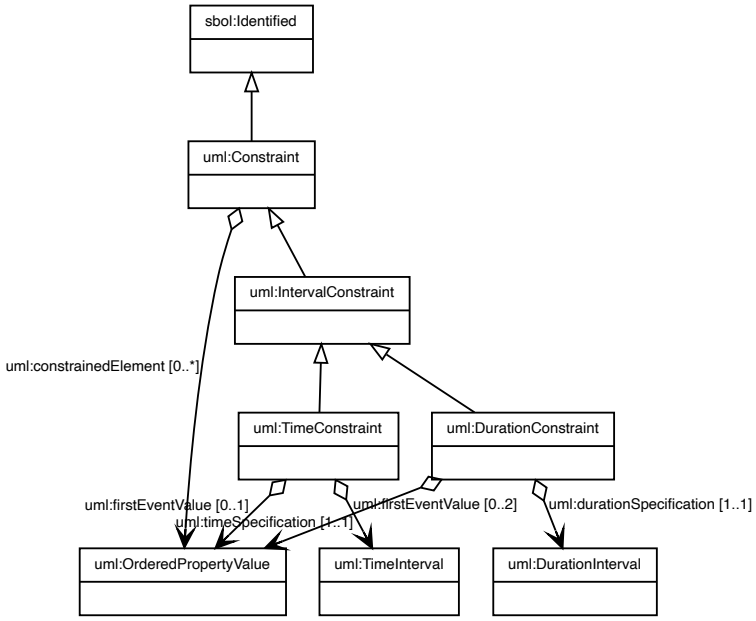

Figure 7: Constraint

4.3.1 IntervalConstraint

An `uml:IntervalConstraint` is a `sbol:Constraint` that is specified by an `uml:Interval`.

The `uml:IntervalConstraint` class is shown in Figure 7. It is derived from `uml:Constraint` and includes the following specializations: `uml:TimeConstraint`, `uml:DurationConstraint`.

4.3.1.1 TimeConstraint

A `uml:TimeConstraint` is a `sbol:Constraint` that refers to a `uml:TimeInterval`.

The `uml:TimeConstraint` class is shown in Figure 7. It is derived from `uml:IntervalConstraint`.This class includes the following properties: `uml:timeSpecification`, `uml:firstEventValue`.

- The `uml:timeSpecification` property is REQUIRED and contains a URI reference to an associated object of type TimeIntervalTheTimeInterval constraining the duration.
- The `uml:firstEventValue` property is OPTIONAL and contains a URI reference to an associated object of type OrderedPropertyValueThe value of firstEvent[i] is related to event[i] (where i is 1 or 2). If firstEvent[i] is true, then the correspondingobservation event is the first time instant the execution enters event[i]. If firstEvent[i] is false, then the corresponding observation event is the time instant the execution exits event[i].

4.3.1.2 DurationConstraint

A `uml:DurationConstraint` is a `sbol:Constraint` that refers to a `uml:DurationInterval`.

The `uml:DurationConstraint` class is shown in Figure 7. It is derived from `uml:IntervalConstraint`.This class

includes the following properties: `uml:durationSpecification`, `uml:firstEventValue`.

- The `uml:durationSpecification` property is REQUIRED and contains a URI reference to an associated object of type DurationIntervalThe DurationInterval constraining the duration.
- The `uml:firstEventValue` property is OPTIONAL and contains URI references to associated objects of type OrderedPropertyValueThe value of firstEvent[i] is related to event[i] (where i is 1 or 2). If firstEvent[i] is true, then the correspondingobservation event is the first time instant the execution enters event[i]. If firstEvent[i] is false, then the corresponding observation event is the time instant the execution exits event[i].

4.4 Parameter

A `uml:Parameter` is a specification of an argument used to pass information into or out of an invocation of a `uml:Behavior`. See UML 2.5.1 specification section 9.4.

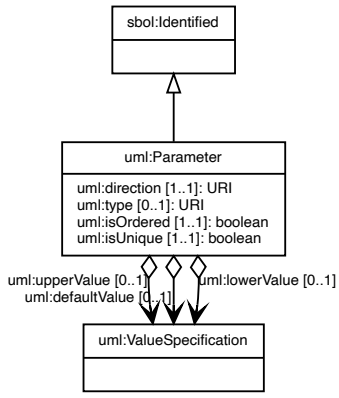

Figure 8: Parameter

The `uml:Parameter` class is shown in Figure 8. It is derived from `sbol:Identified`.This class includes the following properties: `uml:direction`, `uml:type`, `uml:isOrdered`, `uml:isUnique`, `uml:upperValue`, `uml:defaultValue`, `uml:lowerValue`.

- The `uml:upperValue` property is OPTIONAL and contains a URI reference to an associated object of type ValueSpecificationFor MultiplicityElement abstract class; UML 2.5.1 specification section 7.5
- The `uml:defaultValue` property is OPTIONAL and contains a URI reference to an associated object of type ValueSpecificationA ValueSpecification that represents a value to be used when no argument is supplied for the Parameter.
- The `uml:lowerValue` property is OPTIONAL and contains a URI reference to an associated object of type ValueSpecificationFor MultiplicityElement abstract class; UML 2.5.1 specification section 7.5
- The `uml:direction` property is REQUIRED and has a singleton value of type URIIndicates whether a parameter is being sent into or out of a behavioral element.

- The `uml:type` property is OPTIONAL and has a singleton value of type URISpecifies a set of Type instances constraining the allowed values. See UML 2.5.1 specification section 7.5.
- The `uml:isOrdered` property is REQUIRED and has a singleton value of type booleanFor MultiplicityElement abstract class; UML 2.5.1 specification section 7.5
- The `uml:isUnique` property is REQUIRED and has a singleton value of type booleanFor MultiplicityElement abstract class; UML 2.5.1 specification section 7.5

4.5 Behavior

Behavior is an abstract specification of how a state changes over time. This specification may be a prospective definition of a protocol or a capture of an execution trace. See UML 2.5.1 specification section 13.2.

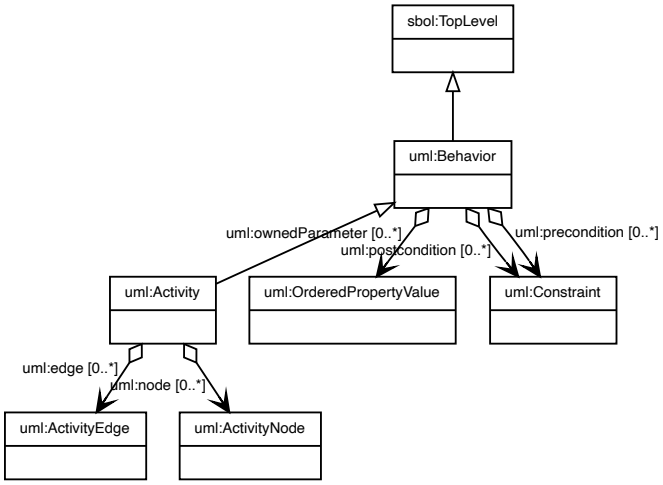

Figure 9: Behavior

The `uml:Behavior` class is shown in Figure 9. It is derived from `sbol:TopLevel` and includes the following specializations: `uml:Activity`. This class includes the following properties: `uml:ownedParameter`, `uml:postcondition`, `uml:precondition`.

- The `uml:ownedParameter` property is OPTIONAL and contains URI references to associated objects of type OrderedPropertyValue a list of Parameters to the Behavior which describes the order and type of arguments that can be given when the Behavior is invoked and of the values which will be returned when the Behavior completes its execution.
- The `uml:postcondition` property is OPTIONAL and contains URI references to associated objects of type ConstraintAn optional set of Constraints specifying what is fulfilled after the execution of the Behavior is completed, if its precondition was fulfilled before its invocation.
- The `uml:precondition` property is OPTIONAL and contains URI references to associated objects of type ConstraintAn optional set of Constraints specifying what must be fulfilled before the Behavior is invoked.

4.5.1 Activity

An prov:Activity coordinates and groups steps in a protocol or workflow. See UML 2.5.1 specification section 15.

The uml:Activity class is shown in Figure 9. It is derived from uml:Behavior.This class includes the following properties: uml:edge, uml:node.

- The uml:edge property is OPTIONAL and contains URI references to associated objects of type ActivityEdge-ActivityEdges expressing flow between the nodes of the Activity.
- The uml:node property is OPTIONAL and contains URI references to associated objects of type ActivityNode-ActivityNodes coordinated by the Activity.

4.6 OrderedPropertyValue

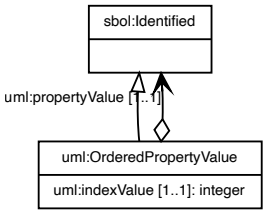

Figure 10: OrderedPropertyValue

The uml:OrderedPropertyValue class is shown in Figure 10. It is derived from sbol:Identified.This class includes the following properties: uml:indexValue, uml:propertyValue.

- The uml:propertyValue property is REQUIRED and contains a URI reference to an associated object of type Identified
- The uml:indexValue property is REQUIRED and has a singleton value of type integer

4.7 ActivityNode

ActivityNode is an abstract class for points in the flow of an prov:Activity connected by ActivityEdges. See UML 2.5.1 specification section 15.2.

The uml:ActivityNode class is shown in Figure 11. It is derived from sbol:Identified and includes the following specializations: uml:ControlNode, uml:ObjectNode, uml:ExecutableNode.

4.7.1 ControlNode

A uml:ControlNode is a kind of uml:ActivityNode used to manage the flow of tokens between other nodes in an prov:Activity. It can manage branching and merging of workflows and the implementation of logic for flow control. See UML 2.5.1 specification section 15.3.

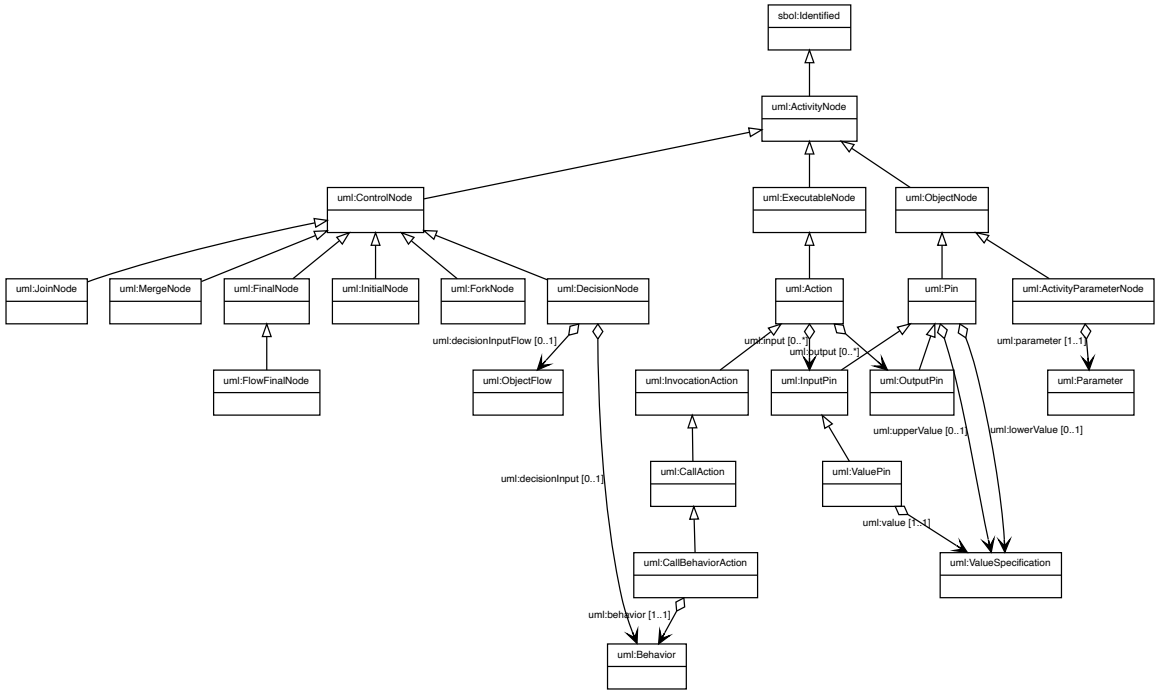

Figure 11: ActivityNode

The uml:ControlNode class is shown in Figure 11. It is derived from uml:ActivityNode and includes the following specializations: uml:InitialNode, uml:FinalNode, uml:ForkNode, uml:JoinNode, uml:MergeNode, uml:DecisionNode.

4.7.1.1 InitialNode

An uml:InitialNode acts as a starting point for executing an prov:Activity. An prov:Activity may have more than sbol:one InitialNodes that start multiple concurrent control flows. See UML 2.5.1 specification section 15.3.

The uml:InitialNode class is shown in Figure 11. It is derived from uml:ControlNode.

4.7.1.2 FinalNode

A uml:FinalNode is a uml:ControlNode at which a flow in an prov:Activity stops. A uml:FinalNode shall not have outgoing ActivityEdges. See UML 2.5.1 specification section 15.3.

The uml:FinalNode class is shown in Figure 11. It is derived from uml:ControlNode and includes the following specializations: uml:FlowFinalNode.

4.7.1.2.1 FlowFinalNode

A uml:FlowFinalNode is a uml:FinalNode that terminates a flow. All tokens accepted by a uml:FlowFinalNode are destroyed. This has no effect on other flows in the prov:Activity. See UML 2.5.1 specification section 15.3.

The uml:FlowFinalNode class is shown in Figure 11. It is derived from uml:FinalNode.

4.7.1.3 ForkNode

A uml:ForkNode is a uml:ControlNode that splits a flow into multiple concurrent flows. A uml:ForkNode shall have exactly sbol:one incoming uml:ActivityEdge, though it may have multiple outgoing ActivityEdges. See UML 2.5.1 specification section 15.3.

The uml:ForkNode class is shown in Figure 11. It is derived from uml:ControlNode.

4.7.1.4 JoinNode

A uml:JoinNode is a uml:ControlNode that synchronizes multiple flows. A uml:JoinNode shall have exactly sbol:one outgoing uml:ActivityEdge but may have multiple incoming ActivityEdges. See UML 2.5.1 specification section 15.3.

The uml:JoinNode class is shown in Figure 11. It is derived from uml:ControlNode.

4.7.1.5 MergeNode

A uml:MergeNode is a control node that brings together multiple flows without synchronization. A uml:MergeNode shall have exactly sbol:one outgoing uml:ActivityEdge but may have multiple incoming ActivityEdges. See UML 2.5.1 specification section 15.3.

The uml:MergeNode class is shown in Figure 11. It is derived from uml:ControlNode.

4.7.1.6 DecisionNode

A uml:DecisionNode is a uml:ControlNode that chooses between outgoing flows. A uml:DecisionNode shall have at least sbol:one and at most two incoming ActivityEdges, and at least sbol:one outgoing uml:ActivityEdge. See UML 2.5.1 specification section 15.3.

The uml:DecisionNode class is shown in Figure 11. It is derived from uml:ControlNode.This class includes the following properties: uml:decisionInput, uml:decisionInputFlow.

- The uml:decisionInput property is OPTIONAL and contains a URI reference to an associated object of type BehaviorA Behavior that is executed to provide an input to guard ValueSpecifications on ActivityEdges outgoing from the DecisionNode.
- The uml:decisionInputFlow property is OPTIONAL and contains a URI reference to an associated object of type ObjectFlowAn additional ActivityEdge incoming to the DecisionNode that provides a decision input value for the guards ValueSpecifications on ActivityEdges outgoing from the DecisionNode.

4.7.2 ObjectNode

An uml:ObjectNode is a kind of uml:ActivityNode used to hold value-containing object tokens during the course of the execution of an prov:Activity. See UML 2.5.1 specification section 15.4.

The uml:ObjectNode class is shown in Figure 11. It is derived from uml:ActivityNode and includes the following specializations: uml:ActivityParameterNode, uml:Pin. This class includes the following properties: uml:type.

- The uml:type property is OPTIONAL and has a singleton value of type URISpecifies a set of Type instances constraining the allowed values. See UML 2.5.1 specification section 7.5.

4.7.2.1 ActivityParameterNode

An uml:ActivityParameterNode is an uml:ObjectNode for accepting values from the input Parameters or providing values to the output Parameters of an prov:Activity. UML 2.5.1 specification section 15.4.

The uml:ActivityParameterNode class is shown in Figure 11. It is derived from uml:ObjectNode.This class includes the following properties: uml:parameter.

- The uml:parameter property is REQUIRED and contains a URI reference to an associated object of type ParameterThe Parameter for which the ActivityParameterNode will be accepting or providing values.

4.7.2.2 Pin

A uml:Pin is an uml:ObjectNode and MultiplicityElement that provides input values to an uml:Action or accepts output values from an uml:Action. See UML 2.5.1 specification section 16.2.

The uml:Pin class is shown in Figure 11. It is derived from uml:ObjectNode and includes the following specializations: uml:InputPin, uml:OutputPin. This class includes the following properties: uml:isOrdered, uml:isUnique, uml:upperValue, uml:lowerValue.

- The uml:upperValue property is OPTIONAL and contains a URI reference to an associated object of type ValueSpecificationFor MultiplicityElement abstract class; UML 2.5.1 specification section 7.5
- The uml:lowerValue property is OPTIONAL and contains a URI reference to an associated object of type ValueSpecificationFor MultiplicityElement abstract class; UML 2.5.1 specification section 7.5
- The uml:isOrdered property is REQUIRED and has a singleton value of type booleanFor MultiplicityElement abstract class; UML 2.5.1 specification section 7.5
- The uml:isUnique property is REQUIRED and has a singleton value of type booleanFor MultiplicityElement abstract class; UML 2.5.1 specification section 7.5

4.7.2.2.1 InputPin

An uml:InputPin is a uml:Pin that holds input values to be consumed by an uml:Action. See UML 2.5.1 specification section 16.2.

The uml:InputPin class is shown in Figure 11. It is derived from uml:Pin and includes the following specializations: uml:ValuePin.

4.7.2.2.2 ValuePin

A uml:ValuePin is an uml:InputPin that provides a value by evaluating a uml:ValueSpecification. See UML 2.5.1 specification section 16.2.

The uml:ValuePin class is shown in Figure 11. It is derived from uml:InputPin.This class includes the following properties: uml:value.

- The uml:value property is REQUIRED and contains a URI reference to an associated object of type ValueSpecificationThe ValueSpecification that is evaluated to obtain the value that the ValuePin will provide.

4.7.2.2.3 OutputPin

An uml:OutputPin is a uml:Pin that holds output values produced by an uml:Action. See UML 2.5.1 specification section 16.2.

The uml:OutputPin class is shown in Figure 11. It is derived from uml:Pin.

4.7.3 ExecutableNode

An uml:ExecutableNode is an abstract class for ActivityNodes whose execution may be controlled using ControlFlows and to which ExceptionHandlers may be attached. See UML 2.5.1 specification section 15.5.

The uml:ExecutableNode class is shown in Figure 11. It is derived from uml:ActivityNode and includes the following specializations: uml:Action.

4.7.3.1 Action

An uml:Action is the fundamental unit of executable functionality. The execution of an uml:Action represents some transformation or processing in the modeled system. Actions provide the ExecutableNodes within Activities and may also be used within Interactions. See UML 2.5.1 specification section 16.

The uml:Action class is shown in Figure 11. It is derived from uml:ExecutableNode and includes the following specializations: uml:InvocationAction. This class includes the following properties: uml:output, uml:input.

- The uml:output property is OPTIONAL and contains URI references to associated objects of type OutputPinThe ordered set of OutputPins representing outputs from the Action.
- The uml:input property is OPTIONAL and contains URI references to associated objects of type InputPinThe ordered set of InputPins representing the inputs to the Action.

4.7.3.1.1 InvocationAction

UML 2.5.1 specification section 16.3

The uml:InvocationAction class is shown in Figure 11. It is derived from uml:Action and includes the following specializations: uml:CallAction.

4.7.3.1.2 CallAction

A uml:CallAction is an abstract class for Actions that invoke a uml:Behavior with given argument values and (if the invocation is synchronous) receive reply values. See UML 2.5.1 specification section 16.3.

The uml:CallAction class is shown in Figure 11. It is derived from uml:InvocationAction and includes the following specializations: uml:CallBehaviorAction.

4.7.3.1.3 CallBehaviorAction

A uml:CallBehaviorAction is a uml:CallAction that invokes a uml:Behavior directly. The argument values of the uml:CallBehaviorAction are passed on the input Parameters of the invoked uml:Behavior. UML 2.5.1 specification section 16.3

The uml:CallBehaviorAction class is shown in Figure 11. It is derived from uml:CallAction.This class includes the following properties: uml:behavior.

- The uml:behavior property is REQUIRED and contains a URI reference to an associated object of type BehaviorThe Behavior being invoked.

4.8 ActivityEdge

An uml:ActivityEdge is an abstract class for directed connections between two ActivityNodes. See UML 2.5.1 specification section 15.2.

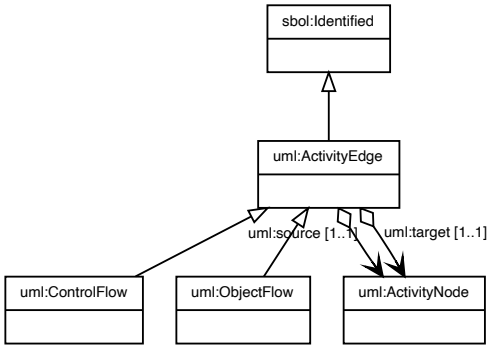

Figure 12: ActivityEdge

The `uml:ActivityEdge` class is shown in Figure 12. It is derived from `sbol:Identified` and includes the following specializations: `uml:ControlFlow`, `uml:ObjectFlow`. This class includes the following properties: `uml:source`, `uml:target`.

- The `uml:source` property is REQUIRED and contains a URI reference to an associated object of type ActivityNodeThe ActivityNode from which tokens are taken when they traverse the ActivityEdge.
- The `uml:target` property is REQUIRED and contains a URI reference to an associated object of type ActivityNodeThe ActivityNode to which tokens are put when they traverse the ActivityEdge.

4.8.1 ControlFlow

A `uml:ControlFlow` is an `uml:ActivityEdge` traversed by control tokens or object tokens of control type, which are use to control the execution of ExecutableNodes. See UML 2.5.1 specification section 15.2.

The `uml:ControlFlow` class is shown in Figure 12. It is derived from `uml:ActivityEdge`.

4.8.2 ObjectFlow

An `uml:ObjectFlow` is an `uml:ActivityEdge` that is traversed by object tokens that may hold values. Object flows also support multicast/receive, token selection from object nodes, and transformation of tokens. See UML 2.5.1 specification section 15.2.

The `uml:ObjectFlow` class is shown in Figure 12. It is derived from `uml:ActivityEdge`.

5 PAML Data Model

5.1 Protocol

A `Protocol` describes how to carry out some form of laboratory or research process. For example, a `Protocol` could describe DNA miniprep, Golden-Gate assembly, a cell culture experiment. At present this class adds no additional information over `uml:Activity`, but may in the future.

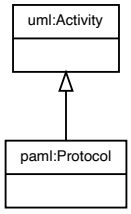

Figure 13: Protocol

The `Protocol` class is shown in Figure 13. It is derived from `uml:Activity`.

5.2 Primitive

A `Primitive` describes a library function that acts as a basic "building block" for a `Protocol`. For example, a `Primitive` could describe pipetting, measuring absorbance in a plate reader, or centrifuging. At present this class adds no additional information over `uml:Behavior`, but may in the future.

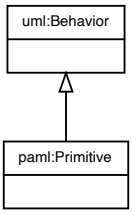

Figure 14: Primitive

The `Primitive` class is shown in Figure 14. It is derived from `uml:Behavior`.

5.3 BehaviorExecution

A BehaviorExecution is a record of how a Protocol, Primitive, or other uml:Behavior was carried out. The execution of the behavior could be either real or simulated.

In specifying a BehaviorExecution, the prov:type field inherited from prov:Activity is used to indicate the uml:Behavior whose execution is being recorded. Precisely sbol:one value of prov:type MUST be a URI for a uml:Behavior. The prov:startedAtTime and prov:endedAtTime fields SHOULD be used to record timing information as this becomes available. Finally, the entity carrying out the execution SHOULD be recorded as a prov:Agent indicated using a prov:Association.

Note that a BehaviorExecution can be used to record both the state of an in-progress execution as well as an execution that has completed. As a BehaviorExecution proceeds, all values of its properties are monotonic, i.e., they are only added to and never changed.

TODO: need to changing completedNormally to allow indication of an in-progress BehaviorExecution TODO: Is there a good ontology for agent roles in association?

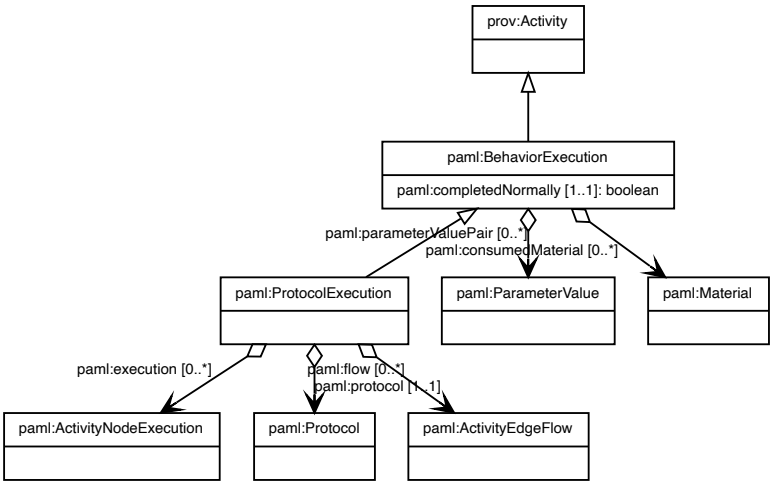

Figure 15: BehaviorExecution

The BehaviorExecution class is shown in Figure 15. It is derived from prov:Activity and includes the following specializations: ProtocolExecution. This class includes the following properties: completedNormally, parameterValuePair, consumedMaterial.

- The parameterValuePair property is OPTIONAL and contains URI references to associated objects of type ParameterValueThe parameterValuePair property is used to record the value that was associated with each uml:Parameter for the uml:Behavior when it was executed, by means of a ParameterValue object. Any uml:Parameter that is not listed is assumed to have had no value assigned. Conversely, every non-optional uml:Parameter for the uml:Behavior MUST have an associated parameter value. Finally, note that this applies both to input uml:Parameter objects, whose value is set before execution begins, and to output uml:Parameter objects, whose value is set by the time execution ends.

TODO: are multiple values allowed, or do those need to be passed as list/set types?

- The `consumedMaterial` property is OPTIONAL and contains URI references to associated objects of type MaterialThis property is used to record the noteworthy consumables used during the execution of the Behavior. For example, a cell culture protocols will consume various reagents and samples of cells. Materials with the same specification SHOULD be consolidated, such that the list of materials SHOULD NOT contain two materials with the same specification.

  For example, consuming 5.0 mL of PBS and 2.0 mL of PBS should be recorded as consuming 7.0 mL of PBS. Complex materials, however, MAY contain the same material more than once in their substructure. For example, M9 media contains glucose, but it would not be necessary to consolidate the glucose in M9 media with additional glucose that was added as a supplement, since that would change the definition of the media.

- The `completedNormally` property is REQUIRED and has a singleton value of type booleanThis boolean should be set to true if the Behavior completed normally and false if there was some exception condition. At present, no further information is being encoded about exceptions, but this is an extension that is anticipated for the future.

5.3.1 ProtocolExecution

A `ProtocolExecution` expands on the information in a `BehaviorExecution` by including records for the nodes and edges defining the Protocol's behavior as a `uml:Activity`. Specifically, the execution property is used to record each firing of a `uml:ActivityNode` and the flow property is used to record each time a token moves along a `uml:ActivityEdge`. Otherwise, a `ProtocolExecution` is used exactly the same way as its parent class `BehaviorExecution`.

TODO: consider dropping the protocol field as redundant with use prov:type field in its parent

The `ProtocolExecution` class is shown in Figure 15. It is derived from `BehaviorExecution`.This class includes the following properties: `execution`, `protocol`, `flow`.

- The `execution` property is OPTIONAL and contains URI references to associated objects of type ActivityNodeExecutionEach instance of this property links to an ActivityNodeExecution that records one firing of a uml:ActivityNode during the execution of its containing Protocol

- The `protocol` property is REQUIRED and contains a URI reference to an associated object of type ProtocolThis property appears to be redundant with the use of prov:type specified by BehaviorExecution, and is likely to be deleted

- The `flow` property is OPTIONAL and contains URI references to associated objects of type ActivityEdgeFlowEach instance of this property links to an ActivityEdgeFlow that records one movement of a UML token along a uml:ActivityEdge during the execution of its containing Protocol

5.4 ParameterValue

This class is used to represent the assignment of a value to a parameter in a BehaviorExecution that records the execution of a `uml:Behavior`. This class is similar to `prov:Usage`, but instead of always pointing to an object it uses an arbitrary literal (which might or might not be an object). An example would be recording that a plate reader absorbance measurement was taken with its absorbance wavelength parameter set to 600 nm

The `ParameterValue` class is shown in Figure 16. It is derived from `sbol:Identified`.This class includes the following properties: `parameter`, `parameterValue`.

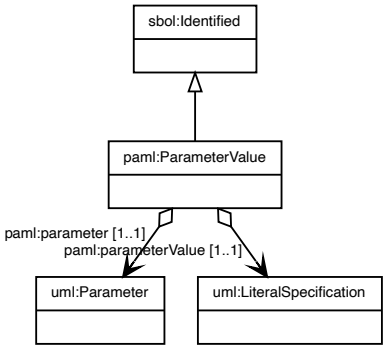

Figure 16: ParameterValue

- The `parameter` property is REQUIRED and contains a URI reference to an associated object of type ParameterThis property points to the uml:Parameter associated with the value (e.g., wavelength for a plate reader absorbance measurement behavior).
- The `parameterValue` property is REQUIRED and contains a URI reference to an associated object of type LiteralSpecificationThis property points to the literal value used for the parameter during execution (e.g., a uml:LiteralIdentified for an om:Measure representing a 600 nm wavelength).

5.5 Material

An amount of material allocated for use during the execution of a behavior. For example a `Material` might be used to specify 1 96-well flat-bottom microplate or 2.5 mL of 10 millimolar glucose.

TODO: consider changing type of specification to allow non-TopLevel descriptions, such as a `ContainerSpec` or sbol:ExternallyDefined TODO: consider adding a field to distinguish between expended vs. reusable materials.

The `Material` class is shown in Figure 17. It is derived from `sbol:Identified`.This class includes the following properties: `amount`, `specification`.

- The `amount` property is REQUIRED and contains a URI reference to an associated object of type MeasureThe amount property of a Material is used to indicate the quantity of material used. For example, 2.5 mL (referring to a fluid) or 3 (with unit "number", referring to a group of microplates)
- The `specification` property is REQUIRED and contains a URI reference to an associated object of type TopLevelThe specification property is used to indicate the type of material used. For example a DNA sample would be described by an sbol:Component.

  TODO: add example for glucose and for 96-well plate

5.6 ActivityEdgeFlow

An `ActivityEdgeFlow` records `sbol:one` movement of a UML token along a `uml:ActivityEdge` during the execution of its containing `Protocol`. If the edge is a `uml:ObjectFlow`, then the value MUST be set. If the edge is a `uml:ControlFlow`, then the value MUST NOT be set.

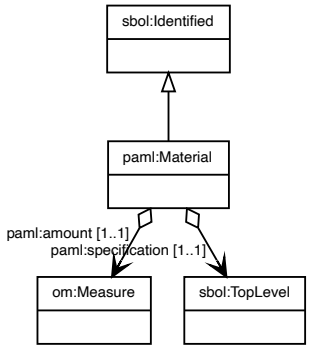

Figure 17: Material

For instance, the `ActivityEdgeFlow` for a `uml:ObjectFlow` might record a measurement being sent to an output `uml:Parameter`, while the `ActivityEdgeFlow` for a `uml:ControlFlow` might record a decision to proceed down a particular branch from a `uml:DecisionNode`.

Note that a `uml:ActivityEdge` might appear in multiple `ActivityEdgeFlow` records associated with a single `ProtocolExecution`, e.g., due to a loop in the `Protocol`. It also might not appear in any, if the `uml:ActivityEdge` is on a path not taken due to branching control flow.

TODO: correct the cardinality: edgeValue is supposed to be optional, not edge

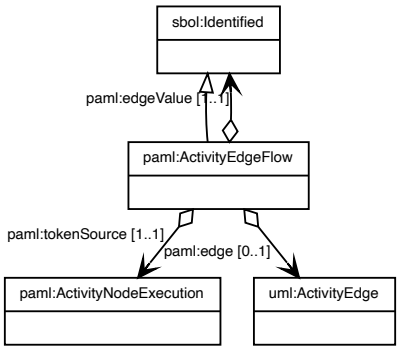

Figure 18: ActivityEdgeFlow

The `ActivityEdgeFlow` class is shown in Figure 18. It is derived from `sbol:Identified`.This class includes the following properties: `tokenSource`, `edge`, `edgeValue`.

- The `tokenSource` property is REQUIRED and contains a URI reference to an associated object of type

ActivityNodeExecutionThis property is used to indicate the ActivityNodeExecution that produced the token.

- The `edge` property is OPTIONAL and contains a URI reference to an associated object of type ActivityEdgeThis property is used to indicate the uml:ActivityEdge down which the token moved.
- The `edgeValue` property is REQUIRED and contains a URI reference to an associated object of type IdentifiedThis property is used to indicate the value of a token that moved on a uml:ObjectFlow edge.

5.7 ActivityNodeExecution

An `ActivityNodeExecution` records `sbol:one` instance in which a `uml:ActivityNode` is executed during the execution of its containing `Protocol`.

For instance, the `ActivityNodeExecution` for a `uml:CallBehaviorAction` to measure absorbance on a plate reader would set its node property to point to the `uml:CallBehaviorAction` and might have incomingFlow properties indicating arrival of information about the samples to measure via a `uml:ObjectFlow` and the arrival a of permission to begin via a `uml:ControlFlow`.

Note that a `uml:ActivityNode` might appear in multiple `ActivityNodeExecution` records associated with a single `ProtocolExecution`, e.g., due to a loop in the `Protocol`. It also might not appear in any, if the `uml:ActivityNode` is on a path not taken due to branching control flow.

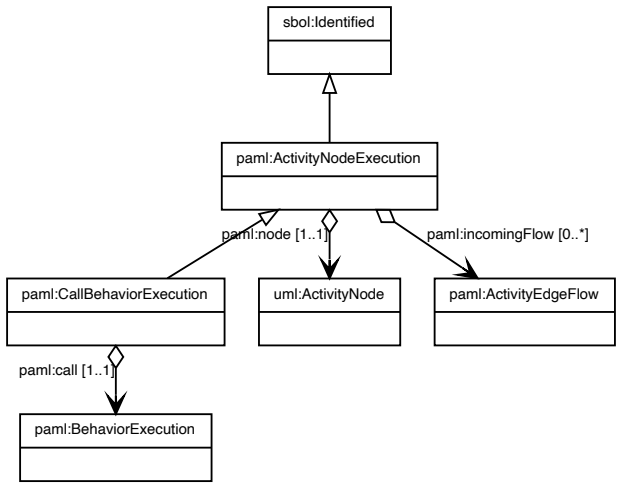

Figure 19: ActivityNodeExecution

The `ActivityNodeExecution` class is shown in Figure 19. It is derived from `sbol:Identified` and includes the following specializations: `CallBehaviorExecution`. This class includes the following properties: `node`, `incomingFlow`.

- The `node` property is REQUIRED and contains a URI reference to an associated object of type ActivityNodeThis property is used to indicate the uml:ActivityNode that has been execcuted.

- The `incomingFlow` property is OPTIONAL and contains URI references to associated objects of type ActivityEdgeFlowThis property is used to indicate an ActivityEdgeFlow that delivered a token consumed during the execution of the uml:ActivityNode.

5.7.1 CallBehaviorExecution

A `CallBehaviorExecution` extends `ActivityNodeExecution` by adding a pointer to a BehaviorExecution record for the `uml:Behavior` that is being executed.

For a primitive action (e.g., measuring absorbance on a plate reader), this is a plain `BehaviorExecution`, while for calling a `Protocol` as a sub-routine (e.g., to run a stage of an Type IIS assembly), this would be a `ProtocolExecution`.

The `CallBehaviorExecution` class is shown in Figure 19. It is derived from `ActivityNodeExecution`.This class includes the following properties: `call`.

- The `call` property is REQUIRED and contains a URI reference to an associated object of type BehaviorExecutionThis property indicates the BehaviorExecution record for the uml:Behavior that was called.

5.8 SampleCollection

SampleCollection is the base class for describing the collections of physical materials that are acted upon by a `Protocol`. For example, a `SampleCollection` might describe a set of 10 cell cultures growing in 96-well plate cells, or a set of 6 streaked agar plates, or a single 500 mL flask filled with media.

There are two types of `SampleCollection`. A `SampleArray` specifies an n-dimensional rectangular array of samples, all stored in the same type of container. A `SampleMask` specifies a subset of a `SampleCollection` by means of an array of Boolean values indicating whether each element is included or excluded from the subset.

Note, however, that a `SampleCollection` is a logical object and not a physical object. Thus, while a `SampleCollection` might describe a set of samples in 96-well plate wells, it does not necessarily identify a particular 96-well plate or the location of those wells. In practice, these will be determined as a result of the specific library calls made to generate `SampleCollection` objects, and may not be determined until the protocol is actually run in a particular execution environment.

This is important for increasing the flexibility with which a `Protocol` can be specified and applied. Consider, for example, a cell culturing protocol that includes a step to measure `sbol:sample` absorbance on a plate reader. Describing this step does not require knowing how the samples are laid out on the plate, and in many cases is even acceptable to run on samples across multiple plates. This flexibility will allow the cell culturing protocol to be applied for experiments with different numbers and arrangements of samples.

The `SampleCollection` class is shown in Figure 20. It is derived from `sbol:Identified` and includes the following specializations: `SampleArray`, `SampleMask`.

5.8.1 SampleArray

A `SampleArray` specifies an n-dimensional rectangular array of samples, all stored in the same type of container. For example, a `SampleCollection` might describe a set of 10 cell cultures growing in 96-well plate cells, or a set of 6 streaked agar plates, or a single 500 mL flask filled with media.

Wells may be full, in which case the contents property should contain a URI to a description of the `sbol:sample`, or empty, in which case the contents should be null.

Note that this is a logical array, and does not necessarily indicate the actual layout of the samples in space. For example, a 2x4 array of samples in 96-well plate wells might end up being laid out as a 2x4 array in wells A1 to B4 or

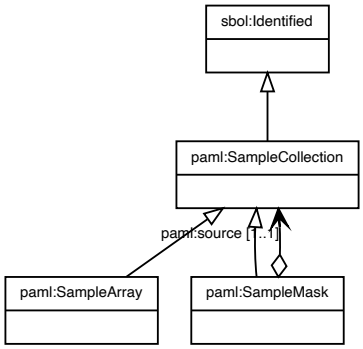

Figure 20: SampleCollection

as a 2x4 array in wells G5 to H8 or as an 8x1 column in wells A1 to H1, or even as eight wells scattered arbitrarily around the plate according to an anti-bias quality control schema.

This also allows for higher-dimensional arrays where each dimension represents an experimental factor. For example, an experiment testing four factors with 3, 3, 4, and 5 values per factor, for a total of 180 combinations, could be represented as a 4-dimensional `sbol:sample` array of 96-well plate wells, and then end up laid out over two plates.

TODO: need to decide on the format of the contents description.

The `SampleArray` class is shown in Figure 20. It is derived from `SampleCollection`.This class includes the following properties: `contents`, `containerType`.

- The `contents` property is REQUIRED and has a singleton value of type stringDescription of the contents. TODO: need to decide whether this is a multi-valued property with associated array coordinates or a single-valued property with an array value. Currently set to string as a "dummy" value that can serialize anything.
- The `containerType` property is REQUIRED and has a singleton value of type URI

5.8.2 SampleMask

A `SampleMask` is a subset of a `SampleCollection`. The subset of samples to be included is defined by an array of Boolean values, where true values indicate that a `sbol:sample` is included and false values indicate that it is excluded.

The dimensions of the mask MUST be identical to the dimensions of the source `SampleCollection`. For this purpose, the dimensions of a masked subset are not reduced, but remain the same as the original `SampleArray`. This allows masks to be composed, such that SampleMask(source=SampleMask(source=X,mask=mask1),mask=mask2) is equivalent to SampleMask(source=X,mask=mask1 AND mask2). Note that this implies masks are commutative and idempotent.

The `SampleMask` class is shown in Figure 20. It is derived from `SampleCollection`.This class includes the following properties: `mask`, `source`.

- The `source` property is REQUIRED and contains a URI reference to an associated object of type SampleCollectionThe source indicates the SampleCollection that is being subsetted via the mask

- The `mask` property is REQUIRED and has a singleton value of type stringThe mask is an N-dimensional array of Booleans values, where each Boolean indicates whether the sample at the corresponding location in the source is included in the subset.

  TODO: format of mask array needs to match the array format chosen for the SampleArray contents property

5.9 SampleData

The `SampleData` class is used to associate a set of data with a collection of samples. This is typically used to capture measurements, e.g., an array of absorbance measurements collected by a plate reader. Using this data structure allows the values in a dataframe to be automatically linked to the descriptions of the samples that the data describes, which is critical for data analysis.

The dimensions of the sampleDataValues MUST equal the dimensions of the `SampleCollection` linked with fromSamples.

TODO: the format of the data values needs to be compatible with the array format chosen for the `SampleArray` contents property. In this case, however, we also need to consider how we want to support multiple values for each `sbol:sample` (e.g., measurement of both fluorescence and absorbance in a plate reader), as well as links to more complex data (e.g., results of flow cytometry or omics for each sample)

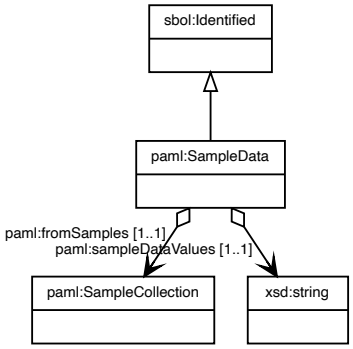

Figure 21: SampleData

The `SampleData` class is shown in Figure 21. It is derived from `sbol:Identified`.This class includes the following properties: `fromSamples`, `sampleDataValues`.

- The `fromSamples` property is REQUIRED and contains a URI reference to an associated object of type SampleCollectionThe fromSamples property indicates the SampleCollection from which the data were collected.
- The `sampleDataValues` property is REQUIRED and contains a URI reference to an associated object of type stringThe sampleDataValues are an array of data values, one for each sample, format to be determined.

5.10 ContainerSpec

A `ContainerSpec` is used to indicate the type of container to be used for a `SampleArray`, e.g., a standard 96-well flat-bottom transparent plate.

TODO: determine if we want to use this format or modify it in some way.

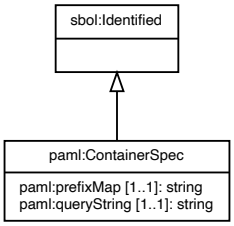

Figure 22: ContainerSpec

The `ContainerSpec` class is shown in Figure 22. It is derived from `sbol:Identified`.This class includes the following properties: `prefixMap`, `queryString`.

- The `prefixMap` property is REQUIRED and has a singleton value of type stringA prefix map in JSON-LD format, to be applied to a queryString.
- The `queryString` property is REQUIRED and has a singleton value of type stringA query string, in OWL Manchester syntax, to be used to find matching containers in the ContainerSpec.

6 Serialization

In order for PAML objects to be readily stored and exchanged, it is important that they are able to be *serialized*, i.e., converted to a sequence of bytes that can be stored in a file or exchanged over a network. To this end, PAML builds upon the Resource Description Framework (RDF). RDF is an abstract language for describing conceptual graph-oriented data models, and therefore does not mandate any specific serialization format. Instead, a number of different serialization formats are provided as separate specifications, such as RDF/XML, N-Triples, JSON-LD, and Turtle. These serialization formats are widely supported by RDF libraries such as rdflib for Python and Apache Jena for Java.

All PAML libraries SHOULD support at least RDF/XML, N-Triples, JSON-LD, and Turtle. Other PAML tools SHOULD support at least one of these four formats.

References

Baig, H., Fontanarrosa, P., Kulkarni, V., McLaughlin, J. A., Vaidyanathan, P., Bartley, B., Beal, J., Crowther, M., Gorochowski, T. E., Grunberg, R., Misirli, G., Scott-Brown, J., Oberortner, E., Wipat, A., and Myers, C. J. (2020). Synthetic biology open language (SBOL) version 3.0.0. *Journal of Integrative Bioinformatics*, 17(2-3).

DCMI Usage Board (2012). DCMI metadata terms. DCMI recommendation, Dublin Core Metadata Initiative.

Object Management Group (2017). Omg unified modeling language (omg uml) version 2.5.1. https/www.omg.org/spec/UML/.

Rijgersberg, H., Willems, D., Ren, X.-Y., Wigham, M., and Top, J. (2021). Ontology of units of measure (OM), version 2.0.31. http://www.ontology-of-units-of-measure.org/resource/om-2.