# BBF RFC ?: Synthetic Biology Open Language (SBOL) Version 2.0.0


| | |
|---|---|
| Bryan Bartley | *University of Washington, US* |
| Jacob Beal | *Raytheon BBN Technologies, US* |
| Kevin Clancy | *ThermoFischer Scientific, US* |
| Goksel Misirli | *Newcastle University, GB* |
| Nicholas Roehner | *Boston University, US* |
| Matthew Pocock | *Newcastle University, GB* |
| lots of other community members | all over the world |
| Chris Myers | *University of Utah, US* |
| Anil Wipat | *Newcastle University, GB* |
| Herbert Sauro | *University of Washington, US* |

editors@sbolstandard.org



Version 2.0.0

24 April 2015

# Contents

# 1 Purpose

Synthetic biology builds upon the techniques and successes of genetics, molecular biology and metabolic engineering by applying engineering principles to the design of biological systems. These principles include standardization, modularity, and design abstraction. The field still faces substantial challenges, including long development times, high rates of failure, and poor reproducibility.

To help address these challenges, the Synthetic Biology Open Language (SBOL) introduces (1) a standardized file format for the electronic exchange of biological designs and (2) a standardized data model for the reproducible description of essential design details. Ultimately, SBOL is intended to speed up the research and development of designed biological systems by enhancing the exchange and reproducibility of biological designs between different labs.

The current SBOL standard, Version 1.1, focuses upon representing the structural aspects of genetic designs. To serve as an effective medium for the computational exchange of genetic designs, SBOL must be extended to capture more aspects of a designed system, including both structural and functional information, and the composition of complex structural and functional designs by combining simpler parts. The SBOL data model proposed in this specification provides for addressing the most pressing needs for expanding SBOL Version 1.1.

1. represent structural components of a biological design, including DNA, RNA, proteins, small molecules and other physical components
2. describe behavioral aspects of a biological design, the intended or expected interactions and dynamic behavior
3. associate structure and function together, so that a single design can be understood both in terms of its structure and its behavior
4. support rich annotations of all components, so that data required to describe a design, but not formalized in this specification can be safely exchanged

Taken together, these capabilities allow SBOL sufficient expressivity to support the description and exchange of hierarchical, modular representations of both the intended structure and function of designed biological systems.

To address the need for functional descriptions in SBOL, the proposed data model adds classes for modules, interactions, and models. These classes provide a firm basis for functional representation in SBOL without going so far as to create a new standard for mathematically modeling biology, as there already exist several established languages for doing so, from the Systems Biology Markup Language (SBML) Finney et al. (2006) to CellML Garny et al. (2008) and even MatLab MathWorks (2015). Rather, these classes enable users of SBOL to group components that function together, describe the basic qualitative interactions between these components, and document references to standard mathematical models that are external to SBOL and that provide more detailed descriptions of component function. In other words, a module gathers together a set of component instantiations, a set of interactions between these component instantiations, and a set of references to external models that are expected to be consistent with the module's interactions.

# 2 Copyright Statement

# 3 History and Acknowledgements

SBOL originated in discussions between participants in the Synthetic Biology Open Language Workshops held in Blacksburg, Virginia on January 7-10, 2011, and its further development has continued at a series of subsequent open invitation workshops and through email exchanges on the SBOL Developers mailing list.

Contributors to this work include: Bryan Bartley (University of Washington), Jacob Beal (BBN Technologies), Goksel Misirli (Newcastle University), Chris J. Myers (University of Utah), Matthew Pocock (Newcastle University and Turing Ate My Hamster LTD), Nicholas Roehner (Boston University), Herbert M. Sauro (University of Washington), Anil Wipat (Newcastle University).

# 4 SBOL RDF Serialization

The SBOL serialization is designed to meet several competing requirements. Firstly, it needs to support ad-hoc annotations and extensions. Secondly, it needs to support processing by generic semantic web and database tools that have little or no knowledge of the SBOL data model. Thirdly, it needs to support the generation of light-weight software clients so as to lower the barrier to entry for new API implementations within environments were the community-maintained implementation(s) are not suitable.

The canonical serialization of SBOL is to a strict dialect of XML-RDF. This provides the base from which to meet the requirements. Any XML-RDF-aware tooling can consume and analyze an SBOL file. Where possible, we have re-used predicates from widely-used terminologies (such as Dublin Core REF) to expose as much of the data as practical to standard RDF tooling.

Arbitrary XML-RDF provides a great deal of flexibility in how equivalent data can be serialized. This makes it different to process RDF-XML files using standard off-the-shelf XML tools, such as DOM-OO mappings. To address this, we define a canonical association between the nesting of data structures within the SBOL UML data model and the RDF-XML file. For all associations that are ownership (filled diamonds), the RDF-XML for the owned entity is embedded within that of the owner. For all associations that are by reference (open diamonds), the RDF-XML for the referenced property is linked to via a resource URI.

All first-class SBOL datatypes have an associated identity URI. In the RDF, this is the resource URI for instances of that type. Properties and associations are asserted as nested RDF-XML assertions. Some datatypes are 'top level', which means that they always appear at the top level of the RDF-XML serialization. All other datatypes will always appear nested within their container.

Each instance of a first-class SBOL datatype may have annotations attached. These are composed of a name and a value. These are serialized to RDF as a triple with the subject being the identity of the instance they annotate, the predicate being the name of that annotation, and the object being the value of that annotation. Annotation values are always nested within the RDF-XML serialization of the instance that they annotate.

SBOL supports top-level, user-defined annotations. This is to allow non-standardized but necessary information to be carried around as part of a design. For example, a particular sub-community may have an internal standard for data sheets. Individual data sheets can be represented as a generic top-level annotation with internal structured annotations. This will be serialized into the RDF-XML in the usual way, as a RDF-XML block at the top level of the file. Other objects may refer to this through their annotations by reference, and this generic top-level entity may refer to other entities via references.

Through this design for the XML-RDF serialization, SBOL is able to adapt to future changes in the standard without requiring large-scale alterations to the RDF files. As exactly the same scheme is used to serialize annotations and specification-defined properties and associations, it is possible to update the SBOL standard to recognize a different range of these. Those not recognized by the specification will always be available through the API as annotations. Similarly, by allowing arbitrary top-level entries in an SBOl file, we enable future specifications or extensions to ratify the structure of some of these. They would then become something represented by an explicit data model, but the identical RDF serialization would be used. Applications lacking support for a given extension can safely round-trip the top-level data that is not understood, treating it as a top-level structured annotation, without data loss or corruption. The very regimented control of nesting vs referencing allows the XML structure to be very predictable, enabling XML/DOM-based tooling to work with SBOL XML-RDF files safely.

# 5 Overview of SBOL

Typically, information about a genetic circuit includes the order of its constituents and their descriptions. The exact locations of these constituents and their sequences allow defining genetic circuits unambiguously, and reusing the designs. Interactions between these constituents are then used to construct biologically plausible designs.

In the figure below, a simple toggle switch system is displayed, in which LacI and TetR repress each other transcriptionally. The toggling of the system is controlled by adding IPTG to deactivate LacI, and ATC to deactivate TetR. The components of the system includes genetic elements, proteins, small molecules.

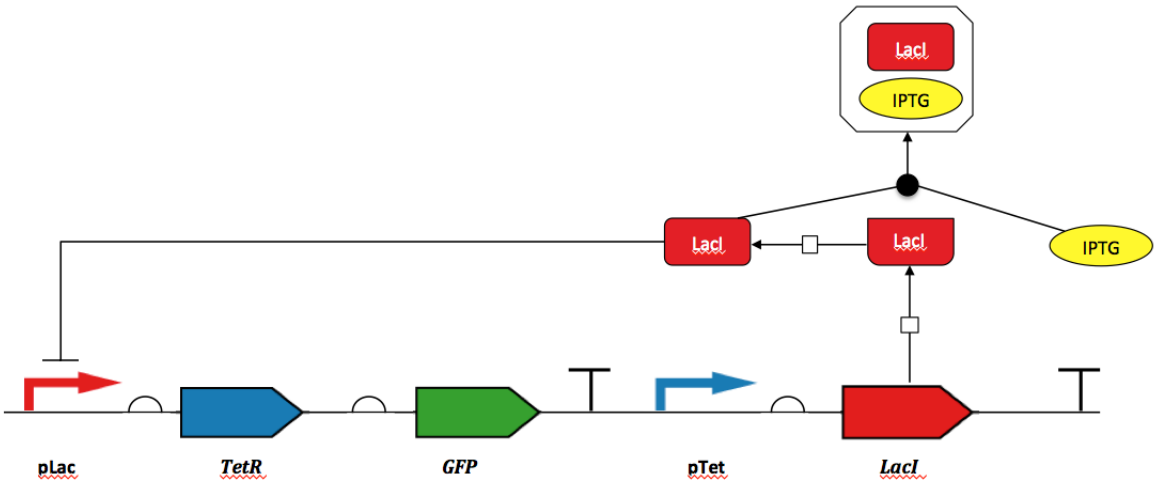


***Figure 1:*** *An example toggle swicth genetic circuit.*

SBOL includes different entities to describe such genetic circuits. Genetic elements such as promoters, RBS, CDSs and terminators are defined with the `ComponentDefinition` entity. Their instances are reused in different designs via the `Component`s that refer to corresponding `ComponentDefinition`s. `ComponentDefinition`s can also represent proteins, RNAs or small molecules. They are associated with sequence information such as nucleotides aminoacids or chemical structure. A full description of a genetic circuit is then represented using `Module`s which contains information about molecular interactions and their participating components. Modules can be associated with quantitative or qualitative models using the `Model` entity, which is used to point to the actual location of a model.

SBOL facilitates the design of complex systems using hierarchical composition. In addition to using simple genetic elements modularly, modules that are composed of different components can also be reused. Such modules can expose some of the design components as inputs and outputs, which can be connected to components from other modules using `MapsTo` entities.

The same toggle switch is now displayed using two LacI and TetR inverter submodules in figure Figure 2. The LacI inverter uses LacI as input and produces the TetR output, and the TetR inverter uses TetR as input and produces the LacI output. These inputs and outputs are mapped in a parent module.

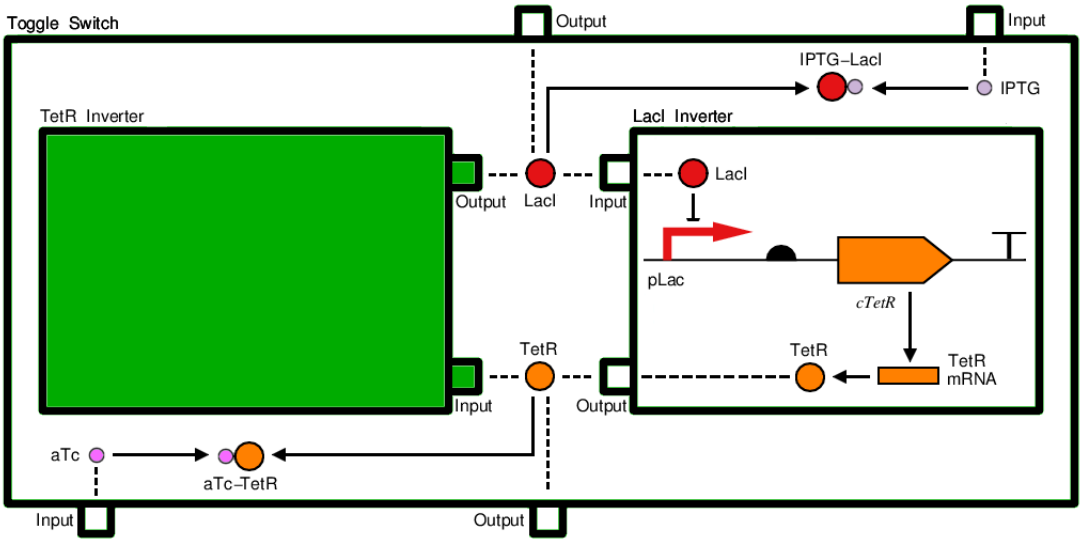


***Figure 2:*** *Representing hierarchical composition of the toggle switch genetic circuit.*

# 6 SBOL Vocabulary

SBOL defines several entity types in order to explicitly define different types of information. At the top level, a SBOL document includes these entities:

| | |
|---|---|
| *list of component definitions(optional)* | (Section 7.6) |
| *list of sequences (optional)* | (Section 7.7) |
| *list of module definitions (optional)* | (Section 7.8) |
| *list of models (optional)* | (Section 7.9) |
| *list of collections (optional)* | (Section 7.10) |
| *list of generic top level entities (optional)* | (Section 7.11.2) |

In addition to these top level entities, a SBOL document may include addiitonal entities that are nested within the top level entities. Short descriptions of all SBOL entities are given below:

`ComponentDefinition`: Represents biological molecules such as DNA, proteins, RNA and small molecules.

`Component`: Represents a specific occurrence of a component within a larger design. For a given component definition, there may be several instances of components specified at different locations within the larger design.

`SequenceAnnotation`: A sequence annotation describes the position and strand orientation (in the case of DNA molecules) of a notable sub-sequence found within the Component being described. Annotations provide the link which describes sequence of a component in terms of other components, subComponents.

`Location`: A Location specifies the coordinates of a genetic locus on a DNA molecule or a residue or site on other sequential macromolecules such as proteins.

`SequenceConstraint`: A Sequence Constraint describes the relative spatial orientation of Components which are assembled together into a biological structure.

`Sequence`: A sequence is a contiguous sequence of monomers in a macromoleculer polymer such as DNA, RNA, and protein. The sequence is a fundamental information object for synthetic biology and is needed to reuse components, replicate synthetic biology work, and to assemble new synthetic biological systems. Therefore, both experimental work and theoretical sequence composition research depend heavily on the sequence associated with component definitions.

`ModuleDefinition`: A Module Definition desribes an abstract functional module that may be composed of many functional interactions between biological components.

`FunctionalComponent`: An occurrence of a component that is intended to have a biological function inside a design, as opposed to a component whose purpose is simply to compose a biological structure.

`Module`: Represents a specific occurrence of a functional module within a larger design. For a given module definition, there may be several instances of modules that specify the overall function of a biological design.

`MapsTo`:

`Interaction`: Interactions describe a functional relationship between biological effectors, such as regulatory activation or repression. Interactions also describe processes from the central dogma of biology, such as transcription and translation.

`Participation`: Describes the role that a component plays in a functional interaction. For example, a transcription factor may participate in an interaction either as a repressor or activator.

*Model*: Links an SBOL representation of biological components and their interactions to quantitative, computational models that may be used to predict the functional behavior of a biological design.

*Collection*: A Collection is an organizational container for Components, a group defined by the user for convenience.

*GenericTopLevel*: This is a data container that can contain custom data added by user applications.

# 7 SBOL Data Model

In this section, we describe the types of biological design data that can belong to an SBOL document and the relationships between these data types. The SBOL data model is specified using Unified Modeling Language (UML) 2.0 diagrams (OMG 2005).

Complete SBOL examples and best practices when using SBOL can be found in Section 8 and Section 10, respectively.

## 7.1 Understanding the UML Diagrams

The types of biological design data modeled by SBOL are commonly referred to as classes, especially when discussing the details of software implementation. Classes are represented in UML diagrams as rectangles labeled at the top with class names. Classes may be connected to other classes by association properties, which are represented in UML as arrows labeled with data cardinalities. Finally, the remaining class properties are listed below class names inside the rectangles and are labeled with their data types and cardinalities (explained below).

Association properties between SBOL classes are directional. An arrowhead indicates the direction in which an association can be traversed, while a diamond at the beginning of an arrow indicates the type of association. Open-faced diamonds indicate shared aggregation, which means that objects of the class at the end of the arrow exist independently of any objects of the class that has the association property. In contrast, filled diamonds indicate composite aggregation or a part-whole relationship, in which objects of the class at the end of the arrow cannot exist independently and must be associated with at most one object of the class that has the association property. For example, it will later be shown how objects of the `SequenceAnnotation` class must associated with an object of the `ComponentDefinition` class (and only that object).

All SBOL properties are labeled with data cardinality restrictions. These are:

- 1 - required, one: there must be exactly one value for this property.
- $0 \ldots 1$ - optional: there may be a single value for this property or it may be absent.
- $0 \ldots *$ - unbounded: there may be any number of values for this property, including none.
- $1 \ldots *$ - required, unbounded: there may be any number of values for this property, as long as there is at least one.
- $n \ldots *$ - at least: there must be at least $n$ values for this property.

Within the SBOL data model, all properties are given a singular or plural name in accordance with their data cardinalities. The forms of these names follow the usual rules of grammar. For example, `sequenceAnnotation` is the singular form of `sequenceAnnotation`s.

Within the RDF serialization of SBOL, however, SBOL properties are always given singular names. The serialization does not use RDF collections to represent multiple values of an SBOL property. Hence, if a SBOL property has five values, then its serialization contains five RDF triples with singular predicate names.

Within an implementing Object-Oriented (OO) API, SBOL properties should be mapped to member accessors that are similarly named and that return idiomatic representations of these properties. For example, a Java implementation would use common Java idioms. In this case, the member accessor for an optional SBOL property could return a Java primitive value, Java object, or null, while the accessor for a multi-valued SBOL property could return a Java `Collection`. In general, OO member accessors for multi-valued SBOL properties should never return null.

As another example, a relational implementation of the SBOL API would store the properties and associations a mixture of data fields and references via foreign keys. The fields in individual tables will correspond to the 'arrowhead' end of an association (in reverse to the direction in the RDF and OO representations), and the name may be modified to reflect this change in directionality. For example, the `sequence` association between a `ComponentDefinition`

and Sequence would be represented by a foreign key field on the Sequence table that references a row in the ComponentDefinition table, and it may be named sequenceOf.

## 7.2 SBOL Document

An SBOL document is a valid RDF/XML document. Accordingly, each SBOL document starts with an XML declaration that has its XML version set to "1.0." As shown in the example below, this declaration is then followed by a rdf:RDF XML element that includes the namespace declarations for RDF and SBOL. The latter namespace is used to indicate which entities and properties in the SBOL document are defined by SBOL.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:sbol="http://sbols.org/v2#">
...
</rdf:RDF>
```

As explained in Section 7.11, SBOL documents can also include custom namespace declarations for the purpose of embedding application-specific data.

## 7.3 Identified

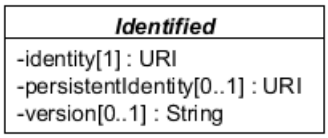


***Figure 3:*** *The Identified abstract class*

All SBOL-defined classes are directly or indirectly derived from the Identified abstract class. This inheritance enables SBOL objects to be identified using unique URIs that may refer to relative locations within an SBOL document or URLs that refer to locations on the World Wide Web. This class includes the following properties: identity, persistentIdentity, and version. (Figure 3).

### *The identity property*

This property is required by all Identified objects and has a datatype of URI. As before in SBOL Version 1.1, this URI serves to uniquely identify an SBOL object. The identity property can be used to point to other objects in the same SBOL document or resources on the Web. Although most SBOL properties are defined under the SBOL namespace, this property is defined under the RDF namespace using the term below.

http://www.w3.org/1999/02/22-rdf-syntax-ns#about.

### *The persistentIdentity property*

The persistentIdentity property is optional and also has a datatype of URI. This property is used to declare that a set of SBOL objects are versions of each other (by virtue of having the same persistent identity). This persistent identity can be used to return the most up-to-date version of a SBOL object.

### *The version property*

This property has a datatype of String and is used along with the persistentIdentity property to indicate the version of an SBOL object for comparison with other SBOL objects with the same persistent identity.

## 7.4 Documented

The Documented abstract class in SBOL represents objects that can be decorated with human-readable properties, such as name and description. This class extends `Identified` with three additional data properties: `displayId`, `name`, and `description` (Figure 4).

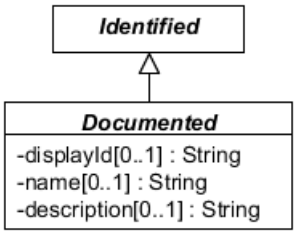


***Figure 4:*** *The Documented class*

### *The `displayId` property*

An optional display ID, with a data type of `String`. It is intended to be used when visualising SBOL objects.

### *The `name` property*

An optional, human readable name property, with a data type of `String`. If a `Documented` object lacks a `displayId`, it is expected that software tools will instead display the entity's `name` or `identity` as a last resort.

### *The `description` property*

An optional, free text description property with the data type of `String`.

## 7.5 TopLevel

`TopLevel` is an abstract class that is extended by any `Documented` object that can be found at the top level of a SBOL file, that is any SBOL object that is not nested inside another object when written to a file. Instead of nesting, composite `TopLevel` objects link to their subordinate `TopLevel` objects via `URI`s when written to a file. For example, a composite `Component` object A would link to its sub-Component object B using B's `URI`. TopLevel classes include `Sequence`, `ComponentDefinition`, `ModuleDefinition`, `Module`, `Collection` and `GenericTopLevel` (Figure 5).

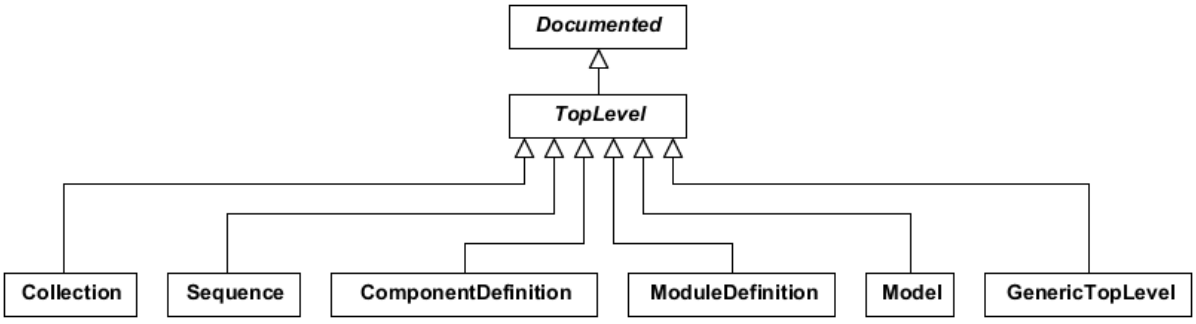


***Figure 5:*** *The TopLevel classes*

## 7.6 ComponentDefinition

The proposed Version 2.0 data model generalizes the DnaComponent class of SBOL Version 1.1 to a `ComponentDefinition` class in order to support an increased range of structural representation. A `ComponentDefinition` can represent genetic components such as DNA, but also RNA and proteins which were unrepresented in Version 1.1. Additionally, the generalized `ComponentDefinition` class can even represent non-genetic components, such as non-biological polymers, small molecules, molecular complexes, and even light.

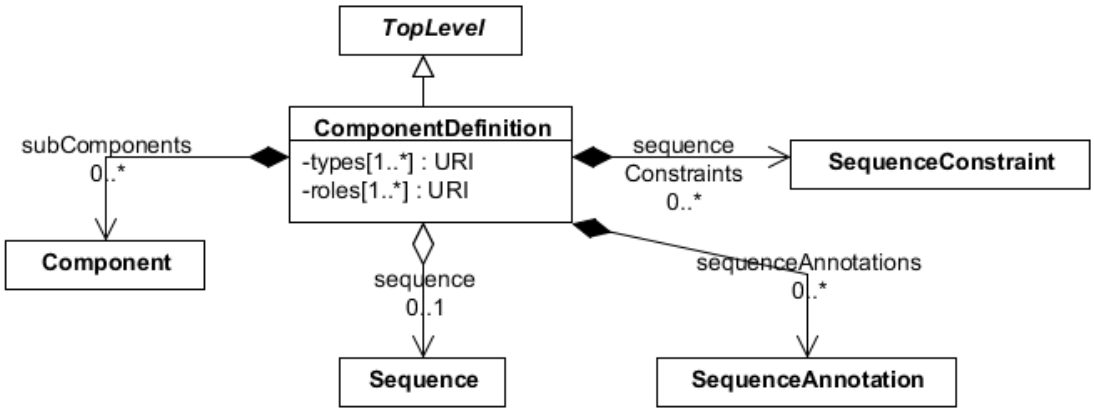


***Figure 6:*** *ComponentDefinition*

### *The `type` property*

In order to specify the blueprint for a molecular component, a `ComponentDefinition` must have at least one types `URI` that identifies a term from an appropriate ontology, such as the ontology of Chemical Entities of Biological Interest (ChEBI) and the BioPAX ontology (Table 1). A type URI documents the basic sort of biochemical or physical entity (for example DNA, protein, or RNA) that a `ComponentDefinition` object abstracts for the purpose of engineering design. If a `ComponentDefinition` object has multiple type URIs, then they must identify synonymous terms.

| Entity Type | BioPAX Term |
|---|---|
| DNA | `http://www.biopax.org/release/biopax-level3.owl#DnaRegion` |
| RNA | `http://www.biopax.org/release/biopax-level3.owl#RnaRegion` |
| Protein | `http://www.biopax.org/release/biopax-level3.owl#Protein` |
| Small Molecule | `http://www.biopax.org/release/biopax-level3.owl#SmallMolecule` |

***Table 1:*** *BioPAX terms to specify the types of ComponentDefinition objects.*

Examples of ontologies for non-molecular type ComponentDefinitions (eg, light)...?

### *The `role` property*

The `role` of a ComponentDefinition object, on the other hand, are analogous to the type of a DnaComponent object in SBOL Version 1.1. Role URIs are expected to identify ontology terms that clarify a `ComponentDefinition` object's potential function in a biological context. For example, a `ComponentDefinition` object of type "DNA" may have a role of "promoter" or "terminator," terms taken from the Sequence Ontology (SO). A ComponentDefinition object of type "protein," on the other hand, may have a role of "transcription factor" or "protease."

### *The subComponent property*

Another significant change from SBOL version 1.1 is that a layer of design abstraction has been added to the ComponentDefinition class. Instances of this class may have subComponents, each pointing to a Component object. The ComponentDefinition class is analagous to a blueprint or parts specification sheet for a biological part. In contrast, the Component class represents specific occurrences of parts within a design. Thus the new version of SBOL supports biological designs that re-use the same component more than once.

### *The sequence property*

The sequence property is optional and includes the URI a corresponding Sequence object.

### *The sequenceConstraint property*

### *The sequenceAnnotation property*

Finally, a ComponentDefinition object can define its structure by linking to objects that belong to the Component, Sequence, SequenceAnnotation, and SequenceConstraint classes. These classes are described below.

### *Serialization*

Parent classes of the ComponentDefinition class includes TopLevel, Documented and Identified. As a result, inherited properties are serialised as explained for these parent classes. The sequence property of a ComponentDefinition object includes a URI to a Sequence object and this URI is serialized as an rdf:resource. The type property may include a collection of type URIs and is serialized as an implicit collection, ignoring the property name "types". The role property is also similarly serialized as an implicit collection of sbol:role properties.

```
<sbol:ComponentDefinition rdf:about="...">
                ...
  zero or one   <sbol:name>...</sbol:name>element
  zero or one   <sbol:description>...</sbol:description>element
  zero or one   <sbol:sequence rdf:resource="..."/>element
  one or more   <sbol:type rdf:resource="..."/>  elements
  one or more   <sbol:role rdf:resource="..."/>  elements
  zero or more <sbol:subComponent>
                 <sbol:Component rdf:about="...">...</sbol:Component>
                </sbol:subComponent> elements
  zero or more <sbol:sequenceAnnotation>
                 <sbol:SequenceAnnotation rdf:about="...">...</sbol:SequenceAnnotation>
                </sbol:sequenceAnnotation> elements
  </sbol:ComponentDefinition>
```

The example below shows the serialization of a simple ComponentDefinition object. The DnaRegion from BioPAX and CHEBI:4705 from CHEBI are used to indicate the type of the biological entity as a DNA molecule. Its role is specified via the generic SO:0000167 (promoter) and more specific SO:0000613 (bacterial_RNApol_promoter) terms.

```
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/Part:BBa_J23119">
  <sbol:name>J23119 promoter</sbol:name>
  <sbol:description>Constitutive promoter</sbol:description>
  <sbol:type rdf:resource="http://identifiers.org/chebi/CHEBI:4705"/>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000613"/>
  <sbol:sequence rdf:resource="http://www.partsregistry.org/Part:BBa_J23119:Design"/>
</sbol:ComponentDefinition>
```

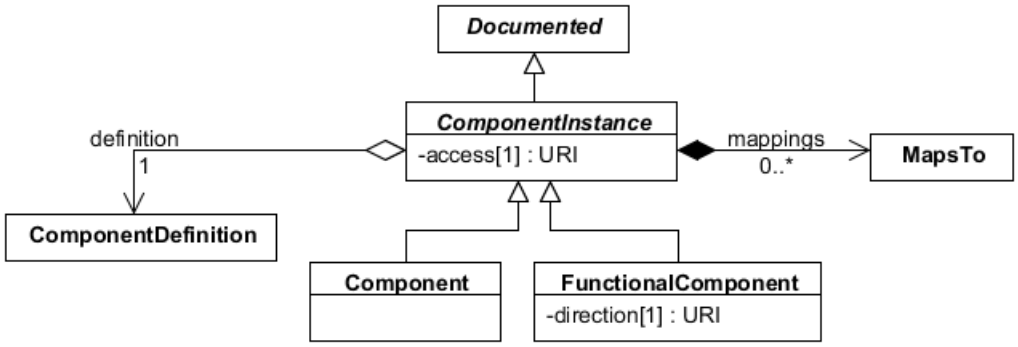


***Figure 7:*** *Component*

### 7.6.1 ComponentInstance

Hierarchical compositions of structure and function is enabled via the ComponentInstance class. Each ComponentInstance object has three data fields. The first data field, “definition”, links to the Component object that is effectively a part of the Component or Module object that owns the ComponentInstantiation object. The second data field, ”access”, defines whether the ComponentInstance object can be referred to by ComponentInstance objects that are higher in the design hierarchy (yes if access is set to ”public”). The third data field, “mappings”, is a set of MapsTo objects that link between other ComponentInstance objects at the same level of the design hierarchy as well as other ComponentInstance objects that are lower in the design hierarchy, thereby composing these objects with greater specificity (see first module example).

There are two subclasses of the ComponentInstance class: the Component and FunctionalComponent classes.

### 7.6.2 Component

Composition of the structural layer of SBOL designs is accomplished using `Component` objects. Each `Component` object is owned by a ComponentDefinition object and serves as an explicit usage of a sub-Component object for the purpose of physical composition.

### 7.6.3 SequenceAnnotation

The `SequenceAnnotation` class describes a location of interest on the `Sequence` object linked by its parent `ComponentDefinition` object and may optionally associate this location with a `Component` object. `SequenceAnnotation` objects specify their location using a `Location` object. As explained below, the `Location` class is extended by several different classes, some of which assert locations other than simple ranges with start and end positions.

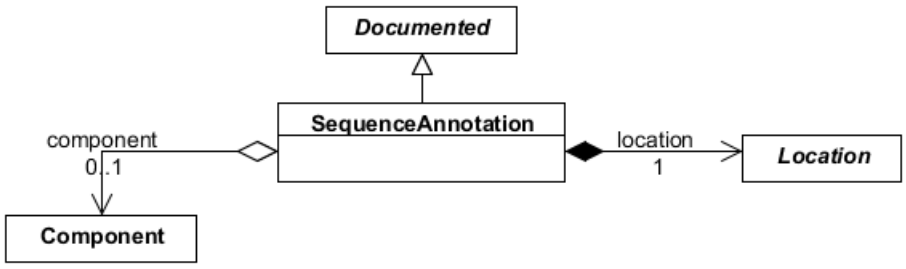


***Figure 8:*** *SequenceAnnotation class*

#### *The `location` property*

Every SequenceAnnotation must have a location property. The value of this property is a nested Location object.

#### *The `component` property*

SequenceAnnotation objects may have this property to link location information to a Component object. The value of this property is a Component URI.

The serialization of SequenceAnnotation objects can be carried out according to the example below. A_LOCATION_SUBCLASS represents one of the Location subclasses.

```
<sbol:SequenceAnnotation rdf:about="...">
                ...
  zero or one <sbol:component rdf:resource="..."/> element
  one         <sbol:location>
                <sbol:A_LOCATION_SUBCLASS rdf:about="...">...</sbol:A_LOCATION_SUBCLASS>
              </sbol:location> element
</sbol:SequenceAnnotation>
```

### 7.6.4 Location

The Location class is extended by the Range, MultiRange, and Cut classes.

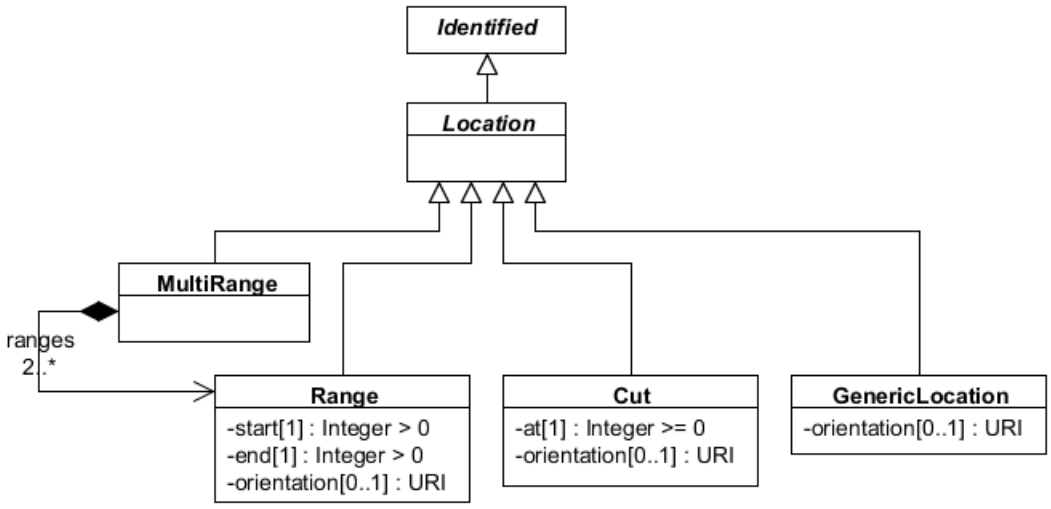


***Figure 9:*** *Location class*

#### *Range*

A Range object specifies inclusive start and end positions. These properties are required in Range objects and they can have `integer` values greater than zero. A Range object also includes an "orientation" property, for example to to specify directionality on a potentially double-stranded Component object.

#### *The `start` property*

Specifies the start of a Range. This property is required and can have `integer` values greater than zero.

#### *The `end` property*

Specifies the end of a `Range`. This property is required and can have `integer` values greater than zero.

#### *The `orientation` property*

This property is optional has a `URI` value. However, for `ComponentDefinition` objects representing DNA molecules, it is recommended that one of the orientation values are chosen (Table 2).

| Orientation Types |
| --- |
| http://sbols.org/v2#inline |
| http://sbols.org/v2#reverseComplement |

***Table 2:*** *URI constants for orientation values*

The serialization of Range objects has the following form:

```
<sbol:Range rdf:about="...">
                ...
  one         <sbol:start>...</sbol:start> element
  one         <sbol:end>...</sbol:end> element
  zero or one <sbol:orientation rdf:resource="..."/> element
</sbol:Range>
```

The example below shows the serialization of a `Range` object. It specifies the region between 56 and 68, and the orientation is given as `inline`.

```
<sbol:Range rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno2/range">
  <sbol:start>56</sbol:start>
  <sbol:end>68</sbol:end>
  <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
</sbol:Range>
```

#### *MultiRange*

A MultiRange object represents a location that is specified by multiple Range objects. For example, this capability can be used to specify a ComponentDefinition object that represents the introns or exons of a eukaryotic gene.

#### *Cut*

The Cut class has been introduced to enable the specification of a location between two indices. Each Cut object has a single integer data field, “at”, that specifies the index just before the location represented by the Cut object. Even though there is no zero index on Structure objects in SBOL, a Cut object with âĂIJatâĂİ equal to zero represents the location just before index one. The OrientedCut class extends the Cut class with an orientation in the same way that the OrientedRange class extends the Range class.

Finally, while the Range and Cut classes are best suited to describing locations on sequential structures, the Location class can be extended in the future to better describe locations on Component Objects with non-sequential structure (see unspecified Moeity class as potential means for specifying locations in more than one dimension).

### *7.6.5 SequenceConstraint*

A `ComponentDefinition` object can link to `SequenceConstraint` objects to assert different kinds of structural restrictions between two `Component` objects that are its subcomponents. A SequenceConstraint object requires `restriction`, `subject` and `object` properties to specify different constraints.

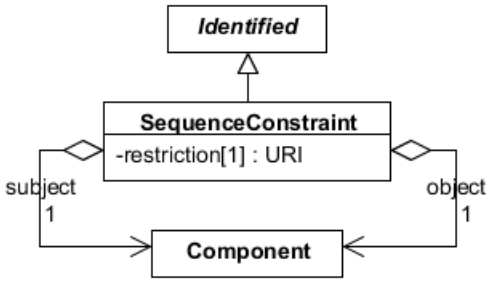


***Figure 10:*** *SequenceConstraint class*

#### *The* `subject` *property*

This property is required to specify the component that a restriction is applied to, and includes the `URI` `identity` of this `Component` object.

#### *The* `object` *property*

This property includes the `URI` `identity` of a `Component` object that is the value of the restriction.

#### *The* `restriction` *property*

This property property qualifies the relationship between the `subject` and `object` `Component`s. It includes a `URI` value, indicating the restriction type. In SBOL Version 1.1 partial designs are made possible by means of SequenceAnnotation objects that can specify that their locations precede those of other SequenceAnnotation objects. In Version 2.0 the "precedes" data field has been generalized to the StructuralConstraint class.

These restrictions could also include "sameOrientationAs" or "oppositeOrientationAs" to enable the relative orientation of Component objects whose ComponentDefinition objects lack associated Sequence objects in partial designs (Table 3).

| Restriction Types |
|---|
| http://sbols.org/v2#precedes |
| http://sbols.org/v2#sameOrientationAs |
| http://sbols.org/v2#oppositeOrientationAs |

***Table 3:*** *URI constants for restriction values*

The serialization of `SequenceConstraint` objects has the following form:

```
<sbol:SequenceConstraint rdf:about="...">
      ...
  one <sbol:restriction rdf:resource="..."/> element
  one <sbol:subject rdf:resource="..."/> element
  one <sbol:object rdf:resource="..."/> element
</sbol:SequenceConstraint>
```

The example below shows the serialization of a `SequenceConstraint` object. In the example, the constraint is included as part of a `ComponentDefinition` for a LacI repressible composite promoter and has a precedes restriction. This restriction states that the subject `Component` for the core promoter precedes the object `Component` for the LacI operator in the composite promoter definition. Such restriction is especially useful to specify incomplete designs and the final design may include other components between the subject and object components.

```
<sbol:SequenceConstraint rdf:about="http://www.partsregistry.org/Part:BBa_K174004/r1">
  <sbol:restriction rdf:resource="http://sbols.org/v2#precedes"/>
  <sbol:subject rdf:resource="http://www.partsregistry.org/Part:BBa_K174004#pspac"/>
  <sbol:object rdf:resource="http://www.partsregistry.org/Part:BBa_K174004#LacIoperator"/>
</sbol:SequenceConstraint>
```

## 7.7 Sequence

The Sequence class is used to encode the primary structure of a ComponentDefinition object and the encoding used to capture this information.

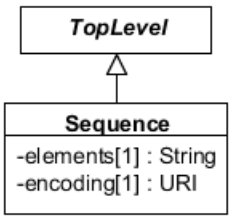


***Figure 11:*** *Sequence class*

### *The `elements` property*

Required. A `String` of characters that represent the constituents of biological molecule, for example nucleic acid symbols for DNA molecules.

### *The `encoding` property*

Required. Sequence objects identify their type of encoding with a `URI`. For example, a Sequence object that encodes a DNA sequence would have an `IUPAC DNA` encoding, while a Sequence object that encodes the chemical structure of glucose might have a `simplified molecular-input line-entry system (SMILES)` encoding (Table 4).

| ComponentDefinition Type | Encoding |
|---|---|
| DnaRegion,RnaRegion | http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html |
| Protein | http://www.chem.qmul.ac.uk/iupac/AminoAcid/ |
| SmallMolecule | http://www.opensmiles.org/opensmiles.html |

***Table 4:*** *URIs for the encoding property and the corresponding ComponentDefiniton types, which are BioPAX terms.*

The serialization of Sequence objects has the following form:

```
<sbol:Sequence rdf:about="...">
      ...
  one <sbol:elements>...</sbol:elements> element
  one <sbol:encoding rdf:resource="..."/> element
</sbol:Sequence>
```

The example below shows the serialization of a Sequence object for a promoter. Nucleotide sequences are represented with the elements property and the encoding is serialized as a URI value.

```
<sbol:Sequence rdf:about="http://www.partsregistry.org/Part:BBa_J23119:Design">
  <sbol:elements>ttgacagctagctcagtcctaggtataatgctagc</sbol:elements>
  <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
</sbol:Sequence>
```

## 7.8 ModuleDefinition

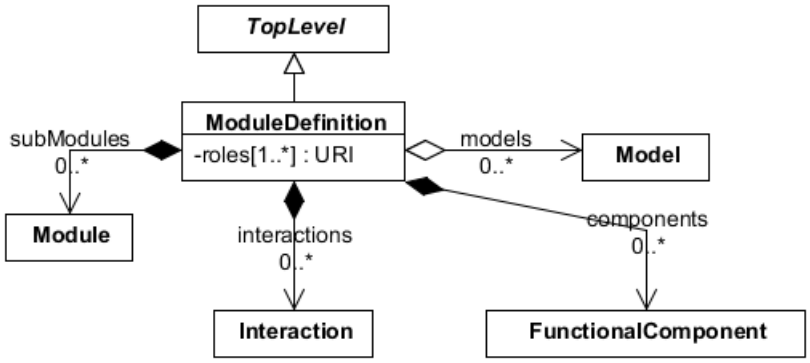


***Figure 12:*** *ModuleDefinition*

The ModuleDefinition class forms the hub for functional description of genetic designs. A Module object is composed from zero or more FunctionalComponent, Module, and Interaction objects, and links to zero or more Model objects. A Module object relies on the "direction" data fields of its FunctionalComponent objects to specify whether they serve as its inputs or outputs.

In addition, each ModuleDefinition object must now have at least one of potentially several roles to indicate its intended usages. For example, the role URIs on a ModuleDefinition object may identify terms for abstract module roles, such as"inverter" or "AND gate", or they may identify terms for biological module roles, such as "metabolic pathway" and "signaling cascade".

### 7.8.1 FunctionalComponent

Composition of the functional layer of SBOL designs, on the other hand, is accomplished using `FunctionalComponent` objects. Each FunctionalComponent object is owned by a Module object and serves as an explicit usage of a ComponentInstance object for the purpose of fulfilling some function. In addition, each FunctionalInstantiation must specify via the "direction" field whether it serves as an input, output, both, or neither for its parent Module object.

### 7.8.2 Module

The Module class enables the composition of Module objects from other sub-Module objects. The first data field, "definition", links to the ModuleDefinition object that is effectively a part of the Module object that owns the Module object. The second data field, "mappings", is a set of MapsTo objects that link between the Component objects at the same level of the design hierarchy as the Module object and the Component objects that are lower in the design hierarchy, thereby composing these objects with greater specificity.

### 7.8.3 MapsTo

The MapsTo class serves as a means of linking between Component objects (both Components and FunctionalComponents) at different levels of the design hierarchy. For example, when a Module object is instantiated inside another Module object, a MapsTo object on the appropriate Module object can be used to link between a Function-

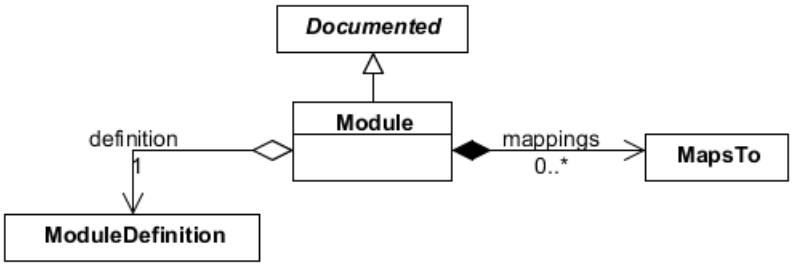


***Figure 13:*** *Module*

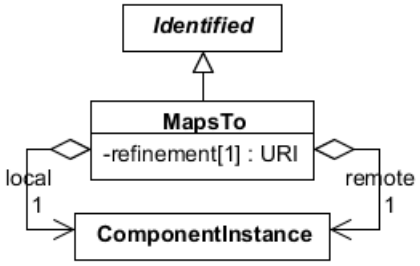


***Figure 14:***

alComponent object in the parent Module (as specified by the “local” data field) and a FunctionalComponent object in the child module (as specified by the “remote” data field). This linking can perhaps be most easily understood via the examples in the next section.

In addition to specifying a link, each MapsTo object must also specify a “refinement” relationship between its local and remote Components. Under this data model, there are four types of refinement: “verifyIdentical” requires that the Component objects link to the same ComponentDefinition object, “useLocal” indicates that the local Component object overrides the remote Component object, “useRemote” indicates the opposite, and âĂIJmergeâĂİ indicates that data fields of the local and remote ComponentInstantiation objects are to be interpreted in combination.

### *7.8.4 Interaction*

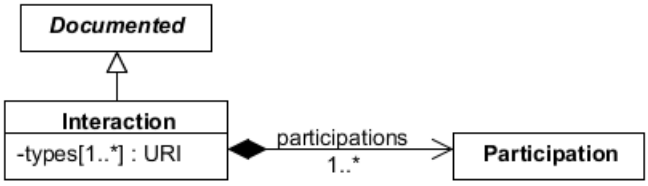


***Figure 15:*** *Interaction*

The Interaction class provides a qualitative basis for asserting the intended function of a given ModuleDefinition object. The proposed data model supports the representation of regulatory interactions, such as activation or

repression, and processes from the central dogma of biology, such as transcription and translation. Other supported interaction types include non-covalent binding between a small molecule and TF, and phosphorylation of a TF by an enzyme.

Each Interaction object must specify its type with at least one URI that identifies an appropriate ontology term, such as a term from the Systems Biology Ontology (SBO). If an Interaction object has multiple type URIs, then they must identify synonymous terms.

Furthermore, each Interaction object must specify its participating ComponentInstantiation objects by linking to one or more objects of the Participation class.

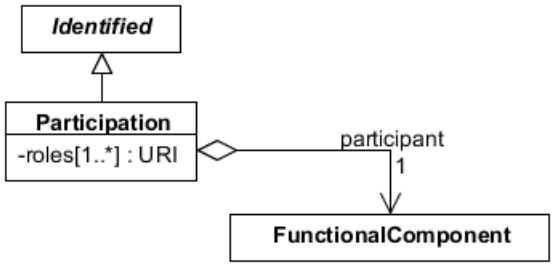


***Figure 16:*** *Participation*

### *7.8.5 Participation*

Each object of the Participation class must specify the role of its participant FunctionalComponent object in its parent Interaction object with at least one URI that identifies an appropriate ontology term. If a Participation object has multiple role URIs, then they must identify synonymous terms.

While the Interaction class provide a qualitative description of genetic function, quantitative descriptions are also needed for genetic design. Instead of introducing a new language for the specification of mathematical models of biology, the proposed data model leverages existing standards and links to them via the Model class.

## 7.9 Model

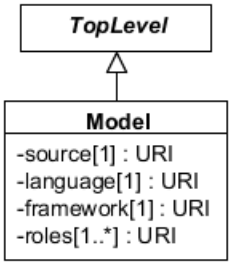


***Figure 17:***

SBOL's `Model` objects are used to link genetic descriptions of biological parts and their interactions to computational models. Each `Model` object specify the location of the actual content of a qualitative/quantitative model, the language the model is implemented with, the modelling framework and the model's role(s). In this way, there is

minimal duplication of standardization efforts and users of SBOL can specify the quantitative function of their ModuleDefinition objects in a well-developed language of their choice. A ModuleDefinition object can link to more than one model since each one can encode different levels of functional detail and play different roles in engineering design.

### *The `source` property*

This property has a data type of `URI`, and is required to specify the actual location of a qualitative or quantitative model.

### *The `language` property*

This property has a data type of `URI`, and is required to specify the language the model is implemented with. Values for the URIs should be chosen from the EMBRACE Data and Methods (EDAM) ontology where possible. Some of the model types and corresponding URI values are shown in Table 5.

| Model Language | URI |
|---|---|
| SBML | `http://identifiers.org/edam/format_2585` |
| CellML | `http://identifiers.org/edam/format_3240` |
| BioPAX | `http://identifiers.org/edam/format_3156` |

***Table 5:*** *Commonly used model languages and their corresponding URIs.*

### *The `framework` property*

This property has a data type of `URI` and is required to specify the modelling framework a model is implemented with. Values for this property should be used from the SBO's modelling framework terms where possible (Table 6).

| Framework | URI |
|---|---|
| Continuous | `http://identifiers.org/biomodels.sbo/SBO:0000062` |
| Discrete | `http://identifiers.org/biomodels.sbo/SBO:0000063` |

***Table 6:*** *Example modelling frameworks and corresponding SBO terms.*

### *The `role` property*

This property has a data type of `URI` and is required to specify what the model is for, such as part or interaction model (Table 7).

| Model Roles |
|---|
| http://sbols.org/v2#component_model |
| http://sbols.org/v2#interaction_model |
| http://sbols.org/v2#module_model |

***Table 7:*** *URI constants for model roles*

The serialization of `Model` objects has the following form:

```
<sbol:Model rdf:about="http://www.sbolstandard.org/examples/toogleswicth">
  ...
  one         <sbol:source rdf:resource="..."/> element
  one         <sbol:language rdf:resource="..."/> element
  one         <sbol:framework rdf:resource="..."/> element
  one or more <sbol:role rdf:resource="..."/> element
</sbol:Model>
```

The example below shows the serialization of a `Model` object. The model object includes information about the models of a toggle switch. The model is implemented in SBML using a continuous modelling framework. The source property shows the physical location of the SBML model, in a model repository.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:Model rdf:about="http://www.sbolstandard.org/examples/toogleswicth">
    <sbol:source rdf:resource="http://virtualparts.org/part/pIKE_Toggle_1"/>
    <sbol:language rdf:resource="http://identifiers.org/edam/format_2585"/>
    <sbol:framework rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000062"/>
    <sbol:role rdf:resource="http://sbols.org/v2#module_model"/>
  </sbol:Model>
</rdf:RDF>
```

## 7.10 Collection

The `Collection` class is another top level class, which groups together `TopLevel` objects that have something in common. For example, a `Collection` object could be the result of a query to find all `Component` objects that function as promoters or all `Module` objects that function as inverters in a given repository.

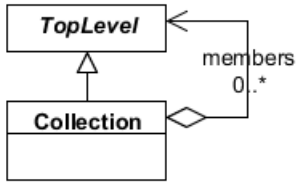


***Figure 18:*** *The Collection class*

### *The member property*

The member property has a data type of `URI` and has the `identity` value of a `TopLevel` entity. Each collection must have at least one member.

The serialization of `Collection` objects has the following form:

```
<sbol:Collection rdf:about="...">
            ...
  one or more <sbol:member rdf:resource="..."/> element
</sbol:Collection>
```

The example below shows the serialization of a `Collection` object. Promoters from a library of constitutive promoters are grouped using the collection example.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:sbol="http://sbols.org/v2#">
```

```
  <sbol:Collection rdf:about="http://parts.igem.org/Promoters/Catalog/Anderson">
    <sbol:displayId>Anderson</sbol:displayId>
    <sbol:name>Anderson promoters</sbol:name>
    <sbol:description> The Anderson promoter collection</sbol:description>
    <sbol:member rdf:resource="http://parts.igem.org/wiki/index.php/Part:BBa_J23119"/>
    ...
    <sbol:member rdf:resource="http://parts.igem.org/wiki/index.php/Part:BBa_J23118"/>
  </sbol:Collection>
</rdf:RDF>
```

## 7.11 Application Specific Data - Annotations

### *7.11.1 Annotating SBOL objects*

SBOL allows embedding application specific data that are not captured by the SBOL standard. Such data are optional, can be computationally generated and exchanged via SBOL documents without getting lost. These custom data are stored in the form of annotations, providing informative metadata about entities in an SBOL document.

Each `Identified` object may have a number of annotations in the form of name/value property pairs. Property names are specified by qualified names as `IRI`s, each formed of a namespace and a local name. Values can be `IRI`s or `Literal`s (for example, `String`, `Integer`, `Double`, `Boolean`) or custom `identity` entities initialized with application specific types. These custom `identity` entities can further be annotated with the scheme described here. These custom entities are either serialized within an SBOL entity being annotated, or referenced using an `IRI` annotation and embedded within the the annotated entity's parent.

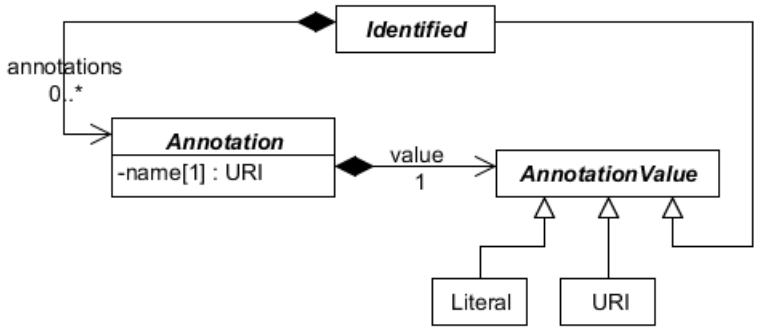


***Figure 19:*** *Annotating SBOL's Identified entities with application specific data.*

Each annotation is serialised as an RDF subject-property-object triplet in which the subject is the SBOL object being annotated, the property is the annotation name, and the object is the annotation value. Simple values URIs are serialised as RDF literals, and URI values are represented with the `http://www.w3.org/1999/02/22-rdf-syntax-ns#resource` RDF property. If the annotation value is another complex object then the object is embedded as an RDF resource, which can further be annotated similarly.

The ComponentDefinition example for a promoter below shows how annotations can be added to SBOL objects. Annotations are added using the relevant information from the Parts Registry. Annotation property names are qualified with the `http://www.partsregistry.org/` namespace, which is prefixed using `pr`. The first annotation is named as `pr:group`, indicating the iGEM group designing the promoter, and has a `String` value. The second `pr:experience` annotation has a `URI` value and is serialised as an RDF resource pointing to the information Web page on the Parts Registry for the promoter. The `pr:information` property represents a complex annotation which is a type of `pr:Information` and includes information about the regulatory details of the promoter using Parts

Registry categories.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:pr="http://www.partsregistry.org/" xmlns:sbol="http://
      sbols.org/v2#">
  <sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/Part:BBa_J23119">
    <pr:group>iGEM2006_Berkeley</pr:group>
    <pr:experience rdf:resource="http://www.partsregistry.org/Part:BBa_J23119:Experience"/>
    <pr:information>
      <pr:Information rdf:about="http://parts.igem.org/cgi/partsdb/part_info.cgi?part_name=BBa_J23119">
        <pr:sigmafactor>//rnap/prokaryote/ecoli/sigma70</pr:sigmafactor>
        <pr:regulation>//regulation/constitutive</pr:regulation>
      </pr:Information>
    </pr:information>
    <sbol:name>J23119</sbol:name>
    <sbol:description>Constitutive promoter</sbol:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://purl.org/obo/owl/SO#SO_0000167"/>
  </sbol:ComponentDefinition>
</rdf:RDF>
```

### 7.11.2 GenericTopLevel

SBOL documents can also be annotated at the top level. SBOL's `GenericTopLevel` is a top-level entity that can include a set of annotations as described above. Entities that are at the top-level and are not recognised by the SBOL standard are loaded into these top level entities. These `GenericTopLevel` entities may have multiple type information and can be used safely by tools to exchange custom data. As with any other top level entities, `GenericTopLevel` entities may include SBOL properties such as `name`, `description`, `displayId` and so on. The type of the generic entity is indicated using the RDF's `type` property.

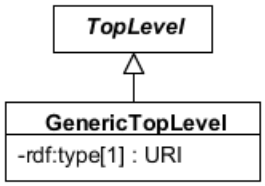


***Figure 20:*** *Annotating SBOL documents with GenericTopLevel entities.*

The example below shows adding a datasheet object to a SBOL document using the `GenericTopLevel` class. The J23119 promoter example is annotated with the URI of a top Level Datasheet object. The annotation properties are defined using the custom `http://www.myapp.org/` namespace and the `myapp` prefix. The datasheet object, with the data type of `myapp:Datasheet`, is accessed using the `URI` value specified by the `myapp:characterizationData` property of the promoter component definition. The datasheet object is further annotated with the transcription rate and the URI for the actual characterization data using the `myapp:transcriptionRate` and `myapp:characterizationData` properties respectively.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:myapp="http://www.myapp.org/" xmlns:sbol="http://sbols.org/
      v2#">
  <sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/Part:BBa_J23119">
    <myapp:datasheet rdf:resource="http://www.myapp.org/datasheet/1"/>
    <sbol:name>J23119</sbol:name>
    <sbol:description>Constitutive promoter</sbol:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://purl.org/obo/owl/SO#SO_0000167"/>
  </sbol:ComponentDefinition>
  <myapp:Datasheet rdf:about="http://www.myapp.org/datasheet/1">
    <myapp:characterizationData rdf:resource="http://www.myapp.org/measurement/1"/>
    <myapp:transcriptionRate>1</myapp:transcriptionRate>
    <sbol:name>Datasheet 1</sbol:name>
  </myapp:Datasheet>
</rdf:RDF>
```

# 8 Examples of Data Model

## 8.1 LacI/TetR Toggle Switch

As an example of structural representation under the SBOL 2.0 data model, Figure 21 specifies the component definitions for one half of a LacI/TetR toggle switch in a UML object diagram. These include DNA component definitions based on parts from the iGEM Registry, the RNA and protein component definitions derived from them, and the small molecule component definition for IPTG. With the exception of the latter, all component definitions in this example are associated with a sequence, though a SMILES encoding could be provided for IPTG if that level of detail is required.

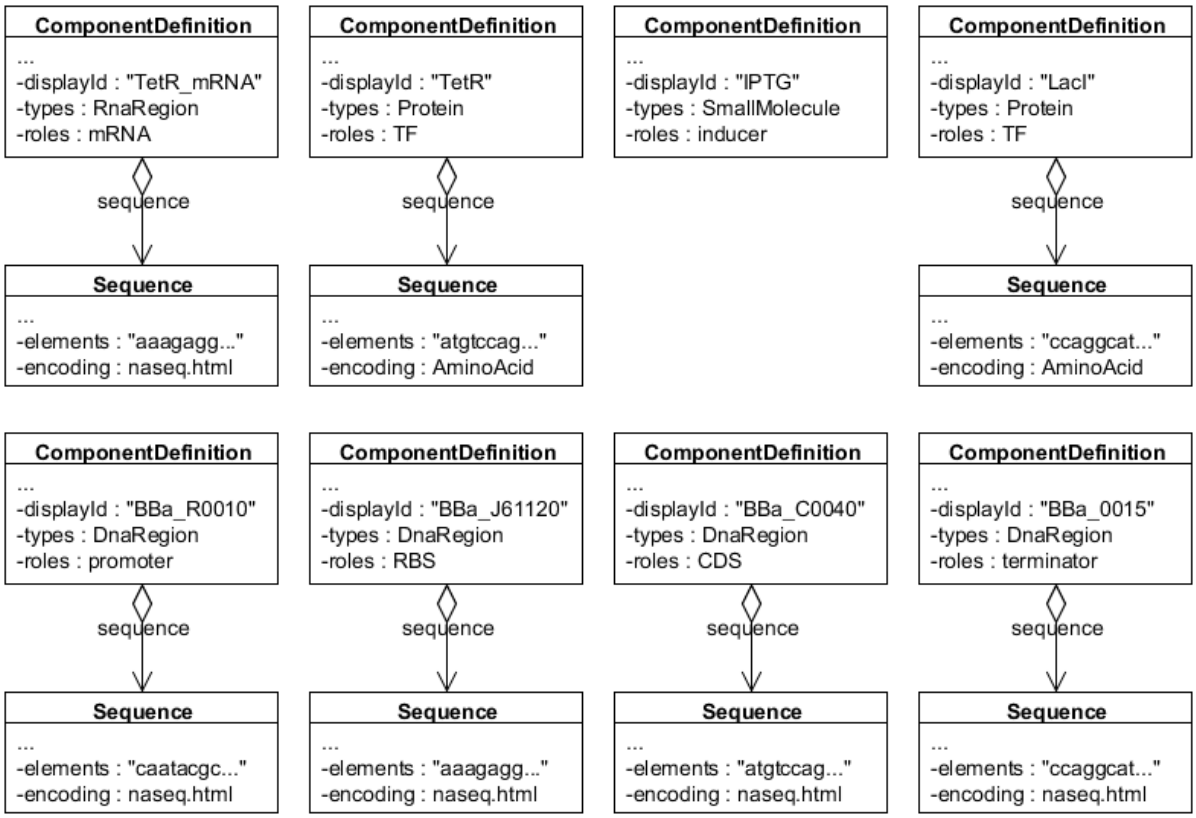


***Figure 21:*** *Examples of component definitions for one half of a LacI/TetR toggle switch.*

As an example of structural composition, Figure 22 specifies the composite component definitions for the IPTG-LacI complex and LacI-repressible TetR gene. In the case of the gene, its subcomponents are positioned as ranges along its sequence using sequence annotations. The complex, however, has no sequence and its subcomponents are aggregated without any data about their relative positions.

As an example of functional representation, Figure 25 specifies the module definition for a LacI inverter. This module definition aggregates and instantiates the component definitions for the LacI-repressible TetR as functional components that participate in interactions. Note that the transcription and translation of TetR is represented using a single genetic production interaction that abstracts away the presence of the intermediate TetR mRNA. If this additional detail becomes necessary, then a new module could be created that includes both transcription and

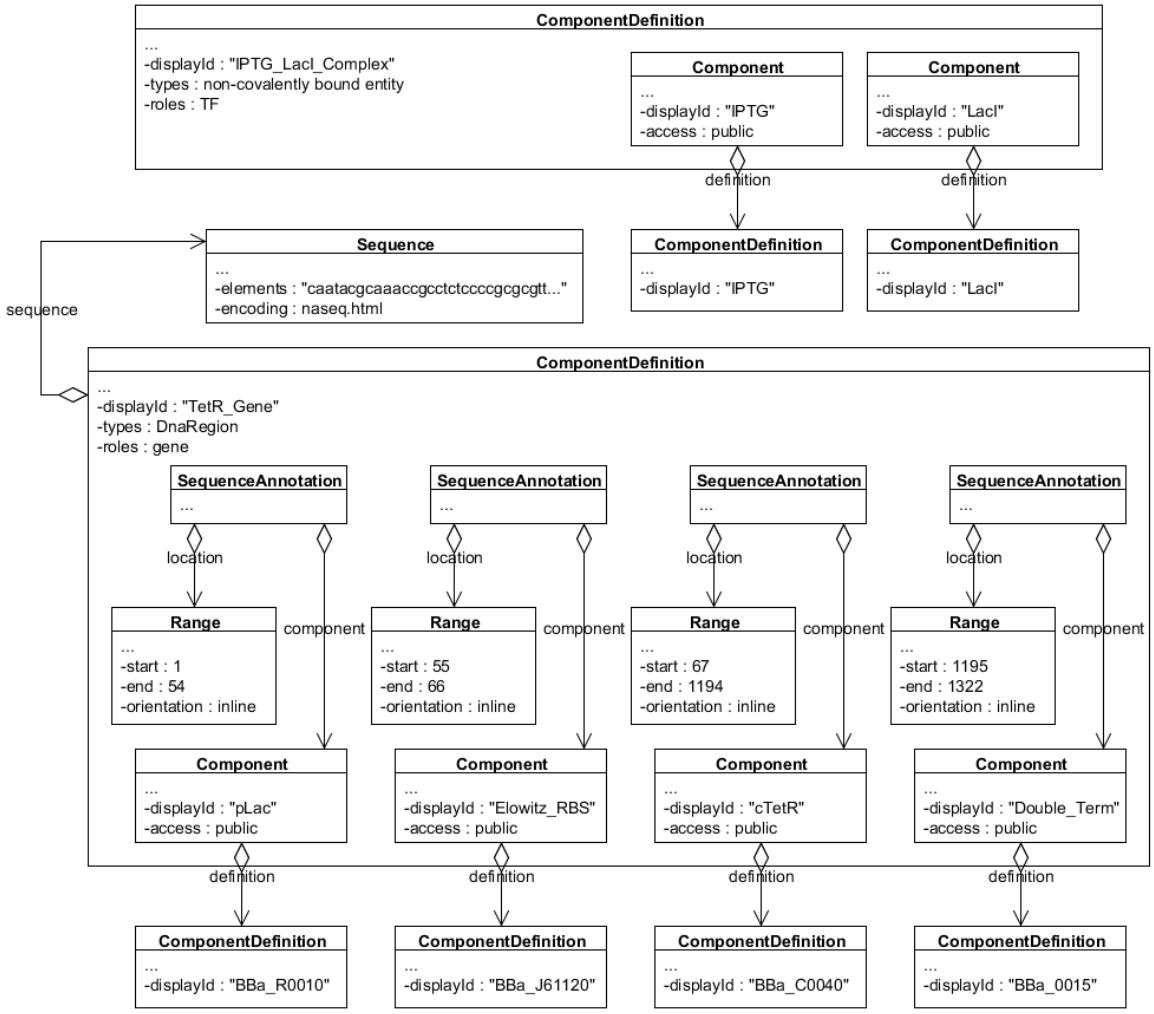


***Figure 22:*** *Example of composite component definitions for one half of a LacI/TetR toggle switch.*

translation interactions and a TetR mRNA functional component. Finally, the module definition is also associated with a continuous SBML model from source file LacI_Inverter.xml.

As an example of functional composition, **??** specifies the composite module definition for the LacI/TetR toggle switch. This module definition aggregates and instantiates the LacI and TetR inverter module definitions as submodules. It also instantiates the LacI and TetR component definitions as the functional component inputs and/or outputs to both the toggle switch and inverter module definitions. To complete the functional composition of the toggle switch, mappings are made between these functional components to indicate that the output of the LacI inverter is the input to the TetR inverter and vice versa

In this example, the output of the LacI inverter is an input of the TetR inverter and vice versa. Also, both inverters accept the instantiation of a small molecule component as input, IPTG in the case of the LacI inverter and aTc in the case of the TetR inverter.

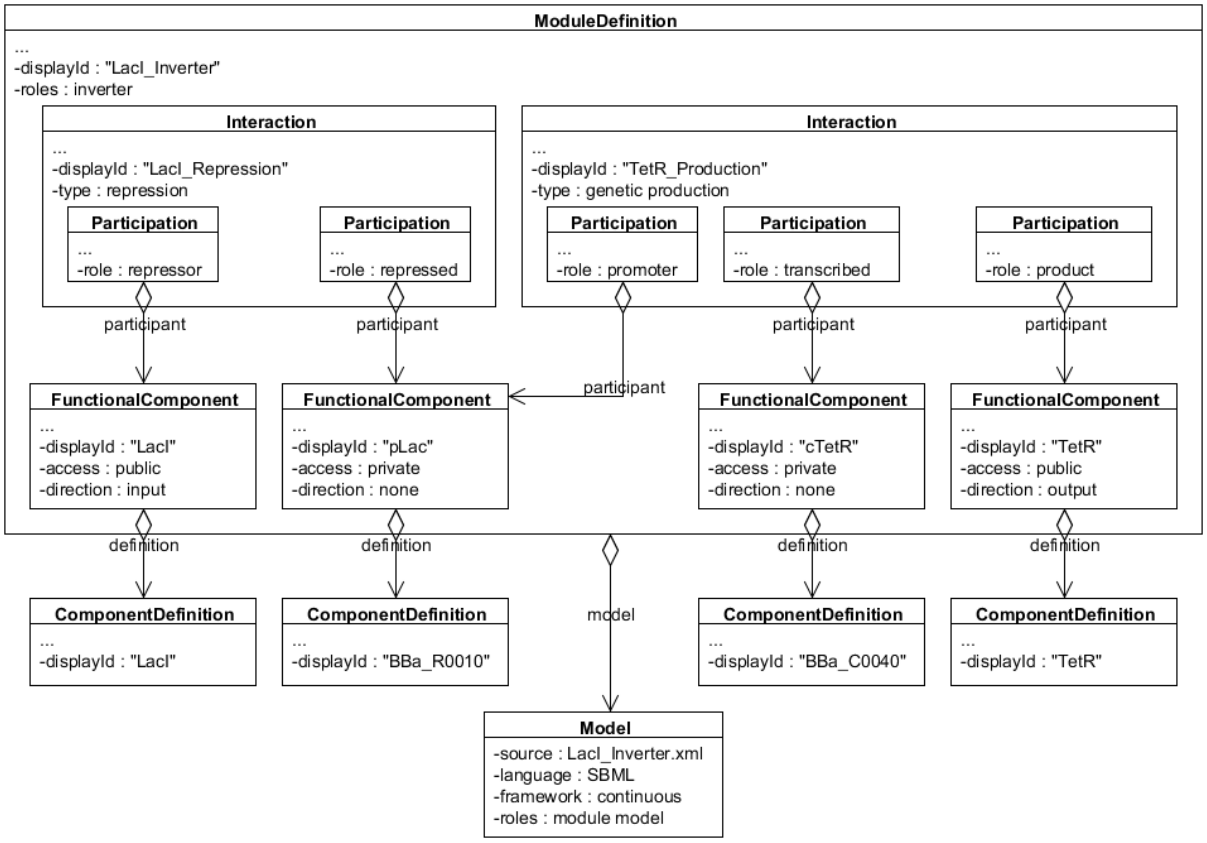


***Figure 23:*** *Example of module definition for a LacI inverter.*

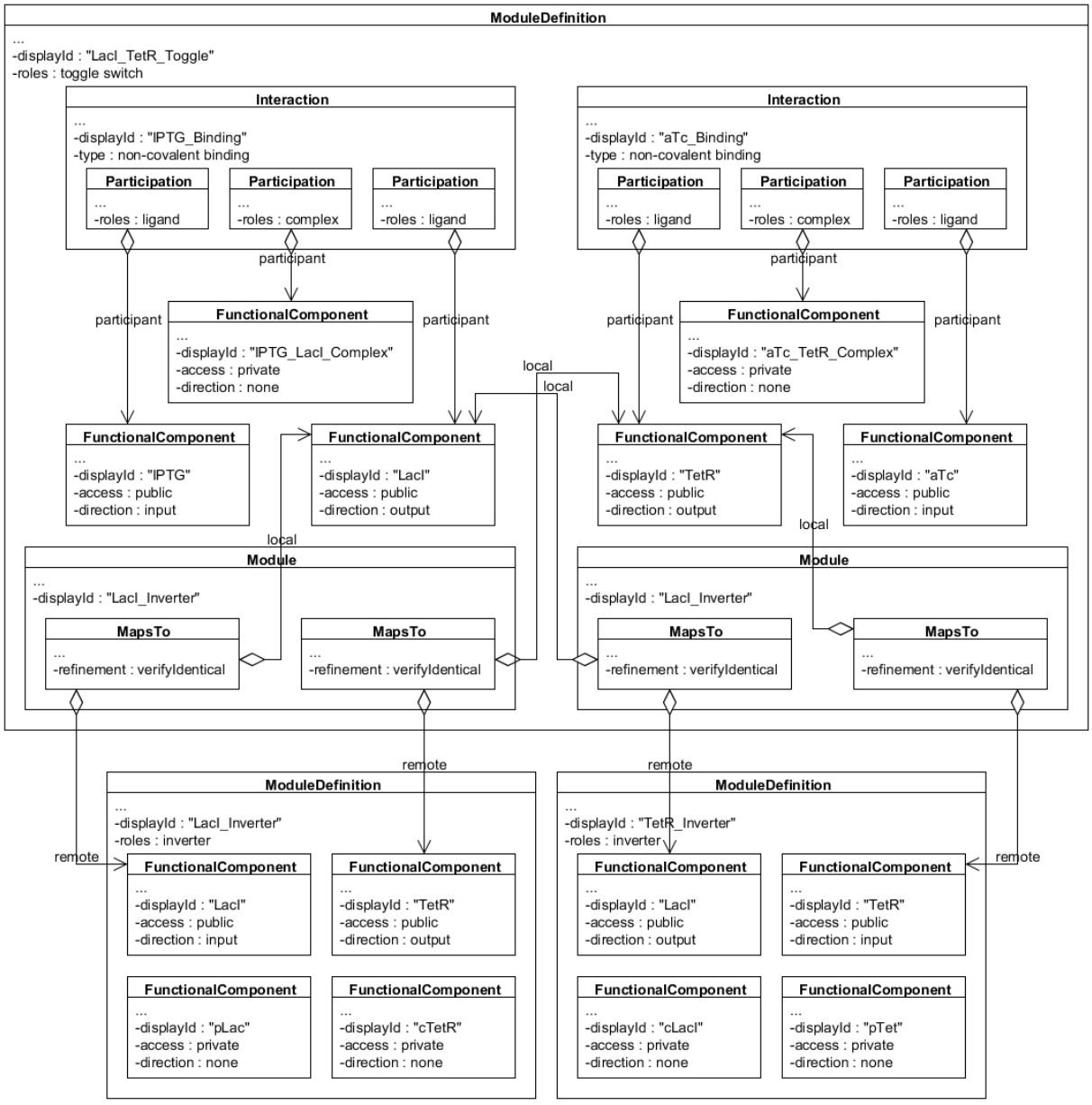


***Figure 24:*** *Example of composing the module definitions for a LacI inverter and TetR inverter into a module definition for a LacI/TetR toggle switch.*

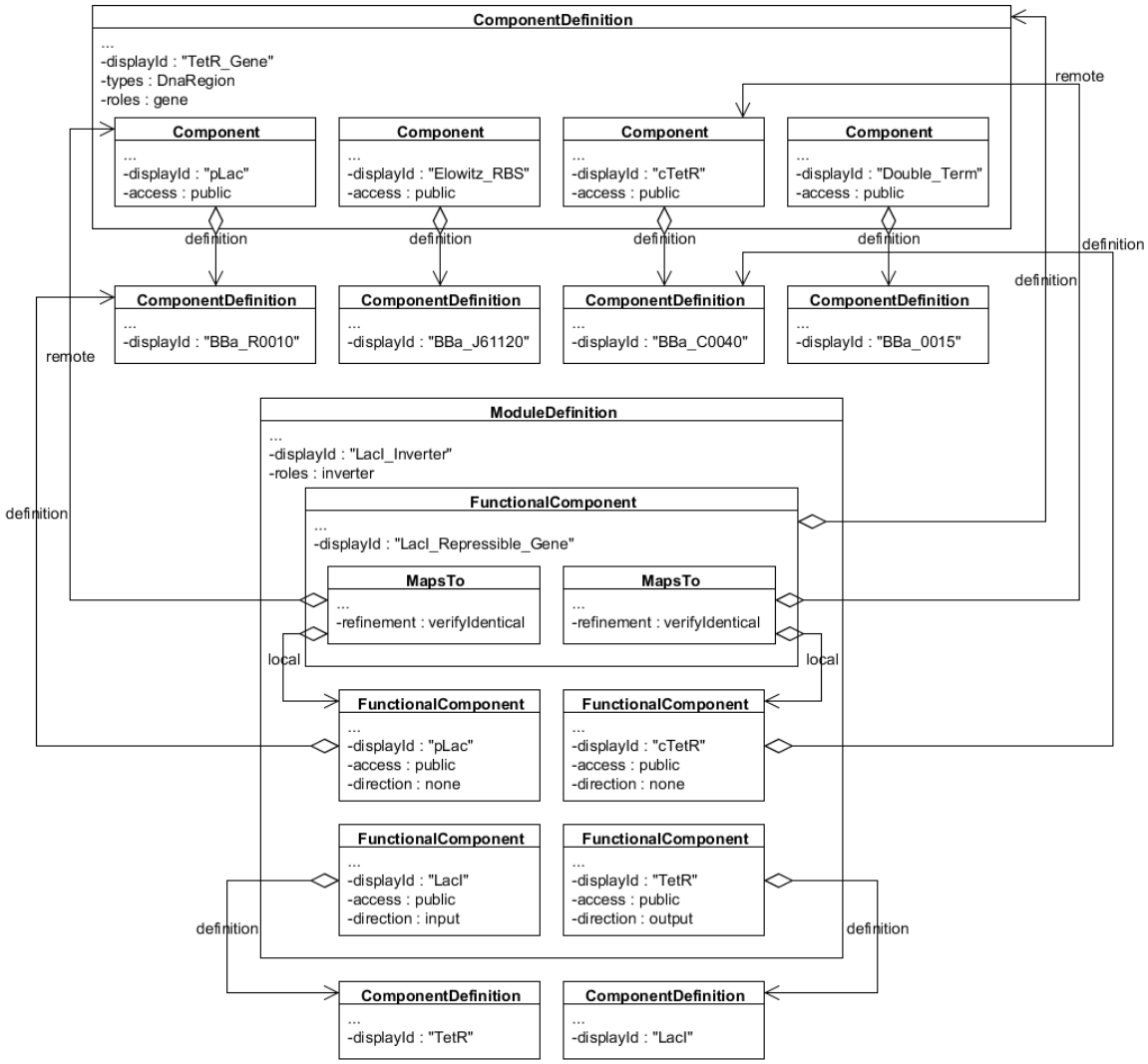


***Figure 25:*** *Example of mapping between the structural and functional components of a LacI inverter.*

# 9 Examples of Serialization

This example shows the serialization of the PoPS Receiver device designed by Canton and co-workers. Details of the device can be found at http://parts.igem.org/Part:BBa_F2620.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:pr="http://www.partsregistry.org/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://
      purl.org/dc/terms/" xmlns:sbol="http://sbols.org/v2#">
  <sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/Part:BBa_R0040">
    <dcterms:title>BBa_R0040</dcterms:title>
    <dcterms:description>TetR repressible promoter</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
    <sbol:sequence rdf:resource="http://www.partsregistry.org/Part:BBa_F2620:Design"/>
  </sbol:ComponentDefinition>
  <sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/Part:BBa_C0062">
    <dcterms:title>BBa_C0062</dcterms:title>
    <dcterms:description>luxR coding sequence</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000316"/>
    <sbol:sequence rdf:resource="http://www.partsregistry.org/Part:BBa_C0062:Design"/>
  </sbol:ComponentDefinition>
  <sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/Part:BBa_B0015">
    <dcterms:title>BBa_B0015</dcterms:title>
    <dcterms:description>Double terminator</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000141"/>
    <sbol:sequence rdf:resource="http://www.partsregistry.org/Part:BBa_B0015:Design"/>
  </sbol:ComponentDefinition>
  <sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/Part:BBa_R0062">
    <dcterms:title>BBa_R0062</dcterms:title>
    <dcterms:description>LuxR inducible promoter</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
    <sbol:sequence rdf:resource="http://www.partsregistry.org/Part:BBa_R0062:Design"/>
  </sbol:ComponentDefinition>
  <sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/Part:BBa_F2620">
    <dcterms:title>BBa_F2620</dcterms:title>
    <dcterms:description>3OC6HSL -&gt; PoPS Receiver</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:00001411"/>
    <sbol:subComponent>
      <sbol:Component rdf:about="http://www.partsregistry.org/Part:BBa_F2620/pLuxR">
        <sbol:access rdf:resource="http://sbols.org/v2#public"/>
        <sbol:definition rdf:resource="http://www.partsregistry.org/Part:BBa_R0062"/>
      </sbol:Component>
    </sbol:subComponent>
    <sbol:subComponent>
      <sbol:Component rdf:about="http://www.partsregistry.org/Part:BBa_F2620/pTetR">
        <sbol:access rdf:resource="http://sbols.org/v2#public"/>
        <sbol:definition rdf:resource="http://www.partsregistry.org/Part:BBa_R0040"/>
      </sbol:Component>
    </sbol:subComponent>
    <sbol:subComponent>
      <sbol:Component rdf:about="http://www.partsregistry.org/Part:BBa_F2620/ter">
        <sbol:access rdf:resource="http://sbols.org/v2#public"/>
        <sbol:definition rdf:resource="http://www.partsregistry.org/Part:BBa_B0015"/>
      </sbol:Component>
    </sbol:subComponent>
    <sbol:subComponent>
      <sbol:Component rdf:about="http://www.partsregistry.org/Part:BBa_F2620/luxR">
        <sbol:access rdf:resource="http://sbols.org/v2#public"/>
        <sbol:definition rdf:resource="http://www.partsregistry.org/Part:BBa_B0034"/>
      </sbol:Component>
    </sbol:subComponent>
    <sbol:subComponent>
```

```
    <sbol:Component rdf:about="http://www.partsregistry.org/Part:BBa_F2620/rbs">
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.partsregistry.org/Part:BBa_C0062"/>
    </sbol:Component>
  </sbol:subComponent>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno2">
      <sbol:location>
        <sbol:Range rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno2/range">
          <sbol:start>56</sbol:start>
          <sbol:end>68</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.partsregistry.org/Part:BBa_F2620/rbs"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno4">
      <sbol:location>
        <sbol:Range rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno4/range">
          <sbol:start>771</sbol:start>
          <sbol:end>900</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.partsregistry.org/Part:BBa_F2620/ter"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno1">
      <sbol:location>
        <sbol:Range rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno1/range">
          <sbol:start>1</sbol:start>
          <sbol:end>55</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.partsregistry.org/Part:BBa_F2620/pTetR"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno3">
      <sbol:location>
        <sbol:Range rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno3/range">
          <sbol:start>69</sbol:start>
          <sbol:end>770</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.partsregistry.org/Part:BBa_F2620/luxR"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno5">
      <sbol:location>
        <sbol:Range rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno5/range">
          <sbol:start>901</sbol:start>
          <sbol:end>956</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.partsregistry.org/Part:BBa_F2620/pLuxR"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequence rdf:resource="http://www.partsregistry.org/Part:BBa_R0040:Design"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/Part:BBa_B0034">
  <dcterms:title>BBa_B0034</dcterms:title>
```

```
    <dcterms:description>RBS based on Elowitz repressilator</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000139"/>
    <sbol:sequence rdf:resource="http://www.partsregistry.org/Part:BBa_B0034:Design"/>
  </sbol:ComponentDefinition>
  <sbol:Sequence rdf:about="http://www.partsregistry.org/Part:BBa_B0015:Design">
    <sbol:elements>
      ccaggcatcaaataaaacgaaaggctcagtcgaaagactgggcctttcgttttatctgttgtttgtcggtgaacgctctctactagagtcacactggctcaccttcgggtgggcctttctgcgtttata
      </sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://www.partsregistry.org/Part:BBa_B0034:Design">
    <sbol:elements>aaagaggagaaa</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://www.partsregistry.org/Part:BBa_C0062:Design">
    <sbol:elements>
      atgcttatctgatatgactaaaatggtacattgtgaatattatttactcgcgatcatttatcctcattctatggttaaatctgatatttcaatcctagataattaccctaaaaaatggaggcaatattatgatgacgctaatttaa
      </sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://www.partsregistry.org/Part:BBa_R0040:Design">
    <sbol:elements></sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://www.partsregistry.org/Part:BBa_R0062:Design">
    <sbol:elements>acctgtaggatcgtacaggtttacgcaagaaaatggtttgttatagtcgaataaa</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://www.partsregistry.org/Part:BBa_F2620:Design">
    <sbol:elements>tccctatcagtgatagagattgacatccctatcagtgatagagatactgagcac</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
</rdf:RDF>
```

# 10 Best Practices

## 10.1 Versioning

Currently, if a developer wishes to change a SBOL object that has been published to the Web, then as a best practice they should create a copy of the SBOL object that incorporates the change, but has a new URI. This practice, however, does not inherently involve a standardized declaration that the second object is a version of the first. Consequently, the `persistentIdentity`, and `version` data fields have been created to provide developers with the means to declare that a set of SBOL objects are versions of each other (by virtue of having the same persistent URI) and label these objects with version Strings.

## 10.2 Using External Terms

External ontologies and controlled vocabularies are integral part of SBOL. Through these resources SBOL utilises existing biological information where possible. New SBOL specific terms are defined only when necessary. Instead, SBOL provides placeholders that can point to external terms. For example, the types of components whether they are DNA or protein based are indicated using BioPAX. Similarly, the functionality of a DNA component is determined via the SO terms. Although preferred ontologies have been explained in relevant sections where possible, other resources providing similar terms can also be used. A summary of these external sources can be found at Table 8.

| SBOL Entity | Property | Preferred External Resource |
|---|---|---|
| **ComponentDefinition** | type | BioPAX |
| | role | SO (if type is *DNA* or *RNA*) |
| | role | CHEBI (if type is *small molecule*) |
| | role | UniProt (if type is *protein*??) |
| **Interaction** | type | SBO |
| **Participation** | type | SBO |
| **Model** | language | EDAM |
| | framework | SBO |

***Table 8:*** *SBOL properties and preferred external resources to choose values from.*

# References


Finney, A., Hucka, M., Bornstein, B. J., Keating, S. M., Shapiro, B. M., Matthews, J., Kovitz, B. K., Schilstra, M. J., Funahashi, A., Doyle, J., and Kitano, H. (2006). Software infrastructure for effective communication and reuse of computational models. In Szallasi, Z., Stelling, J., and Periwal, V., editors, *System Modeling in Cell Biology: From Concepts to Nuts and Bolts*. MIT Press.

Garny, A., Nickerson, D., Cooper, J., dos Santos, R. W., Miller, A., McKeever, S., Nielsen, P., and Hunter, P. (2008). Cellml and associated tools and techniques. *Philos. Transact. A: Math Phys Eng Sci*, 366(1878):3017–43.

MathWorks (2015). Matlab.