A brief description of the CRISPR circuit using SBOL 2.0 data model

We first give a brief description of the CRISPR-based repression module. We use bold font in the following text and figure captions to mark available data model in SBOL 2.0. Detailed description of properties of the data model is available in the Specification (Data Model 2.0).

First, consider the CRISPR-based Repression Template **ModuleDefinition** shown in the center of Figure 1. It provides a generic description of CRISPR-based repression behavior. Namely, it includes generic *Cas9*, *guide RNA* (gRNA), and *target* DNA **FunctionalComponent** instances. It also includes a *genetic production* **Interaction** that expresses a generic target gene product. Finally, it includes a *non-covalent binding* **Interaction** that forms the Cas9/gRNA complex (shown as dashed arrows), which in turn participates in an *inhibition* **Interaction** to repress the target gene product production (shown with a tee-headed arrow). The CRISPR-based Repression Template is then instantiated to test a particular CRISPR-based repression device, CRPb, by the outer CRPb Characterization Circuit **ModuleDefinition**. This outer characterization circuit includes gene **FunctionalComponents** to produce specific products (i.e., mKate, Gal4VP16, cas9m_BFP, gRNA_b, and EYFP), as well as **FunctionalComponents** for the products themselves. Next, it includes *genetic production* **Interactions** connecting the genes to their products, and it has a *stimulation* **Interaction** that indicates that Gal4VP16 stimulates production of EYFP. Finally, it uses **MapsTo** objects (shown as dashed lines) to connect the generic **FunctionalComponents** in the template to the specific objects in the outer **ModuleDefinition**. For example, the outer module indicates that the target protein is EYFP, while the cas9_gRNA complex is cas9m_BFP_gRNA_b.

Modeling CRISPR repression using `libSBOL 2.0`

Creating SBOL Document

All SBOL data objects are organized within an **SBOLDocument** object. The **SBOLDocument** provides a rich set of methods to create, access, update, and delete each type of **TopLevel** object (i.e., **Collection**, **ModuleDefinition**, **ComponentDefinition**, **Sequence**, **Model**, or **GenericTopLevel**). Every SBOL object has a *uniform resource identifier* (URI) and consists of properties that may refer to other objects, including non-**TopLevel** objects such as SequenceConstraint and Interaction objects. `libSBOLj 2.0` organizes the URI collections to enable efficient access, and validation of uniqueness. We first create an **SBOLDocument** object by calling its constructor as shown below.

```
1 Document& doc = *new Document();
2 doc.setHomespace("http://sys-bio.org");
```

The method `setHomespace` sets the default URI prefix to the string "http://sys-bio.org". All data objects created following this statement carry this default URI prefix. The author of any SBOL object, such as the should use a URI prefix that either they own or an organization of which they are a member owns. Setting a deafult namespace is like signing your homework!

Adding CRISPR-based Repression Template module

Creating TopLevel objects

We first create the CRISPR-based Repression Template module shown in Figure 1. In this template, we include definitions for generic *Cas9*, *guide RNA* (gRNA), and *target* DNA **FunctionalComponent** instances. They are encoded as **ComponentDefinition** objects. Creation of the generic Cas9 (line 3) **ComponentDefinition** is done by passing its *displayId* "cas9_generic". The displayId is appended to the default namespace to create the URI "http://sys-bio.org/cas9_generic. The next argument is the required field, emphtype. Every **ComponentDefinition** must contain one or more types, each of which is specified by a URI. A type specifies the component's category of biochemical or physical entity (for example DNA, protein, or small molecule). The generic Cas9's type is `BIOPAX_PROTEIN` (`http://www.biopax.org/release/biopax-level3.owl#Protein`), which is defined as the BioPAX ontology term for protein. Finally, an optional *version* specified by the `version` string may be specied. If *version* is not specified, it will be set by default to Ï.0.0. Other **ComponentDefinition** objects shown below are created in the same way. A **ComponentDefinition** object can optionally have one or more roles, also in the form of URIs. The `gRNA_generic` has a role of `SGRNA` (line 11 below), defined as the *Sequence Ontology* (SO) term

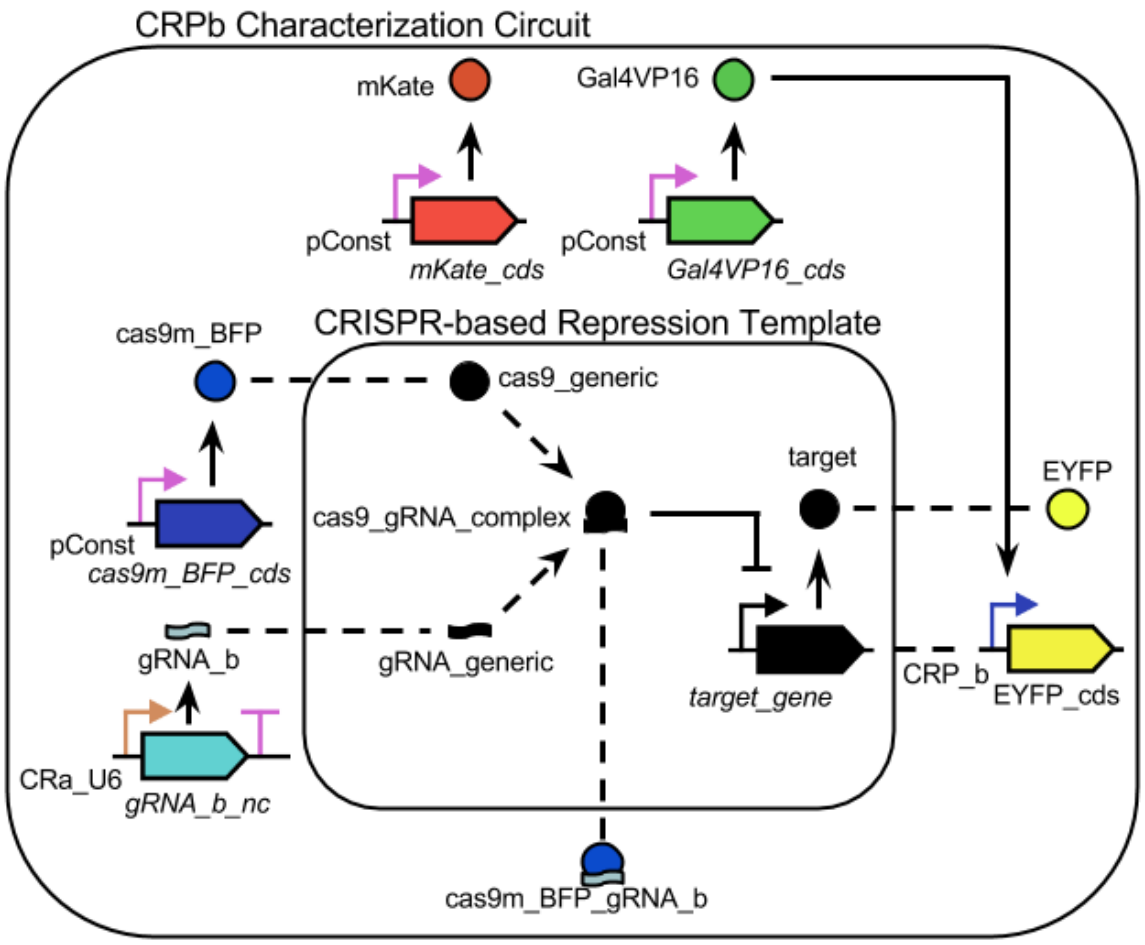

Figure 1: Illustration of a hierarchical CRISPR-based repression module represented in SBOL 2.0 (adapted from Figure 1a in [1]). The CRISPR-based Repression Template **ModuleDefinition** describes a generic CRISPR repression circuit that combines a Cas9 protein with a gRNA to form a complex (represented by the dashed arrows) that represses a target gene (represented by the arrow with the tee arrowhead). These relationships between these **FunctionalComponents** (instances of **ComponentDefinitions**) are represented in SBOL 2.0 using **Interactions**. This **Module** is instantiated in the outer CRPb Characterization Circuit **ModuleDefinition** in order to specify the precise (including **Sequences** when provided) **FunctionalComponents** used for each generic **FunctionalComponent**. The undirected dashed lines going into the template **Module** represent **MapsTo** objects that specify how specific **FunctionalComponents** replace the generic ones.

"SO:0001998". (http://identifiers.org/so/SO:0001998) in the library. Similarly, the target_gene on line 21 below has a role of PROMOTER, defined as SO term "SO:0000167" (http://identifiers.org/so/SO:0000167). We then create the **ModuleDefinition** template by constructing a ModuleDefinition with the displayId "CRISPR_Template".

```
1  String version = "1.0";
2
3  ComponentDefinition &cas9_generic = *new ComponentDefinition("cas9_generic",
       BIOPAX_PROTEIN, "1.0.0" );
4  ComponentDefinition &gRNA_generic = *new ComponentDefinition("gRNA_generic",
       BIOPAX_RNA );
5  gRNA_generic.roles.set(SO "0001998");
6  ComponentDefinition &cas9_gRNA_complex = *new ComponentDefinition("
       cas9_gRNA_complex", BIOPAX_COMPLEX );
7  ComponentDefinition &target_promoter = *new ComponentDefinition("target_promoter",
        BIOPAX_DNA);
8  ComponentDefinition &target_gene = *new ComponentDefinition("target_gene",
       BIOPAX_DNA);
9  ComponentDefinition &target = *new ComponentDefinition("target", BIOPAX_PROTEIN )
10 ModuleDefinition  &CRISPR_template = *new ModuleDefinition("CRISPRTemplate");
```

By default, libSBOL operates in S̈BOL-compliantm̈ode. In SBOL-compliant mode each constructor creates an SBOL-compliant URI with the following form:

http://⟨prefix⟩/⟨displayId⟩/⟨version⟩

using the default URI prefix and provided displayId and version. The ⟨*prefix*⟩ represents a URI for a namespace (for example, www.sys-bio.org/CRISPR_Example). When using compliant URIs, the owner of a prefix must ensure that the URI of any unique **TopLevel** object that contains the prefix also contains a unique ⟨*displayId*⟩ or ⟨*version*⟩ portion. Multiple versions of an SBOL object can exist and would have compliant URIs that contain identical prefixes and displayIds, but each of these URIs would need to end with a unique version. Lastly, the compliant URI of a non-**TopLevel** object is identical to that of its parent object, except that its displayId is inserted between its parent's displayId and version. This form of compliant URIs is chosen to be easy to read, facilitate debugging, and support a more efficient means of looking up objects and checking URI uniqueness.

Specifying Interactions

We are now ready to specify the interactions in the repression template. The first one is the complex formation interaction for cas9_generic and gRNA_generic. We first create an **Interaction** object complex_formation in CRISPR_Template, with the displayId "cas9_complex_formation" and a non-covalent binding type (line 1 to 4). It is recommended that terms from the *Systems Biology Ontology* (SBO) [2] are used specify the types for interactions. Table 11 of the Specification (Data Model 2.0) document provides a list of possible SBO terms for the types property and their corresponding URIs.

Next, we create three participants to this interaction object. Each participant represents a species participating in a biochemical reaction. The components which participate in an interaction must be assigned using the participate

```
1  Interaction& complex_formation = CRISPRTemplate.interactions.create("
       complex_formation");
2  complex_formation.types.add(SBO_NONCOVALENT_BINDING);
3  Participation& A = complex_formation.participations.create("A");
4  Participation& B = complex_formation.participations.create("B");
5  Participation& AB = complex_formation.participations.create("AB");
6  A.roles.add(SBO_REACTANT);
7  B.roles.add(SBO_REACTANT);
8  AB.roles.add(SBO_PRODUCT);
9
10 cas9_generic.participate(A);
11 grna_generic.participate(B);
12 cas9_gRNA_complex.participate(AB);
```

method.

The remaining two interactions, namely the genetic production of the target protein from the `target_gene` and the inhibition of the target protein by the `cas9_gRNA_complex`, are specified using the same method calls.

```
1  // Gene repression
2  Participation& repressor = complex_formation.participations.create("repressor");
3  Participation& repression_target = complex_formation.participations.create("
       repression_target");
4  repressor.roles.add(SBO_INHIBITOR);
5  repression_target.roles.add(SBO_BINDING_SITE);
6
7  // Gene production
8  Participation& promoter = gene_production.participations.create("promoter");
9  Participation& gene = gene_production.participations.create("gene");
10 Participation& gene_product = gene_production.participations.create("gene_product"
       );
11 promoter.roles.add(SBO_PROMOTER);
12 gene.roles.add(SBO_GENE);
13 gene_product.roles.add(SBO_PRODUCT);
14
15 cas9_gRNA_complex.participate(repressor);
16 target_promoter.participate(repression_target);
17 target_promoter.participate(promoter);
18 target_gene.participate(gene);
19 target.participate(gene_product);
```

Creating CRPb Characterization Circuit

So far, we have completed the repression template. In order to construct the the CRPb Characterization Circuit, we must realize the template with specific components. We first create **Sequence** objects for those provided in [1][1] as shown in the code below. For example, to create the sequence for the CRP_b promoter, we call the `Sequence` constructor, as shown on line 57, with the displayId "CRP_b_seq", `version`, the sequence specified by `CRP_b_seq_elements` , and the IUPAC encoding for DNA, which is defined as a URI in the **Sequence** class, referencing http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html.

[1] Unfortunately, as usual, not all sequences are provided in the paper.

```
1     // Create Sequence for CRa_U6 promoter
2     string CRa_U6_seq_elements = "GGTTTACCGAGCTCTTATTGGTTTTCAAACTTCATTGACTGTGCC"
3         "AAGGTCGGGCAGGAAGAGGGCCTATTTCCCATGATTCCTTCATAT"
4         "TTGCATATACGATACAAGGCTGTTAGAGAGATAATTAGAATTAAT"
5         "TTGACTGTAAACACAAAGATATTAGTACAAAATACGTGACGTAGA"
6         "AAGTAATAATTTCTTGGGTAGTTTGCAGTTTTAAAATTATGTTTT"
7         "AAAATGGACTATCATATGCTTACCGTAACTTGAAATATAGAACCG"
8         "ATCCTCCCATTGGTATATATTATAGAACCGATCCTCCCATTGGCT"
9         "TGTGGAAAGGACGAAACACCGTACCTCATCAGGAACATGTGTTTA"
10        "AGAGCTATGCTGGAAACAGCAGAAATAGCAAGTTTAAATAAGGCT"
11        "AGTCCGTTATCAACTTGAAAAAGTGGCACCGAGTCGGTGCTTTTT"
12        "TTGGTGCGTTTTTATGCTTGTAGTATTGTATAATGTTTTT";
13
14    // Create Sequence for gRNA_b coding sequence
15    string gRNA_b_elements = "AAGGTCGGGCAGGAAGAGGGCCTATTTCCCATGATTCCTTCATAT"
16        "TTGCATATACGATACAAGGCTGTTAGAGAGATAATTAGAATTAAT"
17        "TTGACTGTAAACACAAAGATATTAGTACAAAATACGTGACGTAGA"
18        "AAGTAATAATTTCTTGGGTAGTTTGCAGTTTTAAAATTATGTTTT"
19        "AAAATGGACTATCATATGCTTACCGTAACTTGAAAGTATTTCGAT"
20        "TTCTTGGCTTTATATATCTTGTGGAAAGGACGAAACACCGTACCT"
21        "CATCAGGAACATGTGTTTAAGAGCTATGCTGGAAACAGCAGAAAT"
22        "AGCAAGTTTAAATAAGGCTAGTCCGTTATCAACTTGAAAAAGTGG"
23        "CACCGAGTCGGTGCTTTTTTT";
24
25    // Create Sequence for mKate
26    string mKate_seq_elements = "TCTAAGGGCGAAGAGCTGATTAAGGAGAACATGCACATGAAGCTG"
27        "TACATGGAGGGCACCGTGAACAACCACCACTTCAAGTGCACATCC"
28        "GAGGGCGAAGGCAAGCCCTACGAGGGCACCCAGACCATGAGAATC"
29        "AAGGTGGTCGAGGGCGGCCCTCTCCCCTTCGCCTTCGACATCCTG"
30        "GCTACCAGCTTCATGTACGGCAGCAAAACCTTCATCAACCACACC"
31        "CAGGGCATCCCCGACTTCTTTAAGCAGTCCTTCCCTGAGGTAAGT"
32        "GGTCCTACCTCATCAGGAACATGTGTTTTAGAGCTAGAAATAGCA"
33        "AGTTAAAATAAGGCTAGTCCGTTATCAACTTGAAAAAGTGGCACC"
34        "GAGTCGGTGCTACTAACTCTCGAGTCTTCTTTTTTTTTTTCACAG"
35        "GGCTTCACATGGGAGAGAGTCACCACATACGAAGACGGGGGCGTG"
36        "CTGACCGCTACCCAGGACACCAGCCTCCAGGACGGCTGCCTCATC"
37        "TACAACGTCAAGATCAGAGGGGTGAACTTCCCATCCAACGGCCCT"
38        "GTGATGCAGAAGAAAACACTCGGCTGGGAGGCCTCCACCGAGATG"
39        "CTGTACCCCGCTGACGGCGGCCTGGAAGGCAGAAGCGACATGGCC"
40        "CTGAAGCTCGTGGGCGGGGGCCACCTGATCTGCAACTTGAAGACC"
41        "ACATACAGATCCAAGAAACCCGCTAAGAACCTCAAGATGCCCGGC"
42        "GTCTACTATGTGGACAGAAGACTGGAAAGAATCAAGGAGGCCGAC"
43        "AAAGAGACCTACGTCGAGCAGCACGAGGTGGCTGTGGCCAGATAC"
44        "TGCG";
45
46    // Create Sequence for CRP_b promoter
47    string CRP_b_seq_elements =  "GCTCCGAATTTCTCGACAGATCTCATGTGATTACGCCAAGCTACG"
48        "GGCGGAGTACTGTCCTCCGAGCGGAGTACTGTCCTCCGAGCGGAG"
49        "TACTGTCCTCCGAGCGGAGTACTGTCCTCCGAGCGGAGTTCTGTC"
50        "CTCCGAGCGGAGACTCTAGATACCTCATCAGGAACATGTTGGAAT"
51        "TCTAGGCGTGTACGGTGGGAGGCCTATATAAGCAGAGCTCGTTTA"
52        "GTGAACCGTCAGATCGCCTCGAGTACCTCATCAGGAACATGTTGG"
53        "ATCCAATTCGACC";
```

Next, we specify **ComponentDefinitions** for all **FunctionalComponents** in the CRPb Characterization Circuit. The code snippet below first creates a **ComponentDefinition** of DNA type for the `CRP_b` promoter (lines 1-7). Then, we create two **ComponentDefinition** objects, one for the EYFP *coding sequence* (CDS) and another for the EYFP gene (lines 8-17). We use a **SequenceConstraint** object (lines 18-23) to indicate that the `CRP_b` promoter precedes the `EYFP_cds`, because the sequence for the CDS has not been provided and thus cannot be given an exact **Range**. The **restriction** property uses flags defined in the formal specification, which are provided in libSBOL as predefined constants. See the API documentation or the constants.h header file for predefined constants associated with an SBOL property.

Table 1: **ComponentDefinition** objects

component definition	type	role	sequence	sequence constraint
pConst	DNA	PROMOTER	n/a	n/a
cas9m_BFP_cds	BIOPAX_DNA	SO_CDS	n/a	n/a
cas9m_BFP_gene	BIOPAX_DNA	SO_PROMOTER	n/a	cas9m_BFP_gene_constraint
cas9m_BFP	BIOPAX_PROTEIN	n/a	n/a	n/a
CRa_U6	BIOPAX_DNA	SO_PROMOTER	CRa_U6_seq	n/a
gRNA_b_nc	BIOPAX_DNA	SO_CDS	gRNA_b_seq	n/a
gRNA_b_terminator	BIOPAX_DNA	SO_TERMINATOR	n/a	n/a
gRNA_b_gene	BIOPAX_DNA	SO_PROMOTER	n/a	gRNA_b_gene_constraint1 gRNA_b_gene_constraint2
gRNA_b	BIOPAX_RNA	SGRNA	n/a	n/a
cas9m_BFP_gRNA_b	BIOPAX_COMPLEX	n/a	n/a	n/a
mKate_cds	BIOPAX_DNA	SO_CDS	mKate_seq	n/a
mKate_gene	BIOPAX_DNA	SO_PROMOTER	n/a	mKate_gene_constraint
mKate	BIOPAX_PROTEIN	n/a	n/a	n/a
Gal4VP16_cds	BIOPAX_DNA	SO_CDS	n/a	n/a
Gal4VP16_gene	BIOPAX_DNA	SO_PROMOTER	n/a	GAL4VP16_gene_constraint
Gal4VP16	PROTEIN	n/a	n/a	n/a
CRP_b	BIOPAX_DNA	SO_PROMOTER	CRP_b_seq	n/a
EYFP_cds	BIOPAX_DNA	SO_CDS	n/a	n/a
EYFP_gene	BIOPAX_DNA	SO_PROMOTER	n/a	EYFP_gene_constraint
EYFP	BIOPAX_PROTEIN	n/a	n/a	n/a

Table 2: **SequenceConstraint** objects

displayId	restriction type	subject	object
cas9m_BFP_gene_constraint	SBOL_RESTRICTION_PRECEDES	pConst	cas9m_BFP_cds
gRNA_b_gene_constraint1	SBOL_RESTRICTION_PRECEDES	CRa_U6	gRNA_b_nc
gRNA_b_gene_constraint2	SBOL_RESTRICTION_PRECEDES	gRNA_b_nc	gRNA_b_terminator
mKate_gene_constraint	SBOL_RESTRICTION_PRECEDES	pConst	mKate_cds
GAL4VP16_gene_constraint	SBOL_RESTRICTION_PRECEDES	pConst	Gal4VP16_cds
EYFP_gene_constraint	SBOL_RESTRICTION_PRECEDES	CRP_b	EYFP_cds

```
ComponentDefinition& CRP_b = *new ComponentDefinition("CRP_b", BIOPAX_DNA);
CRP_b.roles.add(SO_PROMOTER);
CRP_b.sequence.set(CRP_b_seq);

ComponentDefinition& EYFP_cds = *new ComponentDefinition("EYFP_cds", BIOPAX_DNA);
CRP_b.roles.add(SO_CDS);

SequenceConstraint &EYFP_cds_constraint = EYFP_cds.sequenceConstraints.create();
SequenceConstraint.subject.set(CRP_b.identity.get());
SequenceConstraint.object.set(EYFP_cds.identity.get());
SequenceConstraint.restriction.set(SBOL_RESTRICTION_PRECEDES);
```

Other **ComponentDefinition** objects can be created using the same set of method calls. As an exercise, the reader is encouraged to specify them according to Table 1 and 2. Entries "type" and "roles" column in the table are libSBOL constants corresponding to a SequenceOntology term. URIs for these terms are described in Table 3 of the Specification (Data Model 2.0) document.

We are now ready to create the CRPb Characterization Circuit which realizes the template design. We first create a **ModuleDefinition** object as shown below:

```
ModuleDefinition &CRPb_circuit = *new ModuleDefinition("CRPb_circuit");
```

Table 3: **Interaction** objects

interaction	type	participant	role
mKate_production	SBO_GENETIC_PRODUCTION	mKate_gene mKate	SBO_PROMOTER SBO_PRODUCT
Gal4VP16_production	SBO_GENETIC_PRODUCTION	Gal4VP16_gene Gal4VP16	SBO_PROMOTER PRODUCT
cas9m_BFP_production	SBO_GENETIC_PRODUCTION	cas9m_BFP_gene cas9m_BFP	SBO_PROMOTER PRODUCT
gRNA_b_production	SBO_GENETIC_PRODUCTION	gRNA_b_gene gRNA_b	SBO_PROMOTER SBO_PRODUCT
EYFP_Activation	SBO_STIMULATION	EYFP_gene Gal4VP16	SBO_PROMOTER SBO_STIMULATOR
mKate_deg	SBO_DEGRADATION	mKate	SBO_REACTANT
GAL4VP16_deg	SBO_DEGRADATION	GAL4VP16	SBO_REACTANT
cas9m_BFP_deg	SBO_DEGRADATION	cas9m_BFP	SBO_REACTANT
gRNA_b_BFP_deg	SBO_DEGRADATION	gRNA_b_BFP	SBO_REACTANT
EYFP_BFP_deg	SBO_DEGRADATION	EYFP_BFP	SBO_REACTANT
cas9m_BFP_gRNA_b_BFP_deg	SBO_DEGRADATION	cas9m_BFP_gRNA_b_BFP	SBO_REACTANT

Next, we need to specify all interactions for the CRPb Characterization Circuit. Following the same procedure for creating **Interactions** before, we can create those specified in Table 3.

Now, the CRISPR-based Repression Template and connected to the CRPb Characterization Circuit using the `assemble` method. This method instantiates a submodule in the parent ModuleDefinition. Note that the argument to this method is an std::vector of ModuleDefinitions, in case one wants to add multiple submodules to a parent **ModuleDefinition**. Next we use the `mask` method to indicate that the `target_gene` in the template should be refined to be the `EYFP_gene` specified in the CRPb circuit. This method auto-constructs **MapsTo** objects with the correct **refinement** field.

```
CRPb_circuit.assemble({ CRISPR_template });
EYFP_cds.mask(target_gene);
cas9_mBFP.mask(cas9_generic);
gRNA_b.mask(gRNA_generic);
```

At this point, we have completed the CRISPR circuit model.

One final step is to serialize the complete model to produce an RDF/XML output. This can be done by adding the code below.

```
doc.write("CRISPR_example.xml");
```

References

[1] S. Kiani, J. Beal, M. Ebrahimkhani, J. Huh, R. Hall, Z. Xie, Y. Li, and R. Weiss, "Crispr transcriptional repression devices and layered circuits in mammalian cells," *Nature Methods*, vol. 11, no. 7, pp. 723–726, 2014.

[2] M. Courtot, N. Juty, C. Knüpfer, D. Waltemath, A. Zhukova, A. Dräger, M. Dumontier, A. Finney, M. Golebiewski, J. Hastings, S. Hoops, S. Keating, D. Kell, S. Kerrien, J. Lawson, A. Lister, J. Lu, R. Machne, P. Mendes, M. Pocock, N. Rodriguez, A. Villeger, D. Wilkinson, S. Wimalaratne, C. Laibe, M. Hucka, and N. Le Novère, "Controlled vocabularies and semantics in systems biology," *Molecular Systems Biology*, vol. 7, 2011.