Life Sciences Letters

Received 3 December 2015; accepted 27 February 2016. Date of publication 24 March 2016;
date of current version 14 April 2016.

Digital Object Identifier 10.1109/LLS.2016.2546546

libSBOLj 2.0: A Java Library to Support SBOL 2.0

ZHEN ZHANG[1], TRAMY NGUYEN[1], NICHOLAS ROEHNER[2], GÖKSEL MISIRLI[3], MATTHEW POCOCK[4], ERNST OBERORTNER[5], MEHER SAMINENI[1], ZACH ZUNDEL[1], JACOB BEAL[6], KEVIN CLANCY[7], ANIL WIPAT[3], AND CHRIS J. MYERS[1]

[1]University of Utah, Salt Lake City, UT 84112 USA
[2]Boston University, Boston, MA 02215 USA
[3]Newcastle University, Newcastle upon Tyne NE1 7RU, U.K.
[4]Turing Ate My Hamster, Ltd., Newcastle upon Tyne NE27 0RT, U.K.
[5]DOE Joint Genome Institute, Walnut Creek, CA 94598 USA
[6]Raytheon BBN Technologies, Cambridge, MA 02138 USA
[7]ThermoFisher Scientific Synthetic Biology Unit, Carlsbad, CA 92008 USA

CORRESPONDING AUTHOR: C. J. MYERS (myers@ece.utah.edu).

This work was supported in part by the National Science Foundation under Grant DBI-1356041 and Grant DBI-1355909 and in part by the Engineering and Physical Sciences Research Council under Grant EP/J02175X/1.

ABSTRACT The Synthetic Biology Open Language (SBOL) is an emerging data standard for representing synthetic biology designs. The goal of SBOL is to improve the reproducibility of these designs and their electronic exchange between researchers and/or genetic design automation tools. The latest version of the standard, SBOL 2.0, enables the annotation of a large variety of biological components (e.g., DNA, RNA, proteins, complexes, small molecules, etc.) and their interactions. SBOL 2.0 also allows researchers to organize components into hierarchical modules, to specify their intended functions, and to link modules to models that describe their behavior mathematically. To support the use of SBOL 2.0, we have developed the `libSBOLj 2.0` Java library, which provides an easy to use Application Programming Interface (API) for developers, including manipulation of SBOL constructs, serialization to and from an RDF/XML file format, and migration support in the form of conversion from the prior SBOL 1.1 standard to SBOL 2.0. This letter describes the `libSBOLj 2.0` library and key engineering decisions involved in its design.

INDEX TERMS Application programming interfaces, computational biology, software libraries, software tools, synthetic biology.

I. INTRODUCTION AND MOTIVATION

SYNTHETIC biology is an engineering discipline in which biological components are assembled into modules to perform useful functions. At present, many published synthetic biological systems do not include sufficient information about their structure, function, and design rationale, hindering both their reproducibility and their reusability in engineering new systems [9]. Providing a standardized format for encoding all required artifacts of the design of synthetic biological systems would enable scientists and engineers to readily exchange, store, and integrate the artifacts of a design in an automated fashion. The Synthetic Biology Open Language (SBOL) aims to address this situation for synthetic biology. `libSBOLj` is a Java software library that supports these goals by providing a uniform interface for the adoption of the SBOL standard into genetic design automation tools.

II. SYNTHETIC BIOLOGY OPEN LANGUAGE

Beginning in 2008, SBOL has been developed by an international effort involving both experimental and computational synthetic biologists, and including participants from academic, governmental, and commercial organizations. The core data model established in SBOL 1.1 enables specifications of DNA-level designs [3]. Under this model, biological building blocks are represented as Collections of DnaComponents that are hierarchically composed to provide annotated DNASequences. Encoding a design in SBOL 1.1 allows the design to be assembled, optimized, and tested, as well as exchanged and stored by software tools developed at different organizations, thereby supporting a wide range of collaboration and automation practices.

The SBOL 2.0 standard [1] extends SBOL 1.1 to enable the specification of a wider variety of components, such as RNA, proteins, complexes, and small molecules. SBOL 2.0 also represents relations between components in order to specify intended or observed interactions, such as repression, activation, translation, and complex formation. Components and interactions can be grouped into modules that represent functional blocks and can be further hierarchically connected to form more complex systems. Modules can also be linked to behavioral models in languages, such as the Systems Biology Markup Language (SBML) [6], CellML [5], or MATLAB [8].

IEEE Life Sciences Letters

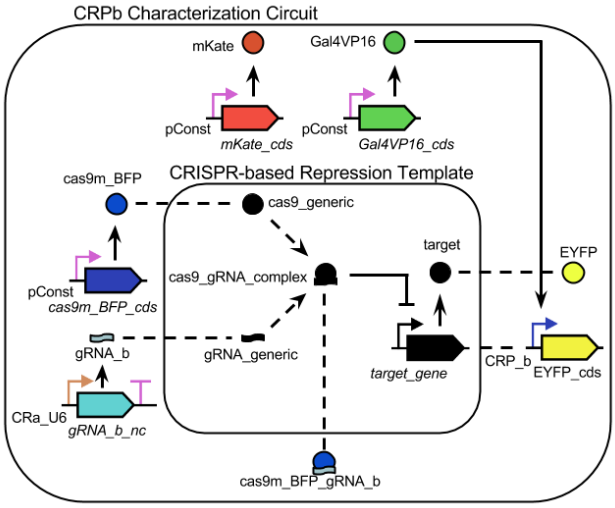

FIGURE 1. Illustration of a hierarchical CRISPR-based repression module represented in SBOL 2.0 (adapted from [7, Fig. 1(a)]). The CRISPR-based repression template ModuleDefinition describes a generic CRISPR repression circuit that combines a cas9 protein with a gRNA to form a complex (represented by the dashed arrows) that represses a target gene (represented by the arrow with the tee-headed 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.

Fig. 1 illustrates how SBOL 2.0 can represent the function of a state-of-the-art design, namely a CRISPR-based repression module from [7]. First, consider the CRISPR-based repression template ModuleDefinition shown in the center of Fig. 1, which is used to provide a generic description of the 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 noncovalent 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 indicating 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.

III. JAVA LIBRARY FOR SBOL 2.0

Crucial to the success of a standard is an infrastructure that supports researchers and software developers for the integration of the standard into tools. A key goal has thus been to develop a library that eases the adoption of SBOL. `libSBOLj 2.0` is a native Java (version 1.7) implementation of the SBOL 2.0 data model, enriched with an application programming interface (API) to instantiate data objects and to define their relations compliant with the SBOL 2.0 data model. That is, `libSBOLj 2.0` enables software tools to use its API to construct objects to store data, such as the system illustrated in Fig. 1. In addition, the library distribution includes detailed documentation of the class definitions and the methods provided by the API.

In particular, `libSBOLj 2.0` organizes all SBOL data within an SBOLDocument object. The SBOLDocument includes a collection of each type of TopLevel object (i.e., Collection, ModuleDefinition, ComponentDefinition, Sequence, Model, or GenericTopLevel). Every 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. The `libSBOLj 2.0` library provides methods to create, access, update, and delete all of the data objects and properties in SBOL 2.0.

While the library can read files with arbitrary URIs, the library only creates compliant URIs that have the following form:

http://⟨prefix⟩/⟨displayId⟩/⟨version⟩

This form is chosen to be easy to read, facilitate debugging, and support a more efficient means of looking up objects and checking URI uniqueness. The ⟨prefix⟩ represents a URI for a namespace (for example, www.sbols.org/CRISPR_Example). The author of a TopLevel object should use a URI prefix that either they own or an organization of which they are a member owns. 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. Finally, 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.

In addition to a URI, each SBOL object can also have a persistentIdentity URI, which is simply its URI without the version when using compliant URIs. The purpose of a persistentIdentity is to allow an object to refer to the latest version of another object using this URI. The latest

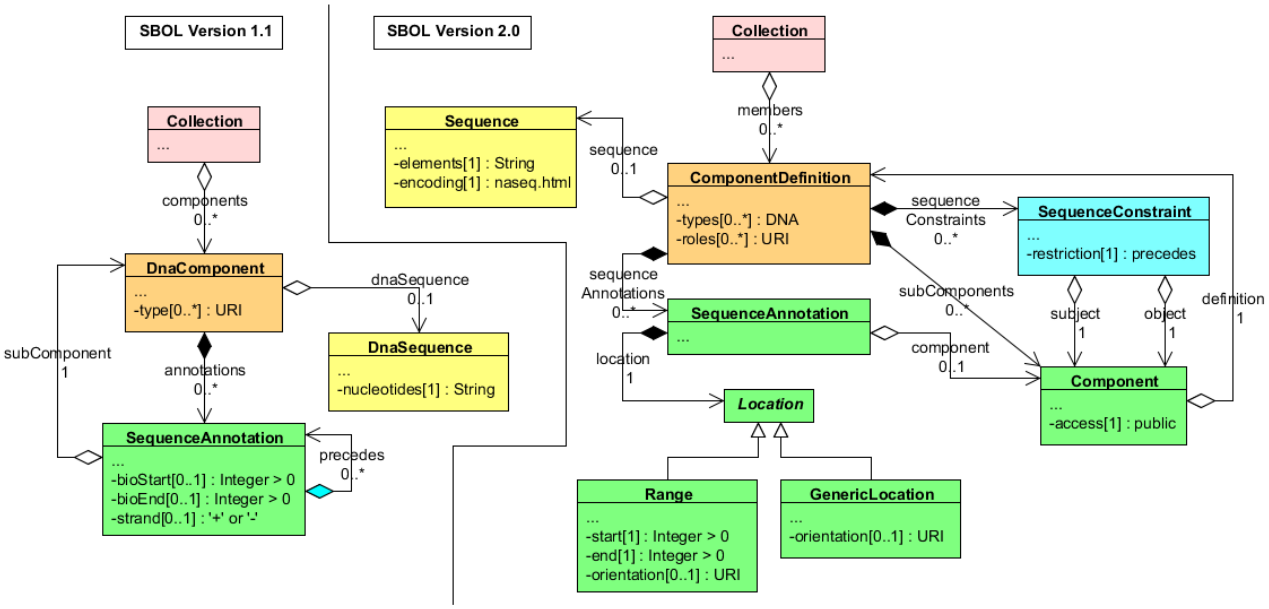

FIGURE 2. **Mapping from the SBOL 1.1 data model to a subset of the SBOL 2.0 data model, indicating equivalences by color. A Collection of DnaComponents maps to a Collection of ComponentDefinitions, among other TopLevel SBOL objects. DnaComponents map to ComponentDefinitions of type DNA. DnaSequences map to Sequences using the IUPAC encoding for nucleotide sequences. SequenceAnnotations with precise start and end positions are mapped to SequenceAnnotations with Range Locations, while SequenceAnnotations with imprecise positions are mapped to SequenceAnnotations with GenericLocations. Each SequenceAnnotation also maps to a Component, which in SBOL 2.0 represents the instantiation or usage of a given ComponentDefinition. Finally, precedes relationships map to SequenceConstraints that specify precedes restrictions.**

version of an object is determined using semantic versioning conventions (see http://semver.org/).

RDF/XML [4] is the main serialization format of both SBOL 1.1 and SBOL 2.0. SBOL 1.1 "inlines" objects whenever they are referenced, meaning that many identical copies of a referenced object appear in a serialized RDF/XML document. SBOL 2.0 serializes each object precisely once and every time the object is referenced, the reference is specified by the object's URI.

SBOL 2.0 includes Annotations and GenericTopLevel objects (i.e., identified annotations) to enable the serialization of additional information that cannot be expressed in the current SBOL 2.0 data model. If the library's read function encounters an unrecognized tag for a TopLevel object, then these data are interpreted as a GenericTopLevel object. Within TopLevel objects, when a tag for a property is not recognized, the data are stored in a custom Annotation object within the TopLevel object. In this way, tools using `libSBOLj 2.0` that do not recognize custom data can still maintain the integrity of such data unmodified when writing and reading SBOL files. Tools that do recognize custom data, on the other hand, can further interpret and manipulate the data after it has been stored in a generic tree-like data structure by the library. In comparison to SBOL 1.1, the SBOL 2.0 solution enhances the extensibility of the SBOL 2.0 RDF/XML documents.

`libSBOLj 2.0` includes a couple of features to ease the transition from SBOL 1.1. First, it includes deprecated versions of the `libSBOLj 1.1` data objects. Users of `libSBOLj 1.1` can immediately switch to `libSBOLj 2.0` with no loss of functionality, but they should then endeavor to migrate their code to use the new SBOL 2.0 data objects as soon as possible. Second, the SBOL 2.0 reader can parse SBOL 1.1 files and automatically convert them to use the SBOL 2.0 data objects using the mapping shown in Fig. 2.

Algorithm 1 illustrates the use of the `libSBOLj 2.0` library using an excerpt of the Java code to express the CRISPR-based repressor design (see Fig. 1) in SBOL 2.0. First, a new SBOLDocument is created (line 1), and is given a default URI prefix (line 2). At this point, ComponentDefinition and Interaction objects are also created for the CRISPR-based repression template ModuleDefinition (not shown). Then, Sequence objects are created for those sequences provided in [7].[1] For example, to create the sequence for the CRP_b promoter, the `createSequence` method is called with the displayId (CRP_b_seq), version (1.0), the sequence, and the encoding used (line 3). Note that this method creates a compliant URI, as described above, using the default URI prefix and provided displayId and version. Next, ComponentDefinition objects are created for each element in the module. For example, a ComponentDefinition of DNA type is created for the CRP_b promoter (lines 4–6). Note that by using compliant URIs, the sequence can be looked up using its displayId, and since no version is provided, it is referenced by its persistentIdentity (line 6). Next, two ComponentDefinitions are created: one for the EYFP coding sequence

[1] Unfortunately, as usual, not all sequences are provided in this letter.

Algorithm 1 Fragments of Java Code to Produce Part of the CRISPR Repression Example Using `libSBOLj 2.0`

```
1  SBOLDocument doc = new SBOLDocument();
2  doc.setDefaultURIprefix("http://sbols.org/CRISPR_Example/");
3  doc.createSequence("CRPb_seq", "1.0", "GCTCCGAATTTCTCGACAGATCTCATGTGAT...", Sequence.IUPAC_DNA);
4  ComponentDefinition CRPb = doc.createComponentDefinition("CRP_b", "1.0", ComponentDefinition.DNA);
5  CRPb.addRole(SequenceOntology.PROMOTER);
6  CRPb.addSequence("CRPb_seq");
7  doc.createComponentDefinition("EYFPcds","1.0", ComponentDefinition.DNA).addRole(SequenceOntology.CDS);
8  ComponentDefinition EYFPgene = doc.createComponentDefinition("EYFPgene","1.0",ComponentDefinition.DNA);
9  EYFPgene.createSequenceConstraint("EYFPgeneCons", RestrictionType.PRECEDES, "CRP_b","EYFPcds");
10 doc.createComponentDefinition("Gal4VP16", "1.0", ComponentDefinition.PROTEIN);
11 ModuleDefinition CRPbCircuit = doc.createModuleDefinition("CRPb_characterization_circuit", "1.0");
12 Interaction EYFPact = CRPbCircuit.createInteraction("EYFPact", SystemsBiologyOntology.STIMULATION);
13 EYFPact.createParticipation("GAL4VP16", "Gal4VP16",SystemsBiologyOntology.STIMULATOR);
14 EYFPact.createParticipation("EYFPgene", "EYFPgene",SystemsBiologyOntology.PROMOTER);
15 Module Template_Module = CRPbCircuit.createModule("CRISPR_Template", "CRISPR_Template", "1.0");
16 Template_Module.createMapsTo("EYFPgene_map", RefinementType.USELOCAL, "EYFPgene", "target_gene");
```

(CDS) and one for the EYFP gene (lines 7 and 8). A SequenceConstraint object is created (line 9) 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. Finally, a protein-type ComponentDefinition is created for the Gal4VP16 protein (line 10). After all the ComponentDefinitions are created, a ModuleDefinition object is created for the CRPb characterization circuit (line 11). Next, the Interactions between the components are specified using terms from the systems biology ontology [2]. One example Interaction is the stimulation of the `EYFPgene` by the Gal4VP16 protein (lines 12–14). Now, the CRISPR-based repression template Module is instantiated and connected to the CRPb characterization circuit using MapsTo objects. For example, a MapsTo object is used to indicate that the `target_gene` in the template should be refined to be the `EYFPgene` specified in the CRPb circuit (line 17).

SBOL does not provide the specification of a mathematical model directly. It is possible, however, to generate a mathematical model using SBML [6] and the procedure described in [10]. Then, the SBOL document can reference this generated SBML model.

IV. SUMMARY

SBOL 1.1 has limitations with respect to the wealth of required artifacts to specify, share, and reproduce designs of synthetic biological systems. SBOL 2.0 extends SBOL 1.1 to specify both structure and function in a hierarchical manner by introducing generalized components, the interactions between them, and modules to group components that collectively implement a common function.

As described in this letter, SBOL 2.0 is supported by `libSBOLj 2.0`, which provides an API, documentation, and additional utilities for managing the encoding and exchange of designs. Improved functionality, particularly error checking, continues to be added to further ease the adoption of SBOL 2.0. The library is freely available from GitHub under the Apache 2.0 License. The SBOL website (http://sbolstandard.org) provides links to the current snapshot, the latest release, the issue tracker, javadocs, a brief tutorial, and examples, including the full code for the CRISPR example. Questions about the library can be sent to: libsbol-team@googlegroups.com.

ACKNOWLEDGMENT

The authors would like to thank B. Bartley and Profs. H. Sauro and J. Gennari (University of Washington), who are developing a C++ version of this library, for their feedback. The authors would also like to thank the SBOL Developers community for their contributions to the SBOL standard. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or their other funding agencies.

REFERENCES

[1] B. Bartley *et al.*, "Synthetic biology open language (SBOL) version 2.0.0," *J. Integrative Bioinformat.*, vol. 12, no. 2, p. 272, 2015.
[2] M. Courtot *et al.*, "Controlled vocabularies and semantics in systems biology," *Molecular Syst. Biol.*, vol. 7, Oct. 2011, Art. no. 543.
[3] M. Galdzicki *et al.*, "The synthetic biology open language (SBOL) provides a community standard for communicating designs in synthetic biology," *Nature Biotechnol.*, vol. 32, pp. 545–550, 2014.
[4] F. Gandon and G. Schreiber. (2014). *RDF 1.1 XML Syntax*, accessed on Mar. 23, 2015. [Online]. Available: http://www.w3.org/TR/rdf-syntax-grammar
[5] A. Garny *et al.*, "CellML and associated tools and techniques," *Philos. Trans. A, Math. Phys. Eng. Sci.*, vol. 366, no. 1878, pp. 3017–3043, Sep. 2008.
[6] M. Hucka *et al.*, "The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, no. 4, pp. 524–531, 2003.
[7] S. Kiani *et al.*, "CRISPR transcriptional repression devices and layered circuits in mammalian cells," *Nature Methods*, vol. 11, no. 7, pp. 723–726, 2014.
[8] MATLAB, MathWorks, Natick, MA, USA, 2015.
[9] J. Peccoud *et al.*, "Essential information for synthetic DNA sequences," *Nature Biotechnol.*, vol. 29, no. 1, p. 22, Jan. 2011.
[10] N. Roehner, Z. Zhang, T. Nguyen, and C. J. Myers, "Generating systems biology markup language models from the synthetic biology open language," *ACS Synth. Biol.*, vol. 4, no. 8, pp. 873–879, 2015.