A Web-Based Validator and Validation API for the Synthetic Biology Open Language

Zach Zundel
Dept. of Bioengineering
zach.zundel@utah.edu

Meher Samineni
School of Computing
m.samineni@utah.edu

Zhen Zhang
Dept. of Elec. and Comp. Eng.
zhen.zhang@utah.edu

Chris J. Myers
Dept. of Elec. and Comp. Eng.
myers@ece.utah.edu

1. INTRODUCTION

The *Synthetic Biology Open Language* (SBOL) [1] is an emerging standard for the expression of both structural and functional data regarding biological constructs. The data is encoded in an RDF-XML format that allows for hierarchical representation of the construct. As with all data standards, files encoding SBOL data must be validated according to a set of validation rules that describe correct formatting and encoding, as well as a minimum acceptable set of information which represents a complete and correct construct. Furthermore, it is useful to be able to convert between files in the SBOL standard and files that other tools are able to use and interpret. Our validator has functionality to convert between SBOL versions 1.1 and 2.0, as well as between SBOL and GenBank, a commonly used format for annotated DNA sequences. Our validation software embeds the libSBOLj [2] validation routines to allow validation of SBOL 2.0 files through web access to validation, as well as RESTful API calls for validation. This software allows for a universal standard for validation that existing tools (such as iBioSim [3] and Cello [4]) can use for validation without requiring tool developers to create, test, and maintain validation routines.

2. VALIDATION REQUIREMENTS

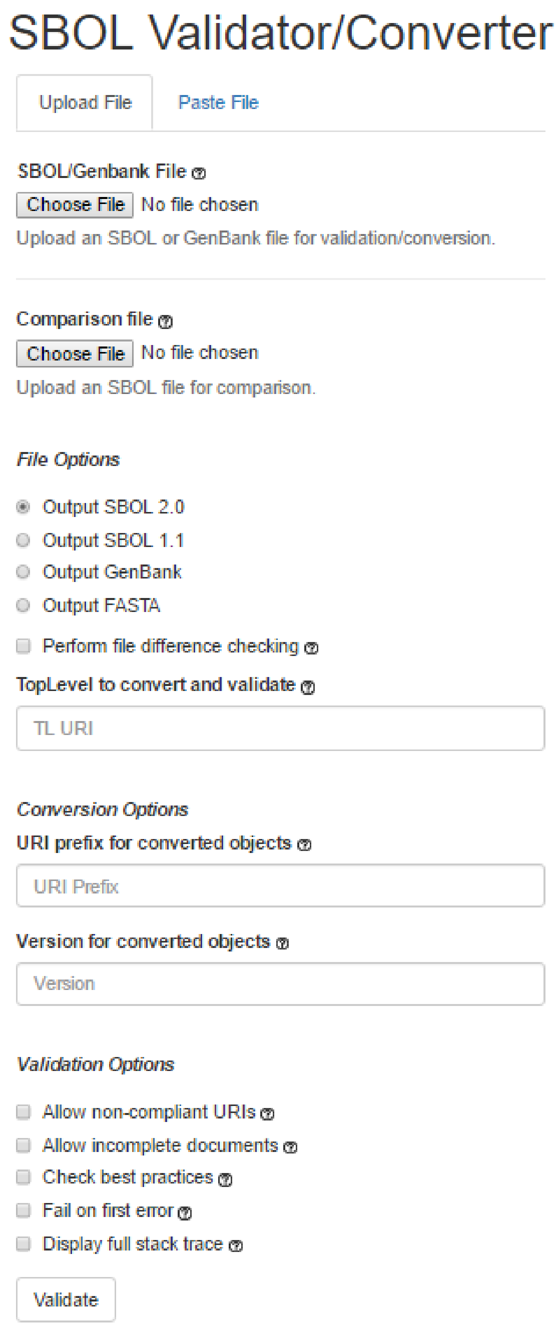
Proper validation of a file has several aspects. SBOL's validation rules are divided into two distinct categories. The first group is the *required validation rules*, and the second is the *best practice validation rules*. Required validation rules are a set of rules set in the SBOL specification that define precise relationships and requirements for valid SBOL [1]. These rules define things such as the number and nature of relationships between specific SBOL objects, attributes, and formats for these objects, and proper encoding and representation of the objects. A file is valid SBOL, if and only if it complies with all of these rules, so they are the most important to check. These rules are developed to be as *machine-checkable* and *unambiguous* as possible. Best practices validation rules describe a set of rules in the SBOL specification that define guidelines that help to ensure sensible and meaningful SBOL. These rules are, by nature, often less *machine-checkable*, though it is still desirable that these retain this trait so that they can be programatically validated. A file can still be valid SBOL if it does not pass these rules, but its sensibility, usefulness, and exchangability may not be the same level as one that passes both validation and best practices checks.

3. THE SBOL VALIDATOR

Our validator is backed by libSBOLj, a Java library for reading, writing, and encoding data in the SBOL standard and includes several validation routines that allow for checking of both required and best practice validation rules. The library performs certain validation checks on reading the RDF/XML file, and a second set of checks once the file has been read into the internal data model. Currently, the standard is version 2.0, which represents a significant departure from both the model and amount of data encoded in its 1.1 version. libSBOLj also provides several methods for converting the SBOL data model to/from other formats. In particular, it allows for the conversion between SBOL versions and between SBOL and the GenBank format.

The web-based validator, shown in Fig. 1, is essentially a PHP wrapper around the Java library. Files can be uploaded or copied into the validation form, and all validation options available in the library are available in the validator. Furthermore, the validator is designed to be able to return any converted files to the user so that all functionality of libSBOLj's validation routines are available. The validator's form processing utility creates several PHP objects that are all components of an overarching ValidationRequeset object. This object uses its component field objects to generate the Java command for validation and execute it. The resulting string is sent to a ValidationResult object for processing and then returned to the user via a web interface.

Our validator also has a RESTful API [5] endpoint to allow programmatic validation of SBOL files through a web service. The validation API extends the framework for the web-based validator. First, its endpoint processing utility accepts a JSON object from the POST request headers and checks to ensure that all necessary fields of the JSON object are included and properly formatted. A properly-formatted JSON object first requires a sub-object which contains Boolean switches and strings for each of the validation options given through the web validator. All of these options must be sent, even if they are empty or false – no values are assumed from missing elements. Secondly, the user must specify whether or not they would like a file returned to them. Finally, all required files must be encoded as separate strings. The endpoint utility accepts and validates all JSON requests, and



SBOL Validator/Converter

Upload File | Paste File

SBOL/Genbank File ⓘ

Choose File No file chosen

Upload an SBOL or GenBank file for validation/conversion.

Comparison file ⓘ

Choose File No file chosen

Upload an SBOL file for comparison.

File Options

[✓ Output SBOL 2.0]
Output SBOL 1.1
Output GenBank
Output FASTA

☐ Perform file difference checking ⓘ

TopLevel to convert and validate ⓘ

TL URI

Conversion Options

URI prefix for converted objects ⓘ

URI Prefix

Version for converted objects ⓘ

Version

Validation Options

☐ Allow non-compliant URIs ⓘ
☐ Allow incomplete documents ⓘ
☐ Check best practices ⓘ
☐ Fail on first error ⓘ
☐ Display full stack trace ⓘ

Validate

Figure 1: Screenshot for the web validator interface.

builds an APIRequest object (which is simply an extension of the ValidationRequest class) using this JSON object. The APIRequest is then used to build and execute the validation command. The result of the command is passed to the requester in the form of another JSON object containing the result of the validation.

4. DISCUSSION

Though any application that wishes to perform validation on any SBOL files can simply include the libSBOLj library as a dependency and build any validation commands itself, providing a centralized service for validation of these files allows for a standard and universal definition of validity that can be applied, developed, and debugged independent of the development of each of the specific libraries. Furthermore, a single validator that is accessible through the web helps to ensure that competing validation routines do not emerge, avoiding risk of multiple definitions of validity that may have slight variance or discrepancies.

At time of writing, no SBOL libraries except libSBOLj implement validation for file conversion. pySBOL, libSBOL, and libSBOLjs are the three other major libraries currently under development that could benefit from the API validation – it shifts the onus of writing and maintaining validation routines to a single source. Therefore, library developers can simply write methods to access the RESTful API, a process that is simple, well-documented, and extensible in many languages. Additionally, many tools currently support reading and writing SBOL documents and more are under development. If these tools read and write SBOL without using one of the four main libraries (libSBOLj, libSBOL, pySBOL, libSBOLjs) they will also be able to validate their output to ensure full compliance with the specification using the web API.

Finally, the developers of the SBOL standard can use the validator as part of the compliance certification process. After dictating a set of constructs to be created in SBOL, one phase of validation of a tool's output would be validating their results, while another would be comparing the output of the tools to a premade file using the validator. A workflow such as this allows the community to define what is and what is not SBOL specification compliance, addressing the emerging issue of tools claiming compliance without the ability to read, write, and interpret valid SBOL documents.

Our validator is currently accessible online at `http://www.async.ece.utah.edu/sbol-validator`, with the API accepting requests at `http://www.async.ece.utah.edu/sbol-validator/endpoint.php`.

5. ACKNOWLEDGEMENTS

This work is funded by the National Science Foundation under Grants CCF-1218095 and DBI-1356041. 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.

6. REFERENCES

[1] B. Bartley *et al.*, "Synthetic Biology Open Language (SBOL) Version 2.0.0," *J Integr Bioinform*, vol. 12, no. 2, p. 272, 2015.
[2] Z. Zhang *et al.*, "libSBOLj 2.0, A Java Library to Support SBOL 2.0," *IEEE Life Sciences Letters*, 2015.
[3] C. Madsen *et al.*, "Design and test of genetic circuits using iBioSim," *IEEE Design and Test*, pp. 32–39, 2012.
[4] A. A. K. Nielsen *et al.*, "Genetic circuit design automation," *Science*, vol. 352, no. 6281, 2016.
[5] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," in *Proceedings of the 22nd International Conference on Software Engineering*, ICSE '00, (New York, NY, USA), pp. 407–416, ACM, 2000.