Extending SynBioHub's Functionality with Plugins

Jeanet Mante,[*,†] Zach Zundel,[*,†,‡] and Chris Myers[*,¶,†,‡]

†*Department of Biomedical Engineering, University of Utah, Salt Lake City, USA*

‡*School of Computing, University of Utah, Salt Lake City, USA*

¶*Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, USA*

E-mail: j.mante@utah.edu; me@zachzundel.com; myers@ece.utah.edu

Abstract

SynBioHub is a repository for synthetic genetic designs represented in the *Synthetic Biology Open Language* (SBOL). To integrate SynBioHub into more synthetic biology workflows, its data processing capabilities need to be expanded. To this end, a plugin interface has been developed. Plugins can be developed for data submission, visualization, and download. This framework was tested by the development of three example plugins, one of each type: one allowing the submission of SnapGene files, one visualizing the co-use of different genetic parts, and one preparing plasmid maps for download.

Synthetic biology is a movement to standardize genetic engineering and make it more repeatable. One important advancement was the development of standardized genetic parts known as *BioBricks*, which can be composed using restriction enzyme assembly .[1,2] Another important advancement was the development of the *Synthetic Biology Open Language* (SBOL), a standard language for describing these parts, among other things.[3,4] Finally, to

share these parts, design repositories, such as SynBioHub,[5] were developed. One recent enhancement to SynBioHub is the introduction of plugin capabilities to allow third-party developers to add functionality. This allows the researchers who know the workflow best to develop new plugins to aid their workflow. Additionally, it allows different SynBioHub instances to be customized to suit the needs of the user whilst streamlining the core for scalability. This paper describes the plugin engine as well as one plugin of each of the 3 types: submission, visualization, and download. The example submit plugin allows the submission of SnapGene .dna files. The example visualization plugin was created to aid in part selection. The example download plugin allows the download of annotated plasmid maps. A plugin template was created in Python. The template was published on GitHub, and used as the base for each of the three example plugins.

Results

SynBioHub was extended with a new plugin engine. Each plugin must be deployed as a web service that is accessible by SynBioHub. Once an appopriate action (e.g. a page is rendered) is taken on SynBioHub the plugin is engaged. The steps for this process are illustrated in Figure 1.

First, the status of the plugin is checked to ensure it is ready to serve requests. Second, the plugin evaluates its ability to run by comparing the data type received to the data types it can handle. Depending on this evaluation, the plugin is run. Finally, SynBioHub waits for a response and asynchronously reports it to the user. New plugins can be configured in the administrative portal of SynBioHub as shown in Figure 2.

Submit Plugin Engine Specifics: Submit plugins are called whenever a new submission is created and a user selects a plugin to handle it. The plugin is sent a list of files included in the submission via the evaluate protocol. The list of files includes some metadata for each file, such as file name and type, and an access link for each file. The submit plugin

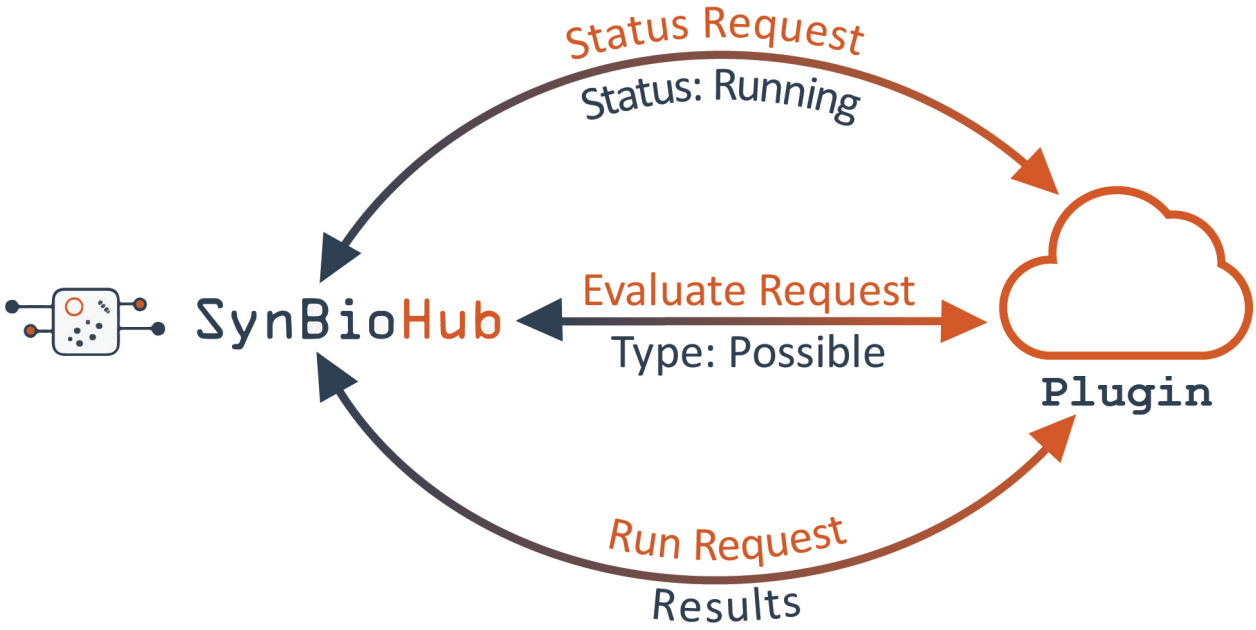

Figure 1: Flow diagram illustrating the different steps of communication between the plugin server and SynBioHub with orange signifying messages from SynBioHub to the Plugin and blue from the Plugin to SynBioHub. First, SynBioHub checks the status of the plugin. If the plugin is running, the type of object is sent to the plugin. If the plugin responds that it can handle the object type, the full data is sent to the plugin to run and return results to SynBioHub.

uses this information to decide if it can process the submission. The plugin then responds: whether or not it can process the submission. If the plugin cannot process the submission, then SynBioHub falls back to the default submit handling code. If the plugin can handle the submission, it does so to create a new submission data set. The resulting data set is then deposited into SynBioHub following the default submission routines.

Example Submit Plugin: The submit plugin accepts SnapGene formatted .dna files and converts them to SBOL which is then uploaded to SynBioHub. The plugin works by submitting the .dna file to a local SnapGene server which returns a GenBank version of the file. This file is then converted to SBOL via the SBOL Converter and Validator (https://validator.sbolstandard.org). The SBOL file is then returned by the plugin to SynBioHub for submission.

Visualization Plugin Engine Specifics: Visualization plugins are called when a part

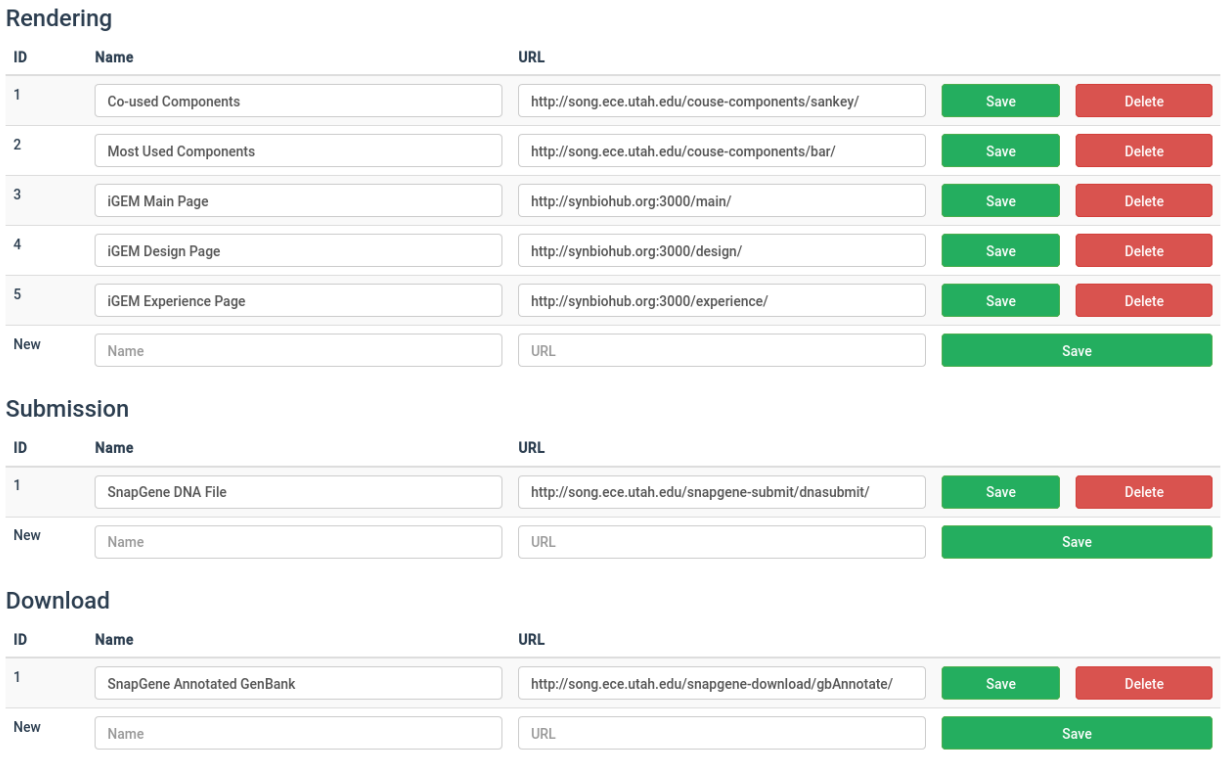

Figure 2: The administrative interface for configuring plugins on SynBioHub. Plugins are given names which are displayed to the user, and the URL of the plugin must be given for it to work properly.

page is rendered. The plugin is sent metadata about the part and links to its SBOL. The plugin can use this data to render a webpage visualizing information about the part. The webpage is described using HTML, which the plugin sends to SynBioHub to be displayed. SynBioHub asynchronously updates the part page to contain the rendered HTML webpage.

Example Visualization Plugin: Our visualization plugin can assist designers to find parts for their designs. In particular, our plugin displays one graph per part page. There is a Sankey diagram that shows other parts that are commonly used with the part displayed on the page of interest. Co-used parts are sorted by their role and their position before or after the part of interest in the genetic sequence (see Figure 3). Some additional features that make this visualisation interactive include the ability to: zoom in and out, scroll across the graphs, show data about a part when hovering over it, dynamically add value comparison lines, download a plot as a PNG, and navigate to a page describing that part in the corresponding

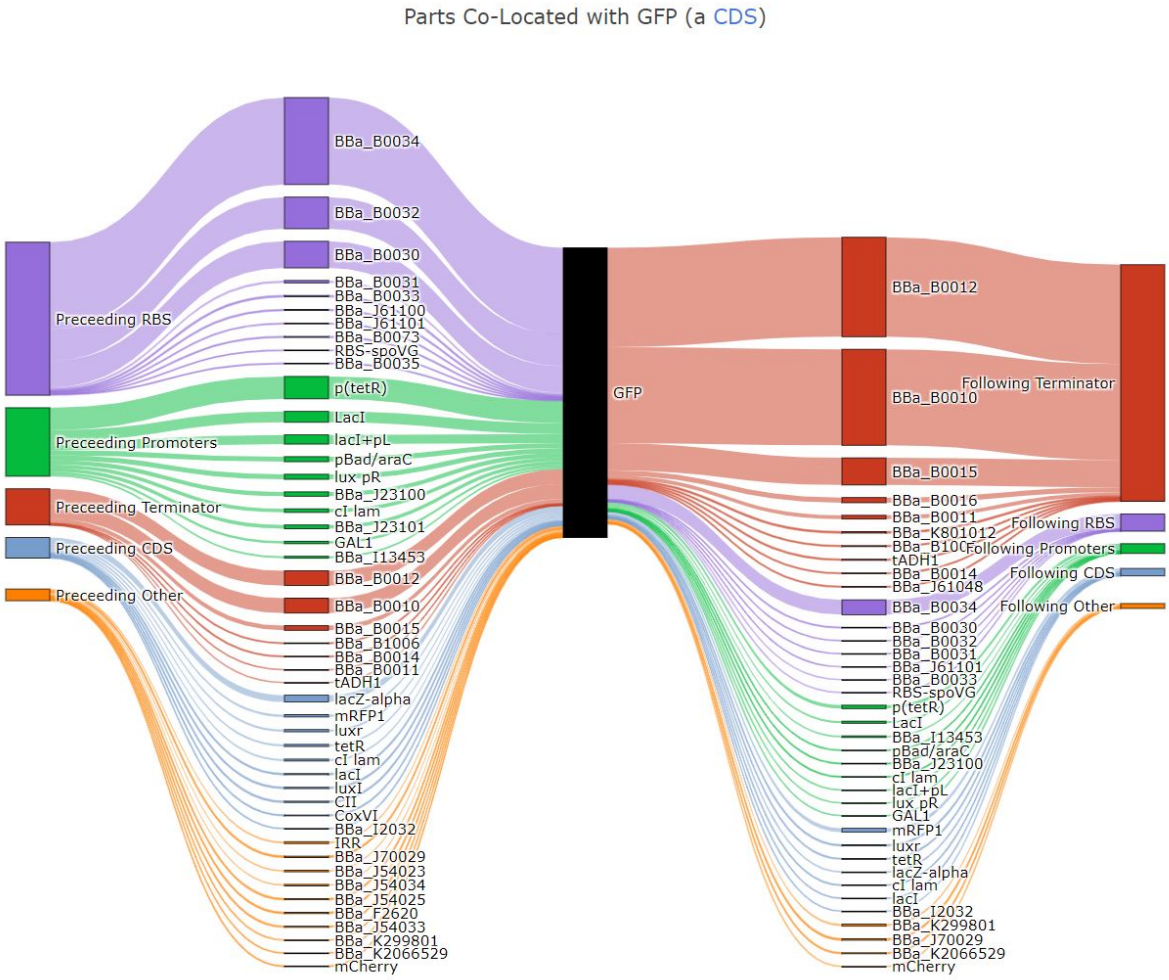

Figure 3: Example visualization created by the plugin for the part BBa_E0040, more commonly known as GFP. Part co-occurrence by type: a diagram showing how the part of interest (GFP) is combined with other parts (e.g. BBa_B0012) and what fraction of these interactions are preceding (BBa_B0012 is before GFP) and following (BBa_B0012 is after GFP).

SynBioHub instance.

Download Plugin Engine Specifics: Download plugins are called when a download is requested. The plugin is sent metadata about the part and links to its SBOL. The plugin can use this data however it likes to create a zip file representing the part. The zip file is then sent to SynBioHub along with some metadata, which is used to update the browser with the zip file to download.

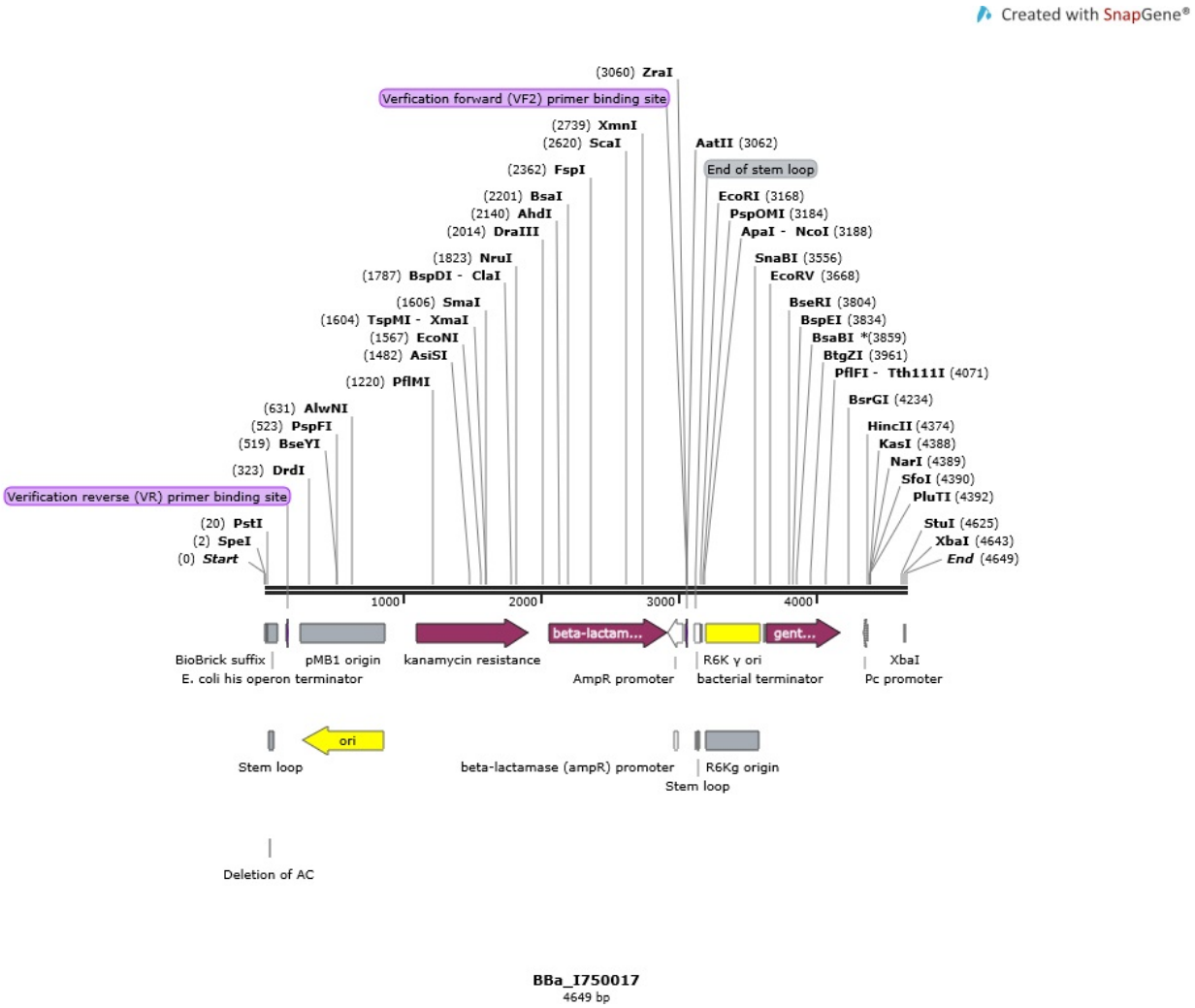

Figure 4: Example of a download output created by the plugin for the part BBa_I750017. This is an annotated linear plasmid map created by the SnapGene server which was called by the download plugin.

Example Download Plugin: The example plugin allows downloading of an annotated

plasmid map for the SBOL part (like Figure 4). The plugin uses the GenBank format of the SBOL part provided by SynBioHub and submits it to a local SnapGene server. The SnapGene server then returns an annotated plasmid map that is returned in a zip file together with a GenBank file. This zip file is made available to SynBioHub for the user to download.

Discussion

The plugin paradigm has clearly demonstrated its usefulness for extending SynBioHub while minimizing the relative complexity of both new features and the SynBioHub core. This highlights some potential future directions for plugin development to bring new kinds of features to SynBioHub.

One type of plugin that would be useful is a *curation* plugin. This type of plugin would allow for the modification of SBOL already in SynBioHub to enhance its usefulness or correctness. For example, a curation plugin could identify duplicates of subcomponents and remove them. This would improve the data available about usage statistics and connect parts with characterization data.

Another type of plugin that may be useful is the *search* plugin. A search plugin could have access to SynBioHub data, and be responsible for responding to search queries. This would enable improvement of search whilst providing a robust and clean interface for interacting with SynBioHub. This could also improve SynBioHub's handling of failures in search functionality by automating fallback to the default search routines. Search plugins would require significantly more access to the SynBioHub triplestore. Additionally, because many search techniques rely on precompiling indices, it would have to have near-constant access to the database. This raises several concerns about data privacy and leakage. The search plugin interface would have to be designed in such a way that a SynBioHub could audit results to ensure a user is not presented with options they do not have access to view. It would likely be up to a SynBioHub instance administrator to determine whether a specific

search plugin is trustworthy enough to grant access to their SynBioHub. To aid in making such decisions a registry of plugins will be developed in the future. This necessary trust could be reduced through techniques such as homomorphic encryption, which would allow the search plugin to operate on data without necessarily having visibility into its meaning.

The code for running and managing plugins is generic enough that adding new plugins which follow the same protocol would be very simple. The most complex additions are around handling the result of the plugin run. Though adding plugins using the existing protocol would be simple, the search and curation plugins were excluded from this work. This is because their functionality cannot be implemented using the existing protocol. For example, search plugins need a way to access the entirety of the data stored in SynBioHub. They could not meaningfully operate on a single SBOL document.

Methods

Submit Plugin: Submit plugins operate on entire submissions, rather than individual SBOL constructs. When a user submits something to SynBioHub, they can select a plugin to handle that submission. A manifest for the submission is prepared, describing each file in the submission. The manifest is first sent to the plugin's evaluate endpoint, which responds with a decision regarding whether the plugin can handle this submission. If the plugin can handle the submission, the manifest is then re-sent to the plugin's run endpoint. If the plugin is successful, then it responds with a ZIP file containing a new set of files to be submitted to SynBioHub. If the plugin's run endpoint is not successful, or if the plugin responds negatively to the evaluate request, SynBioHub falls back to the default submission handler.

The conversion tool is implemented as a plugin deployed on the development server for the reference instance of SynBioHub (`https://dev.synbiohub.org`). The plugin is an HTTP server, written in Python, using the Flask library. When a file is submitted, SynBioHub sends a status and evaluate request to the plugin. If both have been answered correctly (the

plugin is running and can process the file type) SynBioHub then sends a post request to this plugin with the content of the file. This file is then used in an HTTP POST request to a local SnapGene server which creates a GenBank file. This file is then retrieved using an HTTP GET request. The GenBank is then used in an HTTP GET request to the SBOL Converter which returns SBOL. The SBOL is then returned to SynBioHub.

Visualization Plugin: The visualization tool is implemented as a plugin deployed on the development server for the reference instance of SynBioHub (`https://dev.synbiohub.org`). The plugin is an HTTP server, written in Python, using the Flask library. When a part page is opened, SynBioHub sends a status and then evaluate request to the plugin. If both are answered affirmatively (the plugin is running and can handle the part type) SynBioHub sends a post request to this plugin which includes the URL for the part. This URL is used in two different pre-written SPARQL queries which are sent to SynBioHub to gather the necessary data. The data returned is processed to create the input for the visualization. The visualization is made using the plotly library (`https://plot.ly`). Finally, the graphic is transmitted to SynBioHub for rendering on a part page in SVG format embedded in HTML.

Download Plugin: The plasmid map tool is implemented as a plugin deployed on the development server for the reference instance of SynBioHub (`https://dev.synbiohub.org`). The plugin is an HTTP server, written in Python, using the Flask library. When the download button is pushed, SynBioHub checks the plugin is up (status request) and can handle the part type (evaluate request) before sending a post request to the plugin. The post request includes the GenBank URL of the part. This URL is used to obtain the GenBank file for the part which is then submitted to a local SnapGene server using an HTTP POST request. The server creates a PNG file which is retrieved using an HTTP GET request. The PNG and original GenBank is then returned to SynBioHub as a zip file.

Acknowledgement

This work is supported by the National Science Foundation under grants CCF-1748200 and 1939892 and DARPA grant FA8750-17-C-0229. 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 funding agencies.

Supporting Information Available

The links below are to github repositories where the code for each of the elements mentioned can be found.

Plugins:

Visualisation Plugin Repo: `https://github.com/SynBioHub/component-use-plugin`

Submit Plugin Repo: `https://github.com/SynbioHub/Snapgene-submit-plugin`

Download Plugin Repo: `https://github.com/SynBioHub/snapgene-download-plugin`

Plugin template: `https://github.com/SynbioHub/plugin-template`

SynBioHub:

SBOL Validator: `https://github.com/SynBioDex/SBOL-Validator`

SynBioHub: `https://github.com/synbiohub/synbiohub`

SynBioHub Web of Registries: `https://github.com/SynBioHub/Web-of-Registries`

References

(1) Knight, T. *Idempotent Vector Design for Standard Assembly of Biobricks*; 2003.

(2) Shetty, R. P.; Endy, D.; Knight, T. F. Engineering BioBrick vectors from BioBrick parts. *J. Biol. Eng.* **2008**, *2*, 5.

(3) Galdzicki, M. et al. The Synthetic Biology Open Language (SBOL) provides a community

standard for communicating designs in synthetic biology. *Nat. Biotechnol.* **2014**, *32*, 545–550.

(4) Roehner, N. et al. Sharing Structure and Function in Biological Design with SBOL 2.0. *ACS Synth. Biol.* **2016**, *5*, 498–506.

(5) McLaughlin, J. A.; Myers, C. J.; Zundel, Z.; Misirli, G.; Zhang, M.; Ofiteru, I. D.; Goni-Moreno, A.; Wipat, A. SynBioHub: A Standards-Enabled Design Repository for Synthetic Biology. *ACS Synth. Biol.* **2018**, *7*, 682–688.

Graphical TOC Entry