Published in final edited form as:
Biosystems. 2020 July ; 195: 104150. doi:10.1016/j.biosystems.2020.104150.

libsbmljs — Enabling Web–Based SBML Tools

J Kyle Medley[a], **Joseph Hellerstein**[b], **Herbert M Sauro**[a]

[a]Department of Bioengineering, University of Washington, Box 355061, Seattle, WA 98195-5061, USA

[b]eScience Institute, University of Washington, Seattle, Washington, United States of America

Abstract

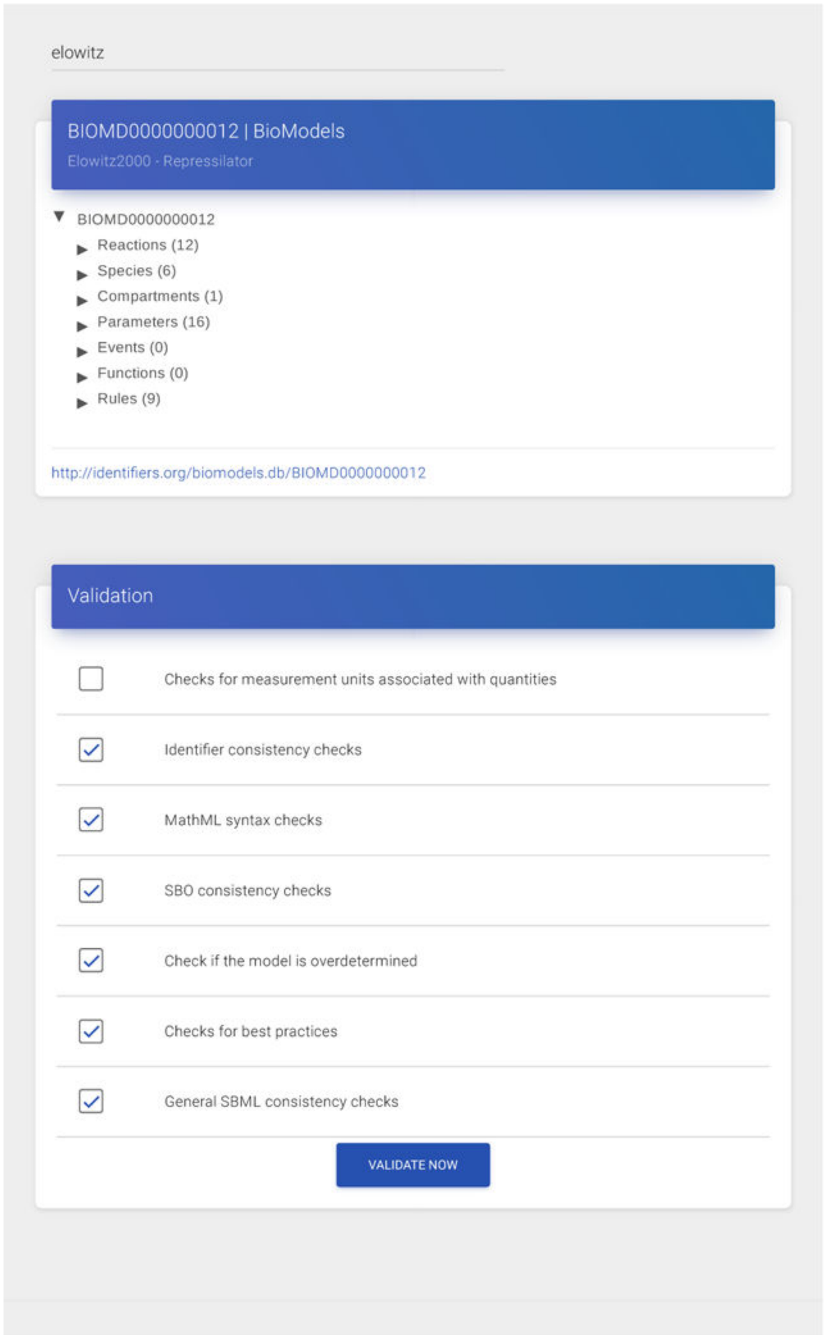
The SBML standard is used in a number of online repositories for storing systems biology models, yet there is currently no Web–capable JavaScript library that can read and write the SBML format. This is a severe limitation since the Web has become a universal means of software distribution, and the graphical capabilities of modern web browsers offer a powerful means for building rich, interactive applications. Also, there is a growing developer population specialized in web technologies that is poised to take advantage of the universality of the web to build the next generation of tools in systems biology and other fields. However, current solutions require **server–side processing** in order to support existing standards in modeling. We present libsbmljs, a JavaScript / WebAssembly library for Node.js and the Web with full support for all SBML extensions. Our library is an enabling technology for online SBML editors, model–building tools, and web–based simulators, and runs entirely in the browser without the need for any dedicated server resources. We provide NPM packages, an extensive set of examples, JavaScript API documentation, and an online demo that allows users to read and validate the SBML content of any model in the BioModels and BiGG databases. We also provide instructions and scripts to allow users to build a copy of libsbmljs against any libSBML version. Although our library supports all existing SBML extensions, we cover how to add additional extensions to the wrapper, should any arise in the future. To demonstrate the utility of this implementation, we also provide a demo at https://libsbmljsdemo.github.io/ with a proof–of–concept SBML simulator that supports ODE and stochastic simulations for SBML core models.

Graphical Abstract

Our project is hosted at https://libsbmljs.github.io/, which contains links to examples, API documentation, and all source code files and build scripts used to create libsbmljs. Our source code is licensed under the Apache 2.0 open source license.

medjk@comcast.net (J Kyle Medley).

Conflict of interest statement
The authors declare that they have no conflict of interest.



elowitz

BIOMD0000000012 | BioModels

Elowitz2000 - Repressilator

▼ BIOMD0000000012
- ▶ Reactions (12)
- ▶ Species (6)
- ▶ Compartments (1)
- ▶ Parameters (16)
- ▶ Events (0)
- ▶ Functions (0)
- ▶ Rules (9)

http://identifiers.org/biomodels.db/BIOMD0000000012

Validation

- [ ] Checks for measurement units associated with quantities
- [✓] Identifier consistency checks
- [✓] MathML syntax checks
- [✓] SBO consistency checks
- [✓] Check if the model is overdetermined
- [✓] Checks for best practices
- [✓] General SBML consistency checks

Keywords

sbml; systems biology; web

Introduction

The SBML [1] standard is used for encoding reaction network models in systems biology research in a reusable, exchangeable, and future–proof manner. One of the factors behind

SBML's wide adoption is the SBML standard's process for introducing extension modules, which allow incremental incorporation of new capabilities. While the core components of the standard are designed for describing kinetic chemical reaction network models, SBML extensions exist for encoding constraint–based models (the "flux– balance constraints" extension [2], employed by the widely used COBRA toolkit for constraint–based modeling [3, 4] as well as other tools such as Escher and the BiGG database [5, 6], FAME [7], SBW Flux Balance [8], ModelPolisher [9] and CellNetAnalyzer[10]), and rule–based models (the SBML "multi" extension [11]). SBML is used in several online model repositories including BioModels [12, 13] and JWS Online [14, 15], which host primarily kinetic reaction network models, and BiGG Models [6], which hosts primarily genome–scale constraint–based models.

Despite this wide–spread adoption and inclusion in several online repositories, no feature–complete JavaScript library currently exists that can run in a web browser (a native Node.js module exists, but cannot run in the browser). Thus, these online repositories must rely on server–side processing of all SBML–related requests. A JavaScript library would allow these services to offload some of their processing to the client, and would also allow for more interactive features on the Web. Furthermore, the Web is becoming a major platform for systems biology tools. With the advent of Web applications for pathway visualization (Escher [5]), gene interaction network visualization (Cytoscape.js [16]), expression analysis (ZBIT [9]) and integrated design systems (Caffeine [17]), the need for a JavaScript library which can read and write SBML becomes imperative.

We present libsbmljs, a feature–complete JavaScript library for reading and writing SBML in the browser and Node.js. libsbmljs uses the full code-base of the libSBML C++ library compiled to the web using Emscripten, a toolset for compiling C++ projects to the web. Emscripten emits WebAssembly [18], a W3C standard for running platform–independent binary code on the web that is supported on all major browsers. We have designed a JavaScript wrapper around this binary format that allows libsbmljs to be used like a normal JavaScript library. Our wrapper supports *all* SBML Level 3 extensions, meaning it can read and write any type of SBML content. Since our library runs in the browser, it does not require a dedicated web server. This is an important consideration for academic software, where long–term maintenance cost is a concern.

Methods

Prior work on implementing the SBML standard has resulted in two libraries: libSBML [19], a C++ library with interfaces for many languages, and JSBML [20, 21], a platform–independent pure Java library. While the existence of these separate implementations is certainly a convenience for C++ and Java developers respectively, it necessitates the maintenance of two independent libraries. Rather than attempt to create a third implementation in pure JavaScript, we have created a web–capable interface for the libSBML C++ library using Emscripten [22], a C++–to–JavaScript compiler. Despite its C++ origins, libsbmljs is completely platform independent and runs on modern browsers on any device which supports web standards.

Compiling a C++ library with Emscripten does not produce a ready–to–use JavaScript library automatically. Instead, Emscripten compiles to WebAssembly [23], a low–level binary format similar to x86 machine code but with additional features for security and platform–independence. Since WebAssembly is very low level, it is difficult to use to design JavaScript web applications. Instead, Emscripten can be used to also compile a JavaScript interface that abstracts the low–level details of calling into WebAssembly and instead allows developers to use familiar JavaScript objects and methods. However, this interface is not generated automatically by Emscripten. Instead, it must be manually specified using WebIDL.

Web IDL is a World Wide Web Consortium (W3C®) standard that specifies *interfaces* to EMCAScript (i.e. JavaScript) objects. For example, the libSBML C++ class SBase has the method getId(), which returns a string.

In WebIDL, this would be specified as:

```
/**
  * SBase: the base class of
  * most SBML elements
  */
[Prefix="libsbml::"]
interface SBase {
   /**
    * Returns this element's
    * id attribute
    */
  DOMString getId();
};
```

In the example above, the body of the getId method is intentionally left blank because it will delegate to the corresponding WebAssembly routine. The "DOMString" type in the above example is not a JavaScript type but a WebIDL type which maps to a JavaScript string and a C++ std::string. Using syntax similar to the above, we manually designed WebIDL interface files for most libSBML classes and methods. However, one issue remains with this approach. The comments entered into the IDL definition above will not appear in the JavaScript interface generated by Emscripten. Thus, there is no way of adding documentation to the generated JavaScript code, which defeats any attempt to generate API documentation. To remedy this issue, we created a script to automatically extract documentation strings from IDL files and insert them into the generated JavaScript code. This allowed us to generate extensive API documentation using documentationjs, a documentation generator for JavaScript.

In order to use the correct argument and return types, we recommand that JavaScript developers consult the API documentation for libsbmljs (https://libsbmljs.github.io/stable/apidoc/), which lists the argument and return types for each method using native JavaScript types. For example, to use the getId method of the SBase object above a developer should consult the documentation for the SBase class (https://libsbmljs.github.io/core/apidoc/#sbase), which shows that the getId method returns a native JavaScript "string."

Special Considerations for Usage

Emscripten–generated WebAssembly/JavaScript libraries are supported on a wide variety of browsers and devices (https://github.com/libsbmljs/libsbmljs lists the browsers we have tested). However, there are minor differences between these libraries and regular JavaScript libraries, which are described below.

Asynchronous Loading

Emscripten–generated libraries load asynchronously. In other words, the library cannot be used immediately as soon as the web page has loaded. This is due to the fact that Emscripten–generated libraries consist of both a JavaScript source file (.js) containing JavaScript classes and methods, and a WebAssembly file (.wasm) containing the compiled C ++ code. The browser may load the JavaScript source file before completely loading and compiling the WebAssembly file. In order to accommodate this, Emscripten libraries provide a 'then()' method for the JavaScript module object similar to a JavaScript Promise. This method accepts a callback that will execute once the WebAssembly is fully downloaded and compiled.

Manual Memory Management

Most modern languages feature some type of automatic garbage collection. However, WebAssembly is a low–level binary–like format, and hence does not provide high–level features like garbage collection. This means that whenever the user creates an object in libsbmljs using the new keyword, the user must also destroy the object using libsbml.destroy(obj). In most cases, this simply amounts to destroying the SBML document when it is no longer needed.

In terms of modern programming languages, this may seem like a significant regression, but it is an unavoidable tradeoff when using C++ compiled WebAssembly, at least for currently available technology (a proposal exists to add garbage collection to WebAssembly [24], but an implementation is not available at the time of writing). In the event that the user forgets to call the libsbml.destroy function, the allocated object will persist in the browser's memory until the browser tab is closed. Since our main target users are developers of web applications, and browser tabs are short–lived, we do not believe this is a significant concern. However, Node.js developers should take care to destroy all created objects. The same requirement also applies to libSBML's native Node.js module.

In SBML, many elements are contained in so–called "listOfX" structures. We have chosen not to include these ListOf structures in libsbmljs because they are merely simple containers for other elements. Moreover, we found that the parent element of a listOf structure always has methods for adding, removing, and iterating over the contained elements, making the listOf structure redundant. For example, the libSBML Model class contains a getListOfReactions method, but also contains createReaction, removeReaction, getNumReactions, and getReaction. Taken together, these methods allow all operations provided by the underlying listOfReactions element. This pattern is repeated for many SBML elements. Excluding these redundant elements also creates a slight size reduction in the JavaScript package. Table 1 shows the elements we have intentionally left out of the wrapper.

Client–side SBML Simulation

We used libsbmljs to construct a fully client–side, web–based SBML simulator (sbml_websim), which supports ODE–based (using the Bulirsch–Stoer algorithm [26, 27] implemented in JavaScript [28]) and stochastic (using the Next Reaction / "Gibson" method [29]) simulations. This simulator is directly connected to the BioModels / BiGG Models browser demo associated with this manuscript (https://libsbmljsdemo.github.io). The simulator supports SBML core features including rate rules and events. Although slow compared to state–of–the–art simulators like libroadrunner [30], the main utility of this simulator is to demonstrate the technological readiness of this approach for creating standards–supporting web–apps. This method could be used, for example, to integrate simulation capability into the BioModels and BiGG Models repositories. Figure 2 shows web–based simulations of each of these models using sbml_websim. Figure 3 shows an example stochastic simulation.

Discussion & Conclusion

Currently, there is no web–capable library that can read and write SBML models. We have presented a WebAssembly / JavaScript library that can read and write all SBML packages. We have provided tutorials, examples and extensive API documentation for potential users. We have also provided a modular build system that can be used to regenerate the wrapper from any recent checkout of the libSBML C++ library from the stable or experimental branch, as well as in–browser tests of the wrapper using the Karma testing engine. Additionally, we have used this wrapper to create the first web– based client–side SBML simulator. We hope these advances will enable the development of systems biology web applications and services that can use the SBML standard.

Acknowledgments

Funding

HMS was supported by NIH grants GM123032-01, NHLBI U01HL12219902, and NIBIB P41EB023912. JKM was supported by NIH grant GM12303201A1. JH was supported by the Moore/Sloan Data Science Environments Project at the University of Washington supported by grants from the Gordon and Betty Moore Foundation (Award #3835) and the Alfred P. Sloan Foundation (Award #2013-10-29).

Bibliography

References

[1]. Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H, Arkin AP, Bornstein BJ, Bray D, Cornish-Bowden A, et al., The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models, Bioinformatics 19 (4) (2003) 524–531. [PubMed: 12611808]

[2]. Olivier BG, Bergmann FT, SBML level 3 package: Flux balance constraints version 2, Journal of Integrative Bioinformatics 15 (1) (Mar. 2018). doi:10.1515/jib-2017-0082. URL 10.1515/jib-2017-0082

[3]. Becker SA, Feist AM, Mo ML, Hannum G, Palsson BØ, Herrgard MJ, Quantitative prediction of cellular metabolism with constraint-based models: the cobra toolbox, Nature protocols 2 (3) (2007) 727. [PubMed: 17406635]

[4]. Schellenberger J, Que R, Fleming RM, Thiele I, Orth JD, Feist AM, Zielinski DC, Bordbar A, Lewis NE, Rahmanian S, et al., Quantitative prediction of cellular metabolism with constraint-based models: the cobra toolbox v2. 0, Nature protocols 6 (9) (2011) 1290. [PubMed: 21886097]

[5]. King ZA, Dräger A, Ebrahim A, Sonnenschein N, Lewis NE, Palsson BO, Escher: a web application for building, sharing, and embedding data-rich visualizations of biological pathways, PLoS computational biology 11 (8) (2015) e1004321.

[6]. King ZA, Lu J, Dräger A, Miller P, Federowicz S, Lerman JA, Ebrahim A, Palsson BO, Lewis NE, Bigg models: A platform for integrating, standardizing and sharing genome-scale models, Nucleic acids research 44 (D1) (2015) D515–D522. [PubMed: 26476456]

[7]. Boele J, Olivier BG, Teusink B, Fame, the flux analysis and modeling environment, BMC systems biology 6 (1) (2012) 8. [PubMed: 22289213]

[8]. Bergmann F, Sbw flux balance, http://fbergmann.github.io/FluxBalance/(2013). URL http://fbergmann.github.io/FluxBalance/

[9]. Römer M, Eichner J, Dräger A, Wrzodek C, Wrzodek F, Zell A, Zbit bioinformatics toolbox: a web-platform for systems biology and expression data analysis, PloS one 11 (2) (2016) e0149263.

[10]. Klamt S, Saez-Rodriguez J, Gilles ED, Structural and functional analysis of cellular networks with cellnetanalyzer, BMC systems biology 1 (1) (2007) 2. [PubMed: 17408509]

[11]. Zhang F, Meier-Schellersheim M, Sbml level 3 package: multistate, multicomponent and multicompartment species, version 1, release 1, Journal of integrative bioinformatics 15 (1) (2018).

[12]. Le Novere N, Bornstein B, Broicher A, Courtot M, Donizelli M, Dharuri H, Li L, Sauro H, Schilstra M, Shapiro B, et al., Biomodels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems, Nucleic acids research 34 (suppl 1) (2006) D689–D691. [PubMed: 16381960]

[13]. Li C, Donizelli M, Rodriguez N, Dharuri H, Endler L, Chelliah V, Li L, He E, Henry A, Stefan MI, et al., Biomodels database: An enhanced, curated and annotated resource for published quantitative kinetic models, BMC systems biology 4 (1) (2010) 92. [PubMed: 20587024]

[14]. Olivier BG, Snoep JL, Web-based kinetic modelling using jws online, Bioinformatics 20 (13) (2004) 2143–2144. [PubMed: 15072998]

[15]. Peters M, Eicher JJ, van Niekerk DD, Waltemath D, Snoep JL, The jws online simulation database, Bioinformatics 33 (10) (2017) 1589–1590. arXiv:/oup/backfile/content_public/journal/bioinformatics/33/10/10.1093_bioinformatics_btw831/2/btw831.pdf, doi:10.1093/bioinformatics/btw831. URL +10.1093/bioinformatics/btw831 [PubMed: 28130238]

[16]. Franz M, Lopes CT, Huck G, Dong Y, Sumer O, Bader GD, Cytoscape.js: a graph theory library for visualisation and analysis, Bioinformatics 32 (2) (2015) 309–311. [PubMed: 26415722]

[17]. Lopez A, Fodor M, Sirunian A, Kaafarani A, Lieven C, Sonnenschein N, Azrak T, Beber ME, Dd-decaf/caffeine: Version 1, 10.5281/zenodo.2616028 (Mar. 2019). doi:10.5281/zenodo.2616028. URL 10.5281/zenodo.2616028

[18]. Webassembly, https://webassembly.org. URL https://webassembly.org/

[19]. Bornstein BJ, Keating SM, Jouraku A, Hucka M, Libsbml: an api library for sbml, Bioinformatics 24 (6) (2008) 880–881. [PubMed: 18252737]

[20]. Le Novere N, Rodriguez N, Wrzodek F, Mittag F, Fröhlich S, Hucka M, Thomas A, Palsson B, Lewis NE, Dräger A, Myers CJ, Watanabe L, Vazirabad IY, Kofia V, Gómez HF, Diamantikos A, Netz E, Matthes J, Eichner J, Keller R, Rudolph J, Wrzodek C, JSBML 1.0: providing a smorgasbord of options to encode systems biology models, Bioinformatics 31 (20) (2015) 3383–3386. arXiv:http://oup.prod.sis.lan/bioinformatics/article-pdf/31/20/3383/17087774/btv341.pdf, doi:10.1093/bioinformatics/btv341. URL 10.1093/bioinformatics/btv341 [PubMed: 26079347]

[21]. Dräger A, Rodriguez N, Dumousseau M, Dörr A, Wrzodek C, Le Novère N, Zell A, Hucka M, JSBML: a flexible Java library for working with SBML, Bioinformatics 27 (15) (2011) 2167–2168. arXiv: http://oup.prod.sis.lan/bioinformatics/article-pdf/27/15/2167/13846361/btr361.pdf, doi:10.1093/bioinformatics/btr361. URL 10.1093/bioinformatics/btr361 [PubMed: 21697129]

[22]. Zakai A, Emscripten: an llvm-to-javascript compiler, in: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, ACM, 2011, pp. 301–312.

[23]. Rossberg A, Titzer BL, Haas A, Schuff DL, Gohman D, Wagner L, Zakai A, Bastien JF, Holman M, Bringing the web up to speed with webassembly, Commun. ACM 61 (12) (2018) 107–115. doi:10.1145/3282510. URL 10.1145/3282510

[24]. Webassembly garbage collection, https://github.com/WebAssembly/design/issues/1079. URL https://github.com/WebAssembly/design/issues/1079

[25]. Elowitz MB, Leibler S, A synthetic oscillatory network of transcriptional regulators, Nature 403 (6767) (2000) 335. [PubMed: 10659856]

[26]. Bulirsch R, Stoer J, Numerical treatment of ordinary differential equations by extrapolation methods, Numerische Mathematik 8 (1) (1966) 1–13.

[27]. Wanner G, Hairer E, Solving ordinary differential equations II, Springer Berlin Heidelberg, 1996.

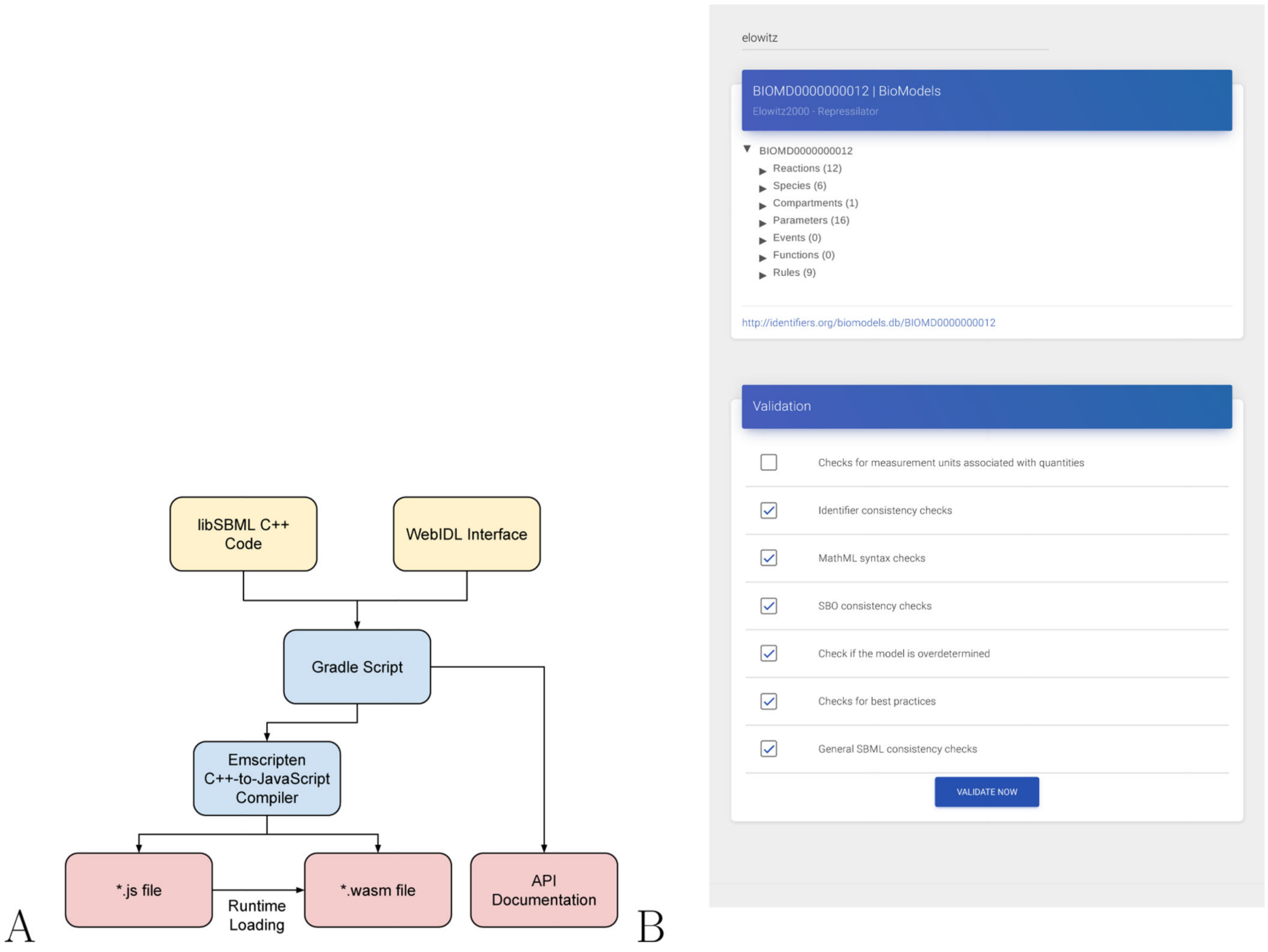
[28]. Smith C, odex-js, https://github.com/littleredcomputer/odex-js. URL https://github.com/littleredcomputer/odex-js

[29]. Gibson MA, Bruck J, Efficient exact stochastic simulation of chemical systems with many species and many channels, The journal of physical chemistry A 104 (9) (2000) 1876–1889.

[30]. Somogyi ET, Bouteiller J-M, Glazier JA, König M, Medley JK, Swat MH, Sauro HM, libroadrunner: a high performance sbml simulation and analysis library, Bioinformatics 31 (20) (2015) 3315–3321. [PubMed: 26085503]

Highlights

- This article describes a feature–complete SBML library for client–side web applications
- Extensive documentation, tests are provided to allow others to build SBML–supporting web applications
- An online demo featuring the ability to browse the BioModel and BiGG Models repositories, validate SBML, and run simulations is described in order to demonstrate technology readiness



elowitz

BIOMD0000000012 | BioModels

Elowitz2000 - Repressilator

▼ BIOMD0000000012
▶ Reactions (12)
▶ Species (6)
▶ Compartments (1)
▶ Parameters (16)
▶ Events (0)
▶ Functions (0)
▶ Rules (9)

http://identifiers.org/biomodels.db/BIOMD0000000012

Validation

- [ ] Checks for measurement units associated with quantities
- [✓] Identifier consistency checks
- [✓] MathML syntax checks
- [✓] SBO consistency checks
- [✓] Check if the model is overdetermined
- [✓] Checks for best practices
- [✓] General SBML consistency checks

VALIDATE NOW

Figure 1:

(A) A workflow diagram of the process used to produce libsbmljs. The libSBML C++ source code and a hand–written WebIDL interface are processed by a Gradle script to produce Emscripten–compiled bytecode and JavaScript API documentation. The Emscripten bytecode is further compiled into separate JavaScript (*.js) and WebAssembly (*.wasm) files. When the JavaScript source file is loaded by the browser, it executes instructions to fetch the corresponding WebAssembly file asynchronously. These two files are then combined into an npm package. **(B)** A screenshot of the demo page showing the Repressilator model [25] in the BioModels database (BIOMD0000000012). After selecting a model via using the demo's search bar or uploading an SBML file, the demo allows the user to view SBML content as a tree–like structure and validate the SBML model subject to the validation options provided by libSBML. This particular model can be viewed at https://libsbmljsdemo.github.io/#/view?m=BIOMD0000000012

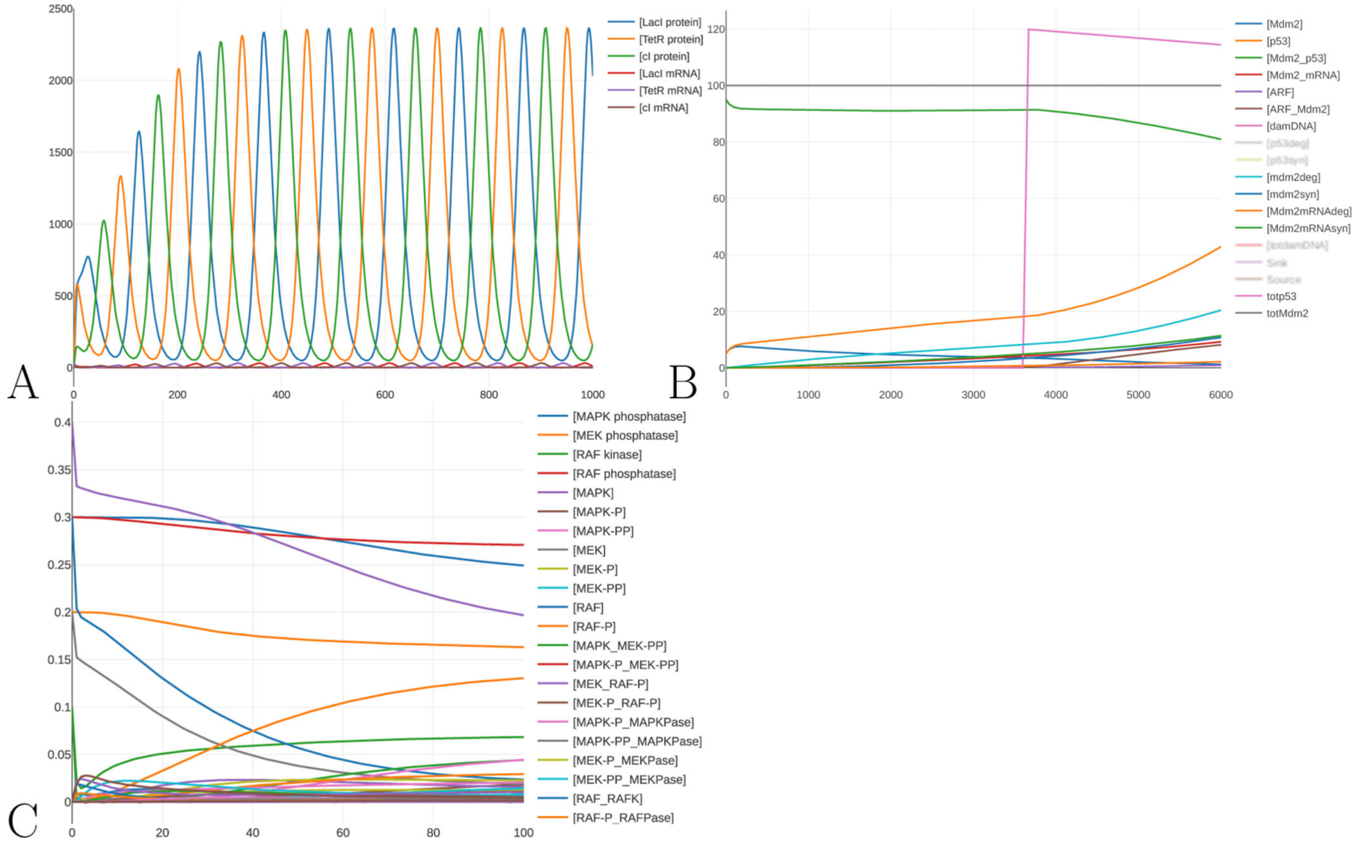


Figure 2:

Example web–based simulations of BioModels: the repressilator (https://libsbmljsdemo.github.io/#/view?m=BIOMD0000000012, 12 reactions, **A**), p53 p14ARF (https://libsbmljsdemo.github.io/#/view?m=BIOMD0000000189, 14 reactions, **B**), and MAPK (https://libsbmljsdemo.github.io/#/view?m=BIOMD0000000014, 300 reactions, **C**) models. These results were separately compared to the libroadrunner simulator to verify accuracy (not shown).

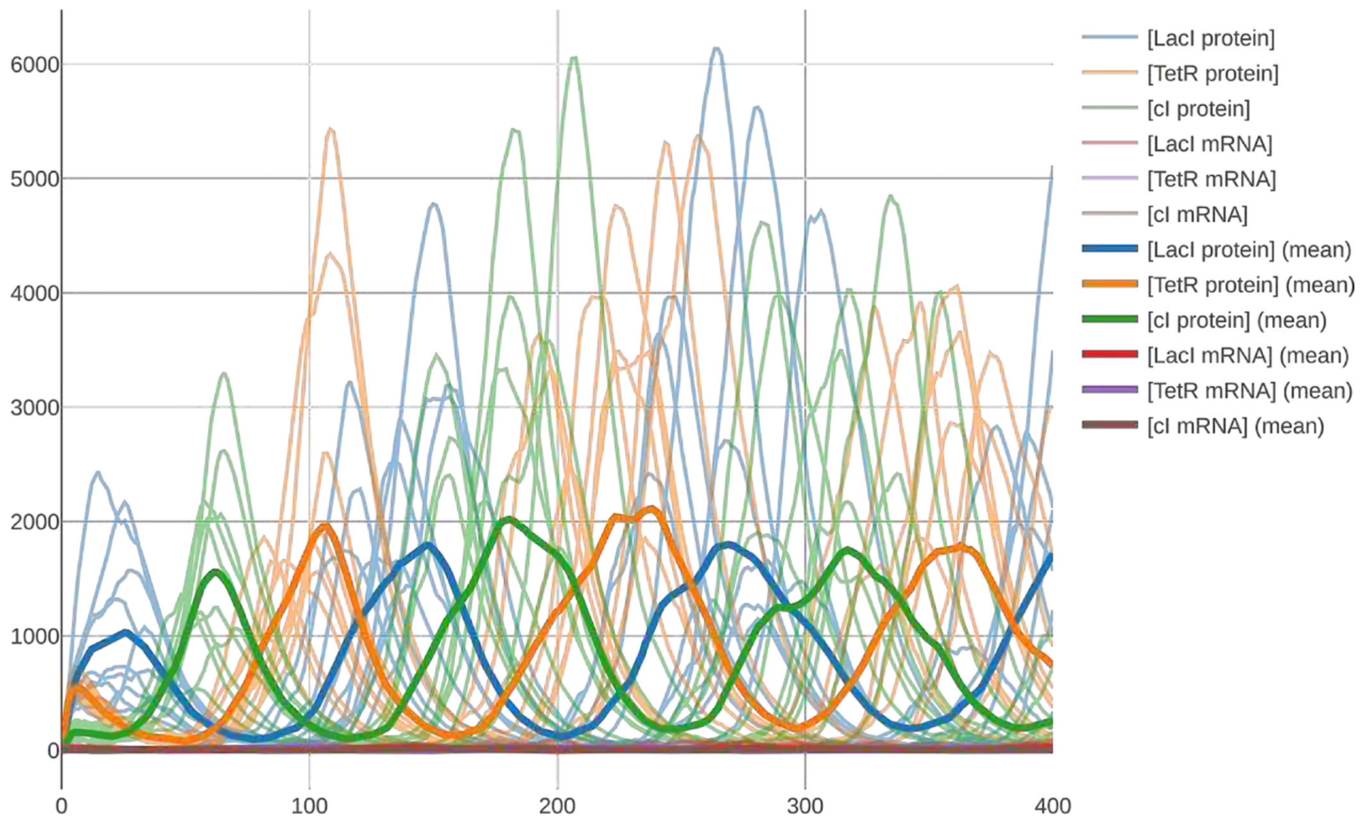


Figure 3:
A stochastic simulation of the repressilator model the Next Reaction Method[29]. sbml websim allows the user to repeat the stochastic simulation for a desired number of replicates (10 here) and plots all replicates (faint lines) in addition to the mean value of each variable over all replicates (solid line).

Table 1:

List of classes and members intentionally left out of the libsbmljs core wrapper.

- SBase
  - appendAnnotation (const XMLNode *annotation)
  - appendNotes (const XMLNode *notes)
  - clone () const
  - deleteDisabledPlugins (bool recursive=true)
  - disablePackage (const std::string \&pkgURI, const std::string \&pkgPrefix)
  - enablePackage (const std::string \&pkgURI, const std::string \&pkgPrefix, bool flag)
  - getAnnotation ()
  - getCVTerms ()
  - getElementName () const
  - getLevel () const
  - getLine () const
  - getModelHistory () const
  - getNotes ()
  - getUserData () const
  - matchesRequiredSBMLNamespacesForAddition (const SBase *sb)
  - read (XMLNode \&node, XMLErrorSeverityOverride_t flag=LIBSBML_OVERRIDE_DISABLED)
  - removeFromParentAndDelete ()
  - removeTopLevelAnnotationElement (const std::string &elementName, const std::string elementURI="", bool removeEmpty=true)
  - setModelHistory (ModelHistory *history)
  - toSBML ()
  - toXMLNode ()
  - unsetModelHistory ()
  - unsetUserData ()
- ListOfUnitDefinitions (all)
- ListOfCompartments (all)
- ListOfSpecies (all)
- ListOfParameters (all)
- ListOfRules (all)
- ListOfReactions (all)
- Model
  - getListOf* () const
- XMLAttributes
- XMLConstructorException
- XMLError
- XMLErrorLog
- XMLInputStream
- XMLLogOverride
- XMLNamespaces
- XMLNode

• XMLOutputStream
• XMLToken
• XMLTriple