JSBML 1.0:

Providing a Smorgasbord of Options to Encode Systems Biology Models

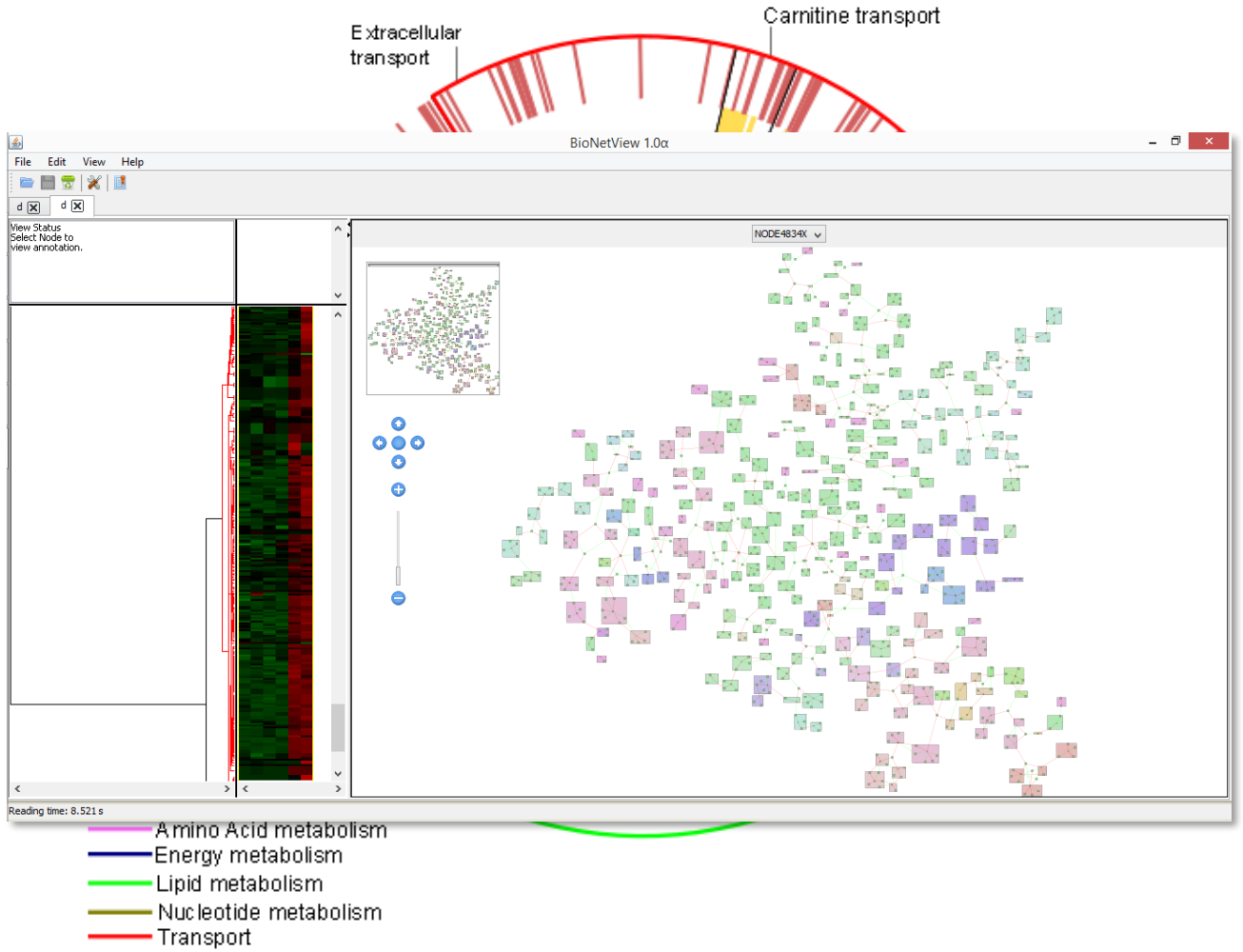
Presented By Alex Thomas

What is JSBML?

- JSBML is a Java language parser and writer for SBML
- *JSBML be used to hold systems biology information, with SBML specified architecture, in a Java data structure for fast access*
- No libSBML dependencies
  - Portable
  - Fluid use with the Java Virtual Machine (JVM)
- Implements SBML specifications
  - SBML (Core) Level 3 Version 1
  - Approved SBML Level 3 extensions
  - All older SBML (Core) levels and versions

JSBML and me

- Joined JSBML development team September 2013
- Research with Nathan Lewis (Primary PI) and Bernhard Palsson
- JSBML development with Andreas Dräger and Nicolas Rodriguez
- Constraints based modeling research
  - network reconstruction
  - data visualization
  - optimization algorithm development
- Use JSBML for encoding constraints based models and graph layout
  - fbc
  - layout



Getting Started

- Downloading and Using JSBML
  - With dependencies
  - Without dependencies
  - Source code
  - Maven (coming soon for 1.0 release)

- Hello World examples

```
cd jsbml-1.0
ant compile
```

Compiling JSBML with Ant; this example uses Bash shell syntax.

Next, if you wish to run the self-tests included with JSBML, you can do so by running the following command:

```
ant test
```

Running the unit tests provided with JSBML.

Finally, if you want to produce a JAR file containing all the JSBML compiled class files, run the following command:

```
ant jar
```

Creating a JAR file.

```java
public class JSBMLexample implements TreeNodeChangeListener {

    public JSBMLexample() throws Exception {
        // Create a new SBMLDocument object, using SBML Level 3 Version 1.
        SBMLDocument doc = new SBMLDocument(3, 1);
        doc.addTreeNodeChangeListener(this);

        // Create a new SBML model, and add a compartment to it.
        Model model = doc.createModel("test_model");
        Compartment compartment = model.createCompartment("default");
        compartment.setSize(1d);

        // Create a model history object and add author information to it.
        History hist = model.getHistory(); // Will create the History, if it does not exist
        Creator creator = new Creator("Given Name", "Family Name", "Organisation", "My@EMail.com");
        hist.addCreator(creator);

        // Create some sample content in the SBML model.
        Species specOne = model.createSpecies("test_spec1", compartment);
        Species specTwo = model.createSpecies("test_spec2", compartment);
        Reaction sbReaction = model.createReaction("reaction_id");

        // Add a substrate (SBO:0000015) and product (SBO:0000011) to the reaction.
        SpeciesReference subs = sbReaction.createReactant(specOne);
        subs.setSBOTerm(15);
        SpeciesReference prod = sbReaction.createProduct(specTwo);
        prod.setSBOTerm(11);

        // For brevity, WE DO NOT PERFORM ERROR CHECKING, but you should,
        // using the method doc.checkConsistency() and then checking the error log.

        // Write the SBML document to a file.
        SBMLWriter.write(doc, "test.xml", "JSBMLexample", "1.0");
    }
}
```

Getting Started

- Downloading and Using JSBML
  - With dependencies
  - Without dependencies
  - Source code
  - Maven (coming soon for 1.0 release)

- Hello World examples

```java
import java.beans.PropertyChangeEvent;
import javax.swing.tree.TreeNode;
import org.sbml.jsbml.*;
import org.sbml.jsbml.util.TreeNodeChangeListener;
import org.sbml.jsbml.util.TreeNodeRemovedEvent;

/** Creates an {@link SBMLDocument} and writes its contents to a file. **/
public class JSBMLexample implements TreeNodeChangeListener {

    public JSBMLexample() throws Exception {
        // Create a new SBMLDocument object, using SBML Level 2 Version 4.
        SBMLDocument doc = new SBMLDocument(2, 4);
        doc.addTreeNodeChangeListener(this);

        // Create a new SBML model, and add a compartment to it.
        Model model = doc.createModel("test_model");
        Compartment compartment = model.createCompartment("default");
        compartment.setSize(1d);

        // Create a model history object and add author information to it.
        History hist = model.getHistory(); // Will create the History, if it does not exist
        Creator creator = new Creator("Given Name", "Family Name", "Organisation", "My@EMail.com");
        hist.addCreator(creator);

        // Create some sample content in the SBML model.
        Species specOne = model.createSpecies("test_spec1", compartment);
        Species specTwo = model.createSpecies("test_spec2", compartment);
        Reaction sbReaction = model.createReaction("reaction_id");

        // Add a substrate (SBO:0000015) and product (SBO:0000011) to the reaction.
        SpeciesReference subs = sbReaction.createReactant(specOne);
        subs.setSBOTerm(15);
        SpeciesReference prod = sbReaction.createProduct(specTwo);
        prod.setSBOTerm(11);

        // For brevity, WE DO NOT PERFORM ERROR CHECKING, but you should,
        // using the method doc.checkConsistency() and then checking the error log.

        // Write the SBML document to a file.
        SBMLWriter.write(doc, "test.xml", "JSBMLexample", "1.0");
    }
```

Using Extensions: Spatial

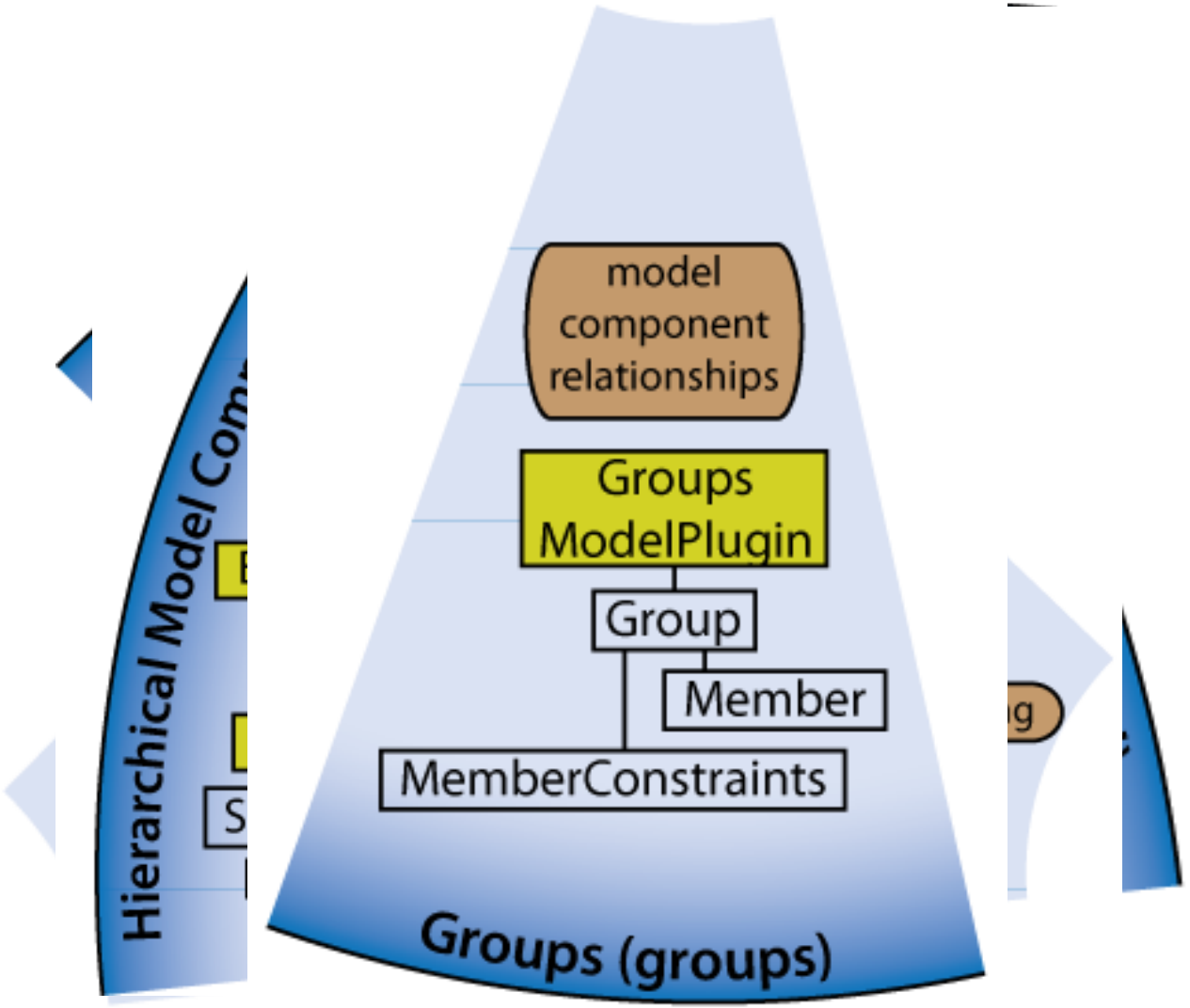
```
public class SpatialTest {

    public static void main(String[] args) throws SBMLException, XMLStreamException {
        int level = 3, version = 1;
        Species spec1,spec2,spec3;
        Reaction rxn1;

        SBMLDocument doc = new SBMLDocument(level, version);
        Model model = doc.createModel("my_model");

        // Normal model

        Compartment comp1 = model.createCompartment("comp1");

        spec1 = model.createSpecies("a", comp1);
        spec2 = model.createSpecies("b", comp1);
        spec3 = model.createSpecies("c", comp1);

        rxn1 = model.createReaction("r1");

        rxn1.addReactant(new SpeciesReference(spec1));
        rxn1.addReactant(new SpeciesReference(spec2));
        rxn1.addProduct(new SpeciesReference(spec3));
```

…

```
// Creating the spatial model extension and adding it to the document

        // Create spatial extensions for model, compartment and species
        SpatialModelPlugin spatialModelPlugin = new SpatialModelPlugin(model);
        model.addExtension(SpatialConstants.getNamespaceURI(level, version), spatialModelPlugin);

        SpatialCompartmentPlugin spatialComp = new SpatialCompartmentPlugin(comp1);
        comp1.addExtension(SpatialConstants.getNamespaceURI(level, version), spatialComp);

        // Add non-SBML-core classes

        Geometry geo = spatialModelPlugin.createGeometry();

        CompartmentMapping spatialCompMap = new CompartmentMapping();
        spatialComp.setCompartmentMapping(spatialCompMap);
        spatialCompMap.setCompartment("comp1");
        spatialCompMap.setDomainType("DomainType1");

        SBMLWriter.write(doc, “Test.xml”);

    }
}
```

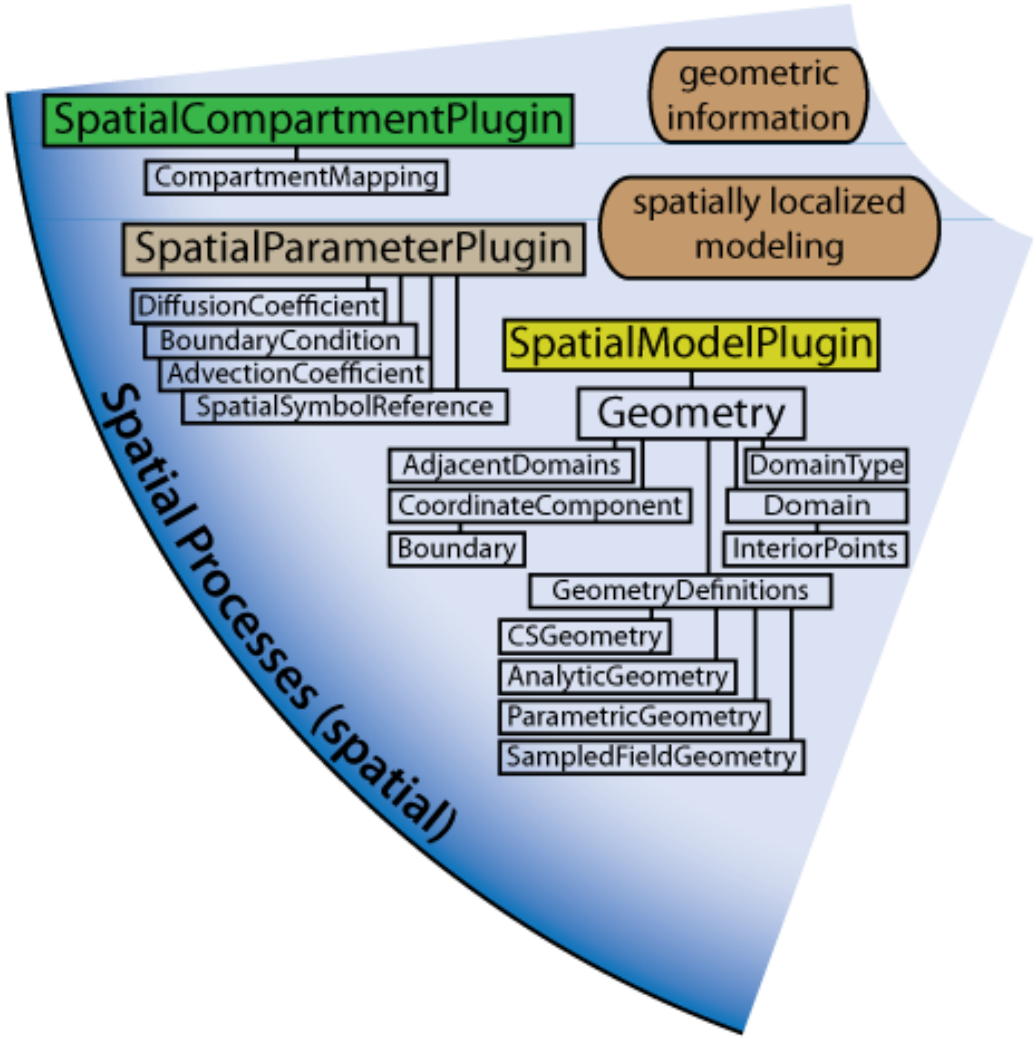
```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" xmlns:spatial="
http://www.sbml.org/sbml/level3/version1/spatial/version1" spatial:required="true" version="1">
    <model id="my_model">
        <geometry/>
        <listOfCompartments>
            <compartment id="comp1">
                <compartmentMapping spatial:domainType="DomainType1" spatial:compartment="comp1"/>
            </compartment>
        </listOfCompartments>
        <listOfSpecies>
            <species id="a" compartment="comp1"/>
            <species id="b" compartment="comp1"/>
            <species id="c" compartment="comp1"/>
        </listOfSpecies>
        <listOfReactions>
            <reaction id="r1">
                <listOfReactants>
                    <speciesReference species="a"/>
                    <speciesReference species="b"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="c"/>
                </listOfProducts>
            </reaction>
        </listOfReactions>
    </model>
</sbml>
```

SBML Level 3 Extension Support

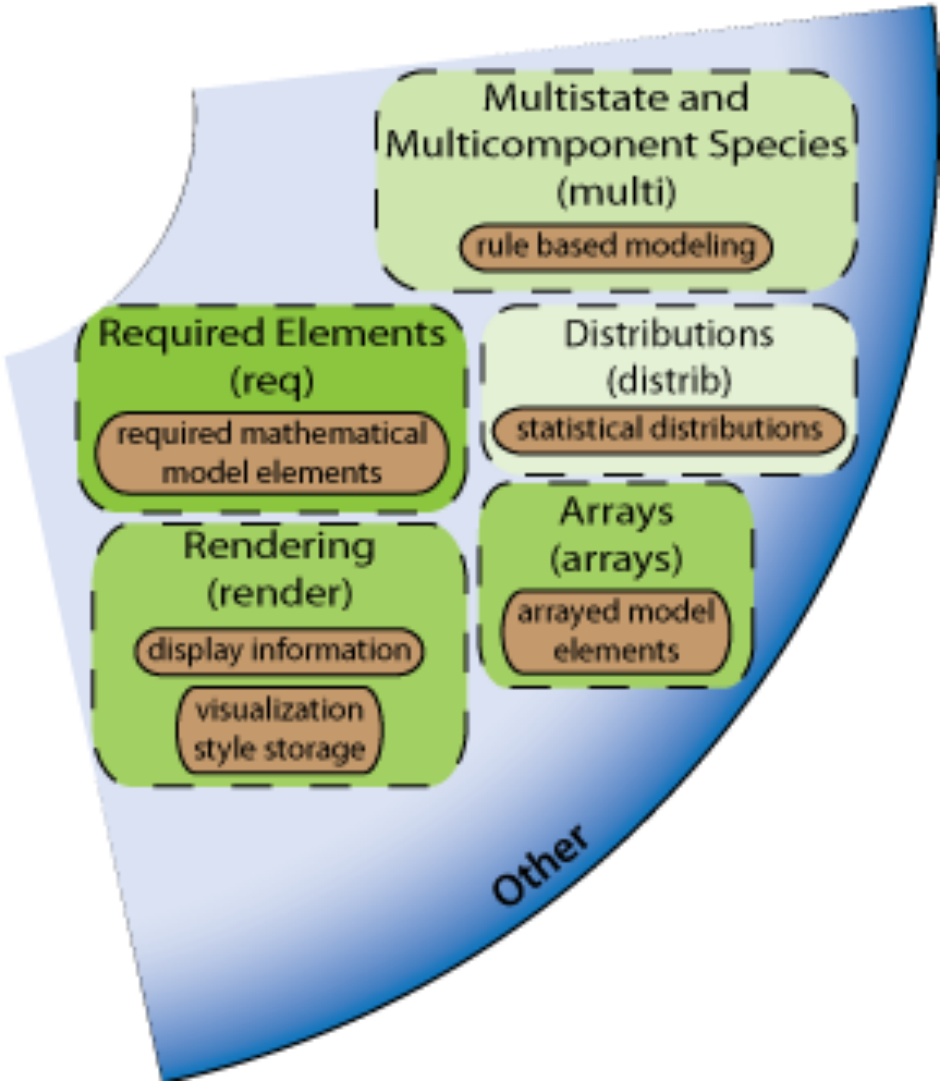


Forays into Coding Extensions: Spatial

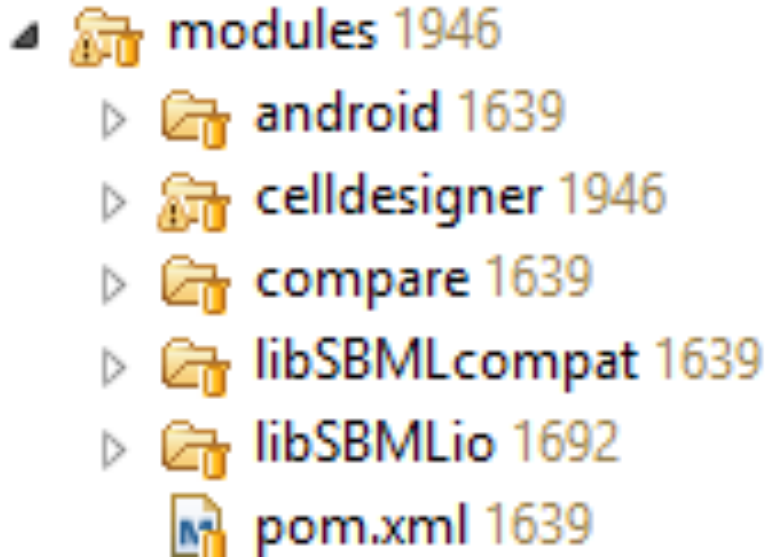
1) Setup Eclipse for JSBML
   1) Java 1.5 compatibility
   2) Add dependencies to build path
   3) Java Annotations processing
2) Set up class hierarchy
   1) “Plugin” classes for SBML core extensions
   2) Code classes that utilize Object Oriented approach to minimize attribute duplication
3) Follow user guide and use code templates
4) Match libSBML functionality
5) Implement reader/writer
6) Add JUnit tests and utility methods



Support for Pending Extensions



Modules for Improved Interfacing

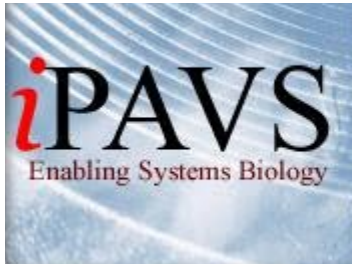


- libSBMLio – reveals libSBML functions that can be used for conversion to JSBML data structure
- CellDesigner – syncs JSBML with CellDesigner's plugin data structure for easy integration
- Android – provides classes from the Java standard distribution for JSBML that may be missing on Android systems
- libSBMLcompat – establishes two way correspondence between JSBML and libSBML Java API
- Compare – draws comparisons between libSBML and JSBML

Easing From LibSBML-Java Bindings to JSBML

- Since version 0.8, JSBML has strived for 100% compatibility with libSBML's Java API
  - Common method and variable names
  - Compatibility (libSBMLcompat) module for further ease of transition
  - ASTNode interface has been updated to mimic libSBML
- Features that libSBML does that JSBML does not include:
  - SBML validation
  - SBML conversion between different levels and versions
- Extra features provided by JSBML
  - ChangeListeners
  - Fast "find" methods
  - Tools for String manipulation
  - Logging facilities

Current Programs that Use JSBML



Current Progress for JSBML 1.0

- All approved SBML Level 3 Extensions are Implemented
- Aim for 1.0 release during September
  - Integrate GSoC Projects
  - Maven support
  - Update user guide
  - Updating spatial package to meet specs in COMBINE
  - Minor bug fixes

Google Summer of Code project integration

- Quick Overview
  - Awarded GSoC funds May 2014 in collaboration with the Open Bioinformatics Foundation
  - Logistics: Duration for 12-weeks, $5500 awarded to each student
  - Met with mentors weekly, underwent midterm and final evaluation
  - Three projects completed to a stable state (working code with documentation)
- Proposed timeline for integration into the main branch:
  - Revised ASTNode interface (Victor Kofia): October 2014
  - Arrays package integration (Leandro Watanabe): September 2014
  - Improved CellDesigner interface (Ibrahim Vazirabad): September 2014

Conclusions

- JSBML is a stable, actively developed Java library for interacting with SBML data
- Easy to get involved in JSBML development
- JSBML 1.0 will be released very soon
  - SBML core and approved SBML Level 3 extensions have support
  - Some support for extensions with drafts
- GSoC is a great way to attract young developers to participate in the community (with funding!)

Acknowledgements

- Core JSBML team
  - Andreas Dräger
  - Nicolas Rodriguez
  - SBML community and international development team
  - jsbml-development@googlegroups.com
- GSoC students
  - Victor Kofia
  - Ibrahim Vazirabad
  - Leandro Watanabe
- The COMBINE Community
- Partial funding and application motivation
  - Nathan Lewis
  - Bernhard Ø. Palsson