molecular systems biology

Programming biological models in Python using PySB

Carlos F. Lopez, Jeremy L. Muhlich, John A. Bachman, Peter K. Sorger

Corresponding author: Peter K. Sorger, Harvard Medical School

Review timeline:

Submission date:	30 August 2012
Editorial Decision:	05 October 2012
Revision received:	21 December 2012
Accepted:	07 January 2013

Editor: Thomas Lemberger

Transaction Report:

(Note: With the exception of the correction of typographical or spelling errors that could be a source of ambiguity, letters and reports are not edited. The original formatting of letters and referee reports may not be reflected in this compilation.)

1st Editorial Decision 05 October 2012

Thank you again for submitting your work to Molecular Systems Biology. We have now heard back from the three referees who agreed to evaluate your manuscript. As you will see from the reports below, while referee #3 is somewhat reserved, the other two referees find the topic of your study of potential interest and are more supportive. Given the balance of opinions expressed by the reviewers, we feel that we can consider a revision of the present work that would address the major points raised in the reports below.

Some of the major issues raised by the reviewers are the following:
- a unit testing framework would address the issue of reproducibility and robust code development raised by reviewers 1 and 3
- ease of installation is of crucial importance to favor the broad dissemination of PySB and all efforts should be undertaken to have very simple, robust, general and well documented installation procedure across platforms.
- some didactic boxes could be included in the manuscript to explain some of the syntactic peculiarities of PySB (eg the specific use of operator overloading in PySB) and perhaps to visualize the class hierarchy (or as supplementary figure).
- with regard to model composition and reuse, mechanisms that would link model components or variables to standardized nomenclature/identifiers (eg for gene/protein names, perhaps using identifiers.org) should be considered or at least discussed.

If you feel you can satisfactorily deal with these points and those listed by the referees, you may wish to submit a revised version of your manuscript. Please attach a covering letter giving details of the way in which you have handled each of the points raised by the referees. A revised manuscript will be once again subject to review and you probably understand that we can give you no guarantee at this stage that the eventual outcome will be favorable.

Reviewer #1 (Remarks to the Author):

Summary
==========
Comprehensive mathematical models are critically needed to accelerate biological discovery and bioengineering. In particular, greater interoperability is needed in the systems biology community to facilitate transparency and collaboration. The authors develop PySB, a simple and yet powerful open-source platform which enables modelers to easily build larger and more comprehensive models by taking advantage of abstraction, composition, and reuse. Importantly, PySB takes advantage of existing and familiar tools including Python, BioNetGen, Kappa, Graphviz, and Matplotlib. Furthermore, the authors use PySB to build three new model-variants of the Bcl-2-mediated cell death.

Overall, PySB is an important step toward better and more collaborative model building. PySB has numerous features which cover most of the needs of most modelers including the abilities to mix declarative and procedural modeling, simulate and fit models with standard numerical methods, and plot and analyze data. One of the strengthens of PySB is that it will become more powerful and more groups develop and share PySB models, providing richer and richer starting points for future researchers. PySB's main limitation is that it currently supports only two simulation engines.

Although PySB is an important step forward, it certainly doesn't solve all of the problems the authors listed in their introduction. Many of the problems are issues with the systems biology community itself, and can't be fixed by a single software library. More than anything else, I think more cohesion is needed within the systems biology community to take full advantage of PySB. Hopefully PySB pushes the community in the right direction.

Here are a few suggestions for the future of PySB and the systems biology community. I certainly don't expect them to be addressed by publication, but it could be useful to comment on them as part of the discussion. (1) The community needs to agree on how to name components. Perhaps PySB can encourage this by asking modelers to map every component onto an accession ID for KEGG (or some other comprehensive database). Otherwise, models will have to be rewritten to be reused in PySB. (2) The community also needs to agree on how to model individual components. Again, otherwise models will need to be rewritten, obviating the advantages of PySB. (3) It would be helpful to have a model repository similar to BioModels for sharing PySB-generated models. Encouraging users to commit to the Git repository is nice, but it would also be helpful to have more user-friendly and searchable interface similar to that at BioModels. Model submitted to the repository should be citable and have their own DOI. (4) It would be helpful to build models declaratively, not only from tables but also from public databases like KEGG. Obviously this requires coming to consensus on the content of those databases. (5) It would be helpful to have a difference feature which works at the level of the model in addition to the text difference program which work at the level of the code. The result of this could be a visual map which highlights the commonality and differences between models. Visual tools would be especially helpful for composing models. Perhaps this tools could be modeled after Simulink or Labview.

Finally -- and here is something that could be brought out more in this work -- the community needs to increase its emphasis on repeatability and testing. PySB could encourage this by providing sample tests for the 15 models discussed in this manuscript and including a testing framework in the PySB release.

Minor comments
==========
Overall, the paper is very well written. Here are several places where the manuscript could be clarified.

- Title, pp3, pp18: Clarify what is meant by "second generation." What do you consider first generation? What else is part of generation #2?

- Introduction: The introduction is very well written, but brings up numerous themes (1) fit-to-purpose vs variant, 2) modifying models, 3) comparing models, 4) collaborative, iterative construction, 5) errors and testing, 6) model sharing, 7) abstraction, composition). It could be helpful to provide a more extensive summary at the end of the first paragraph on pp 3 to orient the reader.

- Introduction: It would be helpful to briefly give broader introduction and motivation to the Bcl-2 biology. In particular, describe why mitochondrial membrane permeabilization is important.

- pp14: Clarify that the Howells model is one of the 7 Shen models. Clarify that the Lopez models are the same as the Green/Andrews models.

- pp14: Clarify how you arrive at 30 = 15 x 2 models. Which module is the extrinsic apoptosis model?

- pp18: Can you provide more information about what Python numerical tools you used for model fitting or point to the methods section?

- Table 2: Clarify that Lopez=Green/Andrews

- Table 2: Clarify that the notation in the ID column refers to the (a) and (b) in the last two row headers.

- Figure 1B: The result corresponding to "PySB declarations" is just a copy of the code; It could be more helpful to display the result of printing the model to stdout

- Figure 1B: It could be helpful to add an arrow head from "Integrate (VODE)" to "plot(...)"

- Figure 3B: Last box the method should be "rec_to_bind" rather than "rec_to_disc"

- Figure 4A: Is there significance to the horizontal alignment of the modules? This could be helpful to describe for the reader.

- Figure 4A: Clarify that Lopez group is the same as Green/Andrews (there's a mismatch between how its written in the text vs the figure)

- Figure 4A: Missing " at end of `Lopez "Embedded`

- Figure 4B: Missing

- Figure 6: Clarify the key in the figure to indicate light->dark means successive annealing generations

- Figure 6: It looks like you need to expand the y-range to show the tick marks.

Reviewer #2 (Remarks to the Author):

Review: "PySB: A second-generation approach to programming biological models"

Overall this is a useful study that produces a tool that has the potential to be quite useful in systems biology. This is a significant achievement. The logic, structure, and proposed solutions to improving the programming of biological models are sound and valid. The claims that it improves upon older

methods like BNG and Kappa are reasonable. The choices of language and implementation techniques are valid. However it should be noted that the integrations mentioned such as NumPy are SciPy, areintegral parts of Python not only of PySB. The concept of using reusable macros and dividing complex networks into modules is conceptually correct and useful. Object/Process orientation and the association with delegates have long been explored and this application here is a particular implementation choice. Code sharing through GIT and other open source methods is useful. Many scientific libraries has been published in such contexts like SciPy (https://github.com/scipy/scipy).
The claim for incremental and collaborative model development is an initial claim and no work has been done to demonstrate these claims as yet. This comment is meant to address the concern that the authors could develop new approaches for the next generation of biomodels sharing. Putting PySB on GIT may allow for code modification to PySB, but that is only the first step in promoting distributed, incremental and collaborative approaches to modeling. One suggestion is to enable PySB to utilize serialized instance reuse, module reuse, and class reuse and allow for dynamical simulation of combined models and modification of such models.
Overall it appears that while this a great tool and has many advantages as cited beyond the current tools, It's improvement in usability and benefit to scientific community is undoubtedly true to me, but remains to demonstrated.

Detailed Comments
As this is a software suite the following issues could be addressed

Code Quality:
Ease of Setup:
- There are no instructions available on the web:
http://docs.pysb.org/en/latest/installation.html#instructions-for-windows-installations

- However, the reviewer's student had no problem setting up given prior python knowledge.
- Example/*.py does not run readily. Code should be modified to run upon down load.

- It would be easier if dependencies could come prepackaged. This should done for those packages that are not restricted by distribution licenses. This problem is somewhat mitigated by the install scripts.

Ease of Use:
- Straight forward black box import method is simple to use and understand for skilled individuals, who know python. This may be of problem if knowledge of python is basic or limited as implementation requires knowledge of operator overloading and order of operation in python to understand and extend this code. This may lead to unpredictable events in the future, if arithmetic rules diverge from biological rules.
- Python syntax is a non-trivial for many computational biologists, which may limit usage Some of the syntax is not common usage (Ex: None >> A()) is unclear of the function in this syntax.
- Process Orientation is excessively used, which may detract from understanding the code and lead to confusion.
o Ex: Model() creating an implicit "model" global reference is not explicitly defined here; which may lead users to wonder why they need to use pint.odesolve(model, t) inside the "synth_deg.py" example.
Ex: "model = Model()" instead of "Model()"
Clarity may be improved by using a combination of Object orientation and polymorphism with process orientation, which is done inside core.py but using explicit references instead of global stores through "class SelfExporter".

Commenting:
- Example code needs to be documented.
- Documentation inside main class like core.py also could be improved; however parameter naming is done well to give descriptive power.
- Use of non-standard expression overloading is confusing and only syntactical sugar for the creators and may not be helpful in building a community of users. See examples below

o "egf(R=1) % egfr(L=1)"
o "egfr(L=None) + egf(R=None) <> egfr(L=1) % egf(R=1)"
Are likely to be unclear and difficult to understand for programmers this needs to be improved by reference to a process diagram

Organization:
- Actual class structure follows standard procedures for encapsulation and acceptable for use and extension.
- However there are too many classes inside the same file (Ex: Core.py). The authors could utilize sub modules and divide large files into smaller files with classes for better source control and extensibility.
- Documentation is missing a process diagram of class hierarchy and information flow, which can improve understanding and reduce the learning curve for modifying the library.

Reviewer #3 (Remarks to the Author):

The well-written manuscript by Lopez et al. reports on a software package for building ODE models of biochemical networks via a programmatic interface called PySB. It is written in Python and can therefore leverage many packages already written in that language for scientific computing. It uses the powerful "rules-based" BioNetGen engine to convert reaction rules into a simulatable set of equations, and utilizes other previously developed tools for both solving these equations and visualizing the results.

The main novelty of this approach lies in the integration of several previously developed tools within a Python umbrella. This is in itself useful, as it provides a means to address some of the shortcomings of using BioNetGen alone, and also as the authors point out, provides a way of modularizing pieces of models so that they are called algorithmically rather than hard coded as equations. However, it is not clear how this is significantly different from for instance the NFSim or Virtual Cell softwares which also implements BioNetGen, but with a sophisticated graphical user interface. This brings up another point in that PySB seems complex to install and use; in fact on the website the authors say it is a "bit tricky to install". If the developers themselves are saying this, one can imagine the curious biologist would find it very difficult to use. Without making it easy for end users, it is likely this software will not be heavily used.

One point the authors bring up several times is the ability to easily integrate models with PySB. They argue that when doing this by the "cut and paste" method, one is left with the problem of unique variable names, and then matching them together. Yet, it is unclear how PySB might automatically solve this problem. It would seem careful analysis of two model's variables is needed regardless of the software package one uses to stitch them together. One could see this being accomplished through very careful annotation perhaps (for example with unique geneIDs tied to model variables), but not simply by making models with programs rather than text files. Moreover many models currently exist in SBML format in public databases which makes sharing them quite straightforward.

They also make the point that by creating "higher order function" such as enzyme catalyzed reactions with a macro, one ensures they are correct. This is only true if the macro itself is correct; how does one ensure this for likely a rapidly growing base of such macros? Moreover, existing software such as Copasi does similar things, and again with a graphical interface and host of associated functions like parameter estimation.

In conclusion, although PySB seems to be a nice piece of software with some potential, the novelty seems incremental. Therefore I would recommend that this manuscript does not represent an advance significant enough for publication in Molecular Systems Biology.

1st Revision - authors' response 21 January 2013

DETAILED RESPONSE TO REVIEW
Reviewer 1:

Suggestions for the future of PySB - addressed in the discussion.

1.1 The community needs to agree on how to name components. *Perhaps PySB can encourage this by asking modelers to map every component onto an accession ID for KEGG (or some other comprehensive database). Otherwise, models will have to be rewritten to be reused in PySB.*

This is an interesting idea (discussed in the cover letter) with which we are sympathetic. We agree that annotating models with meaningful metadata to streamline processes of model integration is an important and non-trivial problem. As a preliminary step to addressing this concern, we have added a basic "Annotation" capability to PySB allowing model components to be associated with external identifiers, which we describe further below. However, we suspect that significant challenges lie in the way of a comprehensive solution to this problem:

- In most projects the identities and activities of biochemical components vary with the purpose, scope and detail of the model. For example, a coarse-grained description of death receptor signaling might model the formation of a Death Inducing Signaling Complex ('DISC') upon ligation of a receptor ('R') by its ligand ('L') as follows:

  L + R <-> LR -> DISC.

  This level of description, which aggregates several distinct gene products into a handful of generic species, is sufficient for models (such as those in EARM 2.0) whose focus is primarily on the downstream reactions of extrinsic apoptosis. However, a model focusing on the detailed mechanism of receptor activation and DISC assembly might include these additional species, such as multiple ligands (FasL, TRAIL), multiple receptors (DR4, DR5), decoy receptors (DcR1 and DcR2), and additional components of the DISC (FADD, FLIP, etc.). This example is intended to underscore the fact that species and interactions used in models are not necessarily intended to represent single, specific gene products.

- It is not yet clear how the MIRIAM annotation framework should be applied to rule-based models. For example, while a Monomer object representing a particular protein could be linked to a UniProt ID, how should that Monomer's sites be annotated? Should site names be keyed to structural features (e.g., a catalytic site or a specific post-translational modification site)? How should site modifications be annotated? This is an active area of research in the rule-based modeling community.

In light of these challenges, the approach we have taken is to add a lightweight "Annotation" class to the PySB core. An Annotation consists of a MIRIAM-style *subject-object-predicate* triple. The *subject* can be any model element, including not only Monomers and Rules but also locally defined functions implementing macros or modules (since functions are first-class objects in Python). The *object* is a string which can be used to link to an external identifier (e.g., an identifiers.org URL). The *predicate* is a string,

which should be chosen from the BioModels qualifiers (e.g., "is" or "isDescribedBy"). For example:

```
Annotation(Bax, 'http://identifiers.org/uniprot/Q07812')
```

(If no predicate is specified, it defaults to "is".)

We used this approach to annotate several of the example models included with PySB as well as the "Albeck" and "Lopez" models in EARM. The Annotations for model M1a were used to generate the MIRIAM annotations for its corresponding SBML version. We consider a full implementation of MIRIAM annotations in PySB outside the scope of the present work but we do anticipate building on the current lightweight implementation as best practices for annotating rule-based models evolve.

We added text describing this feature in the "Integration with the Python ecosystem and external modeling tools" section of the Results ("Finally, to facilitate unique identification of model components…").

1.2 The community also needs to agree on how to model individual components. *Again, otherwise models will need to be rewritten, obviating the advantages of PySB.*

As discussed further in response 3.5, we recognize that the challenge of model re-use is only partly technical—the more fundamental challenge is determining the extent to which a fit-to-purpose model previously written to apply to a particular biological context remains applicable to a new context. This is a human-in-the-loop process that requires biological expertise and judgment. The problem of agreeing on how to model individual components is similar to the problem of agreeing on the "function" of proteins—and our understanding of even the most well-studied proteins remains in flux (e.g., recent findings for *myc*: Nie et al. (2012) *Cell*, 151:1, pp. 68-79). For these reasons we anticipate the immediate application of PySB to take place among groups of researchers working on similar biological pathways or contexts (e.g., extrinsic apoptosis).

The focus of PySB is to find ways to make fit-to-purpose modeling more transparent and manageable—not necessarily to facilitate the creation of "universal" models of biological processes. PySB will facilitate the creation of multiple, alternative models of similar biological processes but we do not see this as necessarily problematic if these models can be a) transparently understood and compared, and b) incrementally modified as new understanding emerges. Our approach has been to facilitate transparency (through the creation of reusable elements and human-readable syntax) and to promote documentation where ambiguities occur. If consensus approaches to modeling components or pathways are to emerge, we see PySB playing a role primarily as allowing for "bottom-up" experimentation with alternative approaches.

We have added additional language to the discussion to clarify our approach to this problem (final two paragraphs).

1.3 would be helpful to have a model repository similar to BioModels *for sharing PySB-generated models. Encouraging users to commit to the Git repository is nice, but it would also be helpful to have more user-friendly and searchable interface similar to that at BioModels. Model submitted to the repository should be citable and have their own*

DOI.

There are two Git repositories associated with the work in this paper. The first Git repository contains the code for the PySB framework itself (https://github.com/pysb/pysb); the second contains the code for the extrinsic apoptosis models (https://github.com/sorgerlab/earm). A key point about PySB models is that the code *is* the model. Developers of novel PySB models can publish their models/code in any of the ways that code can be published—however, we recommend an online distributed version control system such as GitHub.

As noted by the reviewer, although publishing model source code in this fashion has many advantages (chiefly in the ways it facilitates sharing and revision), it does not make use of the extensive work that has already been invested in formally annotatable, searchable, citable model databases such as BioModels.org. In keeping with norms for journals such as MSB we believe that models should still be deposited as exported SBML files in BioModels. We agree with the reviewers' suggestions that greater integration be achieved and propose to work with the BioModels group on the issue of cross-indexing PySB models in GitHub with corresponding SBML models in BioModels. We have added text to this effect to the final paragraph of the discussion ("SBML versions of PySB models can be deposited in biomodels.net…").

As a demonstration of this approach we have deposited the annotated M1a ("Lopez Embedded") model in BioModels. The MIRIAM annotations of components for this generated SBML model were extracted from the Annotation functionality described in 1.1. However, since SBML export results in information loss relative to the PySB model code one really needs both versions of the model. DOIs would facilitate this but assignment of DOIs will need to be accomplished in collaboration with organizations who manage them (such as journals – we were refused a request to become a DOI-issuing organization). However, since PySB can be used to generate diverse models on-the-fly according to templates, a more subtle naming scheme may be required, one in which the DOI can refer to the "abstract" model generation code and not subsidiary "concrete," generated models (which do not exist until the code is run).

1.4 It would be helpful to build models declaratively, not only from tables but also from public databases like KEGG. *Obviously this requires coming to consensus on the content of those databases.*

This is an interesting suggestion. We are still struggling to build predictive node-edge graphs from repositories such as KEGG (which lack mechanistic detail and are quite out of date – see "*Creating and analyzing pathway and protein interaction compendia for modelling signal transduction networks"* BMC Syst Biol. 2012 May PMID: 22548703) so the concept of building a dynamic model directly from databases will require a lot of additional work. The alternative, being currently being pursued by the developers of Kappa and Rulebase.org, is to create curated databases of reusable rules; in its support for rule-based modeling, PySB will be compatible with this approach. This remains an active area of research.

1.5 It would be helpful to have a difference feature which works at the level of the model *in addition to the text difference program which work at the level of the code. The result of this could be a visual map which highlights the commonality and differences*

between models. Visual tools would be especially helpful for composing models. Perhaps this tool could be modeled after Simulink or Labview.

This is an interesting suggestion that will require further work. We now reproduce in the manuscript and supplementary materials various sorts of graphical visualizations possible with PySB models, including reaction network graphs (Fig. 5C), species graphs (Fig. S2), and Kappa-generated contact maps (Fig. 5D). A visual difference feature at the level of reaction networks could potentially be valuable for smaller models but could easily become uninterpretable for even moderately complex models. Difference algorithms for alternative visualizations are possible (e.g. Kappa contact maps) but would require additional work to implement effectively. It is also important to recall that since models are programs in PySB, source code comparison tools and content versioning systems (which are sophisticated text difference tools) *do* serve to highlight key differences in models—much more so than in a format such as SBML, which is only marginally human-readable. We feel that good modeling practice (and good software engineering in general) endeavors to use multi-level abstractions in such a way as to produce code that can be transparently and incrementally modified.

1.6 The community needs to increase its emphasis on repeatability and testing. *PySB could encourage this by providing sample tests for the 15 models discussed in this manuscript and including a testing framework in the PySB release.*

This is an excellent suggestion addressed in the cover letter. We have written a test suite for the extrinsic apoptosis models that demonstrates several different approaches to testing PySB models. We have also added text to the Results briefly describing our use of testing (in the section on Integration: "In addition, models written using PySB can exploit Python tools…" and the section on the EARM 2.0 repository: "To ensure that the re-instantiated models reproduced the behavior of the originally published versions…").

Rather than create a novel testing framework we leveraged the Python modules `unittest`, `doctest`, and `nose`. We have documented these tests and highlighted them in a designated subsection of the online documentation (http://earm.readthedocs.org/en/latest/tests/index.html). For the published models ("Shen models" and "Albeck models") the tests ensure that the PySB versions correctly reproduce the equations or dynamics of the original ODE-based implementations. For the new models described in this manuscript (the "Lopez" models), the tests indicate whether the models reproduce the nominal behavior we describe. In addition, to address the concerns of Reviewer 3 regarding the correctness of the macro implementations, we have added tests to the PySB code that ensures that the macros produce the correct reactions as described in the documentation. In general, we see applications of software testing as an important path forward for the modeling community and anticipate substantial future work in this area.

Minor comments

1.7 Clarify what is meant by "second generation." *What do you consider first generation? What else is part of generation #2?*

We consider *little b*, BNG and Kappa to be relevant "first-generation" applications along with MATLAB tool boxes such as Potter's Wheel. These applications were all developed simultaneously and independently and therefore re-implement many core features (e.g.

Potter's Wheel uses its own non-BNG implementation of "rules.") We have not surveyed all applications in this space but V-Cell is an example of an application that is clearly in its second generation. We have dropped "second-generation" from the title (because it seems likely to cause confusion) but we have added text in the discussion (section "PySB as a second-generation approach").

1.8 Introduction: *The introduction is very well written, but brings up numerous themes (1) fit-to-purpose vs variant, 2) modifying models, 3) comparing models, 4) collaborative, iterative construction, 5) errors and testing, 6) model sharing, 7) abstraction, composition). It could be helpful to provide a more extensive summary at the end of the first paragraph on pp 3 to orient the reader**.***

We have rewritten the introductory paragraphs in an attempt to make these concepts clearer from context. However, a detailed discussion of all of them would likely require a dedicated review article.

1.9 Introduction: *It would be helpful to briefly give broader introduction and motivation to the Bcl-2 biology. In particular, describe why mitochondrial membrane permeabilization is important.*

We agree with the reviewer that this is important and wonder if perhaps our didactic box on this topic was missed (stuck as it was at the end of the manuscript). We have reviewed and revised the box, and trust that in the eventual manuscript it will be much more obvious.

1.10 Clarify that the Howells model is one of the 7 Shen models. *Clarify that the Lopez models are the same as the Green/Andrews models.*

We apologize if our nomenclature was confusing. We have grouped models by the authors of the papers describing the mathematical models, e.g., "Howells" (first author of Howells et al.) or Shen (the last author of the Chen et al. FEBS, Chen et al. BiophysJ, and Cui et al.). The Howells et al. model is grouped with the Shen models because it is largely derivative—it reuses most of the model elements and introduces a few new reactions. This is depicted in Figure 4A and explained in the legend: "the "Shen/Howells Group" incorporates models from three papers from the research group of Shen and a derivative model from Howells et al."

The "Lopez" models are mathematical interpretations of word models introduced by the experimental groups of Green and Andrews. Since there are many possible ways to interpret these word models, we have chosen to identify the specific interpretation of these word models introduced in this paper by the author of the mathematical model ("Lopez").

We have added text to the results making our use of nomenclature more explicit (in Results: "["Lopez Embedded"] implements a mathematical interpretation of recent experimental findings from Andrews and Green.")

Note also that the numbering scheme in Table 2 is unambiguous and does not rely on authors' names.

1.11 pp14: Clarify how you arrive at 30 = 15 x 2 models. Which module is the extrinsic apoptosis model?
Again, we apologize for the lack of clarity. The extrinsic apoptosis reaction model (EARM) repository contains 15 alternative models of how mitochondrial outer membrane permeabilization (MOMP) occurs. The scope of these models is limited to the activity of the Bcl-2 family of proteins.

However, MOMP is just one process, or "sub-module," involved in the pathway of extrinsic apoptosis. To demonstrate how PySB can be used to explore variation in a sub-module in the context of a larger pathway, we incorporated the 15 MOMP sub-modules in the overall extrinsic apoptosis pathway by adding the upstream (receptor to Bid) and downstream (pore formation to PARP cleavage) reactions to each of them. These reactions were taken from our previously published EARM 1.0 model (Albeck et al. (2008) *Mol Cell*). Thus for every hypothesis for MOMP regulation, there is a model of that MOMP sub-module only, and a model incorporating that hypothesis in the full extrinsic apoptosis pathway: 15 MOMP hypotheses x 2 versions = 30 models.

We have added the following additional text to the results to make this more explicit: "we can write programs in which any of 15 models of MOMP are called up along with common `rec_to_bid` and `pore_to_parp` modules to create an additional 15 fully functioning hybrid models of extrinsic apoptosis." We also included the following text: "We collectively denote the resulting set of 30 variant models (15 models only of MOMP plus 15 models of extrinsic apoptosis incorporating the MOMP modules) as Extrinsic Apoptosis Reaction Model version 2.0."

We have also added a detailed explanation of the models on the EARM documentation website: see http://earm.readthedocs.org/en/latest/intro.html and http://earm.readthedocs.org/en/latest/models.html. If desired we can incorporate this documentation in the PDF of the supplement as well.

1.12 Tools (pp18): *Can you provide more information about what Python numerical tools you used for model fitting or point to the methods section?*

The model fitting procedure is described in the "Simulation and parameter estimation" section of the Methods. We used VODE for integration and the simulated annealing algorithm implemented in scipy.optimize.anneal for parameter estimation.

We have included a sentence in the Results to point to the Methods: "we fitted it to data using the simulated annealing algorithm in SciPy (Materials and Methods)."

1.13. Table 2 Revisions: *Clarify that Lopez=Green/Andrews. Clarify that the notation in the ID column refers to the (a) and (b) in the last two row headers.*

We have added footnotes to Table 2 to make this explicit. See also response to 1.10.

1.14. Figure Revisions:

a. Figure 1B: The result corresponding to "PySB declarations" is just a copy of the code; It could be more helpful to display the result of printing the model to stdout

We prefer to keep this as it is currently shown since the figure shows the sequential transformation from one representation (starting with the PySB code) to another (BNGL, ODEs, simulation, etc.).

b. Figure 1B: It could be helpful to add an arrow head from "Integrate (VODE)" to "plot(...)"

Done.

c. Figure 3B: Last box the method should be "rec_to_bind" rather than "rec_to_disc"

This should be "rec_to_bid". Done.

d. Figure 4A: Is there significance to the horizontal alignment of the modules? This could be helpful to describe for the reader.

There is no specific significance to the horizontal alignment of the modules—it is simply a layout that allows us to show the extension relationships (arrows) and family relationships (boxes) neatly in a reasonable amount of space.

e. Figure 4A: Clarify that Lopez group is the same as Green/Andrews (there's a mismatch between how its written in the text vs the figure)

See response 1.10 above.

Figure 4A: Missing " at end of `Lopez "Embedded`

Fixed.

Figure 4B: Missing.

Fixed.

Figure 6: Clarify the key in the figure to indicate light->dark means successive annealing generations

We have simplified our model fitting procedure to use a single objective function and to run in a single round, eliminating the need for the additional curves.

Figure 6: It looks like you need to expand the y-range to show the tick marks.

Fixed.

Reviewer #2
2.1 The claims that it improves upon older methods like BNG and Kappa are reasonable.

Despite this positive comment, we have revised the text to make clear that we "extend" rather "improve upon" BNG and Kappa. PySB leverages BNG and Kappa as enabling technologies and is not meant to replace them. As these other applications are themselves

under active development the benefits of these improvements will accrue to users of PySB as well (see cover letter).

2.2 The integrations mentioned such as NumPy are SciPy, are integral parts of Python not only of PySB.

Of course this is true – we apologize if this was not clear – we have revised the text accordingly (see section "Integration with the Python ecosystem…" in the Results).

2.3 The claim for incremental and collaborative model development is an initial claim and no work has been done to demonstrate these claims as yet.

It is certainly true that PySB has not yet been widely used for incremental model development but the primary aim of our having developed PySB versions of previously developed models was to show how PySB could facilitate an incremental development process. The "mix and match" models we analyze are composed of elements from our own models and models from other groups and insofar as they are new combinations, they are indeed incremental. This retroactive reconstruction attempts to recreate what a community apoptosis modeling effort might look like were PySB available several years ago. Moreover, the "embedded together" version of EARM (model M1a) is clearly incremental insofar as it combines our earlier EARM 1.0 model and new biological insights from Andrews, Green and others.

2.4 This comment is meant to address the concern that the authors could develop new approaches for the next generation of biomodels sharing.

Since PySB required three years of development, it is highly unlikely that we will undertake a new effort of this type. It is certainly true however, that success of PySB will require broad acceptance by the community. The best recent example of this for network biology is CytoScape, and we are in active discussion with Trey Ideker about how he created such a lively open-source community around his program.

2.5 One suggestion is to enable PySB to utilize serialized instance reuse, module reuse, and class reuse and allow for dynamical simulation of combined models and modification of such models.

Perhaps we misunderstand this comment--we believe this functionality already exists and is demonstrated in Figure 3 of the manuscript.

Detailed Comments
***2.6 There are no instructions available on the* web:**

This has been fixed, as described in the cover letter.

2.7 Example/*.py does not run readily.

Good suggestion - we have fixed this. All examples are now tested and documented.

2.8 It would be easier if dependencies could come prepackaged.

As described in the cover letter, the principal issue with installation is that PySB requires a variety of other programs to run (e.g. BNG) as well as a variety of standard Python libraries. Our approach to simplifying installation of PySB is to provide a one-click download of a virtual machine image (Open Virtualization Archive - OVA file). The free and open-source utility *Virtual Box,* itself available for all platforms, can be used to run the PySB virtual machine. The virtual machine contains not only PySB and the EARM models, but also the necessary dependencies (BNG, Numpy, Scipy, Sympy, MatPlotLib, etc.) and useful tools (Git, iPython, GraphViz, Kappa, OCaml). Git itself is preconfigured so that updates to PySB or the models can be obtained using a single command (`git pull`). This differs slightly from a conventional installer but is now standard in cloud computing for scientific applications and is more convenient for both novice and experienced users than an installer involving pre-packaged dependencies. We have modified the web site to describe the installation of the virtual machine as well as installation of specific dependencies for experienced Python users who would prefer a native installation (see http://pysb.readthedocs.org/en/latest/installation.html).

2.9 Straight forward black box import method is simple to use and understand for skilled individuals, who know python*. This may be of problem if knowledge of python is basic or limited as implementation requires knowledge of operator overloading and order of operation in python to understand and extend this code. This may lead to unpredictable events in the future, if arithmetic rules diverge from biological rules.*

We discuss this issue in the cover letter. Briefly, we have modified the paper to include a new didactic box and new supplementary materials that make clear that operator overloading is used to ensure that PySB closely resembles BNGL and takes place strictly within the context of rule definitions. Otherwise, Python operators - including arithmetic symbols - work in a conventional way. Moreover, users can avoid overloading entirely and use only conventional Python commands.

2.10 Python syntax is a non-trivial for many computational biologists *which may limit usage Some of the syntax is not common usage (Ex: None >> A()) is unclear of the function in this syntax.*

While this might have been true in the past, we find that the majority of our systems biology students are now learning Python – it is rapidly replacing PERL for all but conventional bioinformatics applications. MIT now teaches Python instead of LISP for introductory computer science students. Thus, we think it is reasonable to assume that many biologists will have access to adept Python programmers. We do accept that we have some non-standard syntax and have therefore written an additional supplemental section as well as a didactic box covering overloaded operators etc. (see cover letter).

2.11 Process Orientation is excessively used, which may detract from understanding the code and lead to confusion.

- *Ex: Model() creating an implicit "model" global reference is not explicitly defined here; which may lead users to wonder why they need to use pint.odesolve(model, t) inside the "synth_deg.py" example.*
- *Ex: "model = Model()" instead of "Model()"*

- *Clarity may be improved by using a combination of Object orientation and polymorphism with process orientation, which is done inside core.py but using explicit references instead of global stores through "class SelfExporter".*

The reviewer raises an important point regarding the tradeoffs made in the PySB implementation between conventional programming syntax vs. a domain-specific modeling syntax. To borrow a phrase, PySB was designed to make "easy things easy and hard things possible." The "process-oriented" syntax that the reviewer describes, facilitated by the SelfExporter class, allows models to be built up declaratively without retaining references to each created object (this is described briefly in "Component self-export" section of the Materials and Methods, and extensively in the "PySB Syntax" section of the supplementary materials). We have found that this greatly streamlines the most common model-building scenario, in which a single model is constructed declaratively and analyzed.

However, the "object oriented" approach that the reviewer suggests, in which objects are created and explicitly added to one or more models, remains entirely permissible. In complex scenarios involving on-the-fly, combinatorial creation of models and model objects, this is indeed the more transparent approach as it does not rely on side effects of component creation.

We believe that this issue is largely a matter of style and best practices and plan to augment the documentation with a style guide over time. For this publication, we have added a more detailed discussion of the tradeoffs in the supplement.

2.12 Commenting:
● ***Example code needs to be documented.***

Examples (files in pysb/examples) have now been thoroughly documented and tested to ensure that they run correctly.

● ***Documentation inside main class*** *like core.py also could be improved; however parameter naming is done well to give descriptive power.*

We have added considerable new documentation (in the form of docstrings) to the functions and classes in core.py.

2.13. Use of non-standard expression overloading *is confusing and only syntactical sugar for the creators; they are likely to be unclear and difficult to understand for programmers this needs to be improved by reference to a process diagram and may not be helpful in building a community of users.*

As discussed above, we have made extensive additions to the supplementary materials, and added a didactic box, to address this comment (see cover letter). We have also added text to the Results alluding to our use of operator overloading ("To simplify the process of writing rules and to maximize the syntactic match with BNGL…"). This is also addressed briefly in the "PySB syntax" section of the Materials and Methods.

It is not necessary to use overloading in PySB but this results in substantially simpler code and it is our belief that this will facilitate use of the code by biologists and users of

existing rule-based languages. One feature of the shorter "overloaded" expressions is that they can easily be read and understood by non-experts with minimal syntactic overhead. Our goal in this regard was to implement elements of a domain-specific language within Python (the Python package SymPy, which embeds symbolic mathematics capabilities within Python, is an example of a similar approach). However, we accept that experienced programmers may want to use more conventional expressions and they are fully supported as well.

2.14 ***Organization****: There are too many classes inside the same file (Ex: Core.py). The authors could utilize sub modules and divide large files into smaller files with classes for better source control and extensibility.*

We consider this to be a stylistic issue regarding the balance between the number of files vs. the number of classes per file. While we may refactor this module at a later time (e.g. to separate the model component classes from the pattern and expression classes--see the UML class diagram in Supplementary Figure S3) we do not believe it reasonable or necessary to address this at the moment.

2.15 ***Documentation is missing a process diagram*** *of class hierarchy and information flow, which can improve understanding and reduce the learning curve for modifying the library.*

We have created a class diagram of the core modeling components and added it to the supplement (Figure S3). We plan to incorporate the relevant supplementary material on the website itself if and when our manuscript is accepted for publication.

Reviewer #3

3.1. The main novelty of this approach *lies in the integration of several previously developed tools within a Python umbrella.*

We apologize for what seems to be a misunderstanding in this regard. In our opinion the novelty of our approach lies in programmatic modeling – that is, where models are built up using computer programs – and its attendant ability to make use of macros, programmatic abstraction, polymorphism etc. The integration with BNG and other Python tools is important from the point of view of utility, but we make no claim that it is novel or original.

However, it is important to point out that the overall philosophy of PySB is that it should do one thing very well and then seamlessly integrate with community-based efforts in the area of visualization, parameter estimation, rules-based modeling etc. (see cover letter). We think this is a different (and potentially better) approach than building all-in-one applications that represent a compromise in each of their elements. Of course, our approach requires dedication to a completely open-source approach.

3.2. However, it is not clear how this is significantly different from for instance the NFSim *or Virtual Cell softwares which also implements BioNetGen, but with a sophisticated graphical user interface.*

PySB is interoperable with, or makes use of these other applications, but is unique in its approach to program-driven modeling (see 3.1). NFSim is a simulator, which itself could be used to simulate models created using PySB (we are currently developing this functionality). V-Cell is an all-in-one tool outstanding with respect to visualization and spatial modeling, but does not address issues such as composability or modularity. PySB would interoperate well with V-Cell and we intend to pursue such integration at the earliest possible opportunity.

3.3. This brings up another point in that PySB seems complex to install and use.

We have made extensive revisions in response to this concern, as described above. Most importantly, we discuss how the virtual machine image is installed (a one-click, platform-agnostic approach) that should make PySB simple to install. Ease-of-use also relies on better documentation, and we have therefore made extensive modifications to the website (http://pysb.readthedocs.org/en/latest/installation.html) and prepared additional supplementary materials describing PySB syntax.

3.4. One point the authors bring up several times is the ability to easily integrate models with PySB*….. They argue that when doing this by the "cut and paste" method, one is left with the problem of unique variable names, and then matching them together. Yet, it is unclear how PySB might automatically solve this problem.*

We apologize for any confusion in this area. To clarify, "cut-and-paste" refers to an approach to re-use that relies on duplication followed by modification. For ODE-based modeling this has hitherto been the only way to incrementally extend models. This approach is problematic primarily because any explicit connection with the prior model is lost: determining what (if anything) has changed between the models requires careful analysis of the ODEs of both models. This model comparison process is *made more challenging* when the duplicated model renames species and parameters; however, the fundamental problem is the *loss of history and provenance,* rather than the renaming of components *per se*.

PySB addresses this problem by enabling approaches to programmatic re-use that do not involve explicit duplication of code. We illustrate three such approaches in Figure 3 of the manuscript ("instance reuse," "module reuse," and "class reuse."). In each of these approaches, the model or module to be incorporated or extended is *programmatically* imported or invoked and modifications are made in additional code. This procedure is shown explicitly in Figure 4: Figures 4C and 4D show how the base model "Chen FEBS Direct" is subsequently modified and refined in the model "Cui Direct." Using this approach, the basic question "what has changed between the two models?" is answered by reading the modification code (with associated commentary) in the extending model.

3.5. It would seem careful analysis of two model's variables is needed *regardless of the software package one uses to stitch them together. One could see this being accomplished through very careful annotation perhaps (for example with unique geneIDs tied to model variables), but not simply by making models with programs rather than text files.*

This is an important point and echoes the concerns of reviewer #1 regarding the naming of components. It is not our intention to suggest that PySB, in and of itself, will resolve all

issues associated with integration of biological models. As emphasized in our responses 1.1 and 1.2, we anticipate that model integration will remain a "human-in-the-loop" process for some time. PySB addresses a number of technical issues associated with model integration but it does not automatically address the issues associated with the compatibility of models of disparate biological processes, or fundamentally different approaches to modeling the same process. By and large, dynamic models of biological processes remain fit-to-purpose, with applicability limited to a subset of biological questions. Developing all-purpose models of biological processes remains a hard problem and we believe that there is no "free lunch" in this regard (see Krakauer et al. referenced in the manuscript for a more thorough discussion of these issues).

We believe that PySB facilitates model integration primarily by making models easier to understand. Using programs rather than text files is vital because it enables the use of hierarchical abstractions that describe how models are constructed at a variety of conceptual levels (see Figure 7). It is certainly true that composing models requires understanding of how the elements interact. Our approach aims to substitute tedious error-prone work (i.e. "*very careful annotation")* of ODEs with a more abstract representation that is easier to understand and can be modularized using macros and modules.

3.6 Moreover many models currently exist in SBML format in public databases which makes sharing them quite straightforward.

SBML models can be shared (as described above in 1.3 and in the discussion, PySB does not supplant SBML) but they cannot be composed or programmatically extended. Extension of SBML models requires duplication and modification, a pattern we term "cut-and-paste" reuse (see 3.4). The capability we hope to support with PySB is not just sharing, but meaningful community efforts at cumulative model building, modeled after open-source software projects. This requires that models not only be shareable, but also transparent, reusable, and extensible (and testable). It is in these latter aspects that we feel that PySB offers significant capabilities over SBML.

3.7 They also make the point that by creating "higher order function" *such as enzyme catalyzed reactions with a macro, one ensures they are correct. This is only true if the macro itself is correct; how does one ensure this for likely a rapidly growing base of such macros?*

Code that is written once and used repeatedly can be tested for correctness in a way that code written on an *ad hoc* basis cannot. To address the reviewer's specific concern regarding the correctness of macros, we have added a suite of tests that ensure that the macros included with PySB generate the rules and other components as specified in their documentation (see cover letter and 1.6). As a matter of practice, we can impose a requirement that any new general-purpose macros added to the PySB codebase be similarly tested.

We have added text to the Results section on Macros to emphasize this point ("Since macros are tested programmatically they also ensure correct instantiation of forward, reverse and catalytic reactions…").

3.8 Moreover, existing software such as Copasi does similar things, *and again with a graphical interface and host of associated functions like parameter estimation.*

We beg to disagree – Copasi does not use a programming-driven approach or use rules in a sophisticated way (see cover letter and point 3.1 above). Because of close integration with Python libraries such as Numpy and Scipy, PySB models can also take advantage of sophisticated parameter estimation and other numerical algorithms and because these derive from broad community-based efforts they are likely to remain up-to-date. An advantage of modular, open-source tools such as PySB or BNG is that it is possible to decouple the series of steps involved in creating models from calibrating and analyzing them, substituting different algorithms as desired.

PySB is focused on model creation, composition and management (and documentation). Models thus created can be exported via SBML and analyzed in any SBML-compatible tool such as Copasi, which makes available a wide range of numerical tools through a visual interface. However, for users who wish to implement novel numerical algorithms or combine algorithms in new ways, the PySB framework has been designed from the ground up to make this type of extension straightforward.