

Xml2jupyter: Mapping parameters between XML and Jupyter widgets

Randy Heiland¹, Daniel Mishler¹, Tyler Zhang¹, Eric Bower¹, and Paul Macklin¹

DOI: [00.00000/joss.00000](https://doi.org/00.00000/joss.00000)

¹ Intelligent Systems Engineering, Indiana University

Software

- [Review](#) ↗
- [Repository](#) ↗
- [Archive](#) ↗

Submitted: 31 January 2019

Published: 01 March 2019

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC-BY](#)).

Summary

Jupyter Notebooks (Kluyver et al. 2016) provide executable documents (in a variety of programming languages) that can be run in a web browser. When a notebook contains graphical widgets, it becomes an easy-to-use graphical user interface (GUI). Many scientific simulation packages use text-based configuration files (hopefully in some standard format) to provide parameter values. Xml2jupyter is a Python package that bridges this gap. It provides a mapping between configuration files, formatted in the Extensible Markup Language (XML), and Jupyter widgets. Widgets are automatically generated from the XML file and these can, optionally, be incorporated into a larger GUI for a simulation package. Users modify parameter values via the widgets and the values are written to the XML configuration file. Xml2jupyter has been tested using the PhysiCell (Ghaffarizadeh et al. 2018) simulation software and will be used by students for classroom and research projects.

A PhysiCell configuration file defines model-specific user parameters in XML. Each parameter element consists of its name with attributes, defining its data *type* and *units* (optional), and the parameter's default value. The attributes will determine the appearance and behavior of the Jupyter widget. For numeric widgets (the most common type for PhysiCell), xml2jupyter will calculate a delta step size as a function of the default value and this step size will be used by the widget.

To illustrate, we show the following contrived XML example, containing each of the four allowed data types:

```
▼<PhysiCell_settings>
  ▼<user_parameters>
    <radius type="double" units="micron">250.0</radius>
    <threads type="int">8</threads>
    <color type="string">red</color>
    <debug type="bool">True</debug>
  </user_parameters>
</PhysiCell_settings>
```

When we map this into Jupyter widgets, we have the following that shows the rendered widgets (left). The middle snapshot shows the result after the user does a single delta step (up) on the `radius` (note the step size of 10) and the right snapshot after the user single steps the `threads` (note the step size of 1).

radius	250	micron	radius	260	micron	radius	260	micron
threads	8		threads	8		threads	9	
color	red		color	red		color	red	
<input checked="" type="checkbox"/> debug			<input checked="" type="checkbox"/> debug			<input checked="" type="checkbox"/> debug		

In another example, we use an XML configuration file from the *biorobots* sample project included with PhysiCell:

```
<user_parameters>
  <random_seed type="int" units="dimensionless">0</random_seed>

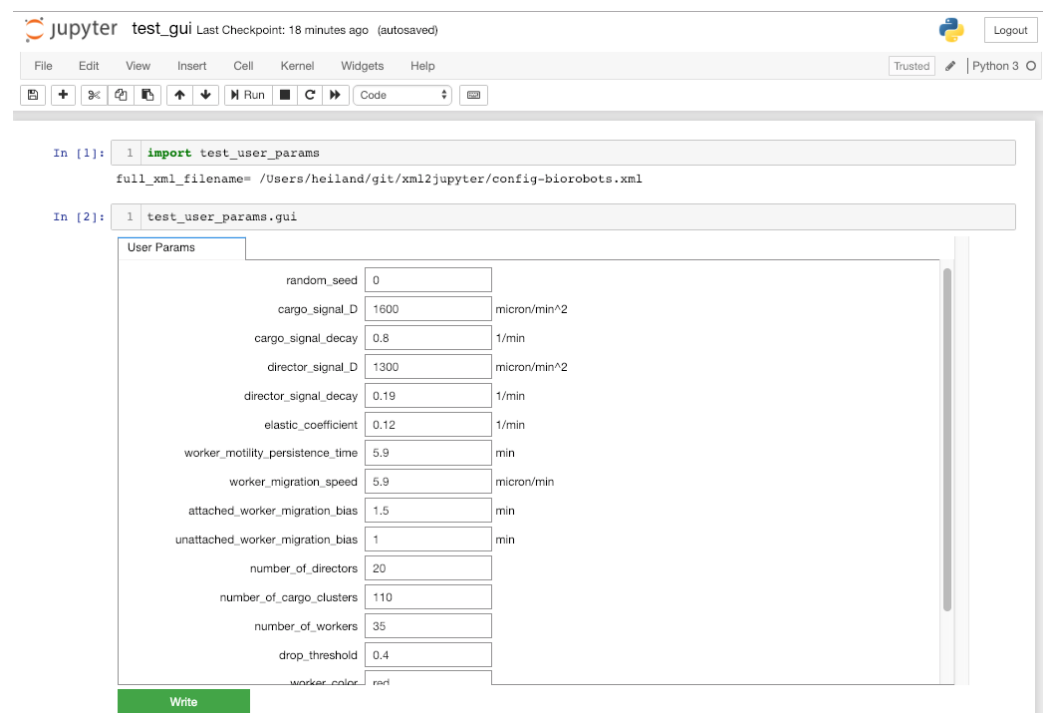
  <!-- for microenvironment setup -->
  <cargo_signal_D type="double" units="micron/min^2">1e3</cargo_signal_D>
  <cargo_signal_decay type="double" units="1/min">.4</cargo_signal_decay>
  <director_signal_D type="double" units="micron/min^2">1e3</director_signal_D>
  <director_signal_decay type="double" units="1/min">.1</director_signal_decay>

  <!-- for cell definitions -->
  <elastic_coefficient type="double" units="1/min">0.05</elastic_coefficient>
  ...
  <director_color type="string" units="none">limegreen</director_color>
</user_parameters>
```

A workflow to demonstrate this project is the following:

- `python xml2jupyter.py <config_file>.xml`

A user would execute a Python script, `xml2jupyter.py`, that parses this XML and generates a Python module containing the Jupyter widgets, together with methods to populate their values from the XML and write their values back to the XML. This can be displayed as a very simple GUI in a Jupyter notebook to provide the mapping to/from the XML parameter values, as shown below:



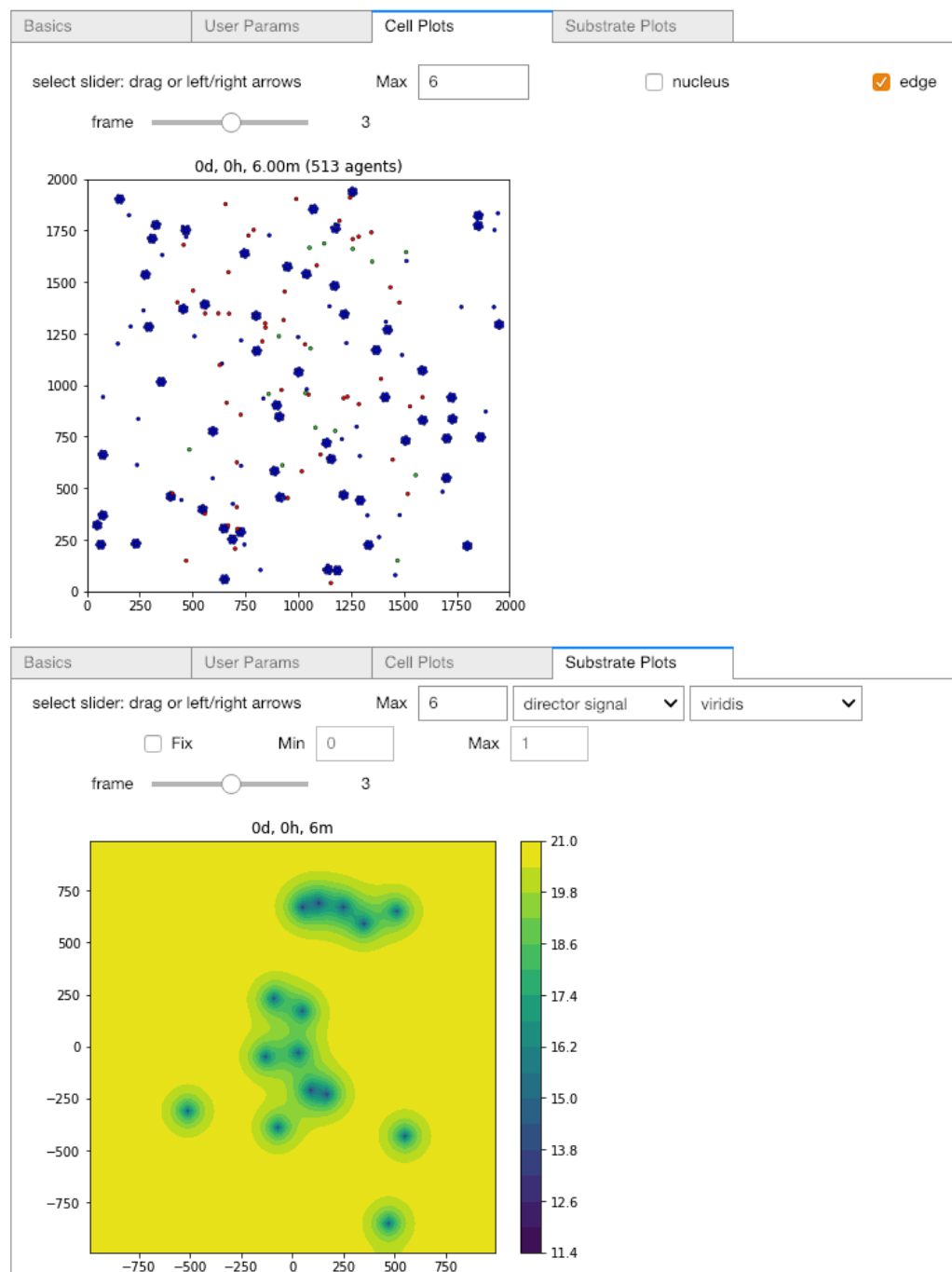
The screenshot shows a Jupyter Notebook titled 'test_gui'. The first cell contains the code: `import test_user_params` and `full_xml_filename= /Users/heiland/git/xml2jupyter/config-biorobots.xml`. The second cell contains `test_user_params.gui`. Below the code, a GUI window titled 'User Params' is displayed. It contains a table with the following parameters and units:

random_seed	0	
cargo_signal_D	1600	micron/min^2
cargo_signal_decay	0.8	1/min
director_signal_D	1300	micron/min^2
director_signal_decay	0.19	1/min
elastic_coefficient	0.12	1/min
worker_motility_persistence_time	5.9	min
worker_migration_speed	5.9	micron/min
attached_worker_migration_bias	1.5	min
unattached_worker_migration_bias	1	min
number_of_directors	20	
number_of_cargo_clusters	110	
number_of_workers	35	
drop_threshold	0.4	
worker_color	red	

A green 'Write' button is located at the bottom of the GUI window.

PhysiCell Jupyter GUI

Our ultimate goal is to generate a functional GUI for PhysiCell users. Xml2jupyter provides one important piece of this - dynamically generating widgets for custom user parameters for a model. With the addition of static components (tabs) of the GUI, a user can also visualize output results from simulations. This additional functionality requires modules not available in the Python standard library, e.g., Matplotlib (Hunter 2007) to display plots, and SciPy to parse PhysiCell output data. We provide instructions for installing these additional dependencies on [github link](#).

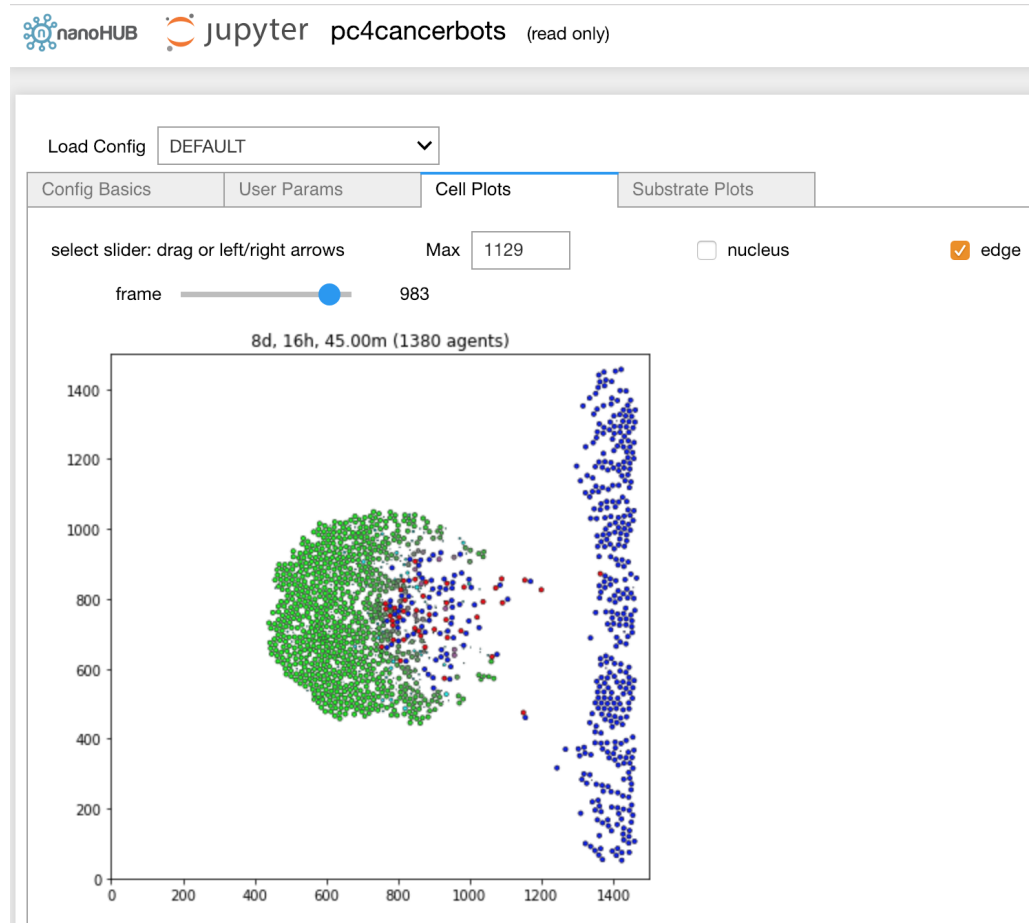


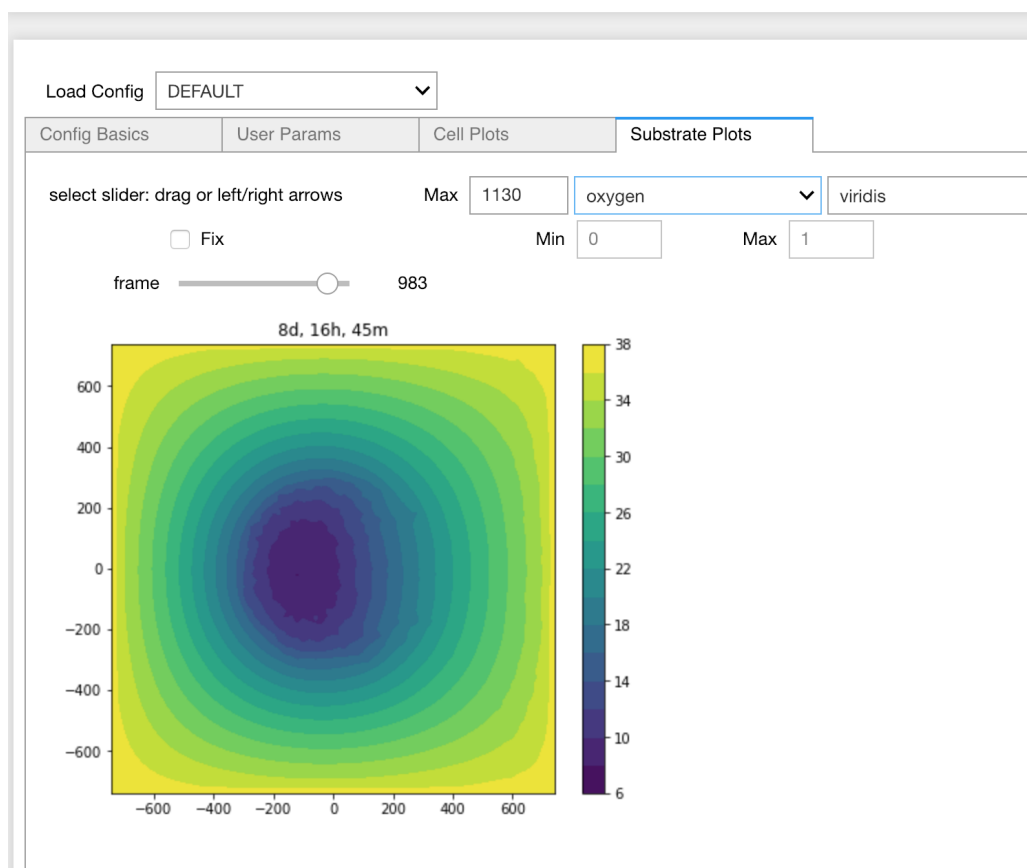
Additional tabs used by the PhysiCell Jupyter GUI.

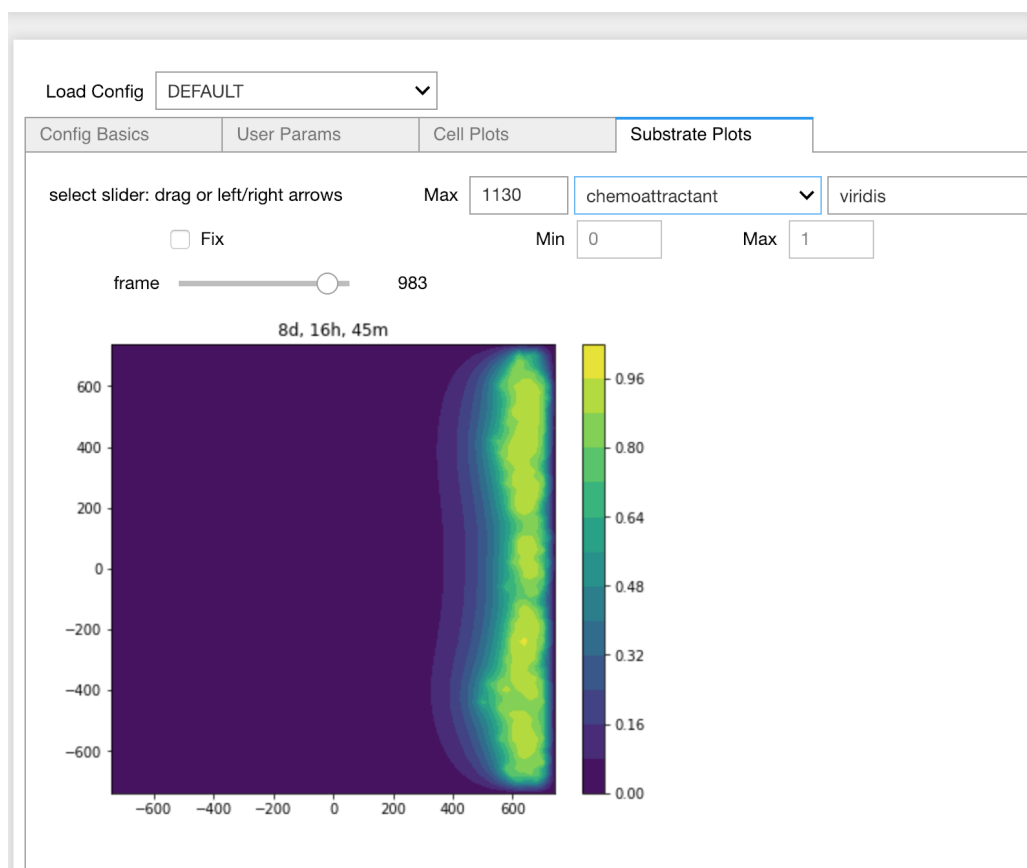
Extensions and Discussion

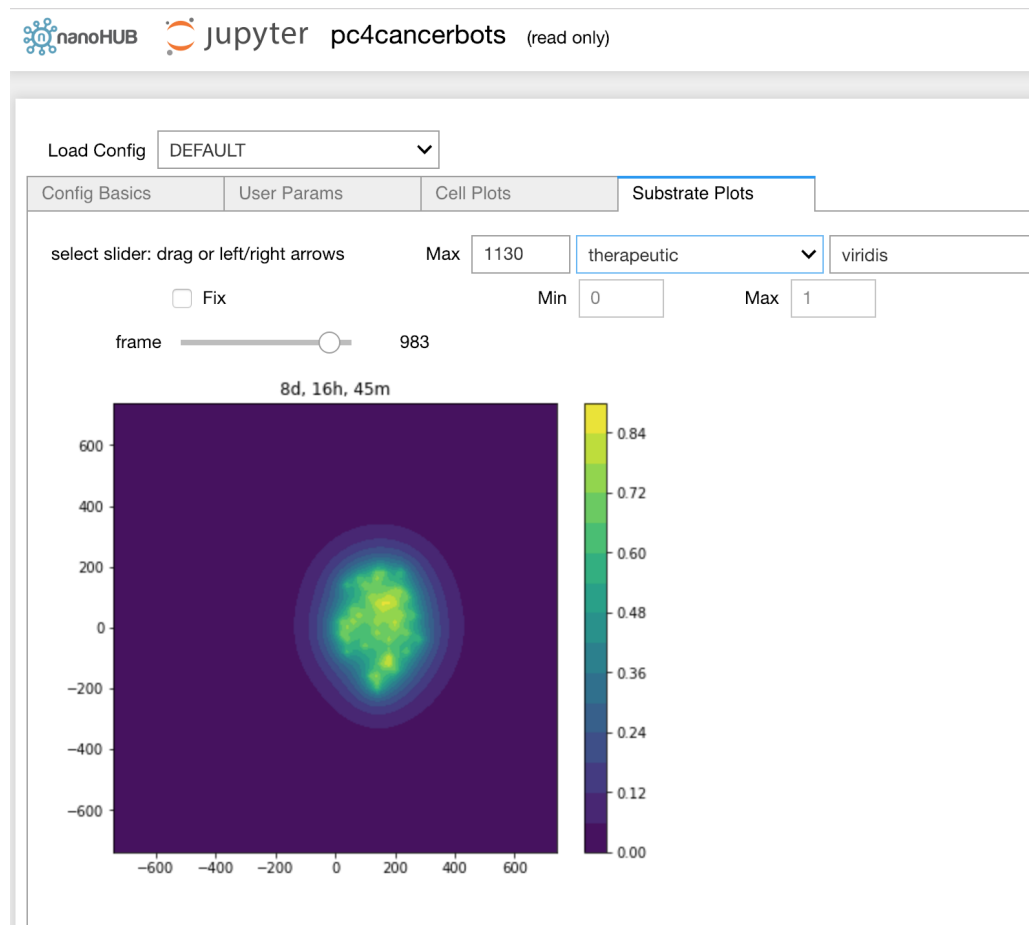
We hope others will be inspired to extend the idea of this project to other text-based configuration files. XML is only one of several data-interchange formats. It just happens to be the one of interest to us for PhysiCell.

Xml2jupyter has helped us port PhysiCell-related Jupyter tools to nanoHUB (Krishna et al. 2013), a scientific cloud for running simulations and visualizing results.









Acknowledgements

We thank the National Science Foundation for providing funding via NSF EEC-1720625. NCI grant #... Paul? We acknowledge support from our collaborators at Purdue University, especially Martin Hunt, who helped port our Jupyter tools to nanoHUB.

References

- Ghaffarizadeh, Ahmadreza, Randy Heiland, Samuel H. Friedman, Shannon M. Mumenthaler, and Paul Macklin. 2018. "PhysiCell: An Open Source Physics-Based Cell Simulator for 3-d Multicellular Systems." *PLOS Computational Biology* 14 (2). Public Library of Science: 1–31. doi:[10.1371/journal.pcbi.1005991](https://doi.org/10.1371/journal.pcbi.1005991).
- Hunter, J. D. 2007. "Matplotlib: A 2d Graphics Environment." *Computing in Science & Engineering* 9 (3). IEEE COMPUTER SOC: 90–95. doi:[10.1109/MCSE.2007.55](https://doi.org/10.1109/MCSE.2007.55).
- Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, et al. 2016. "Jupyter Notebooks – a Publishing Format for Reproducible Computational Workflows." Edited by F. Loizides and B. Schmidt. IOS Press.
- Krishna, Madhavan, Zentner Lynn, Farnsworth Victoria, Shivarajapura Swaroop, Zentner Michael, Denny Nathan, and Klimeck Gerhard. 2013. "NanoHUB.org: Cloud-Based

Services for Nanoscale Modeling, Simulation, and Education.” *Nanotechnology Reviews* 2
(2019): 107. doi:[10.1515/ntrev-2012-0043](https://doi.org/10.1515/ntrev-2012-0043).