NeoCortical Repository and Reports: Database and Reports for NCS

Edson O. Almachar[1] Alexander M. Falconi[1] Katie A. Gilgen[1]
Devyani Tanna[1] Nathan M. Jordan[1] Roger V. Hoang[1]
Sergiu M. Dascalu[1] Laurence C. Jayet Bray[2] Frederick C. Harris, Jr.[1]

Brain Computation Lab
http://www.cse.unr.edu/brain/

[1]Computer Science and Engineering
University of Nevada, Reno
Reno, NV, USA

[2]Bioengineering
George Mason University
Fairfax, VA, USA

Abstract

In the field of Computational Neuroscience, computer based brain simulators help Neuroscientists in the formulation and examination of theories about the inner workings of the brain in the microscopic and cellular level. Brain simulators offer an opportunity to refine and build upon the compendium of scientific knowledge in Neuroscience. One of these brain simulators is the NeoCortical Simulator (NCS). In operating a computational powerhouse, there is a need of an interface with which a user would interact with to communicate with the simulator. We propose a browser based web application that a user may browse to in order to use a plethora of services for the simulator. Two of these services include a Repository Service which allows a user to save a brain model onto a database and a Reporting Interface which allows a user to view the data output by the simulator. This paper details the design and implementation of those services, called NeoCortical Repository and Reports (NCR).

Keywords: Simulation, Computational Neuroscience, Database, Reports

1 Introduction

The NeoCortical Simulator from the University of Nevada, Reno, is a joint research venture between the colleges of Science, Engineering, and Medicine within the University of Nevada, Reno[4, 5]. It is a tool for researchers to perform CPU/GPU based simulations with biological brain models. The NCS uses brain model data as arguments to the simulator and then outputs data in a parsable text format. Previously, the brain models had to be coded in, which was an inconvenience for researchers unfamiliar with programming. Additionally, the outputs were not conveniently displayed and required extra time and effort to understand the data.

Past projects have expanded on the idea of a friendlier user experience when interacting with the simulator. One of which is the NCS-NeuroML translator [6] which allowed the conversion of NCS input files from the standard NeuroML input language to the native input language via the usage of a java-gnome user interface library thereby streamlining interaction with the simulator. Another is the 3D Neuron visualizer, or NeoCortical View[3], which allowed a user to visualize the current live network state of the simulation in 3D.

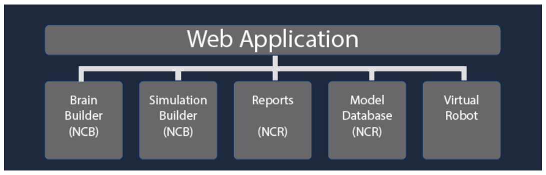


Figure 1: The architecture of the NCS Web Application.

To continue this trend of streamlining the experience, we wish to further improve the interface. We propose the NCS Web Application; an intuitive, browser based application that users may browse to on their web browser and begin interacting with the NCS without need of a command line interface. The NCS Web Application is comprised of five main components that encourage a smooth user experience within the NCS. These components are outlined in Figure 1.

There are multiple development teams in charge

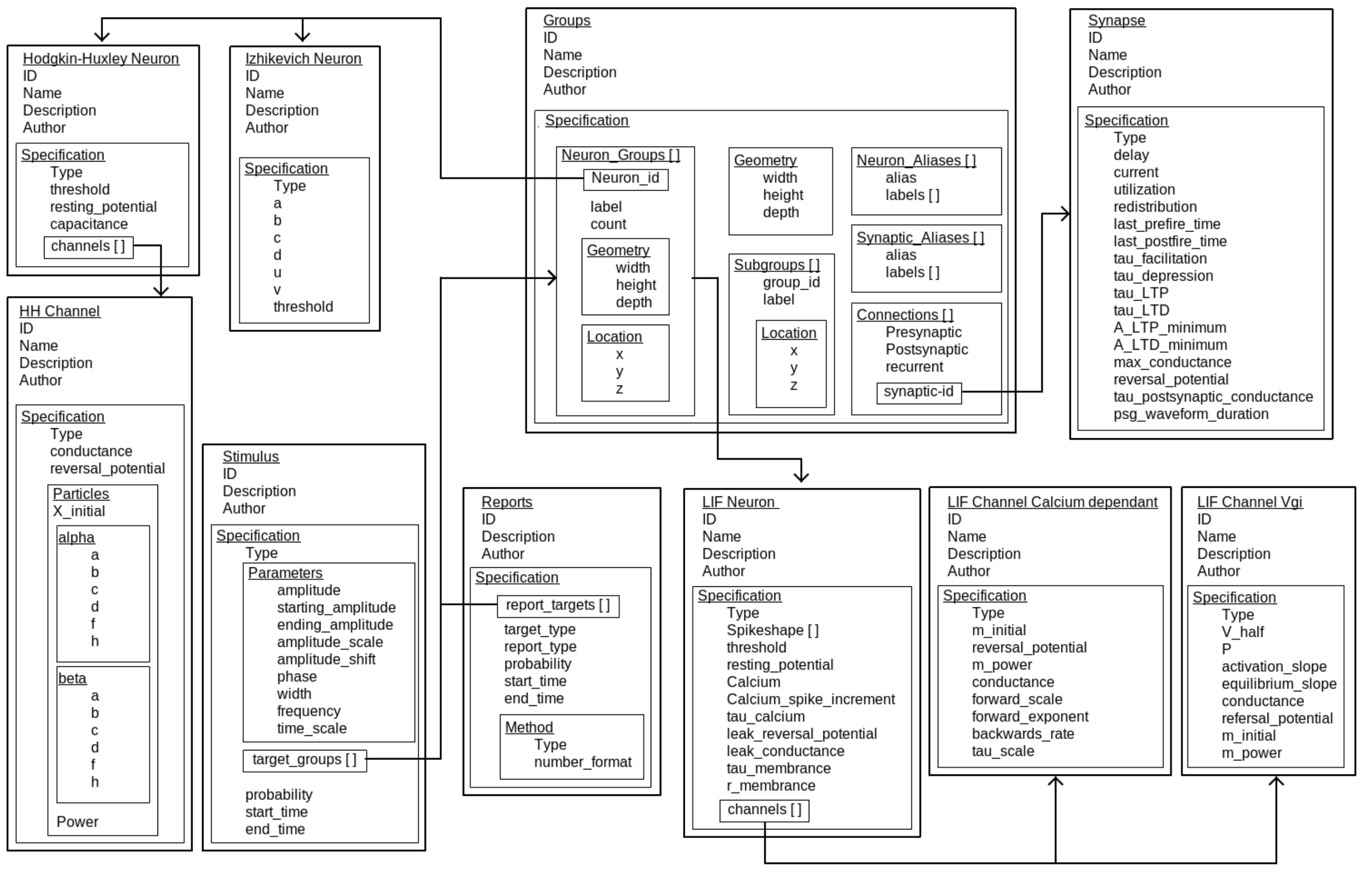


Figure 2: MongoDB documents design

of the construction of this web application. The NeoCortical Builder team is responsible for Brain Builder and Simulation builder [1]. Whereas, the development team, NeoCortical Repository and Reports, is concerned with the development of the Model Database and the Graphical Reports, and as such, is the main topic of discussion for this paper. The two components developed by NCR are described by the following:

Firstly, when a user chooses to interact with the Database, the user must be aware that NCS has the ability to simulate three biological neuron models: Izhikevich, Leaky integrate-and-fire, and Hodgkin-Huxley. NCR is concerned with the implementation of a Model Database that would need to understand these brain models in order to conduct services like storage, search, and updating a model.

Secondly, further down the cycle in an active simulation, there would have to be an output. NCR is concerned with the implementation of a Reporting Interface comprised of line graphs and raster plots, which are the standard graphing mediums in Neuroscience. These reports make interpretation easy and swift so as to promote efficient use of time and better productivity.

As a result, the implementation of NCR within the NCS mainframe will accomplish a multitude of goals that encourage a user friendly experience with the NCS. To help outline these goals, the design overview of the NCR components and their use cases are detailed in Section 2. The User interface, a key asset in the human to computer interaction of the application, is detailed in Section 3. The paper wraps up with a discussion of future work in Section 4.

2 Design Overview

The main functionality of NCR is based on its manifestation as a website. The site is hosted on a web server base constructed using FLASK[9], a python based micro-framework. The model database is designed using MongoDB[7], which is an easily scalable non-relational database. Hosted over the internet, the database can be used to upload or store brain models. The reporting interface is constructed via D3.js[2], a javascript graphing library, and jQueryUI[10], a

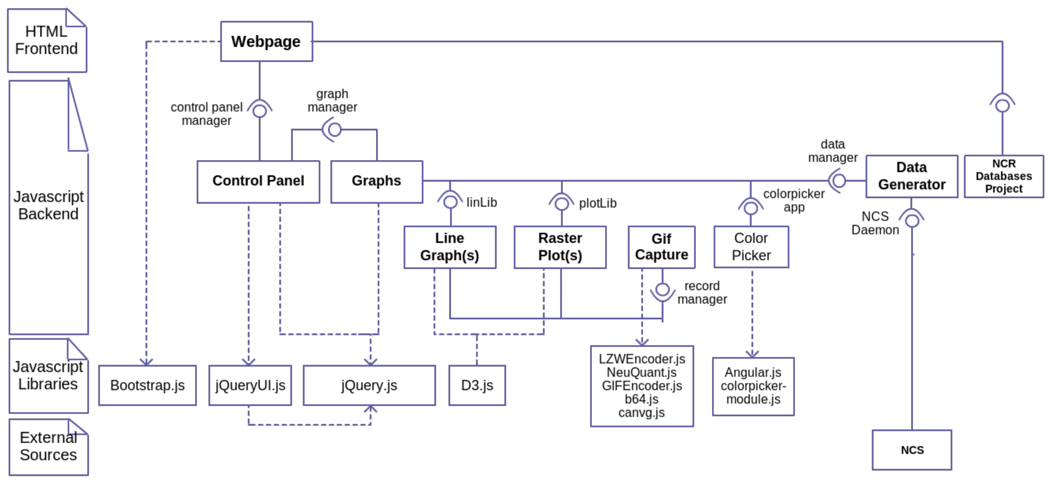
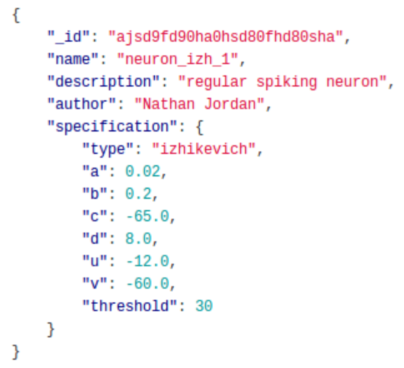


Figure 3: The low level design of the reports interface.



```
{
    "_id": "ajsd9fd90ha0hsd80fhd80sha",
    "name": "neuron_izh_1",
    "description": "regular spiking neuron",
    "author": "Nathan Jordan",
    "specification": {
        "type": "izhikevich",
        "a": 0.02,
        "b": 0.2,
        "c": -65.0,
        "d": 8.0,
        "u": -12.0,
        "v": -60.0,
        "threshold": 30
    }
}
```

Figure 4: JSON document for Izhikevich Neuron

javascript user interface library. Everything the user does to interact with the NCS would be done so through the users' web browser.

2.1 Database Design

The brain model database gives users of the NCS web application the ability to easily collaborate with others and save valuable time. The model database is designed in order to store and query various brain models. MongoDB is an ideal choice for the database since it is free, open source, flexible in terms of schema, and uses JSON-style(Javascript Object Notation) documents as shown in Figure 4.

The MongoDB document design is shown in Figure 2. Big boxes are documents, and boxes inside boxes are sub-documents. Sub-documents are good for faster queries. Each document belongs to one of the 6 collections in the database: Groups, Neurons, Channels, Synapses, Stimuli, and Reports. By default, MongoDB does not enforce schema. In order to have structured schema and validation layer, MongoKit [8] is used. A database schema is created for each document type, and is used as the format for the models within MongoDB.

The simplistic structure of the search panel provides an intuitive way for a user to find a useful model in the database without the need for searching through various text files and downloads. The database stores models from users who have uploaded or created a model using the Model Builder tab of the application, meaning that users from around the world may publish a model for other users to view.

2.2 Reports - High Level Design

The Graphical Reports tab of the NCS web application aims to provide as much of an intuitive interface as possible in an attempt to maximize comprehension of the data and to minimize the complication of technicality. In order to provide such an interface, a fully dynamic environment is generated that would allow a user to manage and manipulate graphs to their liking. Users may manage graphs by dynamically creating or deleting them. Users may apply spatial manipulations by dragging and placing the graphs from one area of the web page to another, allowing the user to reorganize the graphs to their liking.

When considering a large amount of data, accuracy in representation must be considered when abstracting output from the NCS. Scalable Vector Graphics (SVG) are the main abstraction medium for the data and utilized for dynamic representation. D3.js provides the interface between the data and the SVG representing the data. As data is continuously fed to the client from the server, the SVG's must change dynamically over time. To accomplish this, D3 allows for animated SVG's.

Presentation both on the NCS web application and off are important. When considering the applications ability to save graphs onto disk, accurate representation has to be priority. To facilitate this concern, client side graph capture of the browser generated SVG was an optimal design choice. Users may capture the SVG elements within the webpage and have its context downloaded as an SVG file. For an animated Graphics Interchange Format (GIF), the SVG is continually contextualized into a HTML5 Canvas element. That element is then captured and inserted frame by frame into a GIF object and downloaded as a GIF file.

2.3 Reports - Low Level Design

The graphical reporting interface is comprised of multiple components whose functionalities vary widely, but each having an important role in the construction of a reporting window. The components interaction is shown in Figure 3.

Control Panel - The component representing the control panel that would allow a user to manipulate the environment that the graphs will be instantiated into.

Graphs - An singleton that would allow a user to add or remove graphs. This entity is also responsible for maintaining and feeding data to graphs.

Data Generator - An entity that manages the data received from NCS and would continuouly inject that data into the line graphs and raster plots.

Line Graph(s) - A D3 based entity that holds and manages the line graph SVG.

Raster Plot(s) - A D3 based entity that holds and manages the raster plot SVG.

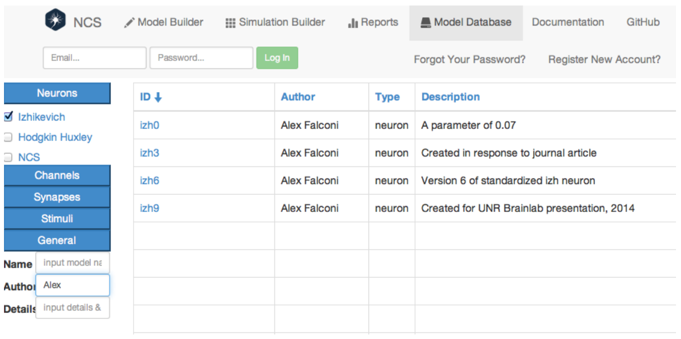
GIF Capture - A service that allows a Line graph or Raster plot to be recorded via GIF or SVG and saved to file.

3 User Interface

3.1 Repository

The model database tab of the NCS Web Application is used to search through the brain model database using a simplistic search panel. The model list on the tab shows the models in the database that match the search criteria specified by the user. Within the search panel, the user can filter the list based on the model types, which are collected into groups in the search panel. Selecting a model type opens a collapsible list of search-able parameter values. The user may enter an exact value, or a range of values delimited by a dash.

The search panel is located on the left side of the model database tab. The initial panel shows groupings for model types that can be expanded to reveal model type selection boxes. Selecting a model type box adds models of the selected type to the filtered results. If the name of the type is selected, the search panel expands to show the parameter search options. Here, the user enters an exact or range of values into the search box and clicks the search icon to update the model list. Only the models of the selected type that have the specified value appear in the list. A user can specify as many included types and parameter values as desired for a search. The general filter group includes filtering the list based on name, author, description, and scope values. An example of the list filtered by selecting the group Neuron, type Izhikevich, and specifying the author name are shown in Figure 5.



ID ↓	Author	Type	Description
izh0	Alex Falconi	neuron	A parameter of 0.07
izh3	Alex Falconi	neuron	Created in response to journal article
izh6	Alex Falconi	neuron	Version 6 of standardized izh neuron
izh9	Alex Falconi	neuron	Created for UNR Brainlab presentation, 2014

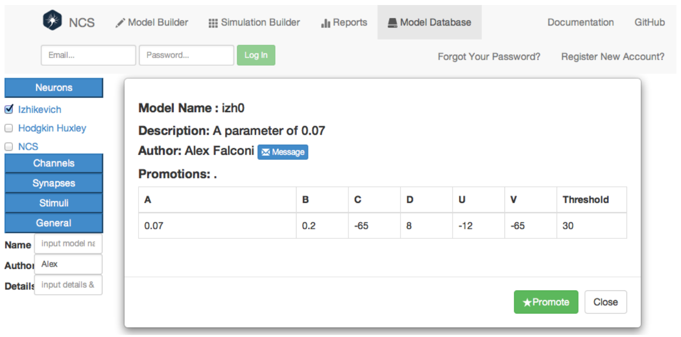
Figure 5: The models in the list are populated based on the search filter values, located in the left search panel.

By selecting the name of a list item, a user can view the models details. A model view opens on the page and shows the models general information, such as the name, author, and description. The detail view includes a table of parameters and the models parameter values. Each detail view is specific to the selected model type in order to intuitively display the information, as shown in Figure 6.

3.1.1 Use Cases - Repository

UC01 Search Models
A user selects model types to be included in the search results, and then expands the parameter dropdown for each type to enter parameter values. The list is populated based on the filters by type selections and entered parameter values.

UC02 Examine Model
After a user has applied search filters to the model list, the list contains relevant models. The user selects the



NCS | Model Builder | Simulation Builder | Reports | Model Database | Documentation | GitHub

Email... | Password... | Log In | Forgot Your Password? | Register New Account?

Neurons
- ✓ Izhikevich
- Hodgkin Huxley
- NCS

Channels | Synapses | Stimuli | General

Name: Input model na
Author: Alex
Details: Input details &

Model Name : izh0

Description: A parameter of 0.07

Author: Alex Falconi Message

Promotions: .

A	B	C	D	U	V	Threshold
0.07	0.2	-65	8	-12	-65	30

Promote | Close

Figure 6: Selecting a model name in the list opens the detailed view which shows the parameter values and the promote option.

name of a model in the list and a detail view window opens. The user views the parameter values for the model.

UC03 Upload and Download Model
A user can choose to upload a model from database to model builder and download model from model builder to database.

3.2 Reports

Upon entering the reports tab of the application, users will be greeted by a reports control panel. From here, users will be able to control a myriad of functions which allow the user to add graphing instances into the environment, add graphing columns, change settings, and view the reporting status of the simulator. When a user adds a graph instance, it is either in the form of a line graph or a raster plot. The graph is instantiated and placed in the environment. The environment which houses the graphs is comprised of graphing columns. A graphing column is a container that houses and displays graphs vertically (e.g. if a user has one graphing column, it is possible to have one "stack" of instantiated graphs. If a user has two graphing columns, it is possible to have two "stacks" of graphs). A key feature enabled by these graphing columns is the ability to drag graphs from one graphing column to another. This is the main implementation that allows for spatial customization where a user can customize a certain region of the webpage to have a certain set of graphs.

When a graph is instantiated, each graph will have its own set of buttons at the header of the instantiated graph. These buttons grant the user a series of customization tools: add or delete lines, change the color of a line, zoom in or zoom out within a graph, change the vertical dimensions of a graph, pause a graphs' reporting state, resume a graphs' reporting state, record the current state of the graph as an animated GIF or static SVG.

3.2.1 Use Cases - Reports

UC01 Add Line Graph or Raster Plot
A user has the ability to add reporting windows that can abstract the data in the form of a Line Graph or a Raster Plot.

UC02 Zoom In or Zoom Out
A user can zoom into or out of a reporting window to accomodate their viewing preferences. The differences in zoom levels can be viewed in Figure 7.

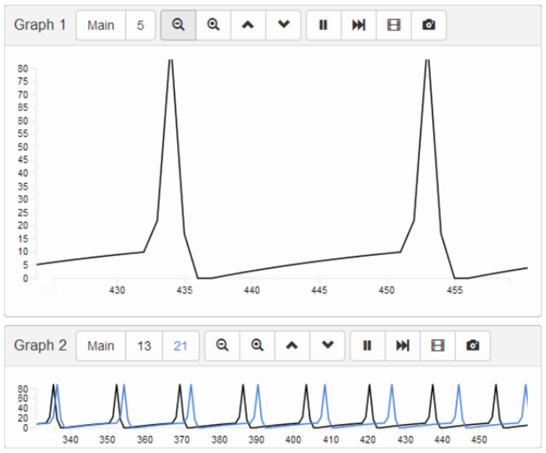


Figure 7: An example of multiple zoom levels and window dimensions.

UC03 Change Color
A user can choose to change the color of a line in a line graph to any color using a gradient color picker for easier viewing of a certain cell as seen on Figure 8.

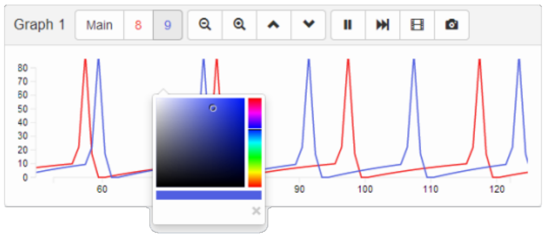


Figure 8: An example of multiple cell channels and color changes to facilitate easy interpretation.

UC04 Play or Pause
As a graph is dynamically reporting data, a user can choose to pause the reporing process and the graph will halt at the current data shown. When the user is ready, he or she may continue to view the reports by pressing the play button. Play and pause buttons are viewable in Figure 9.

UC05 Position Slider
Should the user wish to view old data that has already

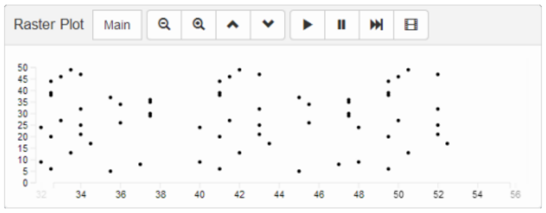


Figure 9: An example of the raster plot reporting window.

been reported, the user may drag the position slider as nesessary to view data as they wish.

UC06 Graph Recording
A user has the ability to record a frame of the graph and save to file. Upon the recording options is an animated GIF, a static GIF, or an SVG.

4 Conclusion and Future Work

4.1 Conclusion

The NCR project follows a fundamental principle: a fluid harmony between the powerful back end brain simulator and the intuitive front end array of tools is essential to the human paradigm of easy to use, easy to share, and easy to understand. NCR allows users to view available brain models with ease and intuitively view a graphical abstraction of the simulation output. Combined, these various components form a product which encourage an easy work flow of user control when interacting with something as complex as the NCS.

4.2 Future Work

Future work for the NCR project includes the addition of non-core features; for example, the implementation of a note taking tool to attach notes or comments to models in the database, or a discussion forum for users so as to expand their collaborative potential with other users across the world. Other ideas include the ability to copy brain models from other existing databases, and the flexibility to accept different types of models that may not currently be compatible with the database.

Other types of future work include the migration onto other platforms such as mobile devices like smart phones or tablets. Adapting the application to table type technology or other touch screen devices may include utilizing the full potential of a touchscreen interface; for example, allowing the user to pinch-zoom a graph within the reports tab, or to swipe through a listing of brain models pulled from the database instead of clicking through a pagination scheme.

References

[1] J. Berlinski, C. Rowe, D. M. Chavez, N. M. Jordan, D. Tanna, R. V. Hoang, S. M. Dascalu, L. C. J. Bray, and F. C. Harris, Jr. NeoCortical Builder: A Web Based Front End for NCS. In *Proceedings of the 27th International Conference on Computer Applications in Industry and Engineering (CAINE-2014)*, 2014.

[2] M. Bostock. D3.js: Data-driven documents. `http://d3js.org/`, 2014. (Retrieved May 19, 2014).

[3] J. E. Cardoza, A. K. Jones, D. J. Liu, R. V. Hoang, D. Tanna, L. C. Jayet Bray, S. M. Dascalu, and F. C. Harris, Jr. Design and Implementation of a Graphical Visualization Tool for NCS. In *Proceedings of The 2013 International Conference on Software Engineering and Data Engieering (SEDE 2013)*, 2013.

[4] R. V. Hoang. *An Extensible Component-based Approach to Simulation Systems on Heterogeneous Clusters*. PhD thesis, University of Nevada, Reno, 2014. `http://www.cse.unr.edu/~fredh/papers/thesis/PHD-010-Roger-Hoang/dissertation.pdf` (Retrieved July 24, 2014).

[5] R. V. Hoang, D. Tanna, L. C. Jayet Bray, S. M. Dascalu, and F. C. Harris, Jr. A Novel CPU/GPU Simulation Environment for Large-Scale Neural Modeling. *Frontiers in Neuroinformatics*, 7, 2013.

[6] N. M. Jordan, K. Perry, N. Narala, L. C. Jayet Bray, S. M. Dascalu, and F. C. Harris, Jr. Design and implementation of an NCS-NeuroML translator. In *Proceedings of the International Conference on Software Engineering and Data Engineering (SEDE 2012)*, Los Angeles, CA., June 2012.

[7] MongoDB Inc. MongoDB. `http://www.mongodb.com/`, 2014. (Retrieved May 19, 2014).

[8] Namlook. MongoKit. `http://namlook.github.io/mongokit/`, 2014. (Retrieved May 19, 2014).

[9] A. Ronacher. Flask. `http://flask.pocoo.org/`, 2014. (Retrieved May 19, 2014).

[10] The jQuery Foundation. jQueryUI. `http://jqueryui.com/`, 2014. (Retrieved May 19, 2014).