Design and Implementation of an NCS-NeuroML Translator

Nathan M. Jordan Kimberly B. Perry Nitish Narala
Laurence C. Jayet Bray Sergiu M. Dascalu Frederick C. Harris, Jr.

Department of Computer Science and Engineering
University of Nevada
Reno, NV 89557
{ ljayet@gmail.com}

Abstract

Computational Neuroscience has become an emerging field studying complex dynamics of the brain. Several software systems have been developed to simulate these processes with a wide range of biological detail, from the interaction between thousands of neurons, down to the dynamics between two individual cells. However, each uses a specific programming language or input syntax, which makes it difficult for non-programmers to use. Since each language is different, it also becomes a non-trivial task to compare results and exchange data among researchers, which slows down the discovery of important findings. We propose to create a tool that translates input files between two different languages, the NCS input language and the NeuroML format. It provides a user-friendly interface, which can be used to both create and edit simulations without using a text editor or a complex programming language. This novel tool will not only allow non-programmers to model brain processes and functions, but will also give NCS users an opportunity to compare their modeling results with other well-known simulators.

Keywords: NeuroML, NCS, Conversion, GUI, Editor, Computational Neuroscience.

1 Introduction

Within the past twenty years, the availability and viability of computer-based models of physical processes has increased dramatically. Using these systems, researchers can now simulate complex biological mechanisms that were previously limited to a very small scale, or simply were not feasible to experiment with. Neural network simulators report on the activity of groups of neurons and/or the activity between elements of the neurons themselves. In 2001, The University of Nevada, Reno began the development of a simulator of their own, called the NeoCortical Simulator (NCS) [1,2]. NCS runs on a grid of machines at the University, and has provided researchers the ability to understand high-function brain dynamics and study neurological disorders such as Alzheimer's disease.

As the usage and scale of the simulator increases, there has become a need to benchmark and compare the performance and experimental results of the NCS simulator with other noteworthy simulators such as NEURON [3], GENESIS [4], NEST [5], and PyNN [6]. However this is a more difficult task than what immediate observations would lead one to believe. Each simulator uses a different input syntax, as well as varying levels of assumption with regard to physics and biochemistry [7]. Consequently, a one-to-one mapping of inputs is not possible, and a significant level of higher-level processing is required to create a mapping between two types of input files.

To solve the problem of ambiguous input files, a new input language format is emerging as a de-facto standard for input to neural simulators. NeuroML is an XML-based markup language that describes neurological simulations with a high-level of biological detail [8]. Using this language, several simulators will be able to process the same input and their results can be compared more easily. NeuroConstruct [9], a graphical interface (GUI) file converter can already convert NeuroML into a few simulators' proprietary formats, including NEURON, GENESIS, MOOSE, PSICS, and PyNN. Unfortunately, NCS does not yet have this capability, and prevents comparing results with other simulators. To facilitate the use of NeuroML with the NCS simulator, we devised a language translator, NeuroTranslate, which can convert between the two types of input files. This editor has forms and other controls, including a user-friendly GUI that allow non-technical users to create simulations without the need to understand either NeuroML or the NCS input syntax.

The remainder of this paper is structured as follows: Section 2 outlines the functional and non-functional requirements of the system; Section 3 details the use cases for the application; Section 4 gives a design overview, including the structure and the behavior of the system; Section 5 describes the GUI of the project; Section 6 presents the current status of application; Section 7 draws conclusions and suggests planned future work.

2 Requirements Specification

2.1 Functional Requirements

The system shall provide the following functions and features (functional requirements labeled as (**F**):

F01 Open a NCS file to view.

F02 Open a NeuroML file to view.

F03 Save a file in NCS format.

F04 Save a file in NeuroML format.

F05 Translate NeuroML into NCS.

F06 Translate NCS into NeuroML.

F07 Edit a NCS file.

F08 Edit a NeuroML file.

F09 Create a NCS file.

F10 Create a NeuroML file.

2.2 Non-Functional Requirements

The system shall also fulfill the following non-functional requirements, labeled as (**NF**):

NF01 Run on the Linux operating system.

NF02 Minimum training time shall be required.

NF03 No programming knowledge shall be required.

NF04 Use a GUI to accomplish the aforementioned requirements.

NF05 Include documentation for user convenience.

NF06 The user interface must be easy and intuitive.

3 Use Cases

The functionality of the system is shown in the use case diagram described in Fig. 1. The details of the use cases **(UC)** are the following:

UC01 Open NCS File
The user selects the open option from the menu. He is then presented with a file dialog and searches for a NCS input file. Once the file is selected, the GUI opens the file in the NCS view.

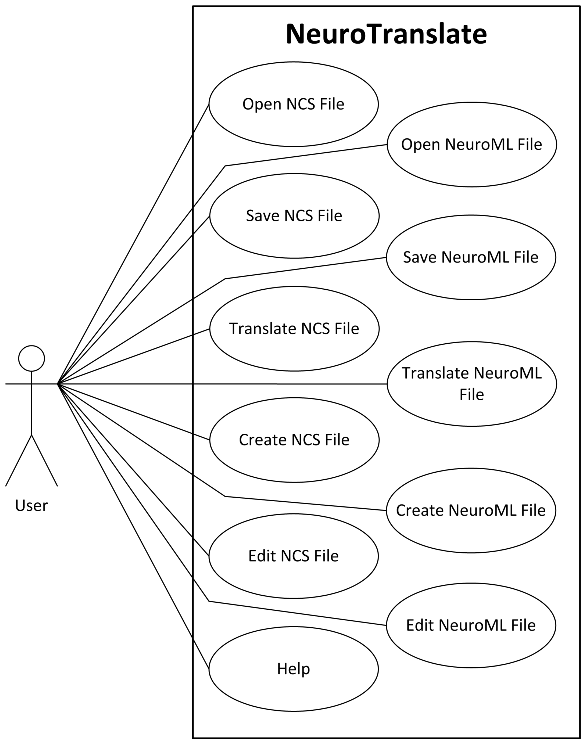


Figure 1: Use Case Diagram.

UC02 Open NeuroML File
The user selects the open option from the menu. He is then presented with a file dialog and searches for a NeuroML input file. Once the file is selected, the GUI opens the file in the NeuroML view.

UC03 Save NCS File
The user selects the save option from the menu. The system will then write the current model to an NCS format file.

UC04 Save NeuroML File
The user selects the save option from the menu. The system will then write the current model to an NeuroML format file.

UC05 Translate NCS File
The user selects the open option from the menu. The user is presented with a file dialog and searches for a NCS input file. Once the file is selected, the system will open and the user clicks the NeuroML button to open the translate dialog.

UC06 Translate NeuroML File
The user selects the open option from the menu. The user is presented with a file dialog and searches for

a NeuroML input file. Once the file is selected, the system will open and the user clicks the NCS button to open the translate dialog.

UC07 Create NCS File
The user selects the new file option and selects the NCS option which opens a blank model in the NCS view.

UC08 Create NeuroML File
The user selects the new file option and selects the NeuroML option which opens a blank model in the NeuroML view.

UC09 Edit NCS File
The user opens a NCS file and modifies parameters within the NCS view.

UC10 Edit NeuroML File
The user opens a NeuroML file and modifies parameters within the NeuroML view.

UC11 Help
The user selects the help option from the menubar. A help page is presented providing information regarding NeuroTranslate, NCS, and NeuroML.

4 Design Overview

4.1 High-Level Design

The software architecture of NeuroTranslate is based off of the Model-View-Controller architectural pattern. The view is represented by the user interface, the model is represented by the data structures containing the NeuroML and NCS data, and the controller is represented by the interconnecting components we have created to make the software design both portable and easy to understand. User interface events are handled through the UIController component, which in turn modifies and queries data from the model. The details of file I/O are left to the FileController, which handles the reading and writing of both NCS and NeuroML files. Figure 2 shows the architecture NeuroTranslate employs for its design.

4.2 Structure

The details of of each class, as described in Figure 3 are the following:

Main
This is the main entry point of the application, it instantiates the UIController and hands over control to the user.

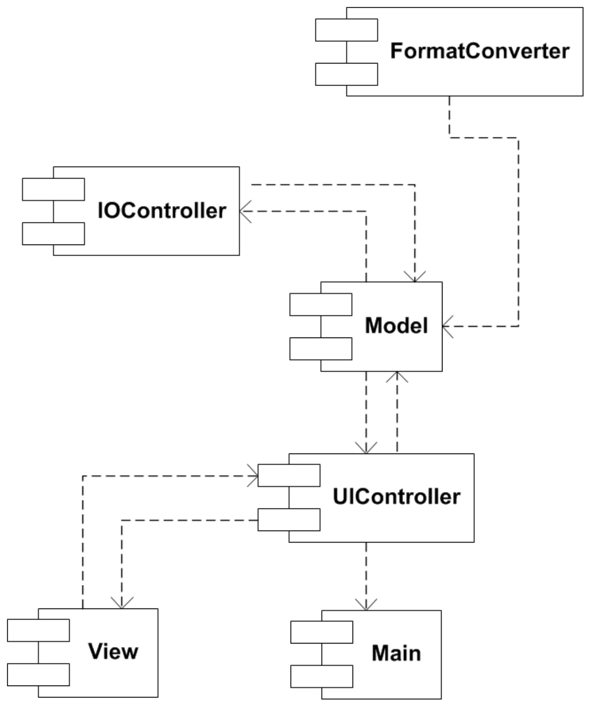


Figure 2: Component Diagram.

MainWindow
The main application window.

NCSData
Contains all the parameters represented in a particular NCS model, represents part of the model.

NeuroMLConverter
Required for switching between XML and the Neuroml object with JAXB.

NCSConversionData
Contains the NCSData object that is a product of the conversion as well as any problems that occurred therein.

Model
Handles transactions to the NCS data or the NeuroML data, dependent on which mode the user is in.

UIController
Responsible for handling events triggered by the user interface, interacts with the model to either get information or make changes.

Neuroml
An object containing the data representative of a NeuroML model.

NeuromlConversionData
Contains the Neuroml object that is a product of the conversion as well as any problems that occurred.

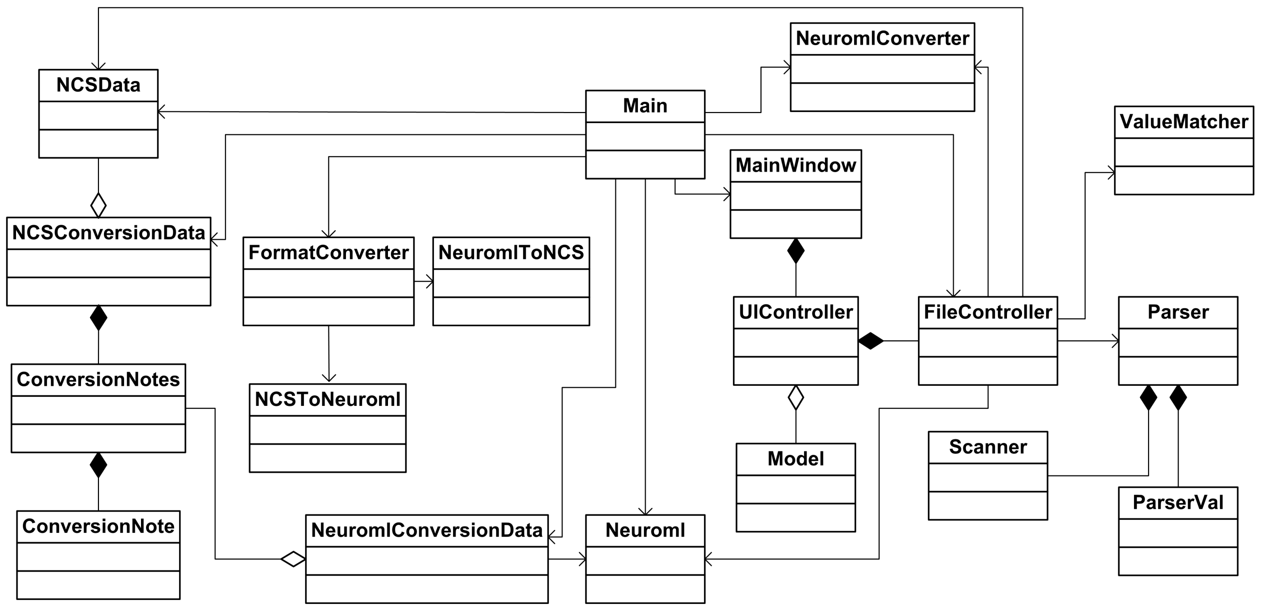


Figure 3: Class Diagram.

FormatConverter
Directs translating between the two file types.

ConversionNotes
Contains a list of information regarding the translation from one format to the other.

NeuromlToNCS
Converts a NeuroML object into an NCSData object and keeps track of problems that may arise during that process.

NCSToNeuroml
Converts a NCS file into a Neuroml object and keeps track of problems that may arise during that process.

ValueMatcher
Matches the names of NCS entities obtained during the parsing of the file to the corresponding objects.

ConversionNote
Results from problems during translation. Contains the name of an entity, the severity of the problem, and a message to direct the user how to solve the problem.

FileController
Opens and saves files to/from the application.

Parser
The NCS parser object produced by BYACCJ.

Scanner
The NCS scanner object produced by JFlex.

4.3 Behavior

When NeuroTranslate is started, the system enters the waiting state where it responds to UI events. While the system is in the waiting state, it will display the selected filetype's view, currently either NCS or NeuroML. The following list describes the different states in which NeuroTranslate can exist, as illustrated in Figure 4.

Open File
The system enters the Open state when the user selects the open file option. The system will prompt the user with a file dialog to enter a filename. The system returns to the waiting state upon exit where the new file parameters are displayed.

New File
The system enters the New File state when the user selects the new file option. The system will clear the current file and replace it with a blank file and re-enters the waiting state.

Save File
The system enters the Save File state when the user selects the save file option. The system will prompt the user with a file dialog to enter a filename. Once a filename has been entered, the system will write the file to the corresponding format and return to the waiting state.

Translate
The system enters the Translate state when the user

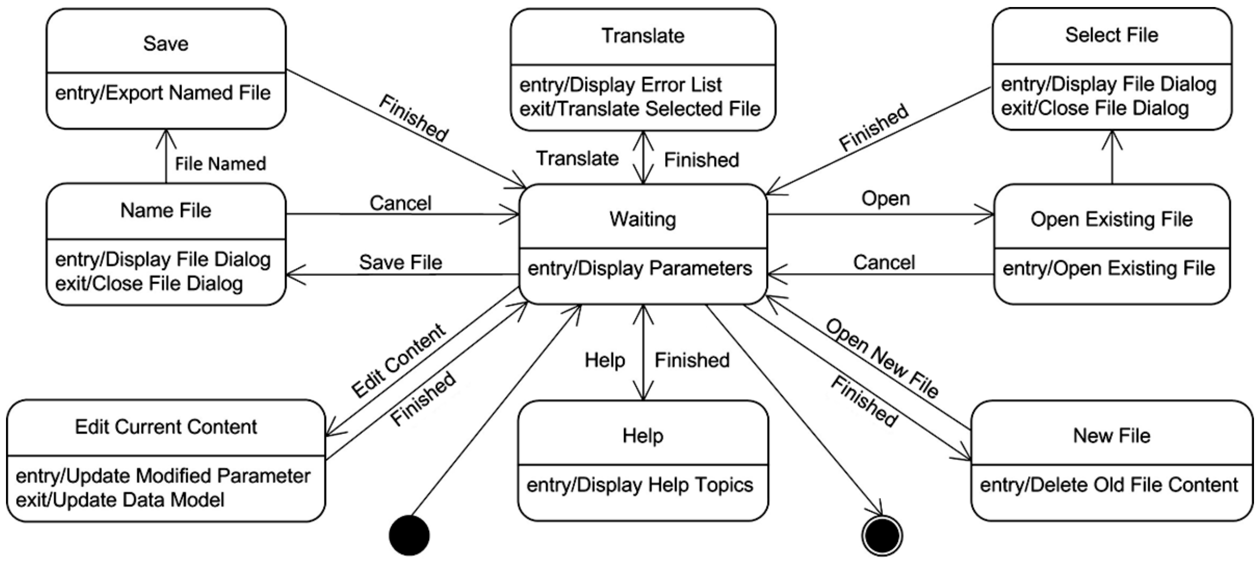


Figure 4: State Diagram.

is ready to translate a file. The system will display an error popup which lists all the warnings and errors if applicable. Upon exiting this state, the system will either translate the file into the new format and reenter the waiting state, or will simply return to the waiting state if either the translation is not possible or the user cancels the translation.

Edit Content
The system enters the Edit Content state when the user updates or modifies a parameter. In this state, the system propagates the changes to the model and other views and returns to the waiting state.

Help
The system enters the Help state when the users selects the help option. In this state, the system will display a help screen with information regarding NeuroTranslate, NCS, and NeuroML.

4.4 Supporting Resources

To aid the development and functionality of NeuroTranslate, we have used a few external libraries and programs.

User Interface
NeuroTranslate required a user interface that was both easy to understand, use and develop, and functioned like a native GNOME application. We considered Swing, AWT, Qt for our user interface design, however, upon review they did not meet our requirements. We instead chose to utilize the java-gnome library [10] for our user interface. Not only does it provided a consistent and quality appearance, it also allowed us to use the libglade user interface library in conjunction with the Glade Interface Designer. These two components facilitated loading a user interface dynamically during runtime, which allowed for greater flexibility during development, and less clutter in our Java code.

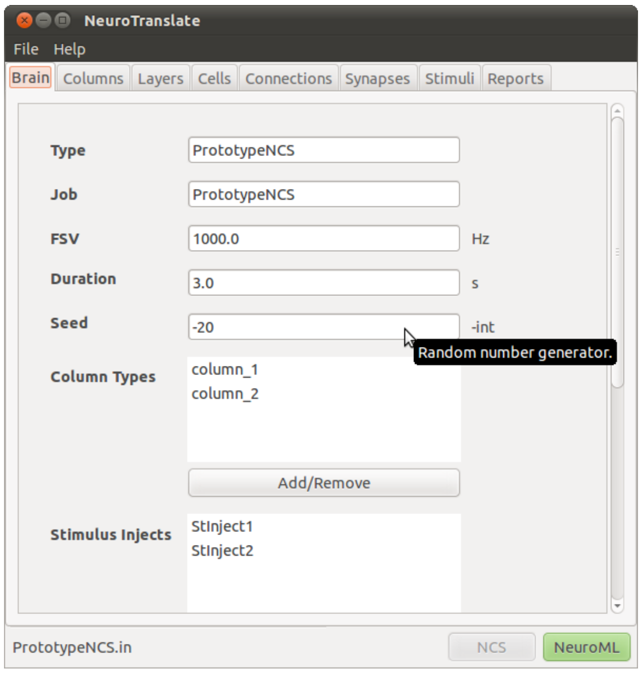
NeuroML
In order to manipulate NeuroML entities within a Java environment, NeuroTranslate needed to read a NeuroML file and represent it with Java objects in a logical manner with respect to the markup language. To do this, NeuroTranslate used the JAXB (Java API for XML Bindings) framework [11]. This framework could both read a NeuroML file and create the corresponding Java objects, and write those Java objects to a syntactically and semantically valid NeuroML file.

NCS
In addition to NeuroML, NeuroTranslate was also required to read a NCS input file into the Java environment. An existing parser for the file type existed (it was used for the simulator itself), however it was written in C, and proved problematic to work with. In order to solve this problem, we created our own lexical scanner and parser using JFlex [12] and BYACCJ [13] for the scanner and parser respectively. These tools dramatically cut down on development time by providing an easier way to create the parser needed to turn the NCS parameters into Java objects for NeuroTranslate to work with.

5 GUI

The NeuroTranslate GUI was created with simplicity in mind. Each of the supported languages (currently NeuroML and NCS input) have their own layouts, which display the parameters in a logical and straightforward manner. The different biological levels of each language are represented in a tabulated list. The most general and higher-level parameters are located on the leftmost tab, and the most biologically specific parameters are located in the rightmost tabs. Within each tab, there are a list of parameters with their names located on the left, their values in the middle, and the units of the parameter (if applicable) are on the right. The changes the user makes within the GUI are reflected immediately in the model and in other tabs as the change is committed, and is visible in the rest of the tabs and fields within the program. Tooltips are available for the parameters to assist the user as well. This layout is demonstrated in Figures 5 and 6 for NCS and NeuroML, respectively.

In order to translate a file from one language to another, the user clicks the languages button (located in the bottom right of the view, see Figure 5) that they wish to translate to. A popup window is then displayed which shows any problems or notifications associated with translation. There are two levels of severity for a problem: warning and error. A warning advises the user that some parameter they have specified in the model might require additional configuration or may not map exactly how they intended. An error specifies that the translation is not possible due to a particular parameter that does not map at all into the other language. Each problem contains the offending parameter, its severity, and a message to help the user discern what the problem is. Examples of this UI element can be found in Figure 7.

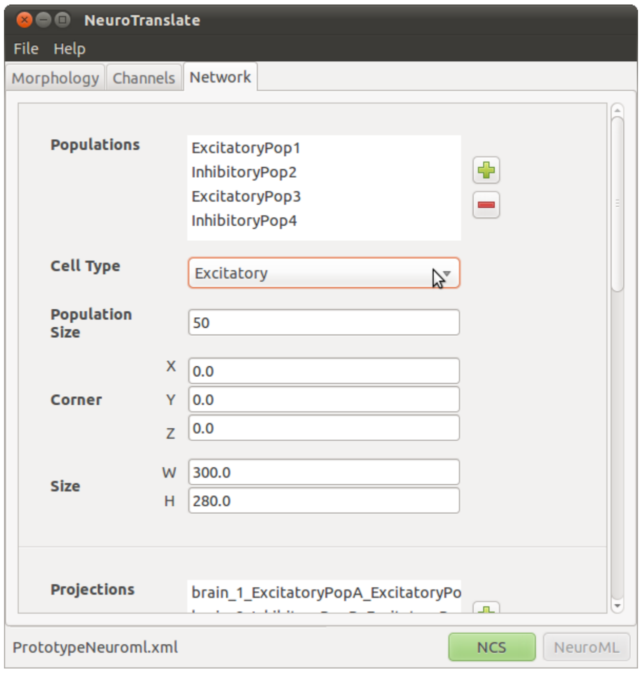


Type	PrototypeNCS	
Job	PrototypeNCS	
FSV	1000.0	Hz
Duration	3.0	s
Seed	-20	-int
Column Types	column_1 column_2	
Stimulus Injects	StInject1 StInject2	

Random number generator.

PrototypeNCS.in

Figure 5: NCS File.



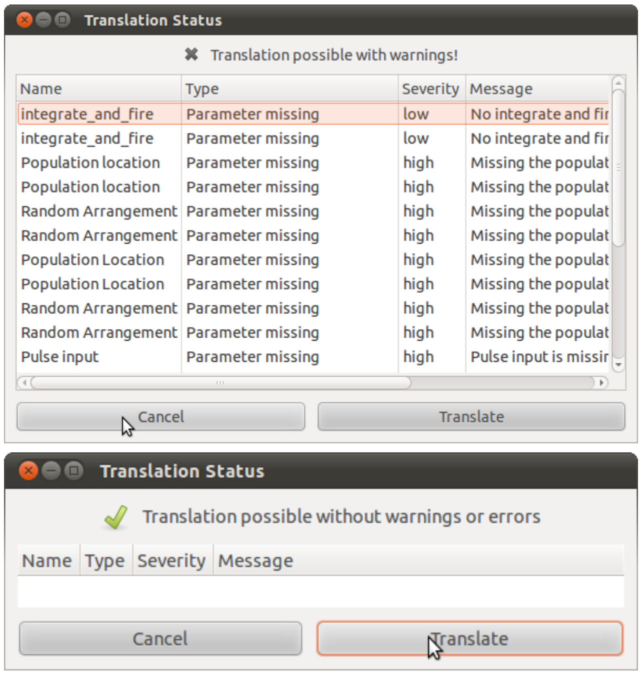
Populations		ExcitatoryPop1 InhibitoryPop2 ExcitatoryPop3 InhibitoryPop4
Cell Type		Excitatory
Population Size		50
Corner	X	0.0
	Y	0.0
	Z	0.0
Size	W	300.0
	H	280.0
Projections		brain_1_ExcitatoryPopA_ExcitatoryPo

PrototypeNeuroml.xml

Figure 6: NeuroML File.

6 Current Status

In its current state of development, NeuroTranslate includes a GUI that allows the user to import a NCS file, add and change certain parameters, and export the file into a valid NeuroML file. This allows researchers to represent neural models at various scales in a simulator independent manner. The system currently provides feedback during translation as to whether or not a certain parameter can properly or completely be mapped to a counterpart in the other language. Some of the more complicated parameters have yet to be implemented, and currently result in warnings/errors during translation.


Translation Status

✖ Translation possible with warnings!

Name	Type	Severity	Message
integrate_and_fire	Parameter missing	low	No integrate and fir
integrate_and_fire	Parameter missing	low	No integrate and fir
Population location	Parameter missing	high	Missing the populat
Population location	Parameter missing	high	Missing the populat
Random Arrangement	Parameter missing	high	Missing the populat
Random Arrangement	Parameter missing	high	Missing the populat
Population Location	Parameter missing	high	Missing the populat
Population Location	Parameter missing	high	Missing the populat
Random Arrangement	Parameter missing	high	Missing the populat
Random Arrangement	Parameter missing	high	Missing the populat
Pulse input	Parameter missing	high	Pulse input is missir

Translation Status

✓ Translation possible without warnings or errors

Name	Type	Severity	Message

Figure 7: Translation Status Diagram - No Problems and Warning Examples.

7 Conclusion and Future Work

NeuroTranslate is a graphical tool to open, create, and edit the NCS and the NeuroML input file format to then translate between them as necessary. With the increasing use of both the NCS simulator and other simulators located around the world, it is becoming increasingly important for researchers to compare results and exchange models and ideas. NeuroTranslate is a step in this direction by allowing computational and non-computational neuroscientists to create models, run simulations and compare results empirically.

The potential of NeuroTranslate for future developments includes a few different features. Given that other simulators besides NCS have not yet adopted the NeuroML standard for an input syntax, the option exists to add additional language support to NeuroTranslate to expand its viability as a tool for neural network simulations. Another feature that has been requested is the ability to interpret the results of the simulation with various forms of data visualizations, such as line charts, and spike train raster plots. In addition to visualizing the results of a simulation, it would also be possible to visualize the model itself using some form of graphical notation (i.e. displaying the location of neurons and other biological objects the way they exist in the simulation), similar to the existing functionality of neuroConstruct [9].

Acknowledgements

This work was supported by a grant from the U.S. Office of Naval Research (N000140110014).

References

[1] R. Drewes. Brainlab: a toolkit to aid in the design, simulation, and analysis of spiking neural networks with the ncs environment. Master's thesis, University of Nevada, Reno, 2005.

[2] C. Wilson, P.H. Goodman, and F.C. Harris Jr. Implementation of a biologically realistic parallel neocortical-neural network simulator, 2005.

[3] N.T. Carnevale and M.L. Hines. *The NEURON Book*. Cambridge University Press, 2006.

[4] J. M. Bower and D. Beeman. Constructing realistic neural simulations with GENESIS. *Methods Mol Biol*, 401:103–125, 2007.

[5] M. Diesmann and M.-O. Gewaltig. NEST: An environment for neural systems simulations, 2001.

[6] J.M. Eppler, M. Helias, E. Muller, M. Diesmann, and M.-O. Gewaltig. PyNEST: a convenient interface to the nest simulator. 2(00012), 2009.

[7] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J.M. Bower, M. Diesmann, A. Morrison, P.H. Goodman, F.C.J. Harris, M. Zirpe, T. Natschlager, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A.P. Davison, S. El Boustani, and A. Destexhe. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of computational neuroscience*, 23(3):349–398, 2007.

[8] P. Gleeson, S. Crook, R.C. Cannon, M.L. Hines, G.O. Billings, M. Farinella, T.M. Morse, A.P. Davison, S. Ray, U.S. Bhalla, S.R. Barnes, Y.D. Dimitrova, and R.A. Silver. NeuroML: A language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Comput Biol*, 6(6):e1000815, 2010.

[9] P. Gleeson, V. Steuber, and R.A. Silver. neuroConstruct: A tool for modeling networks of neurons in 3d space. *Neuron*, 54(2):219–235, 2007.

[10] A. Cowie. The java-gnome user interface library, 2011. Available from: http://java-gnome.sourceforge.net.

[11] E. Ort and B. Mehta. Java architecture for XML binding (JAXB), 2003.

[12] G. Klein. JFlex: The fast scanner generator for java, 2009. Available from: http://jflex.de.

[13] T. Hurka. BYACC/J, 2008. Available from: http://byaccj.sourceforge.net.