Neuroinform (2008) 6:291–309
DOI 10.1007/s12021-008-9021-2

# KInNeSS: A Modular Framework for Computational Neuroscience

**Massimiliano Versace · Heather Ames · Jasmin Léveillé · Bret Fortenberry · Anatoli Gorchetchnikov**

Published online: 10 August 2008


**Abstract** Making use of very detailed neurophysiological, anatomical, and behavioral data to build biologically-realistic computational models of animal behavior is often a difficult task. Until recently, many software packages have tried to resolve this mismatched granularity with different approaches. This paper presents KInNeSS, the KDE Integrated NeuroSimulation Software environment, as an alternative solution to bridge the gap between data and model behavior. This open source neural simulation software package provides an expandable framework incorporating features such as ease of use, scalability, an XML based schema, and multiple levels of granularity within a modern object oriented programming design. KInNeSS is best suited to simulate networks of hundreds to thousands of branched multi-compartmental neurons with biophysical properties such as membrane potential, voltage-gated and ligand-gated channels, the presence of gap junctions or ionic diffusion, neuromodulation channel gating, the mechanism for habituative or depressive synapses, axonal delays, and synaptic plasticity. KInNeSS outputs include compartment membrane voltage, spikes, local-field potentials, and current source densities, as well as visualization of the behavior of a simulated agent. An explanation of the modeling philosophy and plug-in development is also presented. Further development of KInNeSS is ongoing with the ultimate goal of creating a modular framework that will help researchers across different disciplines to effectively collaborate using a modern neural simulation platform.

**Keywords** Behavioral modeling · Compartmental modeling · Object oriented design · Spiking neurons · Software framework

M. Versace (✉) · H. Ames · J. Léveillé · B. Fortenberry · A. Gorchetchnikov
Department of Cognitive and Neural Systems, Boston University, 677 Beacon Street,
Boston, MA 02215, USA
e-mail: versace@cns.bu.edu

M. Versace · H. Ames · J. Léveillé · B. Fortenberry · A. Gorchetchnikov
Center of Excellence for Learning In Education, Science, and Technology, Boston University,
Boston, MA, USA

## Introduction

Advances in functional, anatomical, and behavioral neuroscience techniques have led to an increase in the data available for modeling complex dynamics of biologically inspired neural networks at many levels of abstraction, from in-depth descriptions and analyses of individual membrane channels to large-scale investigations of whole brain activity. This wealth of data is essential for creating realistic neural models and increases our understanding of animal and human behavior. Furthermore, it has pushed the modeling community towards the design of increasingly complex models, incorporating unprecedented amount of biophysical and anatomical constraints. These large-scale neural models are often non-linear dynamical systems which can be analytically intractable and require numerical simulation to gain insight into their behavior. Emergent properties of large-scale neural networks often remain unnoticed until the whole system is simulated and components are allowed to interact (Cannon et al. 2002).

An additional level of complexity is finding a neural simulator and simulation environment that would enable the large variety of researchers from neurophysiology, psychology and computational modeling to share data and

work collaboratively (an excellent review can be found in Brette et al. 2007). Most available software packages are specialized in different applications. For example, CSIM (Maass et al. 2002; Natschläger et al. 2002) and NEST (Gewaltig and Diesmann 2007) make use of single compartmental models whereas KInNeSS, NEURON (Hines 1989, 1993; Hines and Carnevale 1994; Carnevale and Hines 2006), GENESIS (Bower and Beeman 1998), and SPLIT (Hammarlund and Ekeberg 1998) also include functionality for creating multi-compartmental models. Other software, such as XPPAUT (Ermentrout 2002), focus primarily on dynamical systems analysis.

An attempt to integrate this diverse set of neural simulators has led to the development of NeuroML[1] (Neuron Markup Language; Crook et al. 2007), which seeks to provide a common schema for unifying network descriptions and building a database of neural models. While developing such a standard format is a necessary step towards the interoperability of various existing software approaches to neural simulations, it is barely sufficient and far from achieving this goal (Brette et al. 2007; Goddard et al. 2001; Cannon et al. 2007).

The open source[2] KDE Integrated NeuroSimulation Software environment (KInNeSS[3]) follows an XML schema similar to NeuroML and represents a step towards the development of an interdisciplinary, modular neural network software environment. The main goal of KInNeSS is to allow modelers to quickly design, test, and tune their neural models and study their behavior, which can either be the behavior of a simulated agent or the more abstract behavior of a target neural population. For this purpose, KInNeSS users can design simple integrate-and-fire neurons as well as branched structures with complex conduction based models obeying Hodgkin–Huxley dynamics (Hodgkin and Huxley 1952).

KInNeSS is built using C++ modern programming techniques such as Object-Oriented Programming (OOP), polymorphism, multithreading, and functional objects. KInNeSS users with different levels of expertise can quickly and effectively design, run, and analyze simulations at many levels of granularity, from single compartment and single cells to large-scale dynamics recorded in the form of local field potentials (LFP) and current source densities (CSD), and eventually link them to the behavior of the simulated agent. Users without programming skills can take advantage of the simple and intuitive interface and experienced programmers can add features and components to the system without waiting for software updates. Finally, KInNeSS's modular design allows for integration of different simulation projects within the same interface.

The following section gives a general overview of the KInNeSS software and its computational engine, SANNDRA[4] (Synchronous Artificial Neural Networks Distributed Runtime Algorithm). The third section describes the XML schema adapted for KInNeSS and the fourth section gives an overview of the modeling philosophy used in KInNeSS including a description of the equations used in spiking neural modeling architectures. The fifth section illustrates plug-in development that advanced users can employ to expand KInNeSS functionality to their needs. The sixth section presents KInNeSS performance on two benchmark networks and a coarse comparison to other neural simulators. Finally, future developments and conclusions are presented.

## KInNeSS Overview

KInNeSS was originally developed to be a simulation environment for modeling neural systems and neurons whose characteristics are closely linked to experimental data and whose explanatory power encompasses behavioral data. KInNeSS is capable of simulating both conductance based models of cells obeying Hodgkin–Huxley dynamics and simpler systems based on integrate-and-fire models of neurons. KInNeSS is best suited to simulate networks of branched neurons containing approximately 1–4 compartments per branch and biophysical properties such as: membrane potential, voltage-gated or ligand-gated channels, gap junctions or ionic diffusion, neuromodulation channel gating, habituative or depressive synapses, axonal delays, and synaptic plasticity. KInNeSS records the voltage of individual compartments, as well as aggregate cell recordings such as local field potentials (LFP) and current source densities (CSD). Typical simulations that have been performed by KInNeSS include models of hippocampal neurons and spatial navigation (Gorchetchnikov and Hasselmo 2005), models of thalamo-cortical learning (Grossberg and Versace 2008; Leveille et al. 2008), and models of interactions between electric field and cell activity (Berzhanskaya et al. 2007).

KInNeSS can be used independently of any specific programming skills. A friendly point-and-click interface allows the modeler to set all the necessary parameters. The interface also contains functionality for loading projects, which can be written separately from KInNeSS. The project environment contains the tools needed to simulate the environmental and behavioral components of the model. Furthermore, KInNeSS makes use of an XML schema for both import and export of model specifications.

[1] http://www.neuroml.org

[2] KInNeSS is licensed under the GNU general public license; © Anatoli Gorchetchnikov.

[3] http://www.kinness.net

[4] http://www.kinness.net/Docs/SANNDRA/html/index.html

The current KInNeSS 0.3.4 release contains a project environment for modeling spatial navigation tasks, a generic project where an input pattern is provided for the network without specifying how it was created, and a dummy instructional project that illustrates the necessary interactions between the project environment and the core shell for programmers who wish to create new projects. The model interface is implemented as a set of plug-ins, so that the same model can be used in different project environments and different models can be used within the same environment.

Two plug-ins are contained in the current release of KInNeSS: one for creating and editing the network, and the other for running the simulations. The simulation plug-in allows the user to set global simulation parameters and to control the time course of the simulation. The interface allows interruptions at any point in time of the simulation, correcting parameters, and restarting the simulation from the point of interruption or from the original starting point. Additional plug-ins can be created and loaded by the user; see "Plug-in Development" section.

## Computing with SANNDRA

The computational engine behind KInNeSS is SANNDRA (Synchronous Artificial Neural Networks Distributed Runtime Algorithm). SANNDRA was originally designed under the name of SiMeON (Simulation of Memory based On Natural principles) in 1997–1999 and developed using SIMD architecture on the MasPar MP1 parallel computer. It was ported for Linux as a partial SIMD emulation in 2000 (Gorchetchnikov 2000). Initially, SANNDRA was intended to run a long iterative loop through relatively simple computations done in parallel on many similar elements. Due to the SIMD paradigm, it was a synchronous time driven rather than event-driven algorithm from the start. Later design relaxed the SIMD requirements on the similarity of computational elements, but these elements are still synchronized for data exchange. On a sequential computer this leads to an unavoidable performance loss due to synchronization, but ensures that these elements receive the correct input signals.

Each element in SANNDRA can have access to the output of any other element. This design makes SANNDRA capable of numerical integration of large systems of non-homogeneous differential equations and it can be put into much simpler uses like iterative solving of systems of algebraic equations or image processing. For differential equations SANNDRA uses the fourth order Runge–Kutta integration method.

SANNDRA is currently a separately distributed open source[5] library while its main use and testing is done in the KInNeSS project. SANNDRA extensively uses polymorphism to achieve an optimal combination of flexibility and performance. Polymorphism is a key concept of object-oriented design where object-specific individual implementations of methods can be called through a pointer to a virtual method declared in the common base of these objects. The system of equations is combined together from objects derived in the user code from SANNDRA basic objects. In the case of KInNeSS, this user code is part of the package that links the graphical user interface (GUI) and the computational engine. A set of standard building blocks is provided by the library, and the advanced user can always derive the additional blocks from those that are provided. Once combined, the system can be solved independently of its actual structure. There is a slight computational overhead in polymorphism, because a direct function call is faster than a virtual call. For a homogeneous system, when the function for such a direct call can be determined at compilation time, using polymorphism would be disadvantageous. Since SANNDRA is designed to use non-homogeneous systems, the appropriate direct call cannot be determined during compilation, and the choice is between polymorphism and some other kind of run-time detection of the right method to call. In this case polymorphism clearly is the best choice, since it is a part of the C++ language itself.

## Interface Quick Start

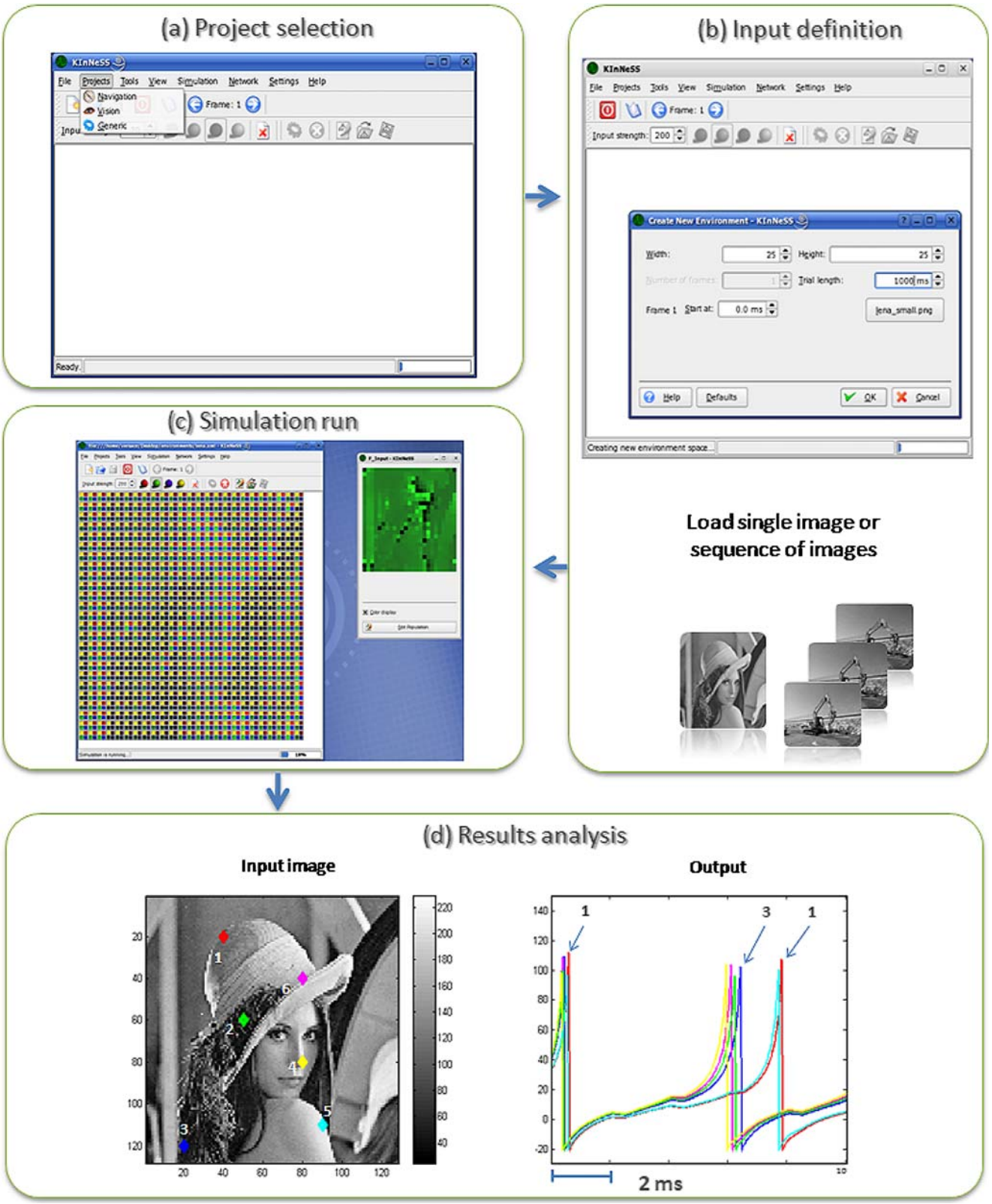
One of the best ways to appreciate the usability of KInNeSS is to experience it in action. In a few simple steps the user can design a neural network, create the input, run the model, and save the results for further analysis. These steps for running a sample network, receptor_example.nml[6], are found in Fig. 1.

When the user starts KInNeSS for the first time, the interface looks similar to the screenshot in Fig. 1a. The number of buttons in the *Project* Toolbar varies depending on the number of installed project environments. The main interface contains the following functionalities:

- *File* Menu: loads and saves environments and other project-specific operations.
- *Settings* Menu: toggles the visibility of the main toolbar and status bar, and contains a dialog window to configure shortcuts, the main toolbar, project toolbar buttons, and the KInNeSS preferences.
- *Help* Menu: accesses the KInNeSS manual, about dialogs, context help, and dialog for bug report submission.

[5] SANNDRA is licensed under the GNU general public license; © Anatoli Gorchetchnikov.

[6] http://kinness.net/Docs/KInNeSS/examples/Example_Receptor_Kinness.rar


(a) Project selection
KInNeSS
File Projects Tools View Simulation Network Settings Help
Navigation
Vision
Generic
Frame: 1
Ready
(b) Input definition
KInNeSS
File Projects Tools View Simulation Network Settings Help
Frame: 1
Input strength: 200
Create New Environment - KInNeSS
Width: 25
Height: 25
Trial length: 1000 ms
Frame 1 Start at: 0.0 ms
lena_small.png
Help
Defaults
OK
Cancel
Creating new environment space...
Load single image or sequence of images
(c) Simulation run
(d) Results analysis
Input image
Output
1
2
3
4
5
6
2 ms

◀ **Fig. 1** Setting up and running a simulation in KInNeSS is easy. After selecting the environment (**a**), the user defines the input either by point-and-click in the input grid or by loading one or more .png files. In the example, the user loads lena_small.png. (**b**) After loading the network receptor_example.nml, the user can start the simulation and monitor in real time (**c**) the input in the main interface (*left*) and the membrane potential of the neural population(s) (*right*), where *bright green colors* represent high membrane potentials, and *dark green* represent low membrane potentials. Finally, the results can be analyzed by other plotting and data analysis software (e.g., MATLAB®). Panel (**d**) shows a plot (*right*) of the membrane potential of six neurons that correspond to six different spatial locations (*left*) in the receptor population. The *right plot* highlights the spike timing of neurons 1 and 3

- Main Toolbar: accesses all commands in the three menus listed above and in the *Project* Menu.
- Status Bar: visualizes the progress of the simulation.
- *Projects* Menu and Toolbar: contains the list of project environments available to KInNeSS. The current release includes:
  - *Navigation*: creates mazes, places rewards in the mazes, and a virtual animal to run through the maze;
  - *Generic*: provides four input channels to the network symbolized by red, green, blue, and yellow colors; and
  - *Vision*: shows the example of the necessary skeleton to implement a new environment.

For more in-depth information and installation instruction see the KInNeSS user manual online[7]. A web-based tutorial that covers the basics of setting up a small network, running it, and collecting the simulation results is also available online[8].

## Behavioral Simulations

One of the unique components of KInNeSS is the abililty to link empirical model simulations and the resulting behavior. KInNeSS was originally designed to create large scale dynamical neural network models embedded in a perception-action loop so that the user can input realistic stimuli (e.g. images) and visualize the resulting actions of a simulated agent exposed to these stimuli. The behavioral level and the network level interact through both the input and output of the network. The user is able to control which network population drives the behavior of the simulated agent. KInNeSS contains three predesigned project environments for visualizing behavioral results of the simulations. These environments are the *Navigation* Project, the *Generic* Project, and the example *Vision* Project. Additional project environments can be designed by advanced users based upon the skeleton provided in the *Vision* Project.

### *The Navigation Project Environment*

This environment is designed to create and run projects in which a simulated agent (for example, a rat) navigates in a simulated environment. This project has been extensively used for "rat in the maze" simulations (Gorchetchnikov and Hasselmo 2002, 2005). Users can create simple mazes of variable dimensions, specify the location of the animal with the "rat tool", and edit the environment by adding obstacles with the "shovel tool", and rewards of variable salience with the "cheese tool". After the animal is placed in the environment, the *Network Editor* plug-in (see "Network Editor" section) is enabled and allows the user to load or create the neural network that controls the animal's interaction with the environment. Editing of the environment is possible at any time, even during the simulation run.

Figure 2 illustrates an example of a model that makes use of the Navigation project environment. The model describes a cortico-hippocampal circuit used by a simulated rat to flexibly navigate toward any arbitrary goal or multiple goals that change on a trial-by-trial basis (Gorchetchnikov and Hasselmo 2005).

### *The Generic Project Environment*

This environment is appropriate for simulations that involve arbitrary external inputs. After the user selects this project through the *Project* Menu, additional menu options appear in the interface and the related preferences pages appear in the KInNeSS preferences dialog. Selecting *File→New* opens the dialog that sets the dimensions of the new environment and the number of input time frames to load (optional). A point-and-click interface allows the user to edit the spatial input pattern in order to test different spatial–temporal configurations of inputs to the neural network. It is also possible to load input patterns from any graphics file in a format supported by the Qt library.

## Network Editor

Flexibility in network editing is essential to neural network modeling. Network setup and manipulations are implemented in KInNeSS within a plug-in accessible through the *Network* Menu or the buttons embedded in the *Project* Environment Toolbar. The *Network Editor* handles the network structure. In addition to the standard functionality of loading and saving the networks, this plug-in has an easy to use point-and-click interface that provides an intuitive framework for quickly designing and testing large-scale networks with complex connectivity patterns.

[7] http://www.kinness.net/Docs/KInNeSS/manual/index.html
[8] http://www.kinness.net/kinness_tutorial.wmv

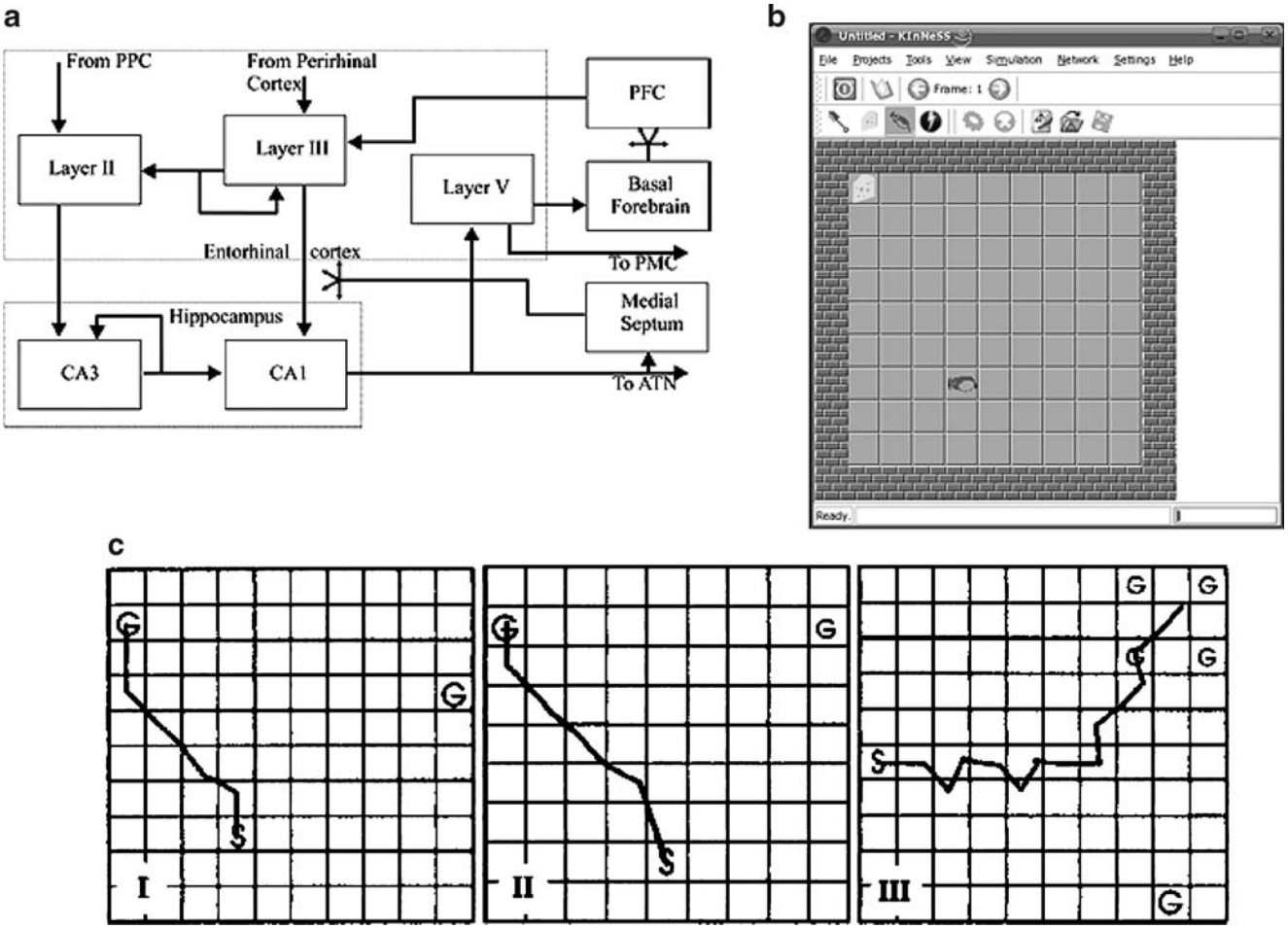



**Fig. 2** (**a**) Structure of the model describing the cortico-hippocampal circuit used by a simulated rat to flexibly navigate toward any arbitrary goal or multiple goals that change on a trial-by-trial basis. PPC, posterior parietal cortex; PFC, prefrontal cortex; ATN, anterior thalamic nucleus, which relays the output to the cingulate motor area, shown to control reward-driven movements. Another possibility is to control the animal by output through deep layers of Entorhinal Cortex (EC). *Split arrowheads* represent diffuse projections. The model includes 1,333 neurons with 2,180 compartments. Reprinted with permission from Gorchetchnikov and Hasselmo 2005. (**b**) The *Navigation* Project Environment allows setting up environments where simulated agents can navigate and obtain rewards (**c**). The trajectory of the simulated animal in the open field simulations. *S* designates the starting location of the simulated animal and *G* stands for goal location with the size of letter representing salience. (*I*) Simulation set 1: selection of the closest goal. The initial vertical segment does not show a strong preference towards the closer goal to the left. When the difference in distances towards both goals gets big enough, the trajectory starts to bear left. (*II*) Simulation set 2: selection of the most salient goal. Initial bearing to the left does not show a strong preference towards a more salient goal. When the difference in distances towards both goals becomes big enough to affect the behaviour, the trajectory bends further left. (*III*) Results for the simulation of a single salient goal versus multiple less salient goals. Reprinted with permission from Gorchetchnikov and Hasselmo 2005

The *Network Editor* uses a tree-like representation of the network; see Fig. 3. The *Add Population* Button will open the *Population Editor*, where the user can manually set the size of the population. Alternatively, the size of the population can be set to depend on the size of the input, which would allow the user to run the same network with different input structures.

To facilitate creating large-scale neural networks, KInNeSS can clone whole populations of cells with their respective input and output projections. The user manual provides additional information on how the connectivity is managed when populations are cloned.

## XML Support

KInNeSS stores the primary representation of a network as a single XML (Extensible Markup Language; Bray et al. 2006) file. Synaptic weights, conduction delays and input stimuli may also be represented in separate XML files. The advantages of using a declarative format such as XML are discussed in Goddard et al. (2001).

KInNeSS supports XML for both import and export of neural network architectures; see Fig. 4. KInNeSS is used both to load and save a network from and to its description in a XML file, respectively. This same XML file is directly

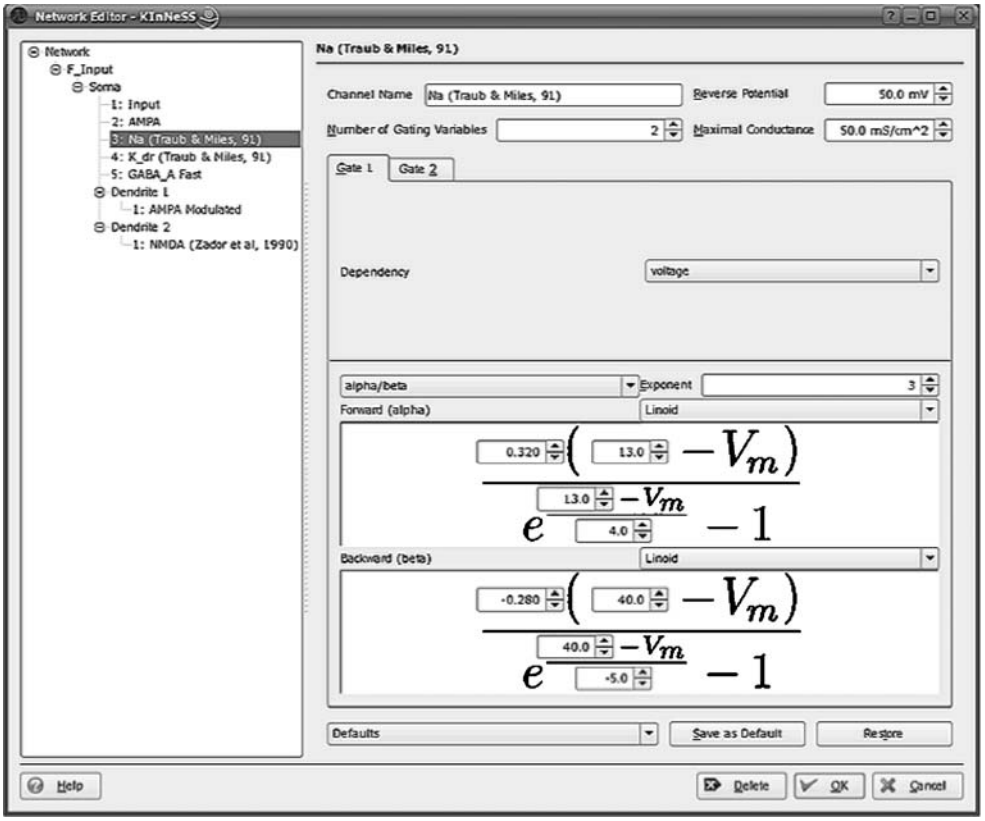



**Fig. 3** The *Network Editor* is one of the key features that enable KInNeSS users to explore different modeling choices including creation of networks (**a**), populations (**b**), compartments (**c**), and channels (**d**) in an easy to use interface

accessed by the KBrain plug-in which interfaces with SANNDRA; see Fig. 4b. Thus, once a network representation in XML is available, it is possible to bypass the KInNeSS GUI and access the network directly from KBrain. This might be useful, for example, to run batch simulations of the same network with different parameters, a feature currently under development.

The XML network files follow an XML schema. This schema is similar to NeuroML standards for such aspects as network topology (NetworkML), biophysical cell properties (ChannelML and Biophysics) and cell shape (MorphML). Figure 4 highlights some of the differences between the KInNeSS (a) and NeuroML (c) schemas. Both formats encode populations of neurons and projections between them. KInNeSS network files are laid out in a purely hierarchical form where each population contains the set of projections it receives. NeuroML files, on the other hand, encode the sets of populations and projections as siblings. Also, the population's cell characteristics in KInNeSS are described in the *OrientedSubStructure* element where it is a child of that population element. These are separately described in NeuroML in the *Cells* element.

Since networks are implemented as XML files, it is in principle possible to translate at least some network architectures from other simulation packages to a format admissible by KInNeSS (via XSL transformations[9]). However, XML standards are not quite sufficient for full interoperability between simulators because they require an additional layer between the standard and the user (Cannon et al. 2007). NeuroML represents a key step in standardizing this layer. Planned KInNeSS improvements include altering the network XML schemas to render it fully compatible with the latest NeuroML version.

## Spiking Neurons in KInNeSS: Basic Modeling Principles

KInNeSS relies on the compartmental modeling approach in which a neuron is implemented as a set of compartments representing sections of the neuron's dendritic tree and soma (Bower and Beeman 1998), where each compartment is in turn implemented as an equivalent electrical circuit (Rall 1964). KInNeSS requires knowledge of the different elements of computational modeling and how to exploit them. This section provides a concise description of the neural formalism adopted in KInNeSS.

Neurons are implemented as branched multi-compartmental structures. The first compartment always defaults to the cell soma and its activity can be visualized during simulations. Up to four dendritic branches can be added to the soma and each dendritic compartment can have up to two child branches. Whereas dendritic compartments are optional, it is necessary to have a somatic compartment.

[9] http://www.w3.org/TR/xslt.html

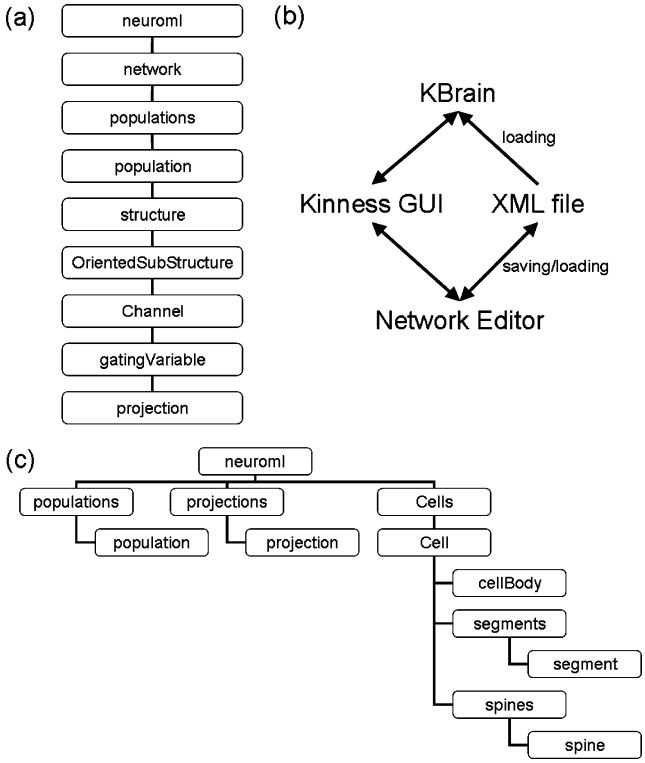



**Fig. 4** Relationship between KInNeSS, SANNDRA, and the XML file. (**a**) In KInNeSS, populations, cell characteristics and projections are stored as part of a hierarchical subtree. (**b**) KInNeSS accesses and saves the network and related structures in XML format. Network instantiation is then carried out via the KBrain plug-in which interfaces with SANNDRA. (**c**) In NeuroML, populations, cell characteristics, and projections are stored as separate subtrees

Compartmental dimensions are set in the *Network Editor* interface. When more than one compartment is introduced, an additional parameter must be assigned to determine the resistances between neighboring compartments.

The compartmental membrane potential $V$ (mV) is based on the following Equation[10]:

$$C_m \frac{dV}{dt} = -\sum_i I_i, \tag{1}$$

where $C_m$ is the membrane capacitance (μF) and $I_i$ is the $i$-th current. For convenience, both sides of Equation 1 are divided by the area of the membrane $\pi dl$ and thus converted to:

$$C_M \frac{dV}{dt} = -\sum_i J_i, \tag{2}$$

where $C_M$ is the specific membrane capacitance (μF/cm$^2$) and $J_i$ is the density of the $i$-th current (μA/cm$^2$). While the area of the membrane $\pi dl$ factor is hidden within the conductance density in many of the currents described below, it appears explicitly in the intercompartmental current equations because the axial conductance is independent of the membrane area.

In KInNeSS, all currents are modeled according to Equation 3:

$$J_i = \overline{g}_i \prod_p g_{ip} \left( V - V_i^{Eq} \right), \tag{3}$$

where $\overline{g}_i$ stands for the maximum conductance density (ms/cm$^2$) of the current (or current channels) when all the gating variables $g_{ip}$ in that current are fully open ($\prod_p g_{ip} = 1$). The equilibrium potential of the current is denoted $V_i^{Eq}$ (mV). A dimensionless *gating variable* $g_{ip}$ refers to any of a number of model cell membrane mechanisms that may dynamically influence the conductance of a current density $J_i$.

Each compartment may receive an arbitrary number of currents in which the user can set through the *Add Channel* Button. Each current channel has four settings: name, number of gating variables, reverse or threshold potential (disabled for some currents), and maximal conductance (disabled for injected currents). These settings apply to all gating variables $g_{ij}$ acting on the relevant current channel. KInNeSS provides a set of default values for current channel parameters obtained from the literature, but a user may choose to customize these parameters.

The existence of the various types of gating variables $g_{ij}$ gives the user freedom in defining currents with Equation 3. The user can select the number and nature of the gating variables. KInNeSS requires at least one gating variable per current but allows no more than three. All gating variables can be raised to arbitrary integer power. For example, the typical $Na^+$ current as defined by (Hodgkin and Huxley 1952):

$$I_{Na} = \overline{g}_{Na} m^3 h \left( V - V_{Na}^{Eq} \right), \tag{4}$$

has a total of two gating variables ($m^3$ and $h$). Here $m$ is the fraction of the *activation gates* being open, and $h$ is the fraction of the *inactivation gates* being open.

Obviously, the gating variables in a single current definition modulate each other's effect. Table 1 lists the types of gating variables implemented in KInNeSS. Each gate is described in the following paragraphs.

## Injection and Input

The injection and input gating variables are used when a cell compartment is current-injected or voltage-driven, respectively, as is commonly done in experimental studies and modeling applications. These are used to provide external input stimuli to the neural network.

[10] The equations in "XML Support" section follow sign conventions found in the literature. Note that in the KInNeSS interface, this convention is reversed.

**Table 1** Gating variables

| Gating variable | Specific use in KInNeSS |
|---|---|
| Injection | Current injection |
| Input | Voltage input that drives compartment towards a certain voltage. In the case of infinite conductance it turns into a perfect voltage clamp. |
| Gap | Gap junctions |
| Voltage | Classical voltage gates: |
| | *Exponential* |
| | *Sigmoid* |
| | *Linoid*, |
| | Generalized version that incorporates all three of the above: |
| | *Parameterized* |
| | And two gates for $g^{\infty}/\tau$ form of representation |
| | *Simple tau* |
| | *Thalamic reticular tau* |
| Ligand | Synaptic currents |
| Voltage block | Voltage dependent blocking of a channel (e.g. $Mg^{2+}$ for NMDA) |
| Modulation | Neuromodulatory effect on a current |
| AHP/ADP | After-hyperpolarization or after-depolarization current |
| Reduced | Pseudo current with quadratic integrate-and-fire to replace a set of Hodgkin–Huxley currents |

A channel is obtained by combining up to three of the gating variables listed here

There are four external input channel sources in KInNeSS, allowing the user to simultaneously apply four independent inputs to the network. Input channel sources are color-coded as Red, Green, Blue, and Yellow. The strength of each input source at a given location is noted $V_{color}$ in Equation 6 and must be within [0 255]. This strength is indicated by the luminance of the corresponding colored square in the *Generic* Project Environment[11]. The use of four color-coded sources as the external input makes it easy to import image files where each pixel is defined by a four (or less) dimensional vector.

In the case of an input gating variable, the current density (Equation 3) simplifies to:

$$J_i = \overline{g}_i\left(V - V_i^{Eq}\right), \tag{5}$$

where the driving potential, $V_i^{Eq}$, is determined by the input stimulus as in:

$$V_i^{Eq} = V^* + s_{red}^i V_{red} + s_{green}^i V_{green} + s_{blue}^i V_{blue} + s_{alpha}^i V_{alpha}, \tag{6}$$

[11] When loading spatial input patterns from a .png file, the yellow channel represents the alpha channel.

where the sensitivity variables $s_{red}^i$, $s_{green}^i$, $s_{blue}^i$ and $s_{alpha}^i$ are constants set by the user. Voltage $V^*$ corresponds to the resting potential of the compartment. Thus, in the absence of any input it acts as an extra leakage. In the case of an injection gating variable, the current density reduces to:

$$J_i = s_{red}^i J_{red} + s_{green}^i J_{green} + s_{blue}^i J_{blue} + s_{alpha}^i J_{alpha}. \tag{7}$$

where the strength of each injection source is denoted $J_{color}$ and must be within [0 255]. Note the absence of a maximum conductance factor ($\overline{g}_i$) in Equation 7.

Equations 6 and 7 show that the magnitude of contribution of a given input source is determined by its associated sensitivity setting. For example, in the case of an injection gate, if the value of an input source were 122 units and its associated sensitivity $s_{color}^i$=10 pA/cm$^2$, then the resulting contributed current would be 1.22 nA/cm$^2$. Given built-in lower and upper bounds on the sensitivity parameters, input currents in KInNeSS can range over multiple orders of magnitude.

## Inter-compartmental Currents

Gap junction gates and inter-compartmental currents can also be modeled with Equation 3 in which $V_i^{Eq}$ represents the membrane potential of the neighboring gap-connected or contiguous compartments. However, inter-compartmental gating variables are neither combined with other gating variables nor varied with time. Thus, the simplification used in Equation 5 also applies here.

For currents from compartment $k$+1 to $k$, the conductance is derived based on the Equation 3.3 in Segev and Burke (1998) which states that intercompartmental current $I_{k+1,k}$ is:

$$I_{k+1,k} = \frac{V_{k+1} - V_k}{(R_{k+1} + R_k)/2.} \tag{8a}$$

Converting axial resistances $R_k$ and $R_{k+1}$ to specific axial resistances $R_A$ using Equation 5.6 in Bower and Beeman (1998) yields:

$$I_{k+1,k} = \frac{\pi}{2R_A}\frac{V_{k+1} - V_k}{\left(\frac{l_{k+1}}{d_{k+1}^2} + \frac{l_k}{d_k^2}\right)}, \tag{8b}$$

and since Equation 1 for compartment $k$ was divided on both sides by $\pi d_k l_k$, the "conductance density" term $\overline{g}_i$ for the intercompartmental current in Equation 5 is given by:

$$\overline{g}_i = \frac{d_k}{2R_i^A l_k^2}\cdot\frac{1}{1 + \frac{l_{k+1}d_k^2}{l_k d_{k+1}^2}}, \tag{9}$$

where $d_k$ and $l_k$ stand for the diameter and length of a compartment $k$ and $R_i^A$ is the specific axial resistance along the branch, defined for each pair of connected compart-

ments. The values of $d_k$, $l_k$ and $R_i^A$ are set by the user in the *Compartment Editor*. Although $I_{k+1,k}$ is symmetric, its effect on the voltage in Equation 2 will be different for compartments $k+1$ and $k$, depending on their respective sizes. The ratio in Equation 9 places a soft upper bound on the number of compartments with realistic parameters that can be used to represent a dendritic segment. If more compartments are desired, less realistic parameters which seek to decrease the ratio in Equation 9, or a shorter integration step, will need to be used. However, this is not a major problem given that KInNeSS is designed for network simulations that do not involve such highly detailed representations of the dendritic structure. In the case of a branching intersection, each dendritic branch and the trunk are explicitly coupled to a branching point via current densities:

$$J_{B,k} = \frac{d_k}{2R_k^A l_k^2}(V_B - V_k), \tag{10}$$

where $V_B$ is the potential of the branching point:

$$V_B = \frac{\sum_s \frac{d_s^2}{R_s^A l_s} V_s}{\sum_s \frac{d_s^2}{R_s^A l_s}}, \tag{11}$$

where summation is done over all compartments $s$ connected by the branching point (Fig. 5).

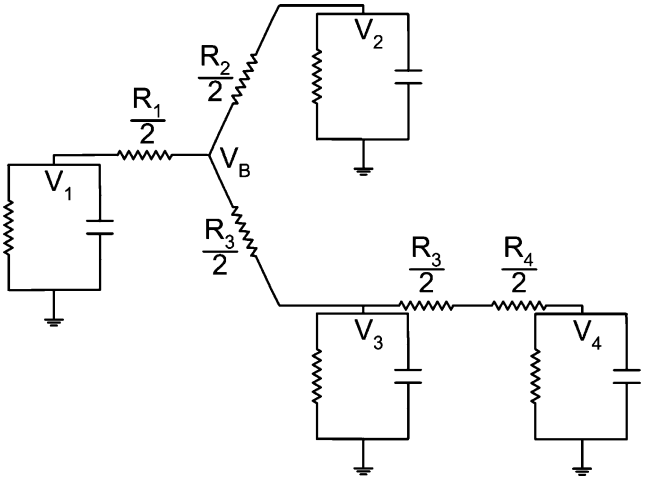



**Fig. 5** Equivalent circuit representation of branching neuron. $V_B$ is the potential at the branching point (Equation 11). Note that $\frac{R_k}{2} = \frac{2l_k R_k^A}{\pi d_k^2}$ corresponds to the actual resistance for branch $k$ in a branching intersection (Equation 8a). When two compartments are connected by a non-branching intersection (e.g. compartments $V_3$ and $V_4$), the specific axial resistance $R_k^A$ is the same for both compartments (i.e. $R_3^A = R_4^A$)

## Gap Junctions and Passive Leak Currents

Gap junction currents also follow Equation 5 with the gating variable calculated:

$$\overline{g}_i = \frac{g_i^J}{\pi d_k l_k}, \tag{12}$$

where $g_i^J$ is a conductance set in the interface and the denominator derives from the division of Equation 1 by the membrane area. Each compartment has one passive leak current, whose conductance density and reverse potential are manipulated in the *Compartment Editor*.

## Voltage-gated Channels

Voltage-gated channels are approximated using the Hodgkin–Huxley formalism (Hodgkin and Huxley 1952), which makes use of voltage-dependent rate variables ($\alpha(V)$ and $\beta(V)$). KInNeSS provides six function definitions for these rates, summarized in Table 2. The user also has the option to define the behavior of the voltage gates through $\tau_j(V)$ and $g_j^\infty$ instead of using the rates $\alpha(V)$ and $\beta(V)$. According to this notation, $\tau_j(V)$ represents the voltage-dependent activation time constant and $g_j^\infty$ is the voltage-dependent steady-state.

## Ligand-gated Channels

Ligand-gated channels are normally closed and open only when the neurotransmitter from the pre-synaptic cell binds to the receptor and opens the channel. The conductance change triggered by the pre-synaptic spike can cause either an excitatory post-synaptic potential (EPSP) or an inhibitory post-synaptic potential (IPSP). KInNeSS uses a dual exponential approximation of the conductance change (Bower and Beeman 1998). Let $t_s$ represent the arrival time of a pre-synaptic spike from unit $i$ to the synaptic terminal from $j$ to $i$. Then, according to the convention in Equation 2, the contribution of that spike to the conductance density $g_{ij}$ at time $t$ of ligand gating variable is defined as:

$$g_{ij}(t) = \begin{cases} \frac{p}{\tau_f - \tau_r}\left(e^{-\frac{t-t_s}{\tau_f}} - e^{-\frac{t-t_s}{\tau_r}}\right) & \text{if } \tau_f \neq \tau_r \\ \frac{t}{\tau_f} e^{1-\frac{t-t_s}{\tau_f}} & \text{otherwise} \end{cases}, \tag{13}$$

where $p$ is a normalizing constant, and $\tau_r$ and $\tau_f$ are manually set rise and fall time constants, respectively. The normalizing constant is governed by Equation 14:

$$\max\left(\frac{p}{\tau_f - \tau_r}\left(e^{-\frac{t-t_s}{\tau_f}} - e^{-\frac{t-t_s}{\tau_r}}\right)\right) = 1. \tag{14}$$

In practice $p$ ensures that $g_{ij}(t)$ can span the interval [0, 1] due to the action of a single pre-synaptic spike. In the case of a pre-synaptic spike train, Equation 13 is summed over the last two spikes using:

$$g_{ij}(t) = g_{ij1} + g_{ij2} - g_{ij1}g_{ij2}. \tag{15}$$

This ensures that both spikes have appropriate contribution to the conductance, but the total $g_{ij}(t)$ remains within

**Table 2** Six types of voltage dependence

| Dependency type | Equation form ($\alpha(V)$=or $\beta(V)$=or $\tau(V)$=) |
|---|---|
| Exponential | $Ae^{\left(\frac{B-V}{C}\right)}$ |
| Sigmoid | $\frac{A}{e^{\left(\frac{B-V}{C}\right)}+1}$ |
| Linoid | $\frac{A(B-V)}{e^{\left(\frac{B-V}{D}\right)}-1}$ |
| Parameterized | $\frac{A+BV}{C+De^{\left(\frac{V+E}{F}\right)}}$ |
| Simple tau | $A+B\cdot 10^{-2}\cdot e^{\left(C\cdot 10^{-2}\cdot V\right)}$ |
| Reticular tau | $A+Be^{\left(\frac{C+V}{D}\right)}+e^{\left(\frac{E+V}{F}\right)^{-1}}$ |

In Hodgkin and Huxley (1952) rate variables $\alpha(V)$ and $\beta(V)$ were fit to variants of the sigmoid, linoid, and exponential forms respectively. Letters $A$ to $F$ stand for parameters that can be set manually, and $V$ is the membrane voltage potential. These functions can be adjusted to account for many types of voltage-dependent conductances. The Exponential, Sigmoid and Linoid dependency types are adapted from Hodgkin and Huxley (1952)

[0, 1]. Note that these are two spikes of exactly the same presynaptic cell, and even if the cell fires with 100 Hz rate, the synapse tracks the last 20 ms. Almost all biological synapses have raise time constants that would bring conductance changes caused by these two spikes to domination over conductance changes caused by all previous spikes. Thus, SANNDRA does not keep track of a long series of spikes, which would significantly increase the computational load (Köhn and Wörgötter 1998). Taking into account the number of receptors $N_{ij}$ per membrane area, the resulting ligand-gated current density for synaptic current from $j$ to $i$ would be:

$$J_{ij} = \overline{g}_i g_{ij} N_{ij} z_j \left(V - V^{Eq}\right), \tag{16}$$

where $V^{Eq}$ is the reversal potential for the chemical channel (e.g. for connection via AMPA channels, $V_i^{Eq} \approx 0\,\text{mV}$) and $z_j$ is the amount of neurotransmitter available in the synapse $j$ at a given time. $z_j$ equals 1 unless specified by the user in the case of a habituative (or depressive) synapse (see below). The term $N_{ij}$ in the interface is set using synaptic weight setting. While $\overline{g}_i$ is constant across all pre-synaptic projections to cell $i$ for a particular synaptic current $J_i$, the weight coefficients $N_{ij}$ are specific for each cell $j$ which corresponds to the traditional weight kernel.

The neurotransmitter released by the pre-synaptic terminal can mediate, or scale, the conductance change triggered at the post-synaptic site (Grossberg 1980; Tsodyks and Markram 1997; Abbott et al. 1997). The accumulation and depletion (or habituation) of neurotransmitter $z_j$ at the synapse is described by:

$$\frac{dz_j}{dt} = \frac{\left(B - z_j\right)}{\tau} - \varepsilon\delta(t) z_j, \tag{17}$$

where $B$=1 is the target level of neurotransmitter at rest, $0<\varepsilon<1$ is the depletion coefficient that can scale the amount of neurotransmitter released at every spike, and $0.1<\tau<2{,}500$ is the recovery rate (in ms) regulating the rate of neurotransmitter accumulation. A spike $\delta(t)$ is defined for Hodgkin–Huxley spike generation as:

$$\delta(t) = \begin{cases} 1 & \text{if } V(t) < 0 \quad \text{and } V(t-\Delta t) > V_\theta \\ 0 & \text{otherwise} \end{cases}, \tag{18}$$

where $V(t)$ is the soma membrane voltage at time $t$, $V_\theta$ is the voltage threshold that is invariably crossed during spikes (−20 mV), $V(t-\Delta t)$ is the soma membrane voltage at time $t-\Delta t$ that precedes the soma voltage crossing 0 mV. In Equation 17, the neurotransmitter $z_j$ accumulates towards $B$=1 at a rate inversely proportional to the recovery rate $\tau$, and habituates (or depletes), by $-\varepsilon z_j$ every time a spike occurs. Neurotransmitter depletion allows the conductance change to be multiplicatively gated by the amount of neurotransmitter available, while still ensuring that $0<g_{ijj}<1$.

## Voltage Block

Certain synaptic channels are normally closed due to the presence of a blocker. This is the case for NMDA (N-Methyl-D-Aspartic Acid) receptor-channels which are blocked by magnesium ($Mg^{2+}$) ions. As the $Mg^{2+}$ ions are removed from the channel opening by increasing the cell's potential, they leave way for a slow, depolarizing calcium ($Ca^{2+}$) current (Zador et al. 1990). A slightly generalized model of this type of blocking mechanism is implemented in KInNeSS which takes into account the rate of binding of the blocker to the receptor site:

$$\textit{Rate of binding} = \frac{A}{\eta\cdot e^{\frac{C-V}{B}}+1}, \tag{19}$$

where $\eta$ is the concentration of the blocking ion in the extracellular medium. When the voltage blocked gating variable is combined with a typical synaptic gating variable, this results in a synaptic current of the form:

$$J_i(t) = z_i g_i N_i \frac{A\left(e^{\frac{-t}{\tau_f}} - e^{\frac{-t}{\tau_r}}\right)}{\eta\cdot e^{\left(\frac{C-V}{B}\right)}+1}\left(V - V^{Eq}\right), \tag{20}$$

where index $j$ from Equation 2 was dropped for simplicity. Parameters $A$, $B$ and $C$ in Equations 19 and 20 allow for better control over the binding mechanism of the blocker.

The formalism for the above-mentioned gates is fairly standard (Bower and Beeman 1998). In addition to these, KInNeSS incorporates a number of additional gating variables not readily found in most simulation packages. These include fast approximations of current modulation, after-hyperpolarization and after-depolarization currents (AHP/ADP), and a quadratic integrate-and-fire equation that can be used to replace a set of voltage gated channels.

### Current Modulation Gating Variable

The modulation gating variable can be used to model the effect of a neuromodulator on neuronal excitability, synaptic conduction, and synaptic modification (Hasselmo 1995). The conductance of this gating variable is defined by:

$$g_m(t) = 1 - g_{ij}(t), \tag{21}$$

where $g_{ij}(t)$ is driven by the activity of the modulating cell and is defined by Equation 15. The modulation gating variable can be used in conjunction with a ligand gating variable, which results in a neuromodulated synapse. An example is given by the AMPA (Amino-3-hydroxy-5-Methylisoxazole-4-Propionic Acid) current suppression with simultaneous learning enhancement mediated by medial septum theta-bound input in models of hippocampal area CA3 (Gorchetchnikov and Hasselmo 2005). If used by itself, this gating variable works as a leak channel that can be closed by modulation.

### Afterhyperpolarization (AHP)/Afterdepolarization (ADP) Gating Variable

Some potassium currents cause a smooth reduction in excitability and have been modeled using AHP currents (e.g. Prescott et al. 2006). The AHP/ADP gate implemented in KInNeSS allows for such control over a cell's membrane potential following a spike. Moreover, the gating variable has a manually set reverse potential. If this potential is greater than the cell's resting potential, the gating variable is an ADP; otherwise, it is an AHP. The Equations governing the AHP/ADP variable are the same that apply for ligand gating variables (Equations 13–15) except that the spikes that govern the gating variable are the ones of the same cell rather than from a pre-synaptic cell.

### Quadratic Integrate-and-fire Model

Ermentrout and Kopell (1986) reduced the Hodgkin–Huxley equations to a simpler one-variable equation in the quadratic Integrate-and-Fire model. The use of this model can speed up the simulation significantly when compared to the original Hodgkin–Huxley model, and its extended version has been found to be a good approximation for a variety of spiking patterns (Izhikevich 2004). The KInNeSS interface enables modulation of the spiking threshold and the source of modulation controlling the cell's excitability. Originally, the current is given by:

$$I = qV^2 - r, \tag{22}$$

where $q$ is a scaling constant and $r$ is a threshold. In order to make the model compatible with the current framework, it is instead implemented in terms of Equation 5 (for derivation see Gorchetchnikov and Hasselmo 2005):

$$J_i = \begin{cases} g_i(V^2 - V \cdot V_\theta) & \text{if } V_\theta \geq V_{rest} = 0 \\ g_i\left(V^2 - V \cdot V_\theta + \frac{V_\theta^2}{2}\right) & \text{otherwise} \end{cases}, \tag{23}$$

where $V_\theta$ is the spiking threshold, which is constant for non-modulated cell and given by:

$$V_\theta = \overline{V}_\theta - \left(\overline{V}_\theta - V_\theta^*\right) g_{ij}(t), \tag{24}$$

for a modulated cell, where $\overline{V}_\theta$ and $V_\theta^*$ are boundaries of threshold change and $g_{ij}(t)$ is defined by Equation 15.

### Connectivity

The user can choose the pre-synaptic cells (through their population) and the type of projection (*all to one*, *many to one*, and *one to one*) in the projection source subpanel of the user interface. The synaptic weight for the *many to one* type is calculated according to a Gaussian curve that is centered at the coordinates of the selected cell and has variances set by *Spread X* and *Spread Y* parameters. The *Border Effect* box allows to *Extend*, *Expand,* or *Wrap* connectivity between cells at the borders of the population. Synaptic connections can be fixed or plastic, with modifiable projections allowing the user to implement spike-timing-dependent plasticity (STDP) according to four variations of STDP learning: Hebbian, pre-synaptically, post-synaptically, or double-gated (Gorchetchnikov et al. 2005). It is also possible to specify axonal conduction delays for synaptic connections.

### Weight Modification and Learning Rules

STDP is supported by empirical evidence (Levy and Steward 1983; Markram et al. 1997; Bi and Poo 2001) and is a central focus in biophysically inspired neural network modeling (e.g. Kitajima and Hara 2000; Abarbanel et al. 2002; Gorchetchnikov et al. 2005; Grossberg and Versace 2008). STDP is defined as the synaptic changes that depend on the precise temporal relationship between the pre-synaptic and post-synaptic spikes. In the most common case, if the pre-synaptic spike precedes the post-synaptic spike, the synapse is potentiated. On the other hand, if the pre-synaptic spike follows the post-synaptic spike, the synapse is depressed.

KInNeSS implements a STDP-based learning rule (see Gorchetchnikov et al. 2005, for analytical derivation of the rule) that uses only local spatial and temporal information; the synaptic modification computations depend on the quantities present at the synapse being modified and at the current time step. In the case of spiking neurons, the temporal component of the pattern is the specific time difference between the presynaptic spike and the postsynaptic spike. This spike time

difference is reliably mapped in synaptic weight magnitude as described by the STDP rule:

$$\frac{dw}{dt} = \lambda\left(X_{pre}X_{post}\left(\hat{w} - \check{w}\right) + w_0 - w\right), \tag{25}$$

where $\lambda$ is a learning rate and $X_{pre}$, $X_{post}$ are the pre- and post-synaptic contributions to the weight change. The part $\left(\hat{w} - \check{w}\right) + w_0 - w$ is used to limit the admissible weight values to the interval $\left[\check{w}, \hat{w}\right]$. The variable $w_0$ is the baseline weight obtained when there is no correlation between pre-synaptic and post-synaptic cells. The learning rate $\lambda$ can be either held constant or dependent on the time since the last spike of a modulatory cell when learning is modulated:

$$\lambda = \overline{\lambda} g_{ij}(t), \tag{26}$$

where $g_{ij}(t)$ is defined by Equation 15. The post-synaptic component is also time dependent and defined by the following expression:

$$f_N\left(V_i^S\right) = \begin{cases} D+1 & \text{if } V_i^S \geq V_i^\theta \\ A(t-s) = D+1 & \text{if } s < t < s - \frac{1}{A} \\ C\left(t - s + \frac{1}{A}\right) + D & \text{if } s - \frac{1}{A} \leq t < s - \frac{D}{C} - \frac{1}{A} \\ 0 & \text{otherwise} \end{cases}, \tag{27}$$

where $A<0$, $C>0$, and $-1<D<0$ are parameters described by Gorchetchnikov et al. (2005). The user sets $\hat{w}$ and *Transition Time* through the interface. KInNeSS automatically sets $D$ as $\frac{-w_0}{\hat{w}}$, $C$ as $-0.04D$, $\check{w}=0$, and $A$ as a negative reciprocal of *Transition Time*. Equation 25 can be extended by adding various gating terms to limit the unbounded growth of the synaptic weights (Grossberg 1980):

$$\frac{dw}{dt} = \lambda\left(X_{pre}X_{post}W\right) \cdot f_G\left(X_{pre}X_{post}\right), \tag{28}$$

where $\left(\hat{w} - \check{w}\right) + w_0 - w$ is denoted $W$ for succinctness. The factor $f_G(X_{prepost})$ gates the learning signals which can be implemented using different equations to capture variations in gating (Gorchetchnikov et al. 2005; applied in Grossberg and Versace 2008; Berzhanskaya et al. 2007; Gorchetchnikov and Grossberg 2007). KInNeSS implements five such variations, as shown in Table 3.

**Table 3** Five types of gating functions

| Gating rule | $f_G\left(X_{pre}X_{post}\right)$ |
|---|---|
| No gate | Constant |
| Pre-synaptically | $X_{pre}$ |
| Post-synaptically | $X_{post}^2$ |
| Dual AND | $X_{pre}X_{post}^2$ |
| Dual OR | $X_{pre} + X_{post}^2$ |

## Data Analysis

KInNeSS outputs neuronal data in a variety of formats easily loaded into software packages, such as MATLAB®, for later analysis. When a detailed analysis of cell biophysics is needed, KInNeSS can store the instantaneous membrane potential[12] for all cells in any cell population of a given model (see Equation 1). If instead only a functional analysis of the network is desired, KInNeSS can store the spike times for all cells. The advantage of the latter approach is a significant reduction in the disk space necessary to store the results as well as in the time it takes to write these results to disk, which results in a consistent speedup in simulation time as shown in "KInNeSS Performance" section.

One distinguishing feature of KInNeSS is its ability to obtain local field potential (LFP) and current source density (CSD) data by simulating the presence of extracellular electrodes in the neural network (Grossberg and Versace 2008). Both types of measurements are stored in external files for analysis and both LFPs and CSDs can be directly visualized in KInNeSS; see Fig. 6:

The potential created by a combination of transmembrane currents of multiple neurons in a certain volume of tissue is referred to as the local field potential. The movement of ions across the cellular membrane causes the appearance of current sources (locations where current appears to flow out of the cells) and current sinks (locations where the current appears to flow into the cells). If $I_k^+$ and $I_j^-$ are the intensities of current source $k$ and of current sink $j$, respectively, and $r_k^+$ and $r_j^-$ are the distance of an extracellular electrode from the sink and source, respectively, then the potential recorded from the tip of an extracellular electrode is (Humphrey 1979):

$$V_e = \frac{1}{4\pi\sigma}\left(\sum_k \frac{I_k^+}{r_k^+} - \sum_j \frac{I_j^-}{r_j^-}\right). \tag{29}$$

Here $\sigma$ is the bulk conductivity of the extracellular medium, set in KInNeSS to a constant value of 15 ms/cm. The calculation of the various transmembrane currents $I_k^+$ and $I_j^-$ for each compartment is simplified by Kirchoff's law; the sum of inter-compartmental currents that do not flow across membrane is equal to the sum of transmem-

[12] The membrane potential outputted by KInNeSS is shifted by an amount equal to the leak potential of the compartment. For example, when the user sets the compartmental leakage reverse potential to -60 mV, a value of zero in the output file corresponds to a membrane potential of -60 mV.

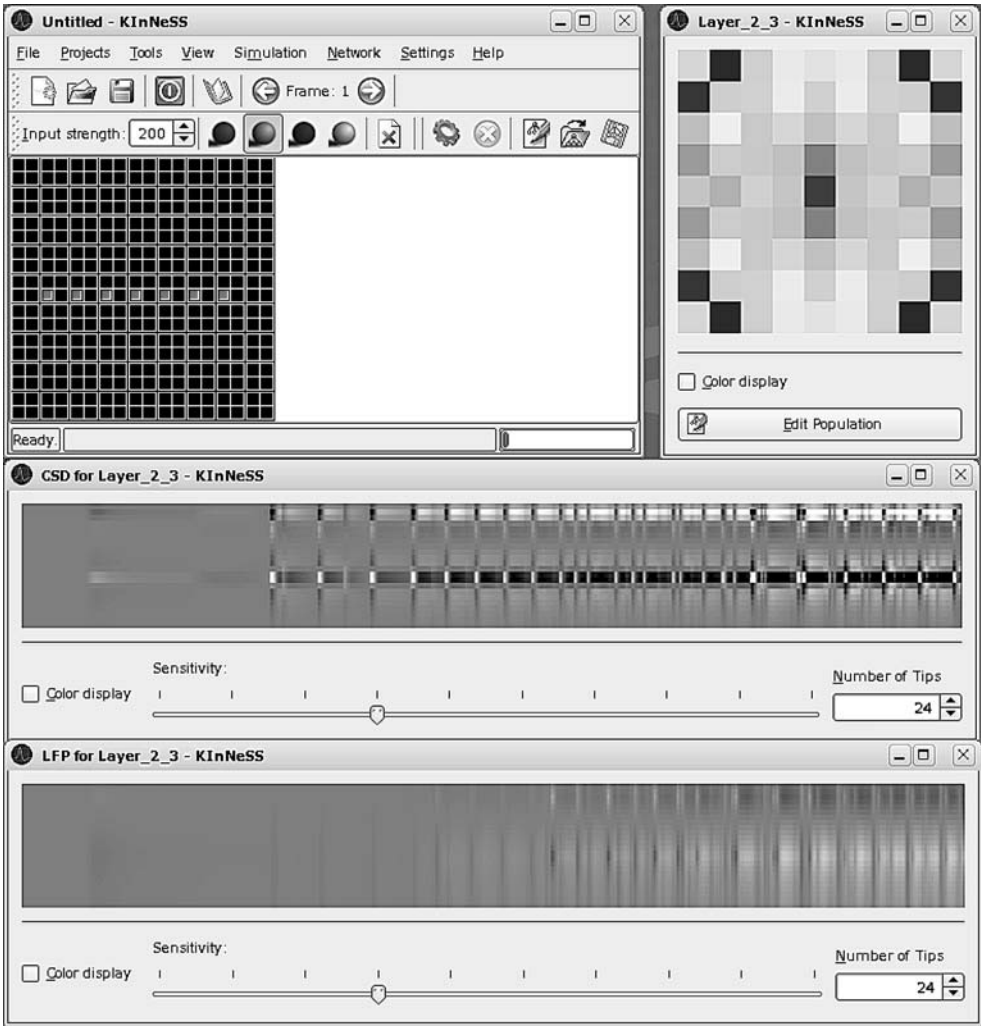



**Fig. 6** Local Field Potentials (LFP) and Current Source Densities (CSD) of 3-compartment layer 2/3 cells of a simulated laminar cortical circuit in a 1 s simulation (Grossberg and Versace 2008). The *display on top right* show the somatic membrane voltage of the 9×9 neural sheet, with *light gray* representing depolarized states and *dark gray* representing hyperpolarized states. The *bottom panels* plot the CSD and LFP measured with a 24-tip electrode of the selected central cell in the 9×9 sheet. *Lighter and darker gray* stands for source/sink and depolarization/hyperpolarization in CSD and LFP, respectively

brane currents. This is particularly convenient because inter-compartmental currents are easier to sum due to their limited number.

Each simulated electrode is composed of a number of electrode tips. Both the approximate placement of the electrode and the number of tips are determined by the user. KInNeSS allows for 2–51 electrode tips. The orientation of the electrode is always perpendicular to the population sheet of cells being monitored and parallel to each of the cells in a population. The distance from the cell at the location selected by the user and the electrode is determined probabilistically within the interval [10–200] μm. The distance of the electrode to all other cells in the population is similarly determined within the interval [10–1,000] μm. This merely emphasizes the contribution of the selected cell to the LFP. The spacing between the electrode tips is determined from the cell length and the number of tips. The first and last tips are positioned across from the ends of cells; intermediate tips are evenly spaced between them.

The CSD is calculated from the recorded extracellular potentials by approximating the second spatial derivative of the recorded voltage. Let $\Delta x$ be the distance between neighboring electrode tips, then:

$$CSD = \frac{V_{e+1} + V_{e-1} - 2V_e}{\Delta x}. \tag{30}$$

## Plug-in Development

KInNeSS users can benefit from various plug-ins which extend its basic functionality. Users with programming skills can create their own plug-ins and incorporate them into the framework. The plug-in toolsets allow for the addition of buttons and tabs in the menu or toolbar to give accessibility to global functions. The plug-in feature is ideal for generic commands or tools that are desired for multiple

models. An example use of the plug-in toolset is to add a feature of loading specialized scripts or a graphing toolset.

KInNeSS is a KDE[13] application that utilizes the KDE KParts functionality which allows creating and linking extensions to existing KParts software[14]. Developers can dynamically load components and actions to merge the plug-in graphical user interface (GUI) with the application's GUI in KParts-based applications. The KParts plug-in implementation process requires four basic steps: implementing a plug-in class, implementing a factory class, adding a GUI and adding the functionality. The plug-in is an extension of an appropriate KPart and cannot be run on its own. Using XML and a KDE defined class KActions, KDE automatically integrates the new plug-in with the host application to add or change the behavior.

Designing plug-ins for KInNeSS requires moderate to advance programming knowledge. The classes defined by KDE are well documented and can be implemented with a relatively small learning curve. The challenge of implementing plug-ins is due to the integration process between the plug-in and host program KInNeSS that makes use of a complex Makefile. A detailed explanation of how to implement a Makefile and plug-ins specific to KInNeSS can be found on the KInNeSS website[15]. Familiarity with Makefiles and object oriented programming is recommended for implementing a plug-in class.

## KInNeSS Performance

Many existing software solutions have been developed to match the growing demands of the modeling community. These different simulators and simulation environments are not interoperable and vary based on the types of simulation strategies and algorithms employed, the level of neural granularity, operating system, XML schemas, and analysis tools (Cannon et al. 2007). A comprehensive review of these different simulators is provided in Brette et al. (2007). Table 4 was reproduced based on Brette et al. (2007) and shows several key features where KInNeSS differentiates itself from the other simulators.

KInNeSS is the only software that currently allows for both XML import and export. Although KInNeSS does not strictly adhere to the NeuroML schema (however future development will focus on this issue), this is a key design feature of KInNeSS that will allow for easier testing and integration of model designs across various neural simulators. To the best of our knowledge, KInNeSS is one of the few software simulators, along with Catacomb2 (Cannon et al. 2002), when compared to the ones evaluated by Brette et al. (2007), that allows a simulated agent to be controlled by a neural model (Gorchetchnikov and Hasselmo 2002, 2005). This is important for simulations that aim to link behavioral data with neural modeling.

In order to take a closer look at the performance of KInNeSS with respect to these software packages, one of the benchmarks listed in the appendix of Brette et al. (2007) was simulated using KInNeSS. These simulations were ran on a 2× AMD Opteron™ Processor 248, 2.2 GHz, cache size 1024 making use of only one of the two cores and 2 GB of RAM running SuSE Linux 10 with custom 2.6.18.1 kernel, gcc 4.0.2 20050901 (pre-release) and KDE 3.5.7. These simulations were run on a network of 3,200 excitatory and 800 inhibitory neurons. The excitatory to inhibitory kernel was 6×6 and the inhibitory to excitatory kernel was 8×8. As a result of applying these kernels, a total of 100 projections were created for each excitatory cell and thus 320,000 total projections were in the network. Thus, 2% of the possible 16 million projections were simulated. An injection current was provided to the excitatory cells to drive the network. This current was randomly generated but remained fixed through the simulation time and across simulations. The simulations were run for 500 ms of simulation time. Code and data from the benchmark is found online[16].

Benchmark 3 in Brette et al. (2007) was described as a conductance-based Hodgkin–Huxley network. Consistent with Brette et al. (2007), this network implementation was based on voltage gate equations (cf. Table 2 and "Voltage-gated Channels" section) and parameters from Traub and Miles (1991). Results of KInNeSS simulations performed on this benchmark are depicted in Fig. 7c. The running time for 500 ms of simulated time was 392 s. When the spike and voltage outputs of all cells with 20 kHz frequency were saved onto the disk, the running time increased to 496 s. When Benchmark 3 was implemented using NEURON, the simulation time was 234 s. Brette et al. (2007) reported a runtime of 256 s for 1 s of simulated time on a single CPU of a Beowulf cluster consisting of six nodes, dual CPU, 64-bit 3.2 GHz Intel Xeon with 1024 KB cache. In Brette et al. (2007), five simulators, NEURON, GENESIS, NEST, CSIM, and SPLIT measured performance on Benchmark 3. Both GENESIS and CSIM did not report the running time of the simulations. NEST reported 125 s running time

[13] http://www.kde.org

[14] KDE always list native KDE classes with K in the front. All names that start with K and are followed by a capital letter are inherited from KDE native classes.

[15] http://kinness.net/

[16] See either ModelDB (https://senselab.med.yale.edu/modeldb/ShowModel.asp?model=113939) or the documentation section of http://www.kinness.net for the code and data.

**Table 4** Feature based comparison of neural network software simulators and simulation environments, adapted from Brette et al. (2007) *Yes* implies that it is either a built-in feature or it can easily be implemented with a few minutes of programming

| | KInNeSS | NEURON | GENESIS | NEST | CSIM | SPLIT |
|---|---|---|---|---|---|---|
| Operating System | Linux Unix (KDE) | Windows Linux Mac-OS | Windows Linux Mac-OS | Windows Linux Mac-OS | Windows Linux | Linux |
| XML import | Yes | No[a] | No | No[a] | No | No |
| XML export | Yes | Yes | No[a] | No[a] | No | No |
| GUI | Yes | Yes | Yes | No, can use Matlab | No, can use Matlab | No |
| Simple Analysis | Yes | Yes | Yes | Yes | No | No |
| Complex analysis | No | Yes | Yes | No | No | No |
| Multi-threading | Yes | Yes | Yes | Yes | No[a] | Yes |
| Event-based | No | Yes | No | Yes | No | No |
| Clock-based | Yes | Yes | Yes | Yes | Yes | Yes |
| Hodgkin–Huxley model | Yes | Yes | Yes | Yes | Yes | Yes |
| Integrate-and-fire models | Quadratic Izhikevich[a] | Leaky Izhikevich | Izhikevich | Leaky Izhikevich | Leaky Izhikevich | No[a] |
| Cable equations | No[a] | Yes | Yes | No | No | Yes |
| Short-term plasticity | Yes | Yes | Yes | Yes | Yes | Yes |
| Long-term plasticity | Yes | Yes | Yes | Yes | Yes | No[a] |
| Conductance-based synaptic interactions | Yes | Yes | Yes | Yes | Yes | Yes |
| Behavioral environment | Yes | No | No | No | No | No |

*No* implies that this feature was unavailable at the time this paper was submitted for publication. [a] implies that the feature is planned to be developed for a future version. *XML import* means that model specifications can be inputted in an XML format. *XML export* means that the model specifications can be exported in an XML format. *Simple analysis* and *complex analysis* imply that the GUI includes tools for i.e. spike counts, correlations, etc, parameter fitting, FFT, matrix operations, etc, respectively. *Cable equations* means that the software can implement compartmental models with dendrites. *Short-term plasticity* includes modeling of facilitation and depression. *Long-term plasticity* includes modeling LTP, LTD, and STDP.

when spikes are suppressed by removing the initial stimulation and for a simulated time of 1 s. NEST used a Sun Fire V40z equipped with four dual core AMD Opteron 875 processors at 2.2 GHz and 32 Gb RAM running Ubuntu 6.06.1 LTS with kernel 2.6.15–26-amd64-server. SPLIT reported 386 s of running time for 5 s if simulated time on a 2 GHz Pentium M machine (Dell D810). Because each of these simulations makes use of significantly different hardware, computations being performed on differing numbers of CPU cores, and different or unknown simulated times, it is very difficult to compare the running time of these simulations.

Benchmark 1 in Brette et al. (2007) was described as a conductance-based leaky integrate-and-fire network based on Vogels and Abbott (2005) and was indirectly simulated with a different version of integrate and fire in KInNeSS. The simulation setup for these simulations was the same as in the simulations of Benchmark 3 and the code is also found online[17]. KInNeSS makes use of the quadratic integrate-and-fire model (Ermentrout and Kopell 1986) explained in "Quadratic Integrate-and-fire Model" section rather than the leaky integrate-and-fire model because it is a biophysically more realistic model. Leaky integrate-and-fire models rely on a threshold crossing for spikes to fire and thus this equation is not concerned with how fast this threshold is crossed. This means that these models only consider the dynamics of the system under threshold. On the other hand, the quadratic integrate-and-fire model uses a dynamical equation to determine when a spike would fire and thus these models rely also on super-threshold dynamics. This means that when the threshold is crossed slowly, the spike develops slowly and when the threshold is crossed quickly, the spike develops more quickly. Results of the KInNeSS simulations on the quadratic integrate-and-fire model are depicted in Figs. 7a,b.

The running time for 500 ms of simulated time was 252 s. When saving on disk of spike and voltage outputs of all cells with the 20 kHz frequency was included, the running time increased to 320 s. In Brette et al. (2007), three simulators, NEURON, NEST, and CSIM, measured performance on Benchmark 1. All three simulators made use of the more simplistic leaky integrate-and-fire model. NEURON reported approximately 256 s running time. NEST reported approximately 27 s of running time when spikes are suppressed by removing initial stimulation and for a simulated time of 1 s.

[17] See either ModelDB (https://senselab.med.yale.edu/modeldb/ShowModel.asp?model=113939) or the documentation section of http://www.kinness.net for the code and data.

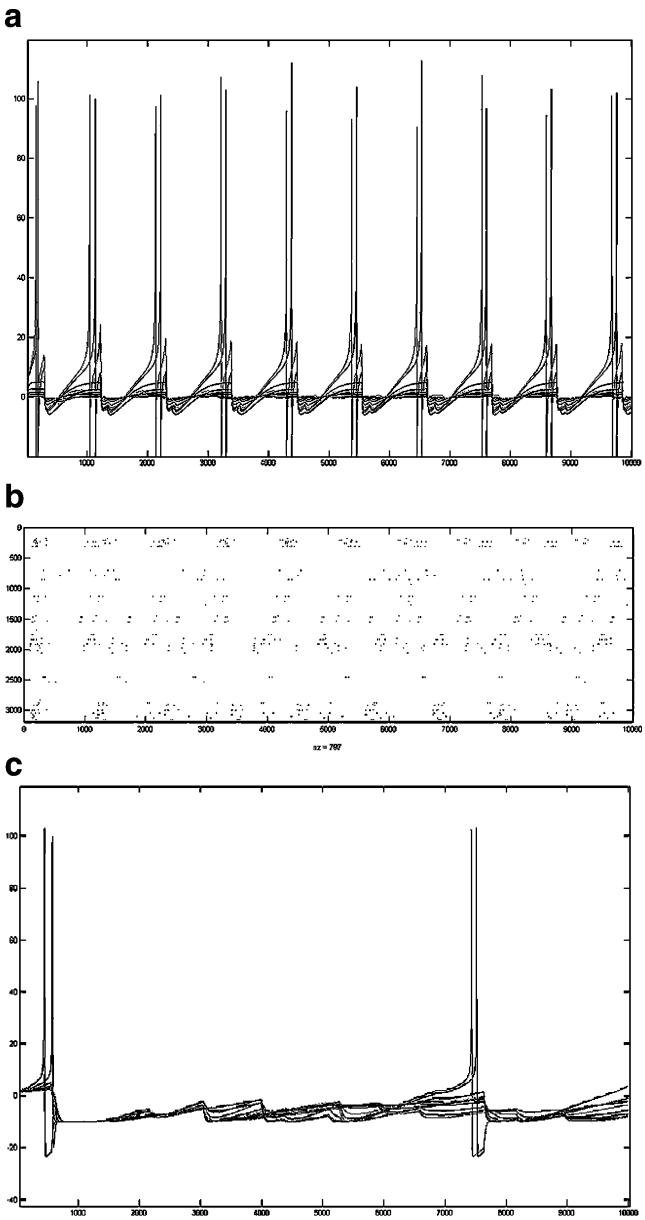



**Fig. 7** KInNeSS benchmark simulation results. **a** Soma membrane potential plot of excitatory neurons 230–240 in the quadratic integrate-and-fire (Benchmark 1) simulation for 500 ms of simulation time. **b** Spike raster of 3,200 excitatory neurons in the quadratic integrate-and-fire (Benchmark 1) simulation. **c** Soma membrane potential plot of excitatory neurons 230–240 in the Hodgkin–Huxley (Benchmark 3) simulation

Although KInNeSS performance and running time has been reported on one of the benchmarks from Brette et al. (2007) and an indirect comparison to a second benchmark, it is difficult to make a comparison to the other simulators (Cannon et al. 2007). A good comparison needs to take into account both running time of simulations as well precision (Rudolph and Destexhe 2007). Furthermore, as was pointed out by Brette et al. (2007), in order for this comparison to be undertaken, there must be a common language or framework for these simulation environments by making use of XML-based specifications. KInNeSS has taken a step in this direction by including both XML import and export functionality. In the future, this will be expanded by adapting XML schemas that more closely adhere to NeuroML.

## Future Directions

KInNeSS is an ongoing research effort. The developers are constantly updating the software and developing new modules. There are currently several projects aimed at improving the functionality of the software. KInNeSS supports the endeavor of creating a common framework across neural simulators. Therefore, KInNeSS will be changing its underlying XML schema to be compatible with the NeuroML schema. This mainly consists of changing the hierarchical layout to encode sets of populations and projections as siblings. In addition, future KInNeSS development will look at adding different integration techniques. A C++ adaptation of the open source version of the CVODE[18] method previously implemented in XPP is currently being tested. It was designed to handle stiff systems of differential equations and will remove the limitation on the number of compartments with realistic parameters.

The Izhikevich neuron model (Izhikevich 2003) will be added to the list of available currents. The inclusion of this model will allow for significant speed-up of simulations presently using Hodgkin–Huxley channels. Two other additions are also being considered for the future. First is the enhancement of online data visualization which will allow the user to plot compartments membrane voltage over time. At the present, this can only be done offline using third party software applications. Also, recent modeling work has used KInNeSS to analyze LFP/CSD data from the entire depth of a simulated 6-layered cortical structure (Grossberg and Versace 2008). Currently, the analysis and visualization of LFP/CSD data from neural structures with several populations at different depth is done in third party software (MATLAB®), but future plans include implementing this capability in KInNeSS.

Ongoing developments of KInNeSS will lead to new releases available for download. KInNeSS users are strongly encouraged to submit suggestions for future developments as well as submit changes to the code that can provide enhancements beneficial to the KInNeSS users' community.

[18] https://computation.llnl.gov/casc/sundials/main.html

## Conclusions

The KDE Integrated Neurosimulation Software (KInNeSS) environment contains a wide variety of software features that are needed by neurophysiologists, cognitive scientists and modelers in a neural simulation environment. The main focus of the software is to allow the modeler to relate fine-grained computational models to simulated behavior. This task is facilitated by an intuitive graphical user interface which speeds up both the initial learning process and continued application. Importantly, KInNeSS offers expandability by providing a base framework for software developers to create and incorporate plug-ins into the system and also allow for networks to be shared between applications by supporting both import and export of XML based model specifications.

KInNeSS allows the modeler to analyze the network dynamics at different levels of granularity, from single compartment and single cells to large-scale dynamics recorded in the form of local field potentials and current source densities. This enables the KInNeSS user to compare the simulation results to a growing repertoire of studies that make use of aggregate cell recordings to investigate the link between behavior, cognition, and brain activity. All these features combined make KInNeSS a useful tool for a multi-disciplinary software platform that would allow closer collaborations to emerge between researchers that gather biophysical data, and modelers that use these data to build computational models of animal and human behavior.

**Acknowledgements** This work was supported by the Center of Excellence for Learning in Education, Science and Technology (NSF SBE-0354378). Massimiliano Versace was supported in part by the Air Force Office of Scientific Research (AFOSR F49620-01-1-0397), the National Science Foundation (NSF SBE-0354378), and the Office of Naval Research (ONR N00014-01-1-0624). Heather Ames, and Jasmin Léveillé were supported in part by the National Science Foundation (NSF SBE-0354378) and the Office of Naval Research (ONR N00014-01-1-0624). Bret Fortenberry and Anatoli Gorchetchnikov were supported in part by the National Science Foundation (NSF SBE-0354378). The authors would also like to thank Prof. Steve Grossberg, Himanshu Mhatre, Prof. Mike Hasselmo, Dash Sai Gaddam, and Jesse Palma for numerous valuable discussions and suggestions for this paper.

## References

Abarbanel, H. D. I., Huerta, R., & Rabinovich, M. I. (2002). Dynamical model of long-term synaptic plasticity. *P. Natl. Acad. Sci.*, *99*, 10132–10137. doi:10.1073/pnas.132651299.

Abbott, L. F., Sen, K., Varela, J. A., & Nelson, S. B. (1997). Synaptic depression and cortical gain control. *Science*, *275*, 220–222. doi:10.1126/science.275.5297.221.

Berzhanskaya, J., Gorchetchnikov, A., & Schiff, S. J. (2007). Switching between gamma and theta: dynamic network control using subthreshold electric fields. *Neurocomputing*, *70*(10–12), 2091–2095. doi:10.1016/j.neucom.2006.10.124.

Bi, G. Q., & Poo, M. M. (2001). Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual Review of Neuroscience*, *24*, 139–166. doi:10.1146/annurev.neuro.24.1.139.

Bower, J. M., & Beeman, D. (1998). *The Book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System* (2nd ed.). New York: Springer.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2006). Extensible Markup Language (XML) 1.0 (Fourth Edition) http://www.w3.org/TR/2006/REC-xml-20060816/#sec-origin-goals

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., et al. (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *The Journal of Comparative Neurology*, *23*(3), 349–398.

Cannon, R. C., Hasselmo, M. E., & Koene, R. A. (2002). From biophysics to behavior: Catacomb2 and the design of biologically plausible models for spatial navigation. *Neuroinformatics*, *1*(1), 3–42. doi:10.1385/NI:1:1:003.

Cannon, R. C., Gewaltig, M. O., Gleeson, P., Bhalla, U. S., Hines, M. L., Howell, F. H., et al. (2007). Interoperability of neuroscience modeling software: current status and future directions. *Neuroinformatics*, *5*(2), 127–138. doi:10.1007/s12021-007-0004-5.

Carnevale, N. T., & Hines, M. L. (2006). *The neuron book*. Cambridge: Cambridge University Press.

Crook, S., Gleeson, P., Howell, F., Svitak, J., & Silver, R. A. (2007). MorphML: level 1 of the NeuroML standards for neuronal morphology data and model specification. *Neuroinformatics*, *5*(2), 96–104. doi:10.1007/s12021-007-0003-6.

Ermentrout, B. (2002). Simulating, analyzing, and animating dynamical systems: a guide to XPPAUT for researchers and students. SIAM.

Ermentrout, B., & Kopell, N. (1986). Parabolic bursting in an excitable system coupled with slow oscillation. *SIAM Journal on Applied Mathematics*, *46*, 233–252. doi:10.1137/0146017.

Gewaltig, M. O., & Diesmann, M. (2007). NEST. *Scholaropedia*, *2*(4), 1430.

Goddard, N. H., Hucka, M., Howell, F., Cornelis, H., Shankar, K., & Beeman, D. (2001). Towards NeuroML: model description methods for collaborative modeling in neuroscience. *Philos. T. Roy. Soc. B*, *356*, 1209–1228. doi:10.1098/rstb.2001.0910.

Gorchetchnikov, A. (2000). An approach to a biologically realistic simulation of natural memory, Master Thesis, Middle Tennessee State University, Murfreesboro, TN.

Gorchetchnikov, A., & Grossberg, S. (2007). Space, time and learning in the hippocampus: how fine spatial and temporal scales are expanded into population codes for behavioral control. *Neural Networks*, *20*(2), 182–193. doi:10.1016/j.neunet.2006.11.007.

Gorchetchnikov, A., & Hasselmo, M. E. (2002). A model of hippocampal circuitry mediating goal-driven navigation in a familiar environment. *Neurocomputing*, *44–46*, 424–427. doi:10.1016/S0925-2312(02)00395-8.

Gorchetchnikov, A., & Hasselmo, M. E. (2005). A biophysical implementation of a bidirectional graph search algorithm to solve multiple goal navigation tasks. *Connection Science*, *17*(1–2), 145–166. doi:10.1080/09540090500140925.

Gorchetchnikov, A., Versace, M., & Hasselmo, M. E. (2005). A model of STDP based on spatially and temporally local information: derivation and combination with gated decay. *Neural Networks*, *18*, 458–466. doi:10.1016/j.neunet.2005.06.019.

Grossberg, S. (1980). How does a brain build a cognitive code. *Psychological Review*, *87*, 1–51. doi:10.1037/0033-295X.87.1.1.

Grossberg, S., & Versace, M. (2008). Spikes, synchrony, and attentive learning by laminar thalamocortical circuits. *Brain Research*, *1218*, 278–312.

Hammarlund, P., & Ekeberg, Ö. (1998). Large neural network simulations on multiple hardware platforms. *The Journal of Comparative Neurology*, *5*, 443–459.

Hasselmo, M. E. (1995). Neuromodulation and cortical function: modeling the physiological basis of behavior. *Behavioural Brain Research*, *67*, 1–27. doi:10.1016/0166-4328(94)00113-T.

Hines, M. (1989). A program for simulation of nerve equations with branching geometries. *International Journal of Bio-Medical Computing*, *24*, 55–68. doi:10.1016/0020-7101(89)90007-X.

Hines, M. (1993). In F. Eeckman (Ed.), *NEURON: a program for simulation of nerve equations in Neural Systems: Analysis and Modeling* (pp. 127–136). Kluwer.

Hines, M., & Carnevale, N. T. (1994). Computer simulation methods for neurons. In M. Arbib (Ed.), *The handbook of brain theory and neural networks*. Cambridge: MIT Press.

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*, 500–544.

Humphrey, D. R. (1979). Extracellular, single-unit recording methods. In D. R. Humphrey (Ed.), *Electrophysiological techniques* pp. 199–259. Bethesda: Society for Neuroscience.

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, *14*, 1569–1572. doi:10.1109/TNN.2003.820440.

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons. *IEEE Transactions on Neural Networks*, *15*, 1063–1070. doi:10.1109/TNN.2004.832719.

Kitajima, T., & Hara, K. (2000). Generalized Hebbian rule for activity-dependent synaptic modifications. *Neural Networks*, *13*, 445–454. doi:10.1016/S0893-6080(00)00028-9.

Köhn, J., & Wörgötter, F. (1998). Employing the Z-transform to optimize the calculation of the synaptic conductance of NMDA and other synaptic channels in network simulations. *Neural Computation*, *10*, 1639–1651. doi:10.1162/089976698300017061.

Leveille, J., Grossberg, S., Mingolla, E., & Versace, M. (2008). Collinear facilitation and visual grouping in the spiking LAMINART model. Vision Science Society Abstract (accepted for VSS 2008) Naples, FL.

Levy, W. B., & Steward, O. (1983). Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus. *Neuroscience*, *8*(4), 791–797. doi:10.1016/0306-4522(83)90010-6.

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Computation*, *14* (11), 2531–2560. doi:10.1162/089976602760407955.

Markram, H., Lubke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, *275*, 213–215. doi:10.1126/science.275.5297.213.

Natschläger, T., Markram, H., & Maass, M. (2002). Computer models and analysis tools for neural microcircuits. In R. Kötter (Ed.), *A practical guide to neuroscience databases and associated tools, chapter 9*. Boston: Kluwer.

Prescott, S. A., Ratté, S., De Koninck, Y., & Sejnowski, T. J. (2006). Nonlinear interaction between shunting and adaptation controls a switch between integration and coincidence detection in pyramidal neurons. *The Journal of Neuroscience*, *26*, 9084–9097. doi:10.1523/JNEUROSCI.1388-06.2006.

Rall, W. (1964). Theoretical significance of dendritic trees for neuronal input–output relations. In R. F. Reiss (Ed.), *Neural theory and modeling* (pp. 73–97). Palo Alto: Stanford University Press.

Rudolph, M., & Destexhe, A. (2007). How much can we trust neural simulation strategies. *Neurocomputing*, *70*, 1966–1969. doi:10.1016/j.neucom.2006.10.138.

Segev, I., & Burke, R. E. (1998). Compartmental models of complex neurons. In *Methods in neuronal modeling. From ions to networks*. Cambridge: MIT Press.

Traub, R. D., & Miles, R. (1991). *Neuronal networks of the hippocampus*. Cambridge: Cambridge University Press.

Tsodyks, M., & Markram, H. (1997). The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability. *P. Natl. Acad. Sci.*, *94*, 719–723. doi:10.1073/pnas.94.2.719.

Vogels, T. P., & Abbott, L. F. (2005). Signal propagation and logic gating in networks of integrate-and-fire neurons. *The Journal of Neuroscience*, *25*, 10786–10795. doi:10.1523/JNEUROSCI.3508-05.2005.

Zador, A., Koch, C., & Brown, T. H. (1990). Biophysical model of a Hebbian synapse. *P. Natl. Acad. Sci.*, *87*, 6718–6722. doi:10.1073/pnas.87.17.6718.