Implementation of the Hodgkin-Huxley model in VRL-Studio

by Myra Huymayer

27.04.2014

1 The Hodgkin-Huxley model

The excitable cell membrane of a neuron can be modeled as a capacitor in parallel with an ionic current. In the squid giant axon the main ionic currents are the sodium and potassium current, yet there is also a weak contribution of other ionic currents (mainly Cl^-), which make up the leak current.
This results in the following formular:

$$C_m \frac{dV}{dt} = I_{ext} - (g_{Na}(V - E_{Na}) + g_K(V - E_K) + g_L(V - E_L)), \tag{1}$$

with C_m denoting the membrane capacitance per unit area $\left(\frac{\mu F}{square\ units}\right)$, I_{ext} $\left(\frac{nA}{square\ units}\right)$ an externally injected current, V (mV) the displacement of the membrane potential from its resting value, E_i (mV) the equilibrium potential and g_i $\left(\frac{mS}{square\ units}\right)$ the conductance of ion type i. The value of g_i at any given time is determined by multiplying the maximal conductance $(\overline{g}_i)$, defined as the conductance per unit area if all the channels of type i are open, with the fraction of channels in the open state (the so called gating variables).

The rectifying potassium current is modeled by a persistent/noninactivating conductance, because it contains only one type of gating mechanism (described by the gating variable n), divided into 4 subgates. Hence g_K can be written as:

$$g_K = \overline{g}_K n^4 \tag{2}$$

Sodium channels only open transiently when the membrane potential is depolarized because they are gated by two processes with opposite voltage dependences, the activation (m) and inactivation (h) variables. The activation variable m increases due to depolarization, while h increases due to hyperpolarization. Both gates must be open for the channel to conduct, leading to:

$$g_{Na} = \overline{g}_{Na} m^3 h \tag{3}$$

Hence formular (1) changes into:

$$C_m \frac{dV}{dt} = I_{ext} - (\overline{g}_{Na} m^3 h(V - E_{Na}) + \overline{g}_K n^4 (V - E_K) + g_L(V - E_L)) \tag{4}$$

The gating variables n, m and h are ordinary differential equations (ODEs) of the following form

with $i = \{n, m, h\}$:

$$\frac{di}{dt} = \alpha_i(V)(1 - i) - \beta_i(V)i \tag{5}$$

$$\tau_i(V)\frac{di}{dt} = i_\infty(V) - i \tag{6}$$

where

$$\tau_i(V) = \frac{1}{\alpha_i(V) + \beta_i(V)} \tag{7}$$

and

$$i_\infty(V) = \frac{\alpha_i(V)}{\alpha_i(V) + \beta_i(V)} \tag{8}$$

$\alpha_i(V)$ denotes the opening rate, $\beta_i(V)$ the closing rate, $\tau_i(V)$ the time constant and $i_\infty(V)$ the limiting value. Equation (6) indicates that i approaches the limiting value $i_\infty(V)$ exponentially with time constant $\tau_i(V)$, for a fixed voltage V.
The key elements to calculate the gating variables are the opening and closing rate functions $\alpha_i(V)$ and $\beta_i(V)$, which are defined for all the gating variables below:

$$\alpha_n(V) = \frac{0.01(V + 55)}{1 - exp(-0.1(V + 55))} \tag{9}$$

$$\beta_n(V) = 0.125exp(-0.0125(V + 65)) \tag{10}$$

$$\alpha_m(V) = \frac{0.1(V + 40)}{1 - exp(-0.1(V + 40))} \tag{11}$$

$$\beta_m(V) = 4exp(-0.0556(V + 65)) \tag{12}$$

$$\alpha_h(V) = 0.07exp(-0.05(V + 65)) \tag{13}$$

$$\beta_h(V) = \frac{1}{1 + exp(-0.1(V + 35))} \tag{14}$$

We used the following parameters:

$$\overline{g}_{Na} = 1.2\frac{mS}{mm^2}, \qquad E_{Na} = 50mV$$

$$\overline{g}_K = 0.36\frac{mS}{mm^2}, \qquad E_K = -77mV$$

$$\overline{g}_L = 0.003\frac{mS}{mm^2}, \qquad E_L = -54.387mV$$

The parameters in our program differed from those in the original paper of Hodgkin and Huxley (1952), which also results in slightly different rate functions. Hodgkin and Huxley (1952) postulated that $\frac{dV}{dt} = 0$

at constant voltage V, while we declared $\frac{dV}{dt} = -65$ at constant voltage, which is a more realistic physiological value. For a more detailed description of the Hodgkin-Huxley-model see Keener and Sneyd (2010) and Dayan and Abbott (2001).

2 Solving ordinary differential equations

Many ordinary differential equations (ODEs) cannot be solved analytically, but have to be solved numerically. The latter normally do not have a unique solution and therefore they require an initial condition in order to find a particular solution. This is the so called initial value problem: given the differential equation $y'(t) = f(t, y(t))$ with $f : \Omega \subset \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$, where Ω is an open set of $\mathbb{R} \times \mathbb{R}^n$. A solution to the initial value problem is a function y that is a solution to the differential equation and satisfies $y(t_0) = y_0$.

Initial value problems can be solved with single-step methods (e.g. explicit Euler scheme). These single-step methods have the general form:

$$y_{k+1} = y_k + h\phi(x_k, y_k, y_{k+1}, h)$$

with h denoting the stepsize. The value of y_k is an approximation of the solution to the ODE at time t_k : $y_k \approx y(t_k)$.

In case of the Hodgkin-Huxley model, a system of ordinary differential equations has to be solved.

$$y := \begin{pmatrix} V \\ n \\ m \\ h \end{pmatrix} \tag{15}$$

$$f(t, y(t)) = \begin{pmatrix} I_{ext} - (1.2(V - 50) + 0.36(V + 77) + 0.003(V + 54.387)) \\ (n_\infty(V) - n)/\tau_n(V) \\ (m_\infty(V) - m)/\tau_m(V) \\ (h_\infty(V) - h)/\tau_h(V) \end{pmatrix} \tag{16}$$

with the initial values, given as the steady state at a membrane resting potential of $-65mV$:

$$y(0) = \begin{pmatrix} -65 \\ n_\infty(-65) \\ m_\infty(-65) \\ h_\infty(-65) \end{pmatrix} \tag{17}$$

For the VRL-Plugin we used the DormandPrince853Integrator, which is an embedded Runge-Kutta integrator of order 8 with stepsize control (and automatic step initialization). This solver was obtained from `http://commons.apache.org/proper/commons-math/download_math.cgi` and it is especially useful to solve arrays of ordinary differential equations.

3 The Visual Reflection Library (VRL) and VRL-Studio

VRL is based on the programming language Java and permits the declarative, fully automated creation of graphical user interfaces from Java objects. This is achieved by using the information accessible via the Java Reflection API[1]. The Reflection API represents the classes, interfaces and objects in the current Java virtual machine and is used when examination or modification of the runtime behavior of applications running in the Java virtual machine is required (Krüger and Stark, 2009, Green, 1998).

VRL-Studio is based on VRL and enables text-based and (to a certain degree) visual programming. Three types of visual components are provided by VRL: object, method and parameter representation. For the implementation of more complex projects in VRL-Studio, it is necessary to create a VRL-Plugin (this is a Java Library (.jar File)), which is written and compiled in Netbeans and later installed in VRL-Studio.

4 Setup of the Hodgkin-Huxley Plugin

The basic setup of the Plugin is shown in the class diagram in Figure 1.

VFunction2D, NFunction2D, MFunction2D and HFunction2D are subclasses of the VRL-class Function2D. They inherit the abstract method `run(Double x, Double y)`, which needs to be overriden by the differential equations described above. VFunction2D implements formular (4) in the `run(Double x, Double y)` method. NFunction2D, MFunction2D and HFunction2D all implement an adjusted differential equation as described in formular (6). Tau_i (see equation (7)) and i_∞ (see equation (8)) were employed as independent methods in N-, M- and HFunction2D, so they could be called later independently from the `run(Double x, Double y)` method. In VFunction2D the method `init(double gBarK, double eK, double gBarNa, double eNa, double gBarL, double eL, double cm)` was declared, which received a graphical user interface (GUI) and which enables the user to define the parameters of formular (4). However, default values were already added. Note that not all setter and getter methods are shown in the class diagram, but were naturally present in the program. They were omitted for the sake of lucidity.

The class IFunction enables a limited manipulation of the current-value, the initial time and final time of current-injection by the user. This is accomplished by the method `init(double i, double`

[1] API = Application programming interface

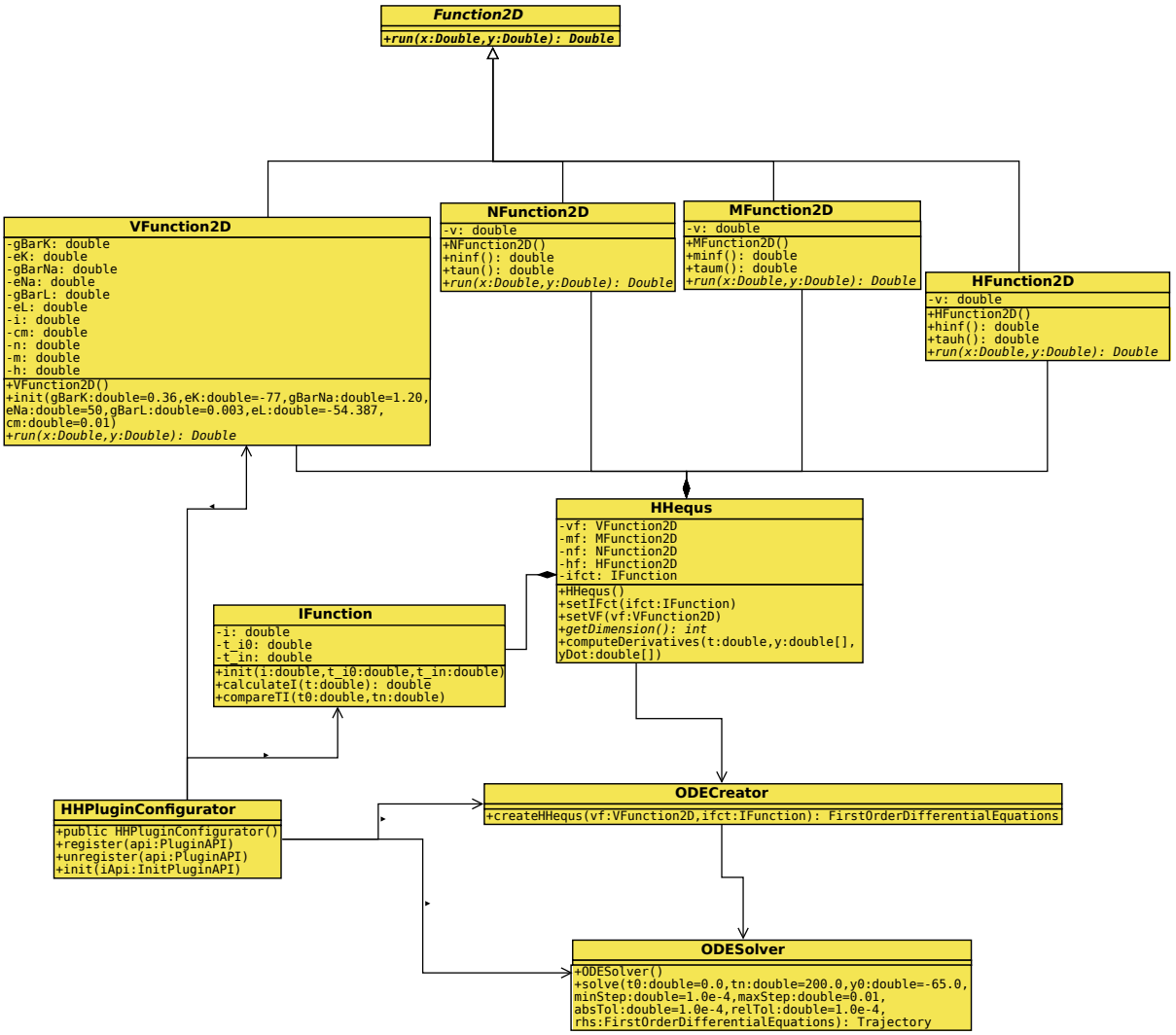

Figure 1: Class diagram of the Hodgkin-Huxley-Plugin. VFunction2D, NFunction2D, MFunction2D and HFunction2D are subclasses of Function2D and inherit all its methods

`t_i0, double t_in)`. Only this method receives a GUI, the other two methods are not to be seen in VRL-Studio. By throwing an exception `init(double i, double t_i0, double t_in)` guarantees that the initial time (t_{i0}) of current injection is not higher than the final time (t_{in}) of stimulation, by throwing an exception upon violation. The user is also not allowed to start/end current injection before starting/ending the simulation (t_0/t_n), which is also caught by exceptions in the method `compareTi(double t0, double tn)`. The current-value is set at the different time points in the simulation by the method `calculateI(double t)`.

Objects of IFunction, V-, N-, M- and HFunction2D are all generated in HHequs.java, which implements the interface FirstOrderDifferentialEquations[2]. Here, the derivatives are computed for the differ-

[2]for more details see `http://commons.apache.org/proper/commons-math/apidocs/org/apache/`

ential equations and the current i is set for each timestep. In HHequs.java the single ODEs are assembled in an array of first order differential equations. This array is returned by the ODECreator.

The core of the Hodgkin-Huxley Plugin is the ODESolver, where the DormandPrince853Integrator[3] is implemted, which solves the previosly defined ODEs. This integrator uses variable step sizes that are handled internally in order to control the integration error with respect to the specified accuracy. The ODESolver returns a Java Object containing an array of trajectories, which can be plotted by several Trajectory plotters. The trajectory plotter is an already existing VRL-Plugin.

Finally, the HHPluginConfigurator marks the program as a VRL-Plugin and defines the classes that receive a GUI in VRL-Studio.

5 Implementation of the Plugin in VRL-Studio

The execution of the Hodgkin-Huxley Plugin in VRL-Studio is displayed in Figure 2.

The classes VFunction2D, IFunction, ODECreator and ODESolver have a graphical user interface

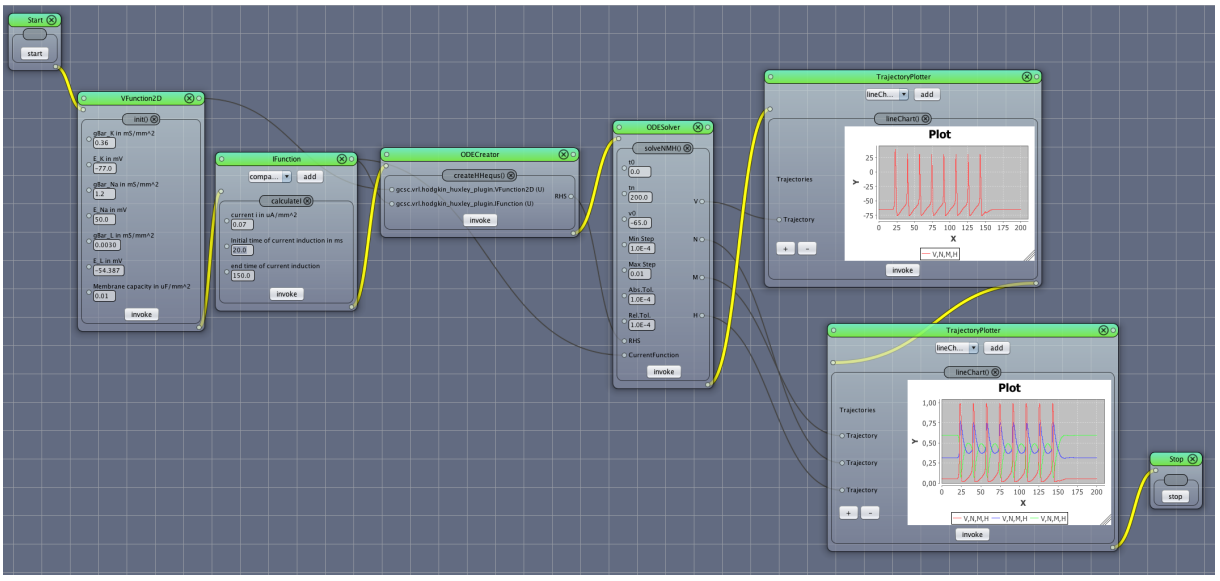

Figure 2: Implementation of the Hodgkin-Huxley Plugin in VRL-Studio at a current of 0.07 $\frac{\mu A}{mm^2}$

(GUI). All the components are connected via the control-flow (yellow lines in Figure 2). Some components contain methods that return values and pass them to another component via the data-flow (dark line in Figure 2), where the data undergo further processing.

commons/math3/ode/FirstOrderDifferentialEquations.html

[3]for further information see: http://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/ode/nonstiff/DormandPrince853Integrator.html

In VFunction2D the user can change the default-values of $\overline{g}_{Na}$, $\overline{g}_K$, $\overline{g}_L$, E_{Na}, E_K, E_L and C_m. In IFunction the user can influence the time when the current injection starts (t_{i0}) and ends (t_{in}) and the current amplitude (current i). The current injected into the modeled cell is a constant current. For the future, it might be interesting to extend this function for currents with complex functions.

User-input is also possible in ODESolver, where the initial value of the voltage (normally the resting potential with the default value = $-65mV$), the start (t_0) and end time (t_n) of the simulation, the maximal and minimal stepsize and the accuracy (absolute and relative tolerance) can be specified by the user.

The Trajectory plotter is an available VRL Plugin and was used to plot the voltage-function and the gating variable functions. In Figure 3 a single action potential is displayed. This was achieved by a short current injection of 0.07 $\frac{\mu A}{mm^2}$ ($t_{i0} = 2$, $t_{in} = 4$ ms) and the time course of the action potential and the gating variables looks very similar to figures in textbooks (compare figure 5.11 in Dayan and Abbott (2001) and figures 4.7 and 4.8 in Keener and Sneyd (2010)).

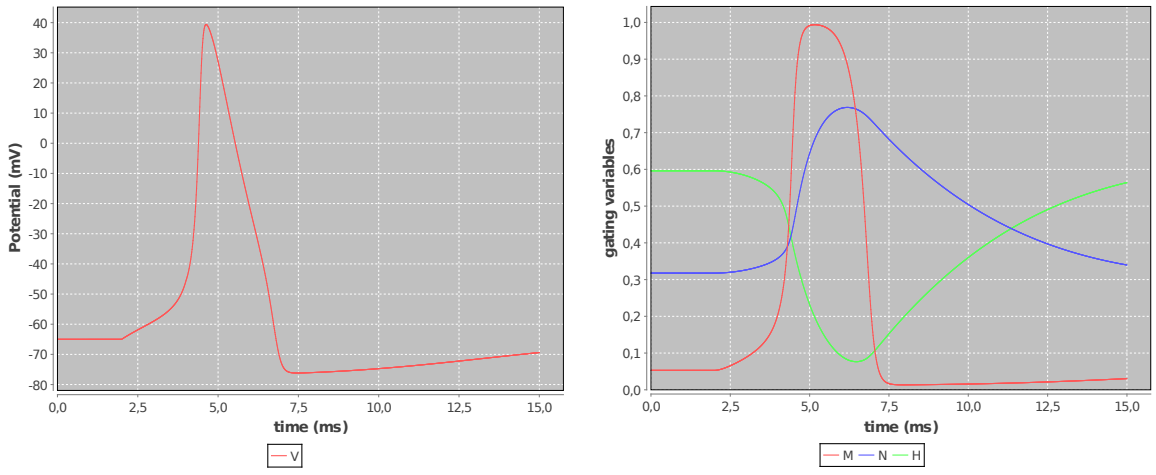

Figure 3: Short current injection (2 ms, 0.07 $\frac{\mu A}{mm^2}$) causing the generation of a single action potential.

For the visualization of a spike train, two test-measurements with different current values (0,07 $\frac{\mu A}{mm^2}$ and 0.7 $\frac{\mu A}{mm^2}$) are shown in Figure 4. As expected the higher current value causes a higher firing rate, yet only the first spike has the original amplitude, while the others have a lower amplitude. This can be explained by the incomplete recovery of the n and h gating variables, which require a long time to reach their original steady states, even overlapping with the interspike interval (ISI). Figures of the gating variables in textbooks show a similar time course. At a current of 0,07 $\frac{\mu A}{mm^2}$ there is enough time during the ISI for the gating variales to completely recover, resulting in a stable action potential amplitude.

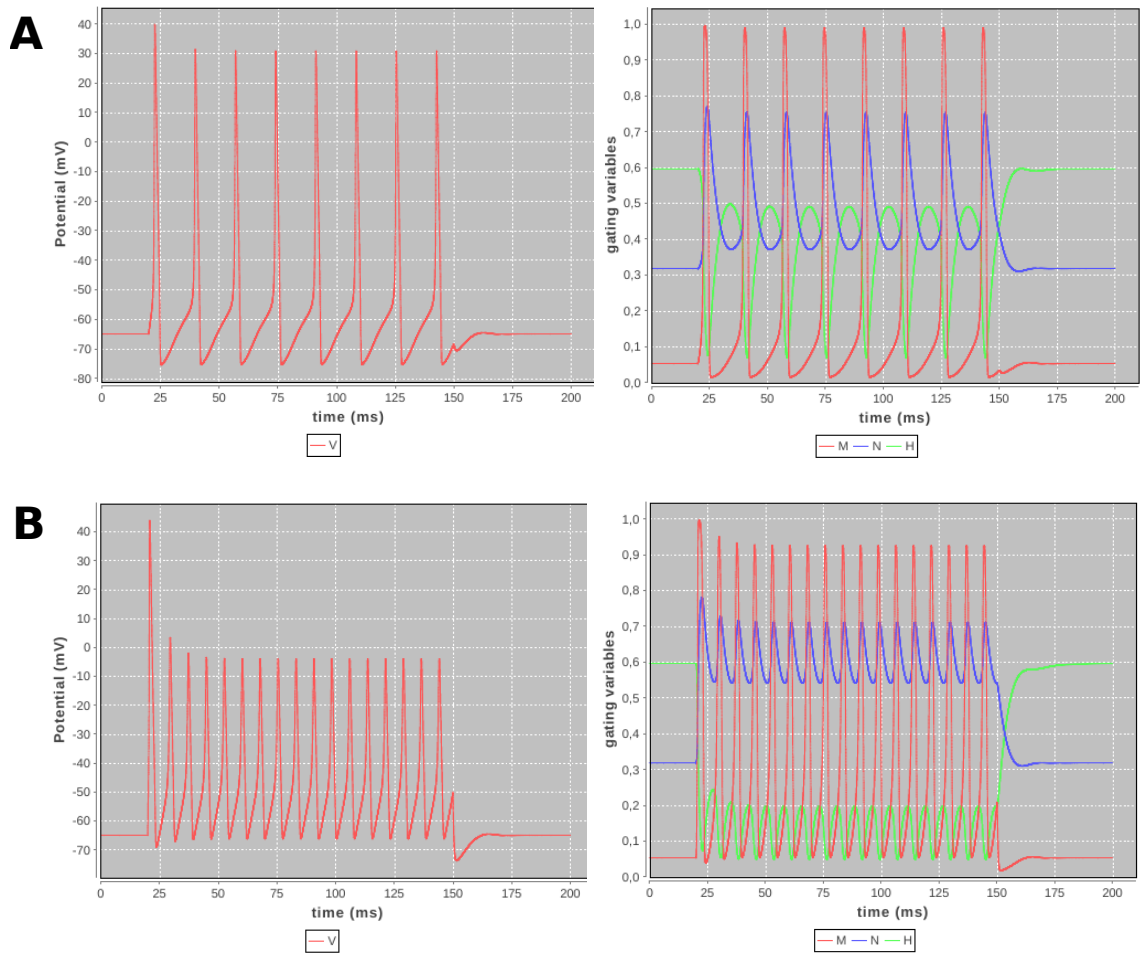

Figure 4: Two test current protocols. On the left side, the voltage calculated from formular (4), on the right side the time course of the gating variables is plotted. For both simulations current injection started 20 ms and ended 150 ms after simulation start.
A) A current of 0.07 $\frac{\mu A}{mm^2}$ was injected. **B)** Current: 0.7 $\frac{\mu A}{mm^2}$.

6 Outlook

As mentioned above, a constant current injection might not display natural stimulation and therefore more complex current courses would be required (e.g. sine-waves or short current-impulses). This could be established by several methods from which the user can choose in the VRL-Plugin. A more interactive solution to this problem could be that the user ”draws” a current with the mouse in a component window and the current is calculated internally. This would make the program even more user friendly.
Another problem is that the Hodgkin-Huxley Plugin described above is a single compartment model, which only changes in time. This does not display the realistic scenario in a neuron where also local information might be of importance. Extending this program for volume information, hence establishing a multi compartment model is also an aim for the future.

References

Peter Dayan and Laurence F Abbott. *Theoretical neuroscience*, volume 31. MIT press Cambridge, MA, 2001.

Dale Green. Trail: The reflection api. *The Java Tutorial Continued: The Rest of the JDK (TM). Addison-Wesley Pub Co*, 1998.

Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

James Keener and James Sneyd. *Mathematical Physiology: I: Cellular Physiology*, volume 1. Springer, 2010.

Guido Krüger and Thomas Stark. *Handbuch der Java-Programmierung*. Pearson Deutschland GmbH, 2009.