



An Understandable, Extensible, and Reusable Implementation of the Hodgkin-Huxley Equations Using Modelica

Christopher Schölzel^{1*}, Valeria Blesius¹, Gernot Ernst^{2,3} and Andreas Dominik¹

¹ Life Science Informatics, Technische Hochschule Mittelhessen - University of Applied Sciences, Gießen, Germany, ² Vestre Viken Hospital Trust, Kongsberg, Norway, ³ Psychological Institute, University of Oslo, Oslo, Norway

OPEN ACCESS

Edited by:

Sanjay Ram Kharche,
University of Western Ontario, Canada

Reviewed by:

Dominic G. Whittaker,
University of Nottingham,
United Kingdom
Bradley John Roth,
Oakland University, United States

*Correspondence:

Christopher Schölzel
christopher.schoelzel@mni.thm.de

Specialty section:

This article was submitted to
Computational Physiology and
Medicine,
a section of the journal
Frontiers in Physiology

Received: 14 July 2020

Accepted: 31 August 2020

Published: 02 October 2020

Citation:

Schölzel C, Blesius V, Ernst G and
Dominik A (2020) An Understandable,
Extensible, and Reusable
Implementation of the Hodgkin-Huxley
Equations Using Modelica.
Front. Physiol. 11:583203.
doi: 10.3389/fphys.2020.583203

The Hodgkin-Huxley model of the squid giant axon has been used for decades as the basis of many action potential models. These models are usually communicated using just a list of equations or a circuit diagram, which makes them unnecessarily complicated both for novices and for experts. We present a modular version of the Hodgkin-Huxley model that is more understandable than the usual monolithic implementations and that can be easily reused and extended. Our model is written in Modelica using software engineering concepts, such as object orientation and inheritance. It retains the electrical analogy, but names and explains individual components in biological terms. We use cognitive load theory to measure understandability as the amount of items that have to be kept in working memory simultaneously. The model is broken down into small self-contained components in human-readable code with extensive documentation. Additionally, it features a hybrid diagram that uses biological symbols in an electrical circuit and that is directly tied to the model code. The new model design avoids many redundancies and reduces the cognitive load associated with understanding the model by a factor of 6. Extensions can be easily applied due to an unifying interface and inheritance from shared base classes. The model can be used in an educational context as a more approachable introduction to mathematical modeling in electrophysiology. Additionally the modeling approach and the base components can be used to make complex Hodgkin-Huxley-type models more understandable and reusable.

Keywords: understandability, cognitive load theory, Modelica, mathematical modeling, software engineering, model engineering, Hodgkin-Huxley, action potential

1. INTRODUCTION

Since 1952, when Alan Hodgkin and Andrew Huxley published their conductance-based model of the action potential generation in the squid giant axon, the Hodgkin-Huxley (HH) model has been the basis of countless research projects to further the understanding of ionic currents and action potentials in neurons and cardiac myocytes (Hodgkin and Huxley, 1952). Today's models feature more types of ion channels and pumps than the three channels identified by Hodgkin and Huxley, but they still use the same electrical analogy and equation structure, which is why we will call them HH-type models in the following (Courtemanche et al., 1998; Inada et al., 2009; Fabbri et al., 2017; Bai et al., 2018). Although other types of ion channel models, such as Markov

Models, are emerging, HH-type models are still the gold standard (Winslow et al., 1999; Fink and Noble, 2009). The descriptions of these HH-type models usually follow one of three explanatory approaches: Either the differential equations are given directly with a short biological explanation of the major variables or a diagram of the electrical analogy is shown and explained in biological terms or a combination of both. Often a biological drawing of the cell is also provided, but it is only used to explain the modeled concepts and not tied to the model equations themselves. This holds for research articles (Courtemanche et al., 1998; Inada et al., 2009), simulation toolkits (Hines and Carnevale, 1997; Jordan et al., 2020) and textbooks (Voit, 2013; Gerstner et al., 2014). However, these approaches pose significant challenges for novices and limit the productivity of experts: A novice has to become familiar with the formalism of differential equations or with circuit diagrams at the same time as they try to understand the model itself. Experts will have overcome this barrier already, but they are also faced with much more complex HH-type models that can easily grow to over 100 equations. These equations all interact with each other in a multitude of feedback loops, making it extremely difficult to spot small errors or to reproduce and extend the model. The risk associated with these barriers is 2-fold: On the one hand, students may choose not to specialize in systems biology or electrophysiology, because they perceive the field to be too difficult. On the other hand, a published model that is described in this way may generate new insights, but prove to be too hard to reuse and extend. For example, the latter seems to be the case for a model by Inada et al. (2009) (116 equations) which has been labeled as “groundbreaking” (Noble et al., 2012) but has only been used for simulations by two other research groups in 10 years. It becomes apparent that there is room for improving the understandability of HH-type models and that this should be a goal of both the initial model design and its presentation in scientific and instructional material.

One area from which such an improvement may originate is software engineering, because software development faces similar problems of understandability: The building blocks of source code are easy to grasp, but creating and maintaining projects with millions of lines of code requires additional organization. A widely established solution to handle this complexity is modularization. Instead of overseeing the whole project at once, software engineers identify individual functions and parts of a system and create small modules to represent them. Each component only has a few lines of code and a limited number of connections to the outside world which makes it understandable. To form the whole system, the modules can be connected at a higher level of abstraction, where each of them can be considered to be a single entity.

In recent years, multiple researchers have advocated to borrow concepts from software engineering for systems biology, culminating in the formulation of the term “model engineering” (Hellerstein et al., 2019). In accordance with this movement, we found that the modeling language and consistent application of relevant language characteristics can have a significant impact on the model quality (Schölzel et al., 2020). In this paper we therefore present a novel modular implementation of the

original HH model that is based on the electrical analogy, but explains and visualizes each component in biological terms. The model is written in the modeling language Modelica and makes heavy use of the features of this language and of software engineering techniques.

Due to the aforementioned anticipated benefits of these techniques we pose the following research questions:

- RQ1 Can the understandability of the HH model be improved by a modular implementation that bridges the gap between biological meaning and electrical analogy?
 RQ2 Can a modular implementation of the HH model serve as a unifying basis for extensions and therefore facilitate the creation of more complex HH-type modules?

For the investigation of these questions, the term *understandability* is central, as our model does not differ from other solutions in terms of its output, but only its presentation. We may find our assessment of what makes a model more or less understandable intuitive, but in a scientific context it is not sufficient to rely on intuition alone. This is especially true, when it involves reasoning about the experiences of other people which may have quite a different background and perspective. Therefore, a model for understandability is needed that is based on scientific evidence. For this task we use cognitive load theory (CLT), a popular and well-validated theory in cognitive psychology which frames understandability in terms of the amount of items that have to be kept in active working memory and the degree of interactivity between them.

CLT is introduced in more detail in section 2.1 along with model engineering, the language Modelica, and the biological basis of the Hodgkin-Huxley model followed by an overview over the software engineering concepts that we apply and the resulting model structure. Section 2 describes our rationale for reducing cognitive load, for the component hierarchy, and the design of the individual model components. Section 3 then shows and explains the resulting model code including the graphical representation followed by a discussion of the implications of the new model structure for understandability and extensibility. Finally, section 4 sums up the answers to our research questions and discusses possible alternative approaches, limitations and future work.

2. MATERIALS AND METHODS

2.1. Background

2.1.1. Model Engineering

To this day, many models are still built for a single purpose without guidelines regarding code quality. However, when models grow beyond a certain point, the modeling process becomes an engineering task and the goal should not only be to produce a model that is working and mathematically sound, but also to build it with an architecture that facilitates the anticipated use cases and makes the code maintainable and accessible to other researchers (Hellerstein et al., 2019). This includes documentation, testing, naming of variables, and the use of established design patterns. In a previous work we have found the consistent use of an appropriate modeling language that is modular, descriptive, (human)-readable, open, graphical,

and hybrid (MoDROGH) to be a major driving factor for model quality in terms of understandability and reproducibility (Schölzel et al., 2020).

2.1.2. Modelica and Object-Oriented Programming

There are a few established and emerging languages that exhibit MoDROGH-characteristics, such as the systems biology markup language (SBML) (Hucka et al., 2003), CellML (Cuellar et al., 2003), Simscape (The MathWorks, Inc., 2020), or embedded domain-specific languages (DSLs) written in Python or Julia (Olivier et al., 2005; Lopez et al., 2013; Elmqvist et al., 2016; Rackauckas et al., 2020)¹. In our opinion, the modeling language Modelica (Mattsson and Elmqvist, 1997) is a particularly interesting example, because it is an industrial standard that emphasizes the engineering aspect of model design. In contrast to SBML, CellML, as well as Python- and Julia-based DSLs, Modelica supports full object oriented design (e.g., through model inheritance), discrete variables for the seamless integration of continuous and discrete model parts, the graphical composition of models via drag and drop, implicit differential/algebraic equations for acausal connections between components via conservation laws, cross-language export and import via the Functional Mockup Interface (Blochwitz et al., 2012), grouping of interface variables to connectors, and unrestricted mixing of implicit and explicit equation formats. It shares these features with Simscape, but unlike Simscape, Modelica provides an open environment, more flexible mechanisms for model inheritance including multiple inheritance and overwriting of variables and equations, and extensible annotations, which could, e.g., be used to implement support for ontological terms to the language. Weak points of Modelica are the lack of existing support for biological terminology and ontologies and the fact that, while the OpenModelica integrated development environment (IDE) (Fritzson et al., 2005) is open-source, many users prefer the proprietary IDE Dymola (Dassault Systèmes, 2020), which has its own compiler that is not always fully compatible with OpenModelica.

In Modelica, modularity is realized by the principles of object-oriented programming (Gamma et al., 1994). Code is structured in classes that can be instantiated to reuse the same code at different positions in a project and that can inherit behavior and interfaces from abstract base classes. This reuse is not only encouraged from one project to the following but also within one project. This corresponds with one of the guiding principles in software engineering called “don’t repeat yourself” (DRY) suggesting that one should avoid writing duplicated code with only very small differences, such as constant values or variable names (Hunt and Thomas, 2000). Both DRY and object orientation are most effective, when the implemented system can be broken down into structurally similar components, which is the case for the HH model.

¹For a comparison of different language candidates see the supplementary note in Schölzel et al. (2020).

2.1.3. The Biological Basis of the Hodgkin-Huxley Model

The Hodgkin-Huxley model explains the time course of the membrane potential of the squid giant axon during an action potential by means of three ion channels: A sodium channel lets Na⁺ cations enter the cell, which increases the potential. A delayed-rectifier potassium channel permits K⁺ cations to leave the cell, lowering the potential back toward the resting state. Finally, a leak channel is responsible for maintaining the resting potential while the other channels are closed. Both the sodium and the potassium channel have voltage-dependent gates—molecules that change their conformation with the membrane potential to activate or inactivate the channel. The sodium channel has both a fast activation gate and a slightly slower inactivation gate, allowing the channel to open for a short period of only a few milliseconds. The delayed-rectifier potassium channel only has an activation gate with slower kinetics while the leak channel is assumed to be always active.

2.1.4. Cognitive Load Theory

As mentioned in the introduction, we use cognitive load theory (CLT) as a model for understandability (Sweller, 2019). In short, CLT is based on the architecture of the human brain, which has a very limited capacity for new information in the working memory, but can easily transfer stored information from long-term memory to working memory. The amount of items that have to be kept in working memory to process an information is called the cognitive load. The main driving factor of this metric is element interactivity. Independent elements can be processed one by one, but when elements have high interactivity they have to be kept in working memory simultaneously. Hence, cognitive load can be reduced in two ways: First, expertise can allow a person that has understood a concept to further on process it as a single item instead of the several items it comprises. Second, a part of cognitive load does not originate from the complexity of the taught concept itself, but from the way it is presented and can therefore be reduced by choosing appropriate methods to present the information and instruct the learner.

2.2. Model Design and Structure

For the sake of simplicity, we will consider the amount of variables, parameters, and equations that constitute a model or model component as an indicator of its cognitive load and thus its understandability. Our goal is therefore to produce small model components that have a low amount of elements and to reduce element interactivity by introducing clearly defined interfaces between these components that allow the learner to view one component as a single item to be kept in working memory once they understand it.

For this task we identify the following components in the HH model: The lipid bilayer that separates the ionic concentrations and therefore electrical charge on the outside and the inside of the cell; the sodium, potassium, and leak channels; the voltage-dependent gates inside the sodium and potassium channels; and the current clamp that holds the current constant in order to measure the voltage with reference to a ground electrode. All of

these components reside on the same hierarchical layer, except for the gates which are components of an ion channel class.

To build the full model, two kinds of connections between non-gate components are required: First, each component has a positive electrical pin on the extracellular side and a negative electrical pin on the intracellular side, allowing current to flow through the component as positive outward current. Second, the voltage-dependent gating molecules of the sodium and potassium channels react more quickly at higher temperatures, which establishes the need for an input-output temperature connection from the lipid bilayer to the ion channels.

In the design of the Modelica model we follow our guidelines established in Schölzel et al. (2020), i.e., implementing small self-contained modules; describing only the “what” of the model, not the “how”; keeping the code human-readable with speaking names and additional documentation strings for variables and parameters; using only open-source tools (namely OpenModelica) for ease of reproduction; and adding a graphical representation for each component. We also published the full code of the model on GitHub as well as in the **Supplementary Data Sheet 1** including every information required to reproduce our results. As mentioned in section 2.1.2, graphical annotations are part of the Modelica code. This encompasses connection annotations that define the coordinates of the line connecting two components and more complex icon annotations that define a component icon as vector graphic. As the latter can be quite large and tend to clutter the otherwise human-readable code, we define all icons in separate classes and include them via inheritance with the `extends` statement.

The model was implemented using OpenModelica version 1.16.0 (Fritzson et al., 2005) and Mo|E version 0.6.3 (Justus et al., 2017) as well as Inkscape version 0.91 (Inkscape Developers, 2020) to add the component icons. Simulations can be replicated with OpenModelica on Windows, Linux, and macOS. The code is available on GitHub under the MIT license at <https://github.com/CSchoel/hh-modelica>.

3. RESULTS

3.1. Model Code

The first thing to consider in a software engineering task are the required interfaces. Hence, we start our implementation of the HH model by defining the following basic connectors:

```
connector TemperatureInput = input Real(unit="degC");
connector TemperatureOutput = output Real(unit="degC");
connector ElectricalPin
  flow Real i(unit="uA/cm2");
  Real v(unit="mV");
end ElectricalPin;
```

Here, `TemperatureInput` and `TemperatureOutput` follow a simple input-output relationship. All components that have a `TemperatureInput` will be connected to a single component with a `TemperatureOutput` that determines the global temperature value. `ElectricalPins` that are connected to each other will all have the same voltage v , but can have different currents i (indicated by the keyword `flow`). During compilation, Modelica will generate an equation following

Kirchhoff's current law that ensures that the sum of all connected currents equals zero. This allows to connect an arbitrary number of components without having to determine the direction of the flow. For the sake of terminology and for a visual distinction `PositivePin` and `NegativePin` are introduced as subclasses without any functional difference from the base class.

```
connector NegativePin
  extends ElectricalPin;
  annotation(...);
end NegativePin;
```

```
connector PositivePin
  extends ElectricalPin;
  annotation(...);
end PositivePin;
```

Annotation code that defines the connector icons is not given here for the sake of brevity and will be completely omitted for further code examples. The same is true for most of the documentation strings. These details can be viewed on GitHub and the resulting visual design can be seen in **Figure 1**.

Since most components will have an electrical connection both to the inside and the outside of the cell, it is beneficial to introduce another base class for those two-pin components:

```
partial model TwoPinComponent
  PositivePin p "positive extracellular pin";
  NegativePin n "negative intracellular pin";
  Real v(unit="mV") "potential difference between pins";
  Real i(unit="uA/cm2") "current flowing through comp.";
equation
  0 = p.i + n.i;
  v = p.v - n.v;
  i = p.i;
end TwoPinComponent;
```

This base class already introduces three small equations that connect the positive (extracellular) and negative (intracellular) pins. The first equation again follows Kirchhoff's current law to ensure that the sum of all currents entering and leaving the components is zero. The other equations just introduce two helper variables: The variable v can be used to measure (or define) the voltage at this component as a difference between the potential at the positive and the negative pin. The variable i measures or defines the current flowing through the component from the negative to the positive pin. The model is declared as `partial` since the number of equations and variables is not balanced. It does not yet specify the current-voltage relationship but leaves it open for implementation in specific subclasses.

The simplest `TwoPinComponent` that specifies this relationship is the `LipidBilayer`:

```
model LipidBilayer
  extends TwoPinComponent;
  extends HHmodelica.Icons.LipidBilayer;
  TemperatureOutput temp = temp_m;
  parameter Real temp_m(unit="degC") = 6.3 "temperature";
  parameter Real c(unit="uF/cm2") = 1 "capacitance";
  parameter Real v_init(unit="mV") = -90 "initial stim.";
initial equation
  v = v_init;
equation
  der(v) = 1000 * i/c;
end LipidBilayer;
```

This model inherits the variables and equations of `TwoPinComponent` and the graphical annotations from the icon `LipidBilayer`. It therefore only has to introduce one additional equation that describes the component as a capacitor, which separates the inside from the outside and is charged when a current is applied. A factor of 1,000 has to be introduced to measure the derivative $\text{der}(v)$ in millivolt instead of volt per second. Since the voltage v only enters the equation as a derivative, the initial value has to be determined. Hodgkin and Huxley used this degree of freedom to simulate a “short initial stimulation” of the Membrane, assuming that it has been kept at a constant $V = 0$ until the time $t = 0$ where the voltage is suddenly changed to $V = V_{\text{init}}$ and is then left to develop under a constant current. In this implementation this behavior is reflected by the parameter `v_init`. Additionally, the membrane temperature is defined through the parameter `temp_m` and propagated via the output connector `temp`.

The temperature is passed on to the Gates, which describe the conformation changes of gating molecules in an ion channel:

```
model Gate "molecule that opens/closes an ion channel"
  replaceable function fopen = expFit(sx=1, sy=1);
  replaceable function fclose = expFit(sx=1, sy=1);
  Real n(start=fopen(0)/(fopen(0) + fclose(0){}),
    fixed=true);
  input Real v(unit="mV");
  TemperatureInput temp;
protected
  Real phi = 3*((temp-6.3)/10);
equation
  der(n) = phi * (fopen(v) * (1 - n) - fclose(v) * n);
end Gate;
```

In contrast to the `LipidBilayer` the `Gate` does not inherit from `TwoPinComponent`, since it is not a component in the electric circuit itself but only a part of the `IonChannel`. Here, n is the gating variable that determines the ratio of gating molecules that are in “open” conformation. The rates with which molecules change formations depend on the current voltage v through the functions `fopen`, which gives the rate of change from closed to open, and `fclose`, which gives the rate of change from open to closed. Instead of having variables α and β that change with an equation as in the original formulation by Hodgkin and Huxley, `fopen` and `fclose` actually can be seen as variables that store the whole fitting functions. This allows us to keep the code DRY by reusing these functions to determine the starting value for n as the steady state that would be achieved by holding the membrane voltage constant as $V = 0$ mV. In the original model, a change in one of the fitting parameters for the equation for α would also require a change in the starting value for n which is not immediately transparent by the description. The functions `fopen` and `fclose` are explicitly declared as `replaceable` so that each ion channel can redefine them as required. For this the three fitting functions `expFit`, `logisticFit` and `goldmanFit` are required in the original HH model. For the sake of simplicity we will only discuss `expFit` here:

```
function expFit "exponential fitting function"
  input Real x "input value";
  input Real sx "scaling factor for x axis";
  input Real sy "scaling factor for y axis";
  output Real y "result";
```

```
algorithm
  y := sy * exp(sx * x);
end expFit;
```

In function definitions, Modelica switches from the usual declarative implementation style to an imperative style as in C or MATLAB. In an algorithm section, equations are variable assignments where an expression on the right-hand side is evaluated and stored in the variable on the left-hand side. This is also indicated by the assignment operator `:=` which has a direction in contrast to the equals sign used for normal equations. During compilation, an algorithm section is transformed to a single equation that depends on all input variables and determines the value of all output variables of the function definition. Here, a simple exponential relationship is defined between the main input x and the output y that can be scaled by the additional fitting parameters `sx` and `sy`. These fitting parameters are fixed to a constant value when the function is instantiated as, for example, function `fclose = expFit(sx=1, sy=1)`. The resulting function now has only x left as the single mandatory input and can therefore be called as `fclose(x)` for any real value x . The functions `logisticFit` and `goldmanFit` follow the same general structure, but realize different fitting functions with additional fitting parameters.

As mentioned before, the `Gate` component is part of an `IonChannel` component. As there are three different ion channels in the HH model, it again makes sense to introduce a common base class:

```
partial model IonChannel "ionic current through membrane"
  extends TwoPinComponent;
  extends HHmodelica.Icons.IonChannel;
  Real g(unit="mmho/cm2") "ion conductance";
  parameter Real v_eq(unit="mV") "equilibrium potential";
  parameter Real g_max(unit="mmho/cm2") "max conduct.";
equation
  i = g * (v - v_eq);
end IonChannel;
```

This component is a two-pin component and inherits a graphical annotation from an icon component. It adds the missing relationship between current and voltage by introducing a conductance variable g . If g is constant, the `IonChannel` behaves as a simple electrical conductor with the only exception that the voltage is relative to the equilibrium potential for the ions transported by this channel. This is true for the `LeakChannel` which only introduces the additional equation $g = g_{\text{max}}$. The sodium and potassium channels, however, have voltage- and temperature-dependent gates. Therefore, `GatedIonChannel` is introduced as another base class that is the same as `IonChannel` but with an additional `TemperatureInput` called `temp`. The delayed-rectifier potassium channel, which lets K^+ cations pass through the membrane when it is open, then becomes:

```
model PotassiumChannel "channel selective for K+ cations"
  extends GatedIonChannel(g_max=36, v_eq=12);
  extends HHmodelica.Icons.Activable;
  Gate gate_act(
  redeclare function fopen=
    goldmanFit(x0=-10, sy=100, sx=0.1),
  redeclare function fclose= expFit(sx=1/80, sy=125),
  v=v, temp=temp
```

```

) "activation gate";
equation
g = g_max * gate_act.n ^ 4;
end PotassiumChannel;

```

This component inherits variables and equations from `GatedIonChannel` and at the same time changes the values for the maximum conductance and the equilibrium potential. It uses a `Gate` component and redeclares the appropriate fitting functions to use for `fopen` and `fclose`. The only additional equation introduced is the dependency between the conductance and the gating variable `n`. As Hodgkin and Huxley determined, four gating molecules have to be in the open conformation simultaneously in order to allow ion transfer through a delayed-rectifier potassium channel. This is realized by taking the fourth power of the gating variable `n`.

The sodium channel, which lets Na^+ cations pass through the membrane, looks similar but a little more complex:

```

model SodiumChannel "channel selective for Na+ cations"
extends GatedIonChannel(g_max=120, v_eq=-115);
extends HHmodelica.Icons.Activatable;
extends HHmodelica.Icons.Inactivatable;
Gate gate_act(
redeclare function fopen=
goldmanFit(x0=-25, sy=1000, sx=0.1),
redeclare function fclose=
expFit(sx=1/18, sy=4000),
v=v, temp=temp
) "activation gate";
Gate gate_inact(
redeclare function fopen=
expFit(sx=1/20, sy=70),
redeclare function fclose=
logisticFit(x0=-30, sx=-0.1, y_max=1000),
v=v, temp=temp
) "inactivation gate";
equation
g = g_max * gate_act.n ^ 3 * gate_inact.n;
end SodiumChannel;

```

Here we have three molecules that form an activation gate and one molecule that forms an inactivation gate. The gates again only differ in the choice of fitting functions and values for their fitting parameters.

With this we already have all individual components that constitute the cell membrane. To measure and to perform experiments, however, we still need a model of the current clamp which keeps the current through the membrane constant in order to measure the voltage relative to a ground electrode:

```

model ConstantCurrent
extends TwoPinComponent;
parameter Real i_const(unit="ua/cm2");
equation
i = i_const;
end ConstantCurrent;

model Ground
PositivePin p;
equation
p.v = 0;
end Ground;

model CurrentClamp
extends HHmodelica.Icons.CurrentClamp;
PositivePin p "extracellular electrode";
NegativePin n "intracellular electrode(s)";

```

```

parameter Real i_const(unit="ua/cm2") = 40;
ConstantCurrent cur(i=i_const);
Ground g "reference electrode";
Real v = -n.v "measured membrane potential";
equation
connect(p, cur.p);
connect(n, cur.n);
connect(g.p, p);
end CurrentClamp;

```

This is the first part in the model where we use Modelica's `connect` equation to connect smaller components to one large component. The `Ground` component simply sets the potential of the extracellular compartment to zero while the `ConstantCurrent` component ensures that the cell has a constant positive outward current. The additional variable `v` captures the actual membrane potential that would be measured by a real current clamp experiment.

Now that all components are defined, putting together the whole HH model becomes as simple as just placing them side by side, connecting positive with positive and negative with negative pins of neighboring components as well as connecting the `TemperatureOutput` of the `LipidBilayer` to all `TemperatureInput` connectors:

```

model HHmodular
PotassiumChannel c_pot;
SodiumChannel c_sod;
LeakChannel c_leak;
LipidBilayer l2;
CurrentClamp clamp;
equation
connect(l2.p, c_pot.p);
connect(c_pot.p, c_sod.p);
connect(c_sod.p, c_leak.p);
connect(c_leak.p, clamp.p);
connect(clamp.n, c_leak.n);
connect(c_leak.n, c_sod.n);
connect(c_sod.n, c_pot.n);
connect(c_pot.n, l2.n);

connect(l2.temp, c_pot.temp);
connect(c_pot.temp, c_sod.temp);
end HHmodular;

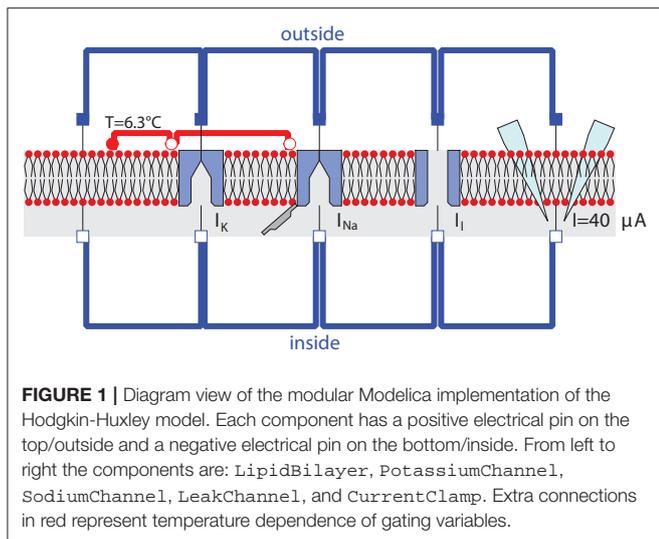
```

Annotations can be used to place the components on a coordinate system and to give the connections a graphical representation. Usually these annotations are not written manually but generated by an IDE like `OpenModelica`, where the components can be placed on a diagram view via drag and drop. Due to restrictions in space we do not show all the annotations here but only an example for the placement of the `LipidBilayer` and the connection between the `LipidBilayer` and the `PotassiumChannel`:

```

...
LipidBilayer l2 annotation(
Placement(visible = true, transformation(
origin = {-67, 3},
extent = {{-17, -17}, {17, 17}},
rotation = 0
))
);
...
equation
connect(l2.p, c_pot.p) annotation(
Line(
points = {{-66, 20}, {-66, 40}, {-33, 40}, {-33, 20}},

```



```

color = {0, 0, 255}
)
);
...

```

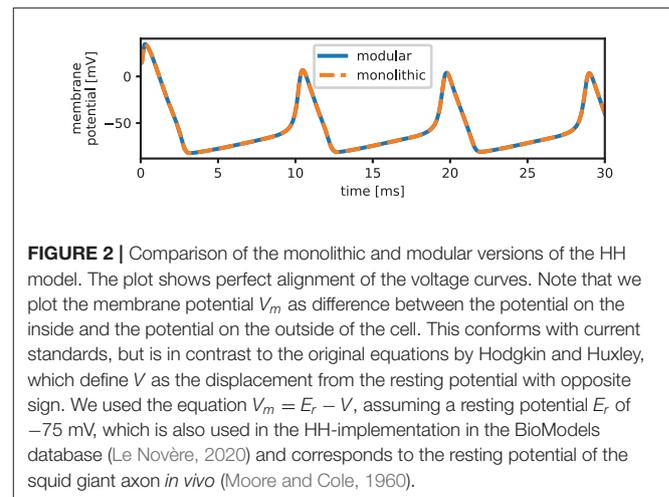
The resulting diagram is shown in **Figure 1**. The lipid bilayer is represented similar to diagrams in biology textbooks with circles on the outside and curved black lines pointing to the inside of the membrane. The channels are displayed as pores that are either open for the leak channel or closed for channels that have to be activated for ions to pass. The sodium channel additionally has a hinged lid to represent the inactivation gate. Finally, the current clamp is represented by two electrodes piercing the membrane.

3.2. Model Validation

When compiled, the modular version of the HH model reduces to the exact same equation system as the original monolithic version. Due to the modularization there are multiple versions of one variable, but this only leads to the addition of a few trivial equations of the form $x = y$ or the form $x = -y$. **Figure 2** shows a plot of the monolithic and modular version to ensure that both are functionally equivalent. This means that there are now two very different implementations with the same functionality which can and have to be analyzed for their suitability according to the research questions we established in our introduction.

3.3. Assessment of Understandability (RQ1)

With RQ1, we asked if the understandability of the HH model can be improved by a modular implementation that bridges the gap between biological meaning and electrical analogy. Using cognitive load theory as a framework, we will assume that a model is more understandable if it requires fewer items to be kept in working memory simultaneously. Although the modular implementation has more lines of code in total, it separates the model into small digestible parts. When a novice, for example, wants to know what a Gate is, they only have to process five variables and one equation, each of which are documented with their physiological meaning. When they



understand this component, they know that its purpose is to produce a value between zero and one which is based on voltage and temperature and represents the ratio of gating molecules in open conformation. Once this concept is stored in long-term memory it can be recalled as a single item into working memory. This means that, when the learner moves on, the two gates in SodiumChannel can be processed as two items instead of twelve². Moving from component to component, the modular version presents the reader with at most two equations and five variables or parameters at the same time. This constitutes a very low cognitive load compared to the 15 equations and 33 variables and parameters of the monolithic version that are presented all at once or in loose groups without clearly defined interfaces.

One way to facilitate the transfer of new concepts in long-term memory is to anchor them to existing knowledge. Our implementation does this by annotating each component and each variable or parameter in that component with biological terms³. The implementation also uses speaking variable names wherever possible to keep a close link between the biological and the mathematical representation—a common best practice in software engineering (that could also be applied to a monolithic version). Some parts of the model, such as the seemingly arbitrary fitting function `goldmanFit`, require more explanation which can be given in Modelica by adding an HTML string to the component for a detailed documentation.

Finally, on the highest hierarchical level, the component HHmodular has still only five variables but ten connect equations. For this model, however, a novice does not need to read any code at all to understand it, because they can use the diagram in **Figure 1** instead. Since it is defined directly in the code and tied to the individual components it is not only a simplified documentation but an accurate graphical reflection of

²This benefit is even more pronounced considering that the formulas for the α and β variables in the original model each consist up to seven seemingly arbitrary mathematical operations that are now given a meaning by introducing the named and documented fitting functions `expFit`, `logisticFit`, and `goldmanFit`.

³Due to spatial limitations we do not show all these annotations in this paper, but they can be found at <https://github.com/CSchoel/hh-modelica/>.

the underlying implementation. This means that understanding the model on this level of abstraction is not any harder than understanding a corresponding biological drawing in a textbook. Assuming that the learner is already familiar with *what* is modeled, the implementation can again be anchored easily to that existing knowledge.

3.4. Assessment of Extensibility and Reusability (RQ2)

With RQ2, we asked if a modular implementation of the HH model can serve as a unifying basis for extensions and therefore facilitate the creation of more complex HH-type modules. The current implementation already could reduce some duplicate equations in the original model by reusing already existing code. The current-voltage relationship of the three ion channels, the rate of conformation change in the three different gates, and the fitting functions `expFit` and `goldmanFit` each had to be defined only once. Furthermore, the introduction of the replaceable functions `fopen` and `fclose` eliminated the need to define starting values for the gating variables. When new components are added to the model, it is highly probable that some of these existing components can be used to reduce the implementation effort. Even more importantly, the argument for the reduction of cognitive load by modularization gets more weight as the model grows in size. In the modular version, the only point where cognitive load may increase due to extensions and therefore make the model less understandable is when there are too many individual models at the highest hierarchical layer. However, even then it is possible to form groups of components (e.g., a group for all potassium-sensitive channels or for all ion pumps) that are then connected on a new even higher hierarchical level. Conversely, the cognitive load associated with a monolithic model will grow with each variable and each equation that is added to the model.

To give one specific example, a reasonable extension could be the inclusion of slow inactivation of sodium channels. In contrast to fast inactivation, that stops the influx of Na^+ cations after a few milliseconds, slow inactivation takes place over seconds or even minutes of prolonged or high frequency depolarizations, reducing the number of sodium channels available for activation (Payandeh, 2018). This could be realized by simply adding another `Gate` component to the `SodiumChannel` and introducing a ratio `p_slow` that determines how much of the total current is attributed to slow as opposed to fast inactivation. The conductance equation would then change from

```
g = g_max * gate_act.n ^ 3 * gate_inact.n;

to:

g = g_max * gate_act.n ^ 3
  * (p_slow * gate_inact_slow.n + (1-p_slow) * gate_inact_fast
    .n);
```

Apart from choosing appropriate fitting functions for the new gate, this would be the only change required. Arguably a monolithic model would not require more changes, but it would be more difficult to first identify which equations have to change and thus it would be easier to make a mistake by missing or interchanging an equation or variable. We encountered this

problem in a previous work with a model of the human cardiac conduction system (Schölzel et al., 2020).

Other extensions might involve defining an alternative `Gate` that uses fitting functions to determine the steady state n_∞ and time constant τ instead of α (`fopen`) and β (`fclose`) (Destexhe and Huguenard, 2000; Goldman et al., 2001) or new components based on `TwoPinComponent`, such as channel formulations based on the Goldman-Hodgkin-Katz flux equation (Huguenard and McCormick, 1992; Destexhe and Huguenard, 2000) or models of ionic pumps (Di Francesco and Noble, 1985; Matsuoka and Hilgemann, 1992). Even in these cases the underlying interfaces can stay the same and parts like the current clamp formulation, common base classes, and fitting functions can be reused.

4. DISCUSSION

We showed that a modular version of the HH model that uses software-engineering techniques to manage complexity is beneficial both for novices and for experts, answering both of our research questions in the affirmative.

RQ1 asked whether the understandability of the HH model can be improved by a modular implementation. We showed that this is the case using CLT as framework and demonstrating a drop of the cognitive load by a factor of 6. The biological concepts can be explained and understood one at a time with an accurate graphical representation at the highest level of abstraction instead of having to navigate through a multitude of equations and variables with high element interactivity. In summary this means that with our implementation a deeper understanding of the HH model can be achieved in less time and it is likely that novices learning the model in this way will make fewer errors when recalling the learned concepts at a later time.

RQ2 asked whether the modular implementation can also serve as a unifying basis for extensions and facilitate the creation of more complex HH-type models. We showed that, in contrast to the monolithic version, adding new components does not significantly increase the cognitive load associated with the model. We also demonstrated that many components of our model are easily reusable which reduces development time and increases interoperability of solutions.

Similar results like ours would have been possible, for example, using CellML, or SBML with the SBML-comp package. In fact, Wimalaratne et al. (2009) also used the Hodgkin-Huxley model as an example to promote the support for hierarchical composition of CellML models. However, we used some Modelica features for our design that do not exist in these other languages. This includes the graphical composition of models, object-oriented programming with multiple inheritance, acausal connections between electrical and chemical components, the grouping of interface variables to connectors, and the annotation of the experiment setup within the model file itself.

One limitation of this approach is that some experts might be much more familiar with the formalism of differential equations than with object-oriented software design. It might be easier for them to reduce a group of equations to a single item in

their working memory than it is to do the same with a piece of code that represents a class. This means that, for models of small to moderate size, the navigation through different classes according to a modular design structure might actually be detrimental to their understanding of the model and to their productivity when working with it. A solution to this problem could be to provide a third, equation-based representation of the respective model in parallel to the existing graphical view and the raw code. Authoring tools like OpenCOR for CellML (Garny and Hunter, 2015) or COPASI for SBML (Hoops et al., 2006) already provide these equation-based views. However, they do not provide an overview of all equations in the whole model, but only of the parts that are currently selected. For Modelica we are only aware of a similar approach to OpenCOR and COPASI in the proprietary IDE MapleSim (Maplesoft, 2020). Implementing such a representation in open-source tools, such as OpenModelica would be possible due to the fact that, like CellML and SBML, Modelica is declarative. OpenModelica already has a feature to “instantiate” a model, which reduces its structure to a “flat” format consisting of a single class with a list of parameters, variables, and equations. Additionally, OpenModelica models can be exported in an XML format that contains all parameters, variables, and equations in a machine-readable form. Based on these existing features, an “equation view” could be implemented in the OpenModelica IDE OMEdit that would allow experts to understand a model at first glance based on the differential equations and without having to traverse its hierarchical structure. Alternatively, such a representation could be part of a documentation website associated with a model or model library. As an added benefit a tool that provides such an automated representation as typeset equations could also provide an export as LaTeX or Word documents, which can then be inserted in articles to guarantee that the published version of the equations is exactly the same as the equations used to simulate the model. We have implemented a first prototype of such an equation-based web documentation using the Julia package Documenter.jl (Piibeleht et al., 2020) and our own package ModelicaScriptingTools.jl (Schölzel, 2020). The resulting experimental documentation for the Hodgkin-Huxley model presented in this article can be found at <https://cschoel.github.io/hh-modelica/dev/>.

REFERENCES

- Bai, J., Gladding, P. A., Stiles, M. K., Fedorov, V. V., and Zhao, J. (2018). Ionic and cellular mechanisms underlying TBX5/PITX2 insufficiency-induced atrial fibrillation: insights from mathematical models of human atrial cells. *Sci. Rep.* 8:15642. doi: 10.1038/s41598-018-33958-y
- Blochwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmquist, H., et al. (2012). “Functional mockup interface 2.0: the standard for tool independent exchange of simulation models,” in *Proceedings of the 9th International Modelica Conference* (Munich), 173–184. doi: 10.3384/ecp12076173
- Courtemanche, M., Ramirez, R. J., and Nattel, S. (1998). Ionic mechanisms underlying human atrial action potential properties: insights from a mathematical model. *Am. J. Physiol. Heart Circ. Physiol.* 275, H301–H321. doi: 10.1152/ajpheart.1998.275.1.H301

Another direction for future research is the application of our techniques to larger and more complex models. We are already using the model developed in this paper as a basis to reproduce a large 116 equation model of the atrioventricular node (Inada et al., 2009). We hope that this and other projects based on the same methodology, be it with Modelica or another MoDROGH language like CellML, may help to increase the quality and speed of scientific progress in systems biology.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are publicly available. This data can be found here: <https://github.com/CSchoel/hh-modelica> (archived as <https://doi.org/10.5281/zenodo.3947848>).

AUTHOR CONTRIBUTIONS

CS and AD conceived the project. CS implemented the models and performed the experiments. VB and GE provided the physiological consultation and critique. CS drafted the manuscript. All authors contributed to manuscript revision, read, and approved the submitted version.

FUNDING

Funding for article processing fees was provided by the central publishing fund of the Technische Hochschule Mittelhessen - University of Applied Sciences.

ACKNOWLEDGMENTS

We thank our two reviewers for their valuable comments. We also thank Annina Hofferberth for proofreading the manuscript.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: <https://www.frontiersin.org/articles/10.3389/fphys.2020.583203/full#supplementary-material>

- Cuellar, A. A., Lloyd, C. M., Nielsen, P. F., Bullivant, D. P., Nickerson, D. P., and Hunter, P. J. (2003). An overview of CellML 1.1, a biological model description language. *Simulation* 79, 740–747. doi: 10.1177/0037549703040939
- Dassault Systèmes (2020). *Dymola*. Available online at: <https://www.3ds.com/products-services/catia/products/dymola/> (accessed August 20, 2020).
- Destexhe, A., and Huguenard, J. R. (2000). Nonlinear thermodynamic models of voltage-dependent currents. *J. Comput. Neurosci.* 9, 259–270. doi: 10.1023/A:1026535704537
- Di Francesco, D., and Noble, D. (1985). A model of cardiac electrical activity incorporating ionic pumps and concentration changes. *Philos. Trans. R. Soc. Lond. B Biol. Sci.* 307, 353–398. doi: 10.1098/rstb.1985.0001
- Elmqvist, H., Henningsson, T., and Otter, M. (2016). “Systems modeling and programming in a unified environment based on Julia,” in *Leveraging Applications of Formal Methods, Verification and Validation: Discussion*,

- Dissemination, Applications, ISO/IEC 2016, Volume 9953 of Lecture Notes in Computer Science (Corfu), 198–217. doi: 10.1007/978-3-319-47169-3_15
- Fabbri, A., Fantini, M., Wilders, R., and Severi, S. (2017). Computational analysis of the human sinus node action potential: model development and effects of mutations. *J. Physiol.* 595, 2365–2396. doi: 10.1113/JP273259
- Fink, M., and Noble, D. (2009). Markov models for ion channels: versatility versus identifiability and speed. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* 367, 2161–2179. doi: 10.1098/rsta.2008.0301
- Fritzson, P., Aronsson, P., Lundvall, H., Nyström, K., Pop, A., Saldamli, L., et al. (2005). “The OpenModelica modeling, simulation, and development environment,” in *Proceedings of the 46th Conference on Simulation and Modelling of the Scandinavian Simulation Society* (Trondheim).
- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Reading, MA: Addison-Wesley.
- Garny, A., and Hunter, P. J. (2015). OpenCOR: a modular and interoperable approach to computational biology. *Front. Physiol.* 6:26. doi: 10.3389/fphys.2015.00026
- Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge: Cambridge University Press.
- Goldman, M. S., Golowasch, J., Marder, E., and Abbott, L. F. (2001). Global structure, robustness, and modulation of neuronal models. *J. Neurosci.* 21, 5229–5238. doi: 10.1523/JNEUROSCI.21-14-05229.2001
- Hellerstein, J. L., Gu, S., Choi, K., and Sauro, H. M. (2019). Recent advances in biomedical simulations: a manifesto for model engineering. *F1000Research* 8:261. doi: 10.12688/f1000research.15997.1
- Hines, M. L., and Carnevale, N. T. (1997). The NEURON simulation environment. *Neural Comput.* 9, 1179–1209. doi: 10.1162/neco.1997.9.6.1179
- Hodgkin, A. L., and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* 117, 500–544. doi: 10.1113/jphysiol.1952.sp004764
- Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., et al. (2006). COPASI—a COmplex PATHway Simulator. *Bioinformatics* 22, 3067–3074. doi: 10.1093/bioinformatics/btl485
- Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., et al. (2003). The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* 19, 524–531. doi: 10.1093/bioinformatics/btg015
- Huguenard, J. R., and McCormick, D. A. (1992). Simulation of the currents involved in rhythmic oscillations in thalamic relay neurons. *J. Neurophysiol.* 68, 1373–1383. doi: 10.1152/jn.1992.68.4.1373
- Hunt, A., and Thomas, D. (2000). *The Pragmatic Programmer: From Journeyman to Master*. Reading, MA: Addison-Wesley.
- Inada, S., Hancox, J. C., Zhang, H., and Boyett, M. R. (2009). One-dimensional mathematical model of the atrioventricular node including atrio-nodal, nodal, and nodal-his cells. *Biophys. J.* 97, 2117–2127. doi: 10.1016/j.bpj.2009.06.056
- Inkscape Developers (2020). *Inkscape-Draw Freely*. Available online at: <https://inkscape.org/> (accessed August 20, 2020).
- Jordan, J., Mørk, H., Vennemo, S. B., Terhorst, D., Peysner, A., Ippen, T., et al. (2020). *NEST*. Zenodo. Available online at: <https://doi.org/10.5281/zenodo.2605422> (accessed August 20, 2020).
- Justus, N., Schölzel, C., Dominik, A., and Letschert, T. (2017). “Mo|E—a communication service between Modelica compilers and text editors,” in *Proceedings of the 12th International Modelica Conference* (Prague), 815–822. doi: 10.3384/ecp17132815
- Le Novère, N. (2020). *BioModels: Hodgkin-Huxley Squid-Axon 1952*. Available online at: <https://www.ebi.ac.uk/biomodels/BLOMD0000000020> (accessed August 20, 2020).
- Lopez, C. F., Muhlich, J. L., Bachman, J. A., and Sorger, P. K. (2013). Programming biological models in Python using PySB. *Mol. Syst. Biol.* 9:646. doi: 10.1038/msb.2013.1
- Maplesoft (2020). *MapleSim—Advanced System-Level Modeling and Simulation*. Available online at: <https://www.maplesoft.com/products/maplesim/> (accessed August 20, 2020).
- Matsuoka, S., and Hilgemann, D. W. (1992). Steady-state and dynamic properties of cardiac sodium-calcium exchange: ion and voltage dependencies of the transport cycle. *J. Gen. Physiol.* 100, 963–1001. doi: 10.1085/jgp.100.6.963
- Mattsson, S. E., and Elmqvist, H. (1997). “Modelica—an international effort to design the next generation modeling language,” in *7th IFAC Symposium on Computer Aided Control Systems Design, CACSD’97* (Gent), Vol. 30, 151–155. doi: 10.1016/S1474-6670(17)43628-7
- Moore, J. W., and Cole, K. S. (1960). Resting and action potentials of the squid giant axon *in vivo*. *J. Gen. Physiol.* 43, 961–970. doi: 10.1085/jgp.43.5.961
- Noble, D., Garny, A., and Noble, P. J. (2012). How the Hodgkin-Huxley equations inspired the cardiac physiome project. *J. Physiol.* 590, 2613–2628. doi: 10.1113/jphysiol.2011.224238
- Olivier, B. G., Rohwer, J. M., and Hofmeyr, J.-H. S. (2005). Modelling cellular systems with PySCeS. *Bioinformatics* 21, 560–561. doi: 10.1093/bioinformatics/bti046
- Payandeh, J. (2018). Progress in understanding slow inactivation speeds up. *J. Gene. Physiol.* 150, 1235–1238. doi: 10.1085/jgp.201812149
- Piibeleht, M., Hatherly, M., and Ekre, F. (2020). *Documenter.jl*. Available online at: <https://github.com/JuliaDocs/Documenter.jl> (accessed August 20, 2020).
- Rackauckas, C., Ma, Y., Widmann, D., Ranocha, H., Levien, E., Short, T., et al. (2020). *DifferentialEquations.jl Documentation*. Available online at: <http://docs.juliadiffeq.org/latest/> (accessed August 20, 2020).
- Schölzel, C. (2020). *THM-MoTE/ModelicaScriptingTools.jl: V1.1.0-alpha.1*. Zenodo.
- Schölzel, C., Blesius, V., Ernst, G., and Dominik, A. (2020). The impact of mathematical modeling languages on model quality in systems biology: a software engineering perspective. *bioRxiv*. doi: 10.1101/2019.12.16.875260
- Sweller, J. (2019). “Cognitive load theory,” in *Advances in Cognitive Load Theory: Rethinking Teaching, 1st Edn.*, eds S. Tindall-Ford, S. Agostinho, and J. Sweller (Abingdon: Routledge), 1–11. doi: 10.4324/9780429283895-1
- The MathWorks, Inc. (2020). *Simscape*. Available online at: <https://www.mathworks.com/products/simscape.html> (accessed August 20, 2020).
- Voit, E. O. (2013). *A First Course in Systems Biology, 1st Edn.* New York, NY: Garland Science.
- Wimalaratne, S. M., Halstead, M. D. B., Lloyd, C. M., Cooling, M. T., Crampin, E. J., and Nielsen, P. F. (2009). Facilitating modularity and reuse: guidelines for structuring CellML 1.1 models by isolating common biophysical concepts. *Exp. Physiol.* 94, 472–485. doi: 10.1113/expphysiol.2008.045161
- Winslow, R. L., Rice, J., Jafri, S., Marbán, E., and O’Rourke, B. (1999). Mechanisms of altered excitation-contraction coupling in canine tachycardia-induced heart failure, II: model studies. *Circ. Res.* 84, 571–586. doi: 10.1161/01.RES.84.5.571

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Copyright © 2020 Schölzel, Blesius, Ernst and Dominik. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.