Progress in Biophysics and Molecular Biology 120 (2016) 100–114

Contents lists available at ScienceDirect

Progress in Biophysics and Molecular Biology

journal homepage: www.elsevier.com/locate/pbiomolbio

Myokit: A simple interface to cardiac cellular electrophysiology

Michael Clerx [a, b], Pieter Collins [a], Enno de Lange [a], Paul G.A. Volders [b, *]

[a] Department of Data Science and Knowledge Engineering, Maastricht University, P.O. Box 616, 6200 MD, Maastricht, The Netherlands
[b] Department of Cardiology, Cardiovascular Research Institute Maastricht, Maastricht University Medical Center, PO Box 5800, 6202 AZ, Maastricht, The Netherlands

ARTICLE INFO

Article history:
Received 18 September 2015
Received in revised form
7 November 2015
Accepted 16 December 2015
Available online 23 December 2015

Keywords:
Computational models
Cardiac action potential
Software tools
Ion channels
Simulation

ABSTRACT

Myokit is a new powerful and versatile software tool for modeling and simulation of cardiac cellular electrophysiology. Myokit consists of an easy-to-read modeling language, a graphical user interface, single and multi-cell simulation engines and a library of advanced analysis tools accessible through a Python interface. Models can be loaded from Myokit's native file format or imported from CellML. Model export is provided to C, MATLAB, CellML, CUDA and OpenCL. Patch-clamp data can be imported and used to estimate model parameters. In this paper, we review existing tools to simulate the cardiac cellular action potential to find that current tools do not cater specifically to model development and that there is a gap between easy-to-use but limited software and powerful tools that require strong programming skills from their users. We then describe Myokit's capabilities, focusing on its model description language, simulation engines and import/export facilities in detail. Using three examples, we show how Myokit can be used for clinically relevant investigations, multi-model testing and parameter estimation in Markov models, all with minimal programming effort from the user. This way, Myokit bridges a gap between performance, versatility and user-friendliness.

1. Introduction

Numerical models of the electrical processes in cardiac myocytes have been successfully used to elucidate the mechanisms of action potential (AP) formation, electrical propagation from cell to cell, and abnormal impulse formation and conduction in the heart (Noble et al., 2012). These models provide a vital bridge between drug targets, for example ion channels or receptors, and the dynamical factors leading to heart-rhythm disorders, such as pathologically altered conduction or repolarization (Weiss et al., 2015). The application of new and refined experimental methods has led to an increased level of detail and specialization in AP models, which may now include elements such as stretch-sensitive channels (Niederer and Smith, 2007), ion-channel phosphorylation (O'Hara et al., 2011) or signaling pathways (Heijman et al., 2011). Conversely, the growing awareness of the need to integrate models on different scales combined with the increased availability of computing power has broadened the scope for application of AP models considerably. As a result, AP models have more users and more uses than ever before, but also a far greater mathematical complexity and a greater reliance on the model builder's correct interpretation of complex multi-modal experimental data. Collaboration between experts from a wide range of disciplines is vital to further refine models and experiments and therefore the knowledge we derive from them (Abriel et al., 2013).

Software tools can aid researchers using AP models in several ways. Firstly, the definition of open and unambiguous formats for model description allows models to be exchanged, inspected, compared, improved and revised. Publishing a model in a widely recognized format is an invitation for external feedback and can assist widespread adoption and recognition of modeling results. Comparison can be automated, allowing models to be tested against previous results from model or experiment each time a change is made. This way, models can be continuously refined without the danger of losing past results.

Secondly, sharing software for simulation and analysis is a way of sharing effort and expertise so that researchers may benefit directly from each other's work. Of particular importance in this respect, is the role of fiber and tissue simulations for the development of single-cell models. Many of the properties that a single-cell model should capture only emerge when cells are coupled, making

* Corresponding author.
E-mail addresses: michael.clerx@maastrichtuniversity.nl (M. Clerx), pieter.collins@maastrichtuniversity.nl (P. Collins), enno.delange@maastrichtuniversity.nl (E. de Lange), p.volders@maastrichtuniversity.nl (P.G.A. Volders).

http://dx.doi.org/10.1016/j.pbiomolbio.2015.12.008
0079-6107/

such simulations a vital part of AP model development. However, the programming effort required to set up these simulations may deter some modelers from taking this route. By sharing software tools such barriers are taken away.

Finally, if tools are not just efficient but also easy to use then the time needed to set up experiments is decreased while the number of people able to do so (or willing to learn) is increased. This can aid collaboration by drawing more non-experts into computational biology while giving experts time to step outside their disciplines.

So how far have these benefits been realized in current software tools? In the next sections we review existing software with a focus on the three goals of sharing models, sharing methods and user friendliness. We discuss the usefulness of these tools for model development and make two central observations: (1) Existing tools cater to AP-model use, more than to model development. These two goals have different, sometimes directly opposite needs. (2) There is a wide gap between easy-to-use simulation software with limited capabilities and powerful simulation tools that require considerable effort and programming skill from the user. Finally, we present Myokit, a tool for model development and analysis which aims to fill this gap.

1.1. Tools for sharing models

The most widely used language for sharing models of the cardiac AP is CellML (Hedley et al., 2001; Cuellar et al., 2003). While there are other exchange formats such as SBML (Systems Biology Markup Language, see Hucka et al. (2003)) in which AP models can be formulated, CellML has broad support in the AP modeling community and a freely accessible repository containing over 160 models of the cardiac AP (http://models.cellml.org). The first specification (version 1.0) was finalized in 2001 and a second (version 1.1) was given definitive status in 2002. An overview of tools capable of working with CellML is available online (http://www.cellml.org/tools). The goal behind CellML is to facilitate universal exchange and re-use of mathematical models, particularly models of the electrophysiological and biochemical processes inside a cell.

A number of tools have been published to convert CellML to other languages, allowing CellML models to be used with a variety of tools. The CellML advanced programming interface (API) defines a number of ways to interact with a model, including translating it to other languages (Miller et al., 2010). An implementation of the API is available on the CellML website and can be used to translate CellML models to C, MATLAB and Python. Another conversion tool is AGOS (Barbosa et al., 2006), which can create simulation code for C++. It was modified by Amorim et al. (2010) to generate simulation code that can be run on graphical processing units (GPUs), thereby facilitating fast multi-cell simulations. PyCml (http://chaste.cs.ox.ac.uk/cellml) is a Python based utility that can be used to read and stringently validate CellML files before creating C++ model definitions.

The CellML language is specified using XML (eXtensible Markup Language), a widely used format for sharing documents over the internet. The choice for XML makes it easy for developers of software tools to incorporate a CellML import into their programs, as software libraries to work with XML documents are freely available for many different platforms. A downside of XML is that, while it is *human-readable* as well as *machine-readable*, it is not necessarily compact, nor easy to read or to write by hand. CellML equations in particular are specified in a subset of MathML (Mathematical Markup Language), which is unambiguous but highly verbose. In addition, good model definitions for archiving and exchange are written out with consistent units, are well annotated with information about all the variables and define a rigid interface through which the model can interact with other models. These features, which are undoubtedly good for exchange formats, can be a hindrance to users wishing to rapidly experiment with different model formulations, making CellML less suited to model development.

1.1.1. Combining models

A major benefit of having models publicly available in a shared format is that models can be *combined*. For example, a model of the cardiomyocyte can be incorporated into an integrative model of the cardiovascular system. This is one of the key ideas behind the IUPS Human Physiome project, which seeks to connect the many specialized mathematical models used in biology (Bassingthwaighte, 2000; Hunter, 2004). The need for such model integration has driven the development of many of the tools discussed here.

A tool for combining models on the molecular level is Virtual Cell, also known as VCell (Moraru et al., 2008). It can model electrophysiology, reaction kinetics, membrane transport and diffusion processes to create a 3D model of a cell that can be related to experimentally obtained images. On a larger scale, tools such as CHeart (http://cheart.co.uk), Continuity (http://continuity.ucsd.edu) and OpenCMISS (Nickerson et al., 2014) can be used to create combined models of electrophysiology, mechanics and fluid dynamics inside the heart.

A key idea in creating these combined, multi-scale models is that the resulting system is *modular*, i.e., that any model of a subsystem (for example a model of the cardiac cellular AP) can be replaced or updated without requiring changes to the other subsystem models. This requires that the models, at least in part, adhere to some well-defined structure about which there is a widespread consensus. Such fixed structures (called *ontologies* in software terms) contrast sharply with the creative aspects of model development where variables, for example currents, are continuously added, removed, split up and redefined. As a result, the aim of integrating different levels of physiology is not always compatible with the aim of accurately incorporating new experimental data.

1.1.2. Comparing model results

AP models represent theories on the electrophysiological functioning of the cell. As such, it should always be possible to compare model results with experimental data or predictions from competing models. A standard defining the Minimum Information for a Cardiac Electrophysiology Experiment (MICEE) has been approved by a large consortium of cardiac electrophysiologists (Quinn et al., 2011). Similarly, a standard defining the Minimum Information About a Simulation Experiment (MIASE) has been created by Waltemath et al. (2011). SED-ML, the Simulation Experiment Description Markup Language, is a format for sharing simulation experiments designed to meet the requirements set out in MIASE. It describes how a simulation should be designed and run in a manner that allows the results to be replicated on different systems. At the time of writing this manuscript, the standard does not yet meet the requirements needed to compare the output of different models or to compare the output of arbitrary models to experimental data. *Functional curation* is a standard proposed by Cooper et al. (2011, 2015b) to describe experiments and post-processing operations independently of a model. Such a standard can be of great use for systematic model development. For example, by creating a series of tests that a model must pass and regularly