CELLULAR OPEN RESOURCE (COR): current status and future directions

BY ALAN GARNY1,*, DENIS NOBLE1, PETER J. HUNTER2 AND PETER KOHL1

1Department of Physiology, Anatomy and Genetics, University of Oxford, Sherrington Building, Parks Road, Oxford OX1 3PT, UK
2Auckland Bioengineering Institute, University of Auckland, Auckland 1142, New Zealand

The need for tools to aid the description and sharing of biological models was highlighted at the launch of the International Union of Physiological Sciences Physiome Project in 1997. This has resulted in the release, in 2001, of the CellML specifications (http://www.cellml.org/specifications/). CELLULAR OPEN RESOURCE (COR) was among the early adopters of this standard, eventually forming the first publicly available CellML-based modelling and collaboration environment. From the onset, COR was designed to provide an environment that could not only be used by experienced modellers, but also by experimentalists, teachers and students. It therefore tries to combine a user-friendly interface with a computationally efficient numerical engine. In this paper, we introduce the philosophy behind COR, explain its user interface and current functionality, including the editing and running of CellML files, highlight lessons learned from user feedback and problems experienced during the development of COR and conclude by exploring future development potential.

Keywords: CellML; extensible markup language; markup language; modelling environment; computational biology

1. Introduction

(a) Model development

Quantitative biomedical research and development increasingly relies on the iteration between ‘wet’ and ‘dry’ tools (Kohl et al. 2000). A PubMed (http://www.pubmed.gov/) search for articles that include either ‘computer model’ or ‘mathematical model’ in their title illustrates the continuous increase in papers devoted to this topic (figure 1).

In particular, during the last decade, there has been a surge in the number of published models, indicative of a trend towards quantitative biomedical research. This, together with the increasing level of complexity of models, makes their publication, sharing and reuse both more important and more challenging.
In the academic world, model development (as other intellectual output) typically involves several stages, which are summarized in a simplified flow chart in Figure 2.

Each of these stages is subject to human error. Mistakes can be introduced during the conceptual stage, when model authors specify their ideas (stage no. 1). These ideas form the basis of a model, which is typically implemented either in a high-level language (e.g. C++, Fortran) or using a mathematical environment such as Matlab (stage no. 2). Model authors usually iterate between the two stages until they are satisfied with model performance, which involves eradication of most errors (e.g. pure mathematical errors) that may have occurred at stage no. 1. Typical errors that can remain unnoticed include inconsistencies in the use of units. Once the model is ready for dissemination, a manuscript describing the model and including its mathematical equations may be written (stage no. 3). Again, it is not uncommon, at this stage, for model authors to omit crucial information (e.g. parameter values, units, metadata) or to include typographical mistakes. Peer review may help in reducing the number

Figure 1. Number of articles, published between 1950 and 2007, that have the term ‘computer model’ or ‘mathematical model’ in their title.

Figure 2. Life cycle of a typical computer/mathematical model in academia.
of such errors, but it rarely removes all mistakes. Similarly, once the manuscript is accepted for publication (stage no. 4), additional errors may be introduced at, or after, the proof stage (this time, by the publisher or the printer). These errors may, or may not, be noted and corrected by model authors. The end result is a published model whose mathematical formulation contains errors that prevent it from being in line with the original description given by its authors. Someone else, interested in using a model (stage no. 5), may thus end up spending extended periods of time to arrive at the original model (stage no. 2), including efforts to fix mistakes introduced during implementation.

(b) Model sharing

To address these issues, and to facilitate the use and distribution of models, some authors have decided to publish the source code of their models online. Thus, the models from the group of Prof. Yoram Rudy are available in both C++ and Matlab (http://rudylab.wustl.edu/research/cell/), with codes that work straight ‘out of the box’. However, Fortran users will have to convert the C++ or Matlab code into their preferred language. While this is a relatively trivial exercise for someone familiar with C++/Matlab and Fortran, it is unlikely to be completely error-free. An important advantage of this generous sharing attitude is that models, which have actually been used and applied by their authors, are being made available in their original form for use by others. Reproduction of published results, for example, can thus form a vital ‘training exercise’ for interested users.

An alternative solution is the development and use of a generic language, even though parallels to the ‘generic’ human language Esperanto may not seem particularly encouraging. This is the approach taken by the International Union of Physiological Sciences (IUPS) Physiome Project (Hunter & Borg 2003). The major goal of the IUPS Physiome Project is to provide a framework for the modelling of the human body using computational methods, which incorporate the biochemistry, biophysics and anatomy of cells, tissues and organs (Hunter et al. 2002; Crampin et al. 2004).

A key component of that framework is CellML, an open format for describing and sharing biological models that could be used in stage no. 2 (i.e. use CellML rather than C++, Fortran, Matlab, etc. as the main format; note that CellML is also relevant to later stages). It is based on the extensible markup language (http://www.w3.org/TR/REC-xml/) and its initial specifications were made public in August 2001 (CellML 1.0; http://www.cellml.org/specifications/cellml_1.0/), before being refined in February 2006 to allow for the import and reuse of model descriptions (CellML 1.1; http://www.cellml.org/specifications/cellml_1.1/).

(c) Cellular Open Resource

Cellular Open Resource (COR; http://cor.physiol.ox.ac.uk/) was the first CellML-based environment to be made publicly available (Garny et al. 2003). It was followed by the Physiome CellML Environment (PCEnv; http://www.pcenv.org/), which is developed by the team behind CellML. These are currently

1 The multicellular capability discussed in our original paper has since been removed from COR.
the only two complete CellML environments that directly interact with CellML files (as opposed to importing them, which may result in a loss of information), and offer facilities such as creation, editing, validation and simulation. Further CellML-related tools have recently been reviewed (Garny et al. 2008), and they include

— visual editing (e.g. Virtual Cell; http://www.nrcam.uchc.edu/; Loew & Schaff 2001);
— CellML validation (e.g. PyCML; https://chaste.ediamond.ox.ac.uk/cellml/);
— units validation (e.g. JSim; http://www.physiome.org/jsim/);
— generation of code (e.g. AGOS; http://www.fisiocomp.ufjf.br/; Barbosa et al. 2006); and
— execution of CellML models (e.g. CESE; http://cese.sourceforge.net/).

In the present paper, we give a quick overview of the general philosophy behind COR, as well as a detailed account of the approaches taken by COR for the editing and simulation of CellML files. To illustrate the latter, we simulate the effects of certain drugs on cardiac electrophysiology, using a guinea pig ventricular cell model. We then discuss future directions for a more generic CellML-based environment, before making some general concluding remarks.

2. General philosophy

(a) Target audience

From the outset, COR has not only targeted modellers, but also experimentalists, teachers and students. The eclectic nature of this target audience means that it is essential for COR to offer both a user-friendly graphical user interface (GUI) and a computationally efficient numerical engine. This approach seems to have resonated with the target user community.

At the time of submission, COR has been downloaded (from http://cor.physiol.ox.ac.uk/Download/) by more than 470 unique individuals from nearly 50 countries worldwide (figure 3). This equates to more than three unique downloads per week (since the site was established). By controlling the source of downloads and tracking the number of unique downloads, we are able to better estimate COR’s uptake in the community, which would have been impossible had COR been released on a source code repository such as SourceForge.net or Google Code (http://sourceforge.net/ and http://code.google.com/, respectively). During the download process, users are asked to state their research interests and, so far, everybody has stated some form of involvement in computer modelling (indicating model-specific interest, rather than general curiosity).

Of all individuals who downloaded COR, roughly 10 per cent are direct collaborators or colleagues of the developers. On one occasion, several users from the same institution downloaded COR, probably in the context of a practical class on computer modelling. Experimentalists with no prior knowledge of computer modelling have been able to use COR productively in their research (e.g. Shattock & Rosen 2006). Interest from industry (including pharmaceutical companies) further highlights the general demand for tools such as COR for research and development.

*Phil. Trans. R. Soc. A* (2009)
Work on COR was initiated in 2002. At the time, a study by RedSheriff\(^2\) showed that 96.8 per cent of computers worldwide were using an operating system of the Microsoft WINDOWS family. As for the remaining 3.2 per cent, it would have probably been safe to assume that the majority of users would have been able to get access to a Microsoft WINDOWS machine, if such need had arisen.

The overwhelming use of Microsoft WINDOWS, and the nature of our target user community, led to the decision to produce a ‘pure’ Microsoft WINDOWS application (including Microsoft WINDOWS VISTA). To opt for a single platform application means that one can choose a programming environment that takes full advantage of the target operating system. One such environment is Delphi (http://www.codegear.com/products/delphi/win32/), which was used to implement COR. The focus on Microsoft WINDOWS and Delphi also avoided issues related to the use of different compilers on different platforms (e.g. Microsoft Visual C++ on Microsoft WINDOWS and GCC on LINUX machines). Clear disadvantages of focusing on a single platform are limited applicability, in particular among some of the most advanced potential users of computational models who may prefer LINUX-based platforms.

(c) Support

A common concern among software developers is unexpected program behaviour, in particular ‘software crashes’. While highly undesired, such events often contain important information about software shortcomings. It is essential,

---
\(^2\)The study was entitled ‘Where do you want to go today? WINDOWS 98 is the most popular operating system to take you there!’ and used to be accessible online from http://www.redsheriff.com/cgi-bin/news20.cgi/Show?_id=d9c6, but has since been closed down. It can, however, be retrieved using the Internet Archive Wayback Machine (http://www.archive.org/).
therefore, to establish a reporting mechanism that enables the user to provide as much information as possible. For this reason, should COR ‘crash’, it automatically generates a report that contains information about the user’s environment and the offending piece of code. This information can be emailed, from within COR, to the development team (optional user choice), to guide work on problem fixing.

Updated versions of COR are made available to users on a regular basis, not only following bug reports but also as a result of improvements to existing features and/or addition of new functionalities.

Figure 4. Settings for customizing COR. (a) Settings specific to the editorial mode (see §3). (b) Settings specific to the computational mode (see §4).

*Phil. Trans. R. Soc. A* (2009)
Since most users are now continuously connected to the Internet, be it at work or at home, COR checks upon start-up whether a new version is available. This occurs by connecting to our server and checking the release log file, unless the user explicitly prevents this check in preference for manual updates. Should a new version be available, COR will display a message to the user, listing the differences between the current and new versions. It is then up to the user to decide whether to upgrade or not.

\( (d) \) Modes of operation

Use of COR will be either predominantly for model development or for running simulations. For this reason, COR offers two modes of operation (editorial and computational; figure 4), which are discussed in §§3 and 4.

3. Editorial mode

\( (a) \) CellML application program interface

Considering the wide target audience of COR (see §2a), it is inevitable that some users will have limited familiarity with CellML (lack of insight into CellML, reluctance to edit CellML files, etc.). For this reason, COR provides a user-friendly way of editing CellML files. The approach taken is illustrated in figure 5. It relies on the use of a CellML application program interface (API), which allows COR to read and write CellML 1.0 files (see §5b for further detail). Upon opening a CellML file, COR uses the CellML API to read its contents (dashed boxes in figure 5, in particular the left dashed box) and creates an in-memory representation of the model described. For editing purposes, this in-memory representation is then converted to a ‘readable’ format (dotted boxes in figure 5, in particular the bottom right dotted box). This was designed with the aim of being faithful to the CellML 1.0 specifications (http://www.cellml.org/specifications/cellml_1.0/), while being less verbose and easier to learn than CellML itself.

Editing of a CellML 1.0 file is thus conducted using the readable format generated by the CellML API. The same API is used to save the edited CellML file, first converting the readable format to an in-memory representation, and then serializing that in-memory representation to a file. The API comes with several services that support tasks such as the conversion of the in-memory representation of a model to machine code (MC; i.e. code that can be directly executed by a computer; see §4a,d). Similarly, COR can export a CellML 1.0 file to both Microsoft Word 2007 and TeX. These exports are relevant for model publication and dissemination (i.e. stage nos. 3 and 4; figure 2). In addition to word processing formats, COR can export to several programming languages (C, C++, Delphi for Win32, Fortran 77, Java, Matlab and Pascal), thus addressing the demands of stage no. 5 (figure 2). Provision of support for additional programming languages can take anything between 2 hours (e.g. Java) and 2 days (e.g. Fortran 77, because of unconventional formatting constraints). It should be noted, though, that these exports are unlikely to work without user intervention. This has, however, proven to be a relatively trivial exercise in practice, since it does not involve altering the
mathematics of the code. To help in assessing the export of a model, some basic programs have been made available online (http://cor.physiol.ox.ac.uk/Exports/).

The API also provides a validation service via which a CellML file can be checked for compatibility with the CellML 1.0 specifications, except for reactions and metadata (note that reactions are likely to be removed from future CellML specifications, and their use is therefore discouraged). To edit a CellML file with metadata is nonetheless possible. The user will simply not be able to edit such data and COR make sense of it.

(b) Graphical user interface

The GUI for the editorial mode of COR is shown in figure 6. It consists of three resizable panels, in addition to the usual menus (including context menus), toolbars and status bar.

— Command viewer. This is used to graphically render units definitions, variable declarations and mathematical equations. Figure 6a illustrates this by rendering an equation, which defines the alpha component of the activation file.
gate of the fast sodium current in the Noble 1962 model (Noble 1962). Had the
mathematical equation (or units definition, or variable declaration) been
malformed, a warning would have been displayed. The command viewer is
fully configurable (figure 4a(iv)). In particular, it is possible to enable or
disable one or several of the following: render anything after an underscore as
a subscript (i.e. \( y_{\alpha} \) will be rendered as \( y_{\alpha} \)); use Greek symbols
wherever possible (i.e. \( y_{\alpha} \) will be rendered as \( \gamma_{\alpha} \)); and group digits
(i.e. 1234.56789 will be rendered as 1,234.567,89). The contents of the viewer
can be copied to the clipboard as a bitmap object, for later use (e.g. to be
pasted in a document or a presentation).

— CellML 1.0 editor. This displays the readable version of a CellML file, as shown
in figure 6b. It consists of a tabbed editor, allowing for one or several CellML files
to be open at the same time. Common editing features are available (e.g. undo/
redo, cut/copy/paste, find/replace) and, as for the command viewer, it is fully
customizable (e.g. font name, font size, keyword display options; figure 4a(i)(iii)). COR is provided with a selection of CellML models (also
available from the CellML Model Repository; http://www.cellml.org/models/),
all of which are ‘locked’ (i.e. read-only) by default. This is to prevent users from
accidentally altering them, although they can be unlocked/relocked, if required.
Locked models can be identified by the reddish (configurable) background of the
editor (light grey in figure 6b). The suggested mode of use is to leave locked
models intact and to edit ‘duplicated’ versions from within COR.

— Messages viewer. As mentioned in §3a, the API has a validation service. Any
error or warning reported by the service is listed in the messages viewer
(figure 6c). For each reported problem, a detailed description is given, which,
when double-clicked, highlights the offending portion of the model. Errors (e.g., an equation that reads \( a + b \)) will prevent a model from being saved, run, etc., while warnings (e.g. a dimensionally inconsistent equation such as \( a = b \), with \( a \) in millivolts and \( b \) in volts) will not prevent the aforementioned actions. The viewer is also used to report non-CellML-related problems (e.g. trying to open a file that is no longer physically present).

Note that the toolbars, command viewer and messages viewer can be undocked and placed anywhere on a user’s computer desktop.

4. Computational mode

(a) Code generation

In order to enter the computational mode, a valid CellML file is needed, in a form that can be used by COR to run a simulation. This means that COR needs to either

— generate the MC (figure 5) or
— generate some C code and compile it to a dynamic link library (DLL), for use by the software.

For the latter, an external compiler is required. COR supports both Intel C++ and Microsoft Visual C++ compilers (http://www.intel.com/cd/software/products/asmo-na/eng/279578.htm and http://www.microsoft.com/express/vc/, respectively; figure 4b(i)). In both cases, the compilers are configured to generate the most computationally efficient code possible. In the case of the Intel C++ compiler, this also involves generating optimized code for the different types of processors that the Intel C++ compiler supports. It is worth noting that

— should the DLL generation fail, COR will automatically revert to generating and using the MC instead and
— of the two aforementioned compilers, the Intel C++ compiler generates the more efficient code. It is, however, targeted at Intel-based machines, so running Intel code on non-Intel-based machines is discouraged, as it may lead to erroneous results or COR ‘crashing’.

COR assumes that no external compiler is available and therefore generates some MC by default (but this can be changed; figure 4b(i)). Certain mathematical functions (e.g. exponential or power functions) are computationally expensive, and use of a special DLL that includes these functions yields better performance. Both an Intel C++ and a Microsoft Visual C++ version of the DLL have been generated and embedded in the COR executable. Upon start-up, COR determines the type of machine used, and extracts and uses the most suitable DLL.

(b) Ordinary differential equation integrators

COR is aimed at solving models that contain one or several ordinary differential equations (ODEs), which must be integrated over time. To solve those ODEs, COR offers a choice of four ODE integrators: forward Euler (FE); second-order
Runge–Kutta (RK2); fourth-order Runge–Kutta (RK4); and CVODE (part of the SUNDIALS library; http://www.llnl.gov/CASC/sundials/; Hindmarsh et al. 2005). FE, RK2 and RK4 use an explicit scheme, together with a fixed time step. The drawback of this approach is that the time step must be small enough to achieve numerical stability, which is determined by the fastest changing parameter in the model (for cardiac action potentials, such a parameter would be responsible for the upstroke, typically requiring a 0.01 ms time step). FE is the simplest ODE integrator available and should be used with caution (note, however, that using the same time step, FE is roughly two and four times faster than RK2 and RK4, respectively).

FE, RK2 and RK4 are mainly provided for user convenience. The recommended (and default) ODE integrator is CVODE (figure 4b(ii)). CVODE is suitable for any type of ODE problem, whether it describes slow or fast phenomena. It relies on a variable order and variable time-step technique, making it particularly efficient for the types of models found in the CellML Model Repository. More information about the configuration of CVODE in COR can be found in the COR help or online at http://cor.physiol.ox.ac.uk/Help/.

CVODE is written in C, so we decided to code FE, RK2 and RK4 in that same language and compile all integrators into a DLL (as was done for the expensive mathematical functions mentioned above; see §4a). The two resulting DLLs (compiled using the two stated C++ compilers) are also embedded in the COR executable, and extracted and used as necessary.

This strategy may explain why COR is between approximately 4 and 140 times faster than any other CellML environment tested (Garny et al. 2008). Note that tests were conducted under matching conditions, using the same CellML models, integrator (CVODE) and integration parameters. The performance comparison was based on using the COR-generated MC. Taking advantage of the Intel C++ compiler, the difference would be even greater.

\[\text{(c) Graphical user interface}\]

Figure 7 shows the computational mode as it is currently implemented in COR. As for the editorial mode (see §3b), it consists of three resizable and dockable/undockable sets of panels.

— **Properties viewer.** This contains two types of data (figure 7a): information related to the simulation itself (i.e. its duration and the interval at which simulation data should be generated) and information related to the CellML model (i.e. all the variables that make it up). The variables are displayed using the containment hierarchy specified in the CellML file and are in one of three types: state variable (computed through an ODE); constant; or computed variable (computed through an algebraic equation). State variables and constants can be modified at any point in time during the simulation, allowing the user to experiment with the model. For convenience, each type of variable can also be shown or hidden, as desired.

— **Graph panels.** A graph panel (figure 7b) is used to plot simulation data. By default, one panel is available, but additional ones can be added/removed, as required. A click on a graph panel selects it to allow the user to associate one or several model variables. For example, in figure 7b, the V variable of the membrane component is associated with the top graph panel. This information is
used by COR to track this model variable during a simulation (note that variables can be added/removed at any point). To facilitate the interpretation of simulation data, different features related to the handling of graph panels are available: clearing of contents; zooming in/out; zooming in a particular region; panning of content; extraction of coordinates; and assessment of ‘horizontal’ and ‘vertical’ distances between two points of a given graph. In addition, the contents of a particular graph panel (or that of all the graph panels) can be exported to a comma-separated value (CSV) format. The CSV format is typically used to export simulation data for use in other packages.

— **Console viewer.** This is used to display text information concerning the current simulation (figure 7c). In the case of successful completion, COR displays how long the computation took. Should the simulation fail, COR displays the reason, i.e. either a problem with the model itself and/or too big a time step for the FE, RK2 or RK4 integrator (these will result in the detection of one or several invalid numerical values, i.e. ‘Not a Number’ (NaN) or ‘Infinite’; see below), or a problem with the CVODE integrator (e.g. the requested accuracy is too high).

As indicated above, a simulation can be paused, modified and resumed at any time. Also, all model state variables and/or constants can be reset to their original values. This is the approach that was used to carry out the simulations described in §4d.

*Phil. Trans. R. Soc. A* (2009)
An important part of model development and application is dedicated to ensuring steady-state behaviour. Standard start-up parameters may not describe steady-state behaviour for user-specific applications (figure 7). The usual approach, to account for conditions such as stimulation frequency or ion concentration gradients, is therefore to run the model until it reaches a new steady state, and use the new steady-state parameters as initial conditions. In COR, this is achieved by saving the computed model, either to replace the original model (this requires the original model to be unlocked) or, better, to create a variant of the model.

Another important feature for model development is the ‘debug mode’ available in COR. It can be enabled/disabled at any time, and it can be used to detect NaN and Infinite errors, which are indicative of model implementation problems and/or bad integration parameters. Note that a defective model will not be reported as such if it is not run in debug mode (this is by design and for optimization reasons). Indeed, detection of NaN and Infinite errors is time consuming and not always necessary.

(d) Case study

As mentioned in §2a, COR has been used for a range of different purposes, such as the theoretical assessment of drug actions, including side effects, on cardiac electrophysiology (Noble 2008). For example, it is estimated that approximately 40 per cent of all drugs have (side) effects on the rapid delayed rectifier K⁺ current ($I_{Kr}$; HERG channel), and assessment of the consequences of this for cardiac electrophysiology has become a key requirement for drug safety evaluation. $I_{Kr}$ is one of the key contributors to repolarization, the process by which the membrane potential in excitable cells (such as cardiomyocytes) is returned from excited (approx. +20 mV) to resting (approx. −85 mV) levels.

Computer modelling can be used to study this behaviour, and its contribution to the development of arrhythmias. Thus, one can theoretically evaluate drug effects in a range of clinical conditions (including genetic mutations, changes in internal ionic concentrations and ischaemia). From these simulations, it has emerged that the induction of arrhythmia is dependent on the interplay of external conditions and drug effects, which explains why the same drug can have benign effects on the vast majority of patients, but be detrimental to the health of a minority population. Such small-group side effects, even if concerning only 1 per cent of the population, often lead to the withdrawal of promising candidate drugs, even if primary effects are, in the majority of cases, highly encouraging in clinical trials. Prior identification of high-risk individuals would, therefore, help not only to prevent negative drug effects, but also to retain potentially valuable compounds for clinical use. A systematic study of parameter combinations (genetic background, health status, specific environmental or drug interaction profiles) that have arrhythmic effects would therefore be highly desirable. There are currently over 360 models available in the CellML Model Repository, most of which are cardiac related, and the availability of an environment such as COR makes it relatively easy to compare different species (e.g. human, mouse, rabbit) under different conditions, including drug effects.

A key advantage of COR is speed of computation. Some of the processes that underlie arrhythmia, such as changes in intracellular ion concentrations, develop slowly over time, requiring computations encompassing tens of minutes of heart
activity. COR can process these computations considerably faster than real time (e.g. 30 min achieved within 40 s of computation on a ThinkPad T61p laptop computer; Intel Core 2 Duo CPU T7500 at 2.20 GHz). Therefore, many explorative simulations for large parameter spaces can be carried out within a reasonable time frame. Earlier software tools required days of computation to achieve these results and, in practice, prevented this type of work.

We will illustrate these applications of COR on examples of drugs with multiple sites of action. Such simulations are important because they reveal that drugs with effects on HERG, which in isolation would be arrhythmogenic, can have valuable therapeutic scopes as a consequence of the combined action on multiple targets. Figure 8a shows the detrimental effect of a 70 per cent block solely of $I_{Kr}$, using the Noble 2000 (N2K) guinea pig ventricular cell model. This is an updated version of the model published in 1998 (Noble et al. 1998), in which the voltage-sensitive L-type Ca$^{2+}$ current ($I_{CaL}$) and the Ca$^{2+}$ dynamics have been refined (the N2K model).
model can be downloaded from the CellML Model Repository: http://www.cellml.org/models/noble_2000_version01 for figure 8a,b and http://www.cellml.org/models/noble_2000_version01_variant01 for figure 8c,d). Under control conditions, the membrane potential returns to its resting potential (figure 8a, trace $V_{m,C}$). In the presence of the 70 per cent block of $I_{K_r}$ (figure 8a, trace $I_{K_r,B}$), the membrane potential (figure 8a, trace $V_{m,B}$) does not repolarize completely, and oscillates around the 0 mV potential level. These oscillations are driven by spontaneous reactivation of $I_{Ca,L}$ (figure 8a, trace $I_{Ca,L,B}$) in the presence of incomplete repolarization (caused by the partial block of $I_{K_r}$). Similar behaviour in the whole heart would be associated with severe rhythm disturbances.

The profile of a real compound (BRL32872; Bril et al. 1995; Noble & Colatsky 2000) with multiple actions was simulated next. This drug affects not only $I_{K_r}$ (70% block) but also $I_{Ca,L}$ (20% block). This additional partial block of the key driver of spontaneous membrane potential oscillations allows the virtual cell to resume its normal rhythm (figure 8a, trace $V_{m,D}$).

Another example, simulating the effect of ranolazine (Maltsev et al. 1998; Belardinelli et al. 2006; Maltsev & Undrovinas 2006; Noble & Noble 2006), is illustrated in figure 8b. This compound causes a substantial block of $I_{K_r}$ (90%), combined with a smaller reduction in the persistent Na$^+$ current ($I_{pNa}$, 50%). Again, selective block of $I_{K_r}$ causes incomplete repolarization, while the combined effect (including partial $I_{pNa}$ block) sustains normal activity. These actions of ranolazine are immediate in cells with a normal physiological ion concentration background. However, in sodium-overloaded cells (typical for many pathological conditions associated with metabolic impairment), the beneficial effect of combined drug action develops over time, as it shifts the balance away from intracellular Na$^+$ accumulation.

This can be illustrated using the reverse scenario of subjecting normal cells to altered levels of $I_{pNa}$. Figure 8c,d illustrates how raising the conductance of $I_{pNa}$ changes internal Na$^+$ concentration, which, at unphysiologically high channel conductances, can reach arrhythmogenic levels within minutes (figure 8d). This simulation therefore illustrates how a compound with a pharmacological profile such as ranolazine may have additional cardioprotective effects.

5. Future directions

(a) Operating system

The RedSheriff study mentioned in §2b is now several years old. A more recent Internet usage-based survey by Net Application (August 2008) shows that the share of computers using a Microsoft WINDOWS system was down to 90.7 per cent from 96.4 per cent in October 2004. During the same period, the market share of Mac systems went from 3.2 to 7.9 per cent, and that of LINUX systems from 0.3 to 0.9 per cent (http://marketshare.hitslink.com/report.aspx?qprid=8). This suggests that a cross-platform CellML modelling solution is becoming increasingly desirable.

Implementation of such a solution will inherently be more challenging, involving the use of different compilers, and the end-product may only be able to take advantage of a reduced set of features common to the targeted platforms.
(b) CellML API

As mentioned in §3a, COR currently handles CellML 1.0, while increasingly models adopt the CellML 1.1 standard. At present, the only CellML 1.1-based modelling environment is PCENV (http://www.pcenv.org/). Clearly, support for CellML 1.1 is required to address an increasing demand by modellers and model users, since it supports the import and reuse of model descriptions.

(c) COR and PCENV

COR and PCENV, two ‘pure’ CellML-based environments (see §1c), have similar goals. For this reason, the teams developing both packages decided to work on a combined tool, which will adopt the code base of PCENV and usability options of COR. The code base relies on the technologies made available by the Mozilla Project (http://www.mozilla.org/). These technologies (e.g. XUL, XBL, XPCOM; http://developer.mozilla.org/), together with others (e.g. C++, JavaScript, CSS), are ideally suited for the development of cross-platform solutions. This is in stark contrast to Delphi, used for developing COR, which is suited solely for Microsoft WINDOWS systems (http://www.codegear.com/products/delphi/win32/).

Another advantage of the merger is the fact that this will develop an even closer association with the CellML team, which has released a CellML API that addresses both CellML 1.0 and CellML 1.1 (http://www.cellml.org/tools/api/). As CellML is an evolving standard, there is ongoing work on releasing CellML 1.2 and an updated API. It will be helpful for modellers and model users, if the next generation of CellML modelling environment is able to take early advantage of such developments, or even contribute to the shaping of standards, where applicable.

COR will bring overall user-friendliness, as well as advanced CellML editing facilities (see §3) and techniques for optimized computation and debugging of CellML models (see §4a–c).

(d) ENCORE, a prototype for merged PCENV/COR functionality

The plan for the move to PCENV/COR builds on the experience of the two development teams and anticipates a three-level modelling environment, where users can organize their CellML files, edit them and run simulations. This finds a reflection in the prototype GUI put together, codenamed ENHANCED COMPUTING OPEN RESOURCE (ENCEORE; figure 9).

(e) Session file

An important concept, initially implemented in PCENV, is that of a session file. A session file provides the user with a model-specific and interactive environment. This combines use of CellML Graphing Metadata (http://www.cellml.org/specifications/metadata/graphs/; Garny et al. 2008; Nickerson et al. 2008; Beard et al. 2009), scalable vector graphics (SVG) diagrams and JavaScript. These files have been specifically created for various CellML models and can be downloaded from the CellML Model Repository. This successful concept will be expanded, so that session files will be automatically created when exiting ENCORE. This will allow users to retrieve an environment identical to the session they left. They could, however, still start ENCORE by loading a default session file or one that was created at some other point in time.
Figure 9a shows the proposed GUI features of ENCORE for organizing CellML files. These include the following.

(f) Organizer

Figure 9a shows the proposed GUI features of ENCORE for organizing CellML files. These include the following.

— (CellML Model) Repository explorer. This will offer a ‘direct’ link to the online CellML Model Repository. It will allow the user to browse and query the repository, as well as to download and upload models. A query for $I_{\text{CaL}}$, for example, will not only return matches for $I_{\text{CaL}}$, but also for Cav1.3 (a gene nomenclature-based synonym), using a pre-existing ontology feature. Also, with the move towards CellML 1.1, import and reuse of model components and unit definitions will be possible, and will require the automatic mapping of model variables, conversion of units (Cooper & McKeever 2008), etc.

— File explorer. This will offer consistency with the repository explorer and allow users to access their local CellML files in the way they would normally do, using any file explorer or manager.

— Models (organizer). This will be used to organize CellML models independently of their physical location. For example, one might wish to physically arrange CellML files based on the species that they represent (e.g. human, mouse, rabbit), or author team, while within ENCORE one may prefer a model arrangement by cell type (e.g. atrial, ventricular), to reflect proximity of model features and behaviour, and to aid component import/export.
In addition to the above, it will be possible to drag and drop CellML files between the two explorers (i.e. download/upload), the organizer and the editor (see below). A feature that is not currently available in either COR or PCENV is to see, at a glance, the differences between two models. The raw CellML format could be used for this, but a readable format (as the one used in COR) will be more relevant for most users in this context (see §3a). This functionality will be made available within both the organizer and the editor.

(g) Editor

The editing functionality of ENCORE (figure 9b) will expand what has been provided in COR, and it includes the following.

— **Viewer.** See the description of the command viewer in §3b.

— **Metadata.** These have become an essential component of CellML (Beard et al. 2009), so ENCORE will offer a user-friendly way to create and edit metadata. This will be done using either raw data or via a dedicated viewer (in case the type of the metadata is known). There are emerging standards for CellML Simulation Metadata and CellML Graphing Metadata, for which dedicated editing facilities can be provided (http://www.cellml.org/specifications/metadata/simulations/ and http://www.cellml.org/specifications/metadata/graphs/, respectively; Nickerson et al. 2008). It is not clear at this time whether the editing of metadata will best be done independently of CellML file editing, or as part of this process (see the next point).

— **Editor.** This will be based on the editing facility in COR (see §3b), i.e. a tabbed editor that allows several CellML files to be opened at the same time. One possibility for making the editor user-friendly is the use of Scintilla (http://www.scintilla.org/), a powerful editing component that supports syntax highlighting, code folding, bookmarks, view splitting, etc. Otherwise, in addition to viewing a CellML file using the readable format developed for COR (see §3a), it will be possible to view a CellML file using the raw CellML format, a tree-like structure (inherited from PCENV), a pure mathematical approach, a graphical approach, as well as other formats that can be incorporated via a plug-in system. In the case of a pure mathematical approach, CellML concepts such as units, components, groups, etc. will be undefined. If one interprets CellML as a format ‘only’, one should allow mathematical equations to be entered in a way similar to what would normally be done on paper. It would then be up to the software to create the necessary units, components, groups, etc. to make the model CellML compliant. The graphical approach will benefit from the work of the *INSILICO* IDE team (http://www.physiome.jp/). In all cases, it will be possible to add comments to a model, using metadata.

— **Messages.** This will work similar to the messages viewer described in §3b.

The export facility in COR has already proven useful, but it will be improved by supporting lookup tables and partial evaluation (Cooper et al. 2006).

Also, as cell models increase in complexity and, in particular, include cellular compartments and even spatial concentration gradients (in ions or second messengers), it will be important to display the structure of the models using...
three-dimensional graphical representations. A markup language standard for encoding spatially varying fields, called FieldML, has been established (Christie et al. 2009) and an open-source graphical software package for displaying FieldML files, called CMGUI, is freely available (http://www.cmiss.org/cmgui/). We intend to incorporate the CMGUI code into EnCORE in order to support rendering of three-dimensional spatial representations that are coupled to CellML models. For example, the availability of structural detail at a resolution of 3–5 nm for cells from electron tomography, and the overlaying of protein distribution data into three-dimensional cell geometric models generated by these data, should greatly enhance our understanding of cellular compartmentalization and signalling.

(h) Simulator

The simulation part of EnCORE is illustrated in figure 9c and will include the following.

— *Simulation*. This will be used to specify the simulation parameters (see the description of the properties viewer in §4c).

— *Properties*. These will be identical to the model variables part of the properties viewer (see §4c).

— *Results*. The functionality of the graph panels in §4c will be extended to allow plotting of simulation data against any model variable (and not only as a function of time). This feature, which is already available in PCENV, involves the tracking of every model variable, and will therefore have to be optional, as it will inevitably slow down the computation and increase the memory requirements (which may be a disadvantage for long simulations; see §4d).

— *SVG*. As mentioned earlier, PCENV supports session files, which include SVG diagram options. Thus, as for graph panels, the user will be able to create one or more links to SVG diagrams, as well as embed JavaScript code to make them interactive.

— *Console*. See the description of the console viewer in §4c.

As for the organizer and editor, it will be useful to monitor differences between two simulations. PCENV currently allows plotting of simulation data from two different simulations. This will be expanded to allow the rendering of combined variables, e.g. $I_{CaL} + I_{Kr} - I_{pNa}$ (as is already possible in JSim).

The debug mode available in COR (see §4c) will be retained and expanded by allowing the user to ‘trace into the model’, i.e. rather than computing the whole model at once, it will be possible to compute one equation at a time, allowing the user to see the effects of that particular equation on overall model behaviour.

Another feature that will also help model developers, as well as users, is the possibility to ‘hijack’ a variable by applying any protocol to it. We could, thus, create a current–voltage curve for the N2K model (see §4d) by hijacking its membrane potential and all of its concentrations. Its membrane potential would be subjected to a linear voltage clamp protocol between the values of, say, −100 and +50 mV, while all ion concentrations would be clamped to their initial value.

*Phil. Trans. R. Soc. A* (2009)
6. Conclusion

Thanks to its usability and speed, COR has proven successful in getting a wide range of users to create, edit, simulate and exchange models described in the CellML 1.0 format. It has also been instrumental in curating the CellML files available in the CellML Model Repository. However, the current trend towards import and reuse of model descriptions (possible only with CellML 1.1), combined with the desirability of a multi-platform solution, indicates scope for improvement. Both of these aspects are already addressed by PCEnv. As COR and PCEnv have similar goals, and complementary strengths, a merger between the two environments is expected to provide modellers, experimentalists, teachers and students with a powerful, up-to-date, user-friendly and cross-platform CellML-based model development and simulation environment.

This study has been supported by a grant from the UK Biotechnology and Biological Sciences Research Council (BB/E024955/1). We acknowledge further support by the European Community. P.K. is a senior research fellow of the British Heart Foundation.

References


