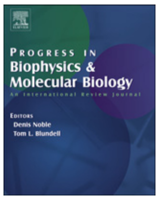
Progress in Biophysics and Molecular Biology 107 (2011) 32–47

Contents lists available at ScienceDirect

Progress in Biophysics and Molecular Biology

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



Original Research

OpenCMISS: A multi-physics & multi-scale computational infrastructure for the VPH/Physiome project

Chris Bradley[a,b,*], Andy Bowery[c], Randall Britten[a], Vincent Budelmann[a], Oscar Camara[d,e], Richard Christie[a], Andrew Cookson[f], Alejandro F. Frangi[d,e], Thiranja Babarenda Gamage[a], Thomas Heidlauf[g], Sebastian Krittian[c], David Ladd[a], Caton Little[a], Kumar Mithraratne[a], Martyn Nash[a,h], David Nickerson[a], Poul Nielsen[a,h], Øyvind Nordbø[i], Stig Omholt[i], Ali Pashaei[d,e], David Paterson[b], Vijayaraghavan Rajagopal[a], Adam Reeve[a], Oliver Röhrle[g], Soroush Safaei[a], Rafael Sebastián[j], Martin Steghöfer[d,e], Tim Wu[a], Ting Yu[a], Heye Zhang[k], Peter Hunter[a]

[a] *Auckland Bioengineering Institute (ABI), The University of Auckland, New Zealand*
[b] *Department of Physiology, Anatomy and Genetics (DPAG), University of Oxford, United Kingdom*
[c] *Computing Laboratory, University of Oxford, United Kingdom*
[d] *Center for Computational Imaging and Simulation Technologies in Biomedicine (CISTIB), Universitat Pompeu Fabra, Barcelona, Spain*
[e] *Networking Biomedical Research Center – Bioengineering, Biomaterials and Nanomedicine (CIBER-BBN), Barcelona, Spain*
[f] *Imaging Sciences & Biomedical Engineering Division, King's College London, United Kingdom*
[g] *Stuttgart Research Centre for Simulation Technology, University of Stuttgart, Germany*
[h] *Department of Engineering Science, The University of Auckland, New Zealand*
[i] *CIGENE, Department of Animal and Aquacultural Sciences, Norwegian University of Life Sciences, Aas, Norway*
[j] *Department of Computer Science, Universitat de Valencia, Spain*
[k] *Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China*

A R T I C L E I N F O

Article history:
Available online 7 July 2011

Keywords:
Computational modelling software
Multi-scale
Multi-physics
Physiome project

A B S T R A C T

The VPH/Physiome Project is developing the model encoding standards CellML (cellml.org) and FieldML (fieldml.org) as well as web-accessible model repositories based on these standards (models.physiome.org). Freely available open source computational modelling software is also being developed to solve the partial differential equations described by the models and to visualise results. The OpenCMISS code (opencmiss.org), described here, has been developed by the authors over the last six years to replace the CMISS code that has supported a number of organ system Physiome projects.

OpenCMISS is designed to encompass multiple sets of physical equations and to link subcellular and tissue-level biophysical processes into organ-level processes. In the Heart Physiome project, for example, the large deformation mechanics of the myocardial wall need to be coupled to both ventricular flow and embedded coronary flow, and the reaction–diffusion equations that govern the propagation of electrical waves through myocardial tissue need to be coupled with equations that describe the ion channel currents that flow through the cardiac cell membranes.

In this paper we discuss the design principles and distributed memory architecture behind the OpenCMISS code. We also discuss the design of the interfaces that link the sets of physical equations across common boundaries (such as fluid-structure coupling), or between spatial fields over the same domain (such as coupled electromechanics), and the concepts behind CellML and FieldML that are embodied in the OpenCMISS data structures. We show how all of these provide a flexible infrastructure for combining models developed across the VPH/Physiome community.

1. Introduction

The last few decades have witnessed a remarkable increase in the computational power that is available to scientists and engineers. The increased power has enabled mathematical and computer modelling of physical systems to advance from looking at

* Corresponding author. Auckland Bioengineering Institute, The University of Auckland, Private Bag 92019, Auckland 1142, New Zealand. Tel.: +64 9 3737599x89924; fax: +64 9 3677157.
E-mail address: c.bradley@auckland.ac.nz (C. Bradley).

0079-6107/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.pbiomolbio.2011.06.015

simple phenomenon in isolation to the analysis of complex coupled multi-scale and multi-physics processes. The increase in model complexity has driven a corresponding increase in the complexity of the computer codes that solve the models. As modern scientific modelling codes take a great deal of effort to develop, test, and maintain, there has been an emerging trend towards collaborative efforts to develop software libraries which can be used by many groups. For the case of the finite element method a number of libraries have been developed. Some of the libraries aim to be general purpose e.g., libMesh (Kirk et al., 2006), deal.II (Bangerth et al., 2007), FETK (Holst, 2001) and OOFEM (Patzák and Bittnar, 2001; Patzák et al., 2001). Other libraries are more specific to a particular scientific area e.g., in the field of cardiac modelling there are, to name a few, Continuity (www.continuity.ucsd.edu), CARP (Vigmond et al., 2003) and Chaste (for Cancer and Cardiac applications) (Pitt-Francis et al., 2009). The computational size of complex coupled models have meant that the computational libraries have increasingly used parallel processing to reduce the run time of the simulations (Bordas et al., 2009).

The VPH/Physiome[1] project is an international collaborative effort to understand both the physiology and pathology of the human body using quantitative, anatomically and biophysically based models that link clinical data at the organ level (e.g., from MRI) to patient-specific molecular data. The complexity and range of spatial and temporal scales involved in this project call for robust data and modelling standards. The development of such standards is described briefly below. Model repositories and visualisation tools, based on these standards, have also been established. The purpose of this paper is to describe the development of a computational software library, called OpenCMISS, which uses the models described by the VPH/Physiome standards for solving coupled biophysically based equations that incorporate multiple spatial and temporal scales.

We describe the design goals for the software and illustrate the data structures with a geometrically simple example, relevant to the cardiac physiome project, which couples biophysical equation sets both within and across tissue regions, and in some cases linking down to cell level ODE models.

2. Physiome standards

As the computational models inevitably become more complex, it is increasingly difficult for anyone other than the author(s) of the publication describing the model to decipher, code, and run the model in order to reproduce the results claimed in a publication. It can also be very difficult to then use this model as one component of a more complex model.

To address these challenges several groups have developed standards for encoding models over the past ten years. These modelling standards typically use the eXtensible Markup Language (XML) developed by the worldwide web consortium (w3c.org), as well as a variety of other standards based on XML, such as MathML for encoding mathematics, and various metadata standards.

Two XML-based model encoding standards are currently being developed under the IUPS Physiome Project (Hunter and Nielsen, 2005; Hunter, 2004; Hunter and Borg, 2003) and the European Virtual Physiological Human (VPH) project (vph-noe.eu). CellML (cellml.org) is designed to encode lumped parameter biophysically based systems of ordinary differential equations (ODEs) and nonlinear algebraic equations – together called differential algebraic or DAE systems (Cuellar et al., 2003). FieldML (fieldml.org) is designed to encode spatially and temporally varying field information such as anatomical structure, the spatial distribution of protein density or computed fields such as the electrical potential or oxygen concentration throughout a tissue (Christie et al., 2009). A third markup language called the ‘systems biology markup language’ or SBML (sbml.org) has been developed by the systems biology community. This has similar expressiveness to CellML but is targeted more specifically to representing models of biochemical and genetic networks.

CellML separates the syntax of a model (e.g., the mathematical equations encoded in MathML) and the semantics (the biological and biophysical meanings of the model components and parameters) defined in the model metadata by reference to suitable ontologies. This facilitates building complex models by importing modular components defined in libraries. SBML is more closely tied to the concepts of biochemical and genetic networks. FieldML deals with the encoding of fields, such as geometry and stress, at multiple spatial scales by allowing hierarchies of material coordinate systems that preserve anatomical relationships (e.g., coronary arteries embedded in a deforming myocardial tissue that is itself part of a heart contained within a torso). These three standards are supported by the US National Institutes of Health (NIH) and the European Commission’s funding agency (currently operating under Framework 7).

Model repositories are available for all three markup languages – PMR2 (Physiome Model Repository 2, see models.cellml.org) for CellML (Lloyd et al., 2008) and FieldML models and Biomodels (biomodels.net) for SBML models. Various minimum information standards are also available including MIRIAM (ebi.ac.uk/miriam) and MIASE (ebi.ac.uk/compneur-srv/miase).

3. Background to CMISS

CMISS is an acronym for ‘Continuum Mechanics, Image analysis, System identification and Signal processing’. Code development for CMISS began in Auckland in 1980 with the goal of creating a bioengineering finite element code for solving multiple coupled partial differential equations. The initial applications were for the heart, primarily for electromechanics (Hunter and Smaill, 1988; Hunter et al., 2003), but it rapidly expanded to include other heart processes (Smith et al., 2004) and to support applications in other organs such as the lungs, with coupling between soft tissue mechanics, heat and moisture transport and airway fluid mechanics (Tawhai et al., 2000, 2004). While finite element methods provided the primary numerical technique, the code was developed in the 1980s to include boundary element methods (for electrical current flow in the thorax) and finite difference equations based on curvilinear grids. The visual display and graphical user interface aspects of the program were developed in parallel in a program called CMGUI (cmiss.org/cmgui) (Christie et al., 2002).

Some of the unique features of CMISS (in comparison to commercial finite element codes) that have been carried over into OpenCMISS are: (i) the use of cubic Hermite basis functions to preserve C^1 or G^1 continuity across element boundaries (Bradley et al., 1997) where this is important for efficient field representation; (ii) the use of variable order basis functions (e.g., bicubic Hermite by linear Lagrange); and (iii) the close relationship that is preserved between the representation of tissue structure and the organ anatomy, through the use of material structure fields defined with respect to embedded material coordinates.

[1] The Physiome Project was begun by the International Union of Physiological Sciences (IUPS) in 1997 to provide an integrative model-based framework for understanding human physiology. The Virtual Physiological Human (VPH) project was initiated in 2006 by the European Commission with a focus on clinical applications of the physiome project. These two projects have now merged into the VPH/Physiome project.

The CMISS code (including CMGUI) has been at the heart of many multi-scale physiome projects over the past 20 years, but as the VPH/Physiome modelling standards evolved it became clear that a redevelopment of both programs was needed to take advantage of both the new modelling standards and repositories, and the increasing move (in the academic world at least) to freely available open source software. The OpenCMISS project was therefore begun in 2005 as an open source collaboration between the University of Auckland in New Zealand and the University of Oxford in the UK. This collaboration was extended last year to include King's College London, Universitat Pompeu Fabra, Barcelona, Spain, the Norwegian University of Life Sciences and the University of Stuttgart, Germany. This year, the Shenzhen Institute of Advanced Technology, in Shenzhen, China, has also joined.

4. Design goals

The first goal was that OpenCMISS would be a flexible library rather than a large monolithic application. A library-based code means that it is considerably easier to incorporate physiome and bioengineering models into clinical or commercial applications as a library that can be wrapped by a customised interface. The library should be modular, extensible, and programmable. This allows for the library itself to be customised and/or extended in whatever way is appropriate for the end application.

The second design goal was generality. Previous experience with the CMISS modelling environment indicated the importance of developing code in as general a way as possible. Generalised data structures, in which the data for diverse modelling problems are expressed in a common format, allow for easier coupling between different problems. This is especially true for unforeseen, coupled problems that may arise from future applications. The goal of generality does, however, often mean that there is some trade-off with the computational performance of code. As the computational size of bioengineering models can be very large it is extremely important that computational performance is carefully considered. But it is our view that it is better to optimise a more general code armed with the knowledge of exactly what the problem is than to prematurely optimise a specific code which could then limit the applicability of that code.

The third design goal was that OpenCMISS should be an inherently parallel code and that the parallel environment should be as general as possible. Parallel processing is required as the computational demands of solving models increases due to increased resolution or complexity of the models. However, optimal parallel processing strategies depend on the particular problem being solved. Also the lifetime of modelling codes is often an order of magnitude greater than the lifetime of the computer hardware, and it is notable that the architecture of parallel machines has changed over the last few decades from vector processors, to symmetric multiple processors (SMPs), to clusters of processors, to clusters of multiple core processors, through to using General Purpose Graphical Processing Units (GPGPUs). Code that assumes a particular parallel algorithm or a particular parallel architecture may not be appropriate for a future problem or future parallel hardware. For these reasons a design goal of OpenCMISS was that the code uses a general $n \times p(n) \times e(p)$ hierarchical parallel environment where n is the number of computational nodes, $p(n)$ is the number of processing systems on the computational node and $e(p)$ is the number of processing elements for each processing system. Examples of this hierarchy are:

a) multi-core or SMP $n = 1$, $p(n) > 1$, $e(p) = 1$
b) pure cluster $n > 1$, $p(n) = 1$, $e(p) = 1$
c) multi-core cluster $n > 1$, $p(n) > 1$, $e(p) = 1$
d) multi-core cluster with GPUs $n > 1$, $p(n) > 1$, $e(p) > 1$

The fourth design goal was that OpenCMISS should be able to be used, understood, and developed by novices and experts alike. Modern bioengineering and physiome science requires a team of scientists, graduate students, and post-doctoral researchers from varied backgrounds, each with a different skill set. It is unrealistic to expect that each member of the team will become an expert in every area of modelling and computation. The design of OpenCMISS thus abstracts and encapsulates model details in a number of objects of hierarchical complexity. The hierarchy of these objects allows complex details to be hidden from the users, if required, and the object interface allows an expert to manipulate object parameters whilst the novice user makes use of sensible default parameter values for the common cases.

The final design goal, as mentioned earlier, was to incorporate Application Programming Interfaces (APIs) for the physiome markup languages CellML (Miller et al., 2010) and FieldML.

5. Software systems

OpenCMISS is written in Fortran 95/2003 with an object-based approach for high level objects. It has bindings for Fortran and C and uses SWIG (swig.org) interfaces for C++ and Python. It uses the Mozilla trilicense (mozilla.org/MPL) that is being used for other open source Physiome projects. Standard software engineering practices are followed, including the use of a source code repository on SourceForge (sourceforge.net/projects/opencmiss), Doxygen for documentation, testing via a Buildbot system, validation against analytic test cases, and a tracker system – all described further below.

Note that although the main source code revision control system for OpenCMISS is hosted on SourceForge using Subversion, we are currently evaluating distributed version control systems (DVCS's) such as Git or Mercurial, with the possibility of migrating from Subversion in the future. One perceived advantage of distributed version control systems is the fit with a research-oriented development model, allowing for code to be written for research that is destined to eventually be released as open source, but kept closed-source initially until the corresponding research articles have been published. DVCS's allow for version control and collaboration during the closed phase, and when released as open source, the original version history can be kept intact, or possibly summarised.

Other SourceForge services used are:

- a wiki (sourceforge.net/apps/mediawiki/opencmiss/index.php?title=Main_Page)
- mailing lists (sourceforge.net/mail/?group_id=201176)
- a web-based source code revision viewer using Trac (sourceforge.net/apps/trac/opencmiss)

Note that the source code revision viewer allows views of the code revision history via a web browser, and also provides an RSS feed of code changes. A mirror of the Trac revision viewer is also available, and is hosted at the ABI (svnviewer.bioeng.auckland.ac.nz/projects/opencmiss).

For project planning, the OpenCMISS project uses the Physiome project issue tracker (tracker.physiomeproject.org). The Bugzilla tracker system is used for managing both feature planning and bug reporting. A draft version of the OpenCMISS development plan is stored in the issue tracker database. The tracker allows tracker items to have dependencies on other tracker items, and these can be viewed by means of interactive web-based tree views and graph views. This allows for dynamic planning, where the views of the plan are updated instantly as each status is updated. Modifications to plan are also instantly visible. Stakeholders have the option of receiving status updates via customisable RSS feeds or by means of customisable e-mail alerts.

The OpenCMISS project makes extensive use of example programs. These examples serve a number of purposes including demonstrating the capability of OpenCMISS, documenting how to solve certain equations and as a means for testing and validating the code. The set of examples is currently stored in the OpenCMISS source repository, but this will most likely change in the near future so that example models and their field descriptions are hosted as part of a physiome model repository. OpenCMISS examples undergo an extensive validation process in which example solutions are compared with an analytic solution to the problem (if available). The example solutions are checked for convergence and tested in parallel. Once the example has demonstrated that it is solving correctly it is then tested nightly against the current code using a Buildbot testing system. The BuildBot automated testing system for OpenCMISS (autotest.bioeng.auckland.ac.nz) is a web-based system used for automated daily building, testing, and documentation generation. BuildBot provides a web-based configuration facility, and views of current and historical test results. These results are also available via e-mail alerts and an RSS feed.

The generated documentation is updated daily to match the source code head revision. The Doxygen documentation system is used to extract specially formatted comments from the source code and also parses the source code structure and generates documentation targeted at developers using the OpenCMISS library, as well as developers contributing to the OpenCMISS project (available from cmiss.bioeng.auckland.ac.nz/OpenCMISS/doc/programmer).

A second Subversion system svn.physiomeproject.org/svn/opencmissextras, hosted by the ABI, is used to assist developers in setting up a development environment for OpenCMISS, mainly by facilitating retrieval of compatible versions of the third-party libraries and tools on which OpenCMISS depends.

Finally, the main OpenCMISS website (opencmiss.org) uses the ModX content management system.

6. Open source libraries

OpenCMISS builds on a number of successful software projects used in the modelling community. In accordance with an open source design philosophy, OpenCMISS aims to use software libraries that are, where possible, themselves open source. In cases where a library is not open source, that library and its functionality are considered optional so that it is possible to build a completely open-sourced product.

For parallel computations in a heterogeneous multiprocessing environment OpenCMISS uses the MPI standard (mpi-forum.org) for distributed parallelisation and the OpenMP (openmp.org) standard for shared memory parallelisation. OpenCMISS has been tested with the MPICH2 (mcs.anl.gov/research/projects/mpich2), Open MPI (open-mpi.org), MVAPICH (mvapich.cse.ohio-state.edu) open source MPI libraries as well as vendor specific MPI libraries from Intel and IBM. Recently, OpenCMISS developers have been investigating parallel computations on GPGPUs using CUDA (nvidia.com/object/cuda_home_new.html). However, as CUDA is a proprietary technology, developers have started to consider OpenCL (khronos.org/opencl) for programming GPGPUs.

In order to calculate optimal mesh partitions, OpenCMISS uses the parallel graph partitioning package ParMETIS (glaros.dtc.umn.edu/gkhome/metis/parmetis/overview). The licensing options for ParMETIS allow it to be used freely only for educational and research purposes by non-profit institutions. For situations where this copyright is too restrictive OpenCMISS also uses Scotch (gforge.inria.fr/projects/scotch) for graph partitioning.

For numerical solvers, OpenCMISS makes use of a number of third-party libraries. Particular use is made of libraries developed through the US Department of Energy SciDAC (scidac.gov), TOPS (scidac.gov/math/TOPS.html) and ACTS (acts.nersc.gov) projects. For linear and nonlinear system solvers OpenCMISS uses PETSc (mcs.anl.gov/petsc) for iterative Krylov sub-space linear system solvers. In addition a number of direct linear solvers such as MUMPS (graal.ens-lyon.fr/MUMPS), SuperLU_DIST (crd.lbl.gov/~xiaoye/SuperLU) and PaStiX (gforge.inria.fr/projects/pastix) are available in OpenCMISS through a PETSc interface. OpenCMISS also uses

- SUNDIALS (computation.llnl.gov/casc/sundials/main.html) for differential–algebraic equations
- Hypre (computation.llnl.gov/casc/hypre/software.html) for preconditioners
- TAO (mcs.anl.gov/research/projects/tao) for optimisation
- SLEPc (grycap.upv.es/slepc) for eigenproblems
- BLACS (netlib.org/blacs) and ScaLAPACK (netlib.org/scalapack) for linear algebra

For input and output OpenCMISS uses the CellML (cellml.org) and FieldML (fieldml.org) APIs. FieldML will ultimately use standard libraries such HDF5 (hdfgroup.org/HDF5) and/or NetCDF (unidata.ucar.edu/software/netcdf) for parallel I/O of large data sets.

7. Multi-physics modelling

The major features in OpenCMISS for dealing with coupled multi-physics models include a flexible system for describing multiple physical models and complex problem workflows, methods for coupling different physical systems together and the ability to handle different spatial and temporal scales using FieldML concepts and CellML models.

In order to provide a general and reusable framework OpenCMISS holds each different set of physical equations (and the data for those equations) in a separate object within the code base. For example, if a coupled system of fluid and solid mechanics was being considered, the equations for the fluid part would be in a separate object to the solid part. Each set of physical equations can then be considered and constructed independently. OpenCMISS further abstracts the coupling process by separating the sets of equations from the solver libraries that perform numerical and computational operations on those equations. The coupled set of equations which can be used with a solver is formed by adding in each individual set of equations and any necessary coupling equations (see below) to the solver equations. This system for coupling equations has the advantages that each physical set of equations can be considered independently, without the complexity of the final coupled equations, and that the solvers can consider just the numerical equations independently of the actual source of the equations. As coupled systems often require a number of different computational operations, OpenCMISS allows for a flexible system to describe the problem workflow in terms of the solvers of the coupled systems.

Coupling of different physical models can occur in a number of ways. Different physical models can be coupled in the same region of interest (e.g., a coupled system of partial differential equations) or in different regions of interest (e.g., a fluid–solid interaction system in which the fluid is in one region and the solid in another). Coupling can also occur between the solvers in a problem workflow (e.g., iterating between two solvers until convergence is reached).

OpenCMISS couples physics in the same region by using a consistent FieldML description of each physical problem and allowing for coupled equations through the sharing of common field variables. Because the data about each problem is stored using the same data structures, the individual equations for each problem can be formulated using variables from different problems as easily as they can be formulated using variables from their problem.

In OpenCMISS coupling between different regions can be accomplished using a number of different methods to relate the equations. These methods impose compatibility conditions between the fields in each region. The conditions are termed interface conditions in OpenCMISS. The simplest interface/compatibility condition is to allow the degrees-of-freedom (DOFs) of the equations in one region to be linearly related to the DOFs in another set of equations in another region. This linear coupling allows for the interface condition between the equations to be imposed directly at each DOF point. In OpenCMISS this is known as a strong interface condition in the sense that it holds at a number of points on the interface. This strong interface condition allows for a coupled set of equations that govern the combined physics to be formed through row and column manipulations of the individual sets of equations. As inter-region coupling involves different regions, the strong interface condition is defined by using explicit interfaces. These interfaces ensure that information is passed between regions in a controlled manner.

Other inter-region coupling methods allow for the interface conditions to be imposed in an integral sense. Examples of these methods supported by OpenCMISS include Lagrange, augmented Lagrange, and penalty methods. Interface conditions from these methods are termed weak interface conditions in that the compatibility condition does not necessarily hold at any particular point in the interface, but rather that the condition holds in the average or integral sense (Nordsletten et al., 2010).

The final key feature for multi-scale, multi-physics problems in OpenCMISS is the use of CellML and FieldML. CellML models allow for the physics at a small spatial scale to be abstracted and viewed as a single point model within another model at a larger spatial scale. OpenCMISS also allows for CellML models to be evaluated and integrated at a different temporal scale than other models. Further spatial and temporal scales can be incorporated into OpenCMISS problems using hierarchical field concepts from FieldML which allow data to be viewed at different levels.

8. OpenCMISS data structures

We illustrate the key FieldML data structures and workflows implemented in OpenCMISS with a geometrically simple example that couples six systems of equations, each representing a different physical process. The example couples the equations from these physical processes, both within and across their spatial domains and across spatial scales. It provides a simple data structure prototype for the anatomically based electromechanical heart modelling in the cardiac physiome project. The physical regions are shown in Fig. 1.

The key OpenCMISS/FieldML concepts of ***regions***, ***meshes***, ***decomposed domains***, ***fields***, ***equation sets***, ***equations***, ***problems***, ***control loops***, ***solvers***, ***solver equations***, and ***distributed vectors and matrices*** will be discussed below with reference to this example.

The physical processes are described briefly below. Note that the equations are given here in their ‘strong’ partial differential equation (PDE) form to highlight the fields needed for their representation in OpenCMISS. The equations solved within OpenCMISS are ‘weak’ or integral equation versions of these PDEs in keeping with the standard finite element Galerkin approach. The references quoted for each equation set give further details including, in most cases, the weak form of the equations.

1. ***Large deformation soft tissue mechanics*** on mesh 1 of region 1. Finite elasticity theory (Malvern, 1969; Nash and Hunter, 2000; Oden, 2006) with incompressible, inhomogeneous, nonlinear, anisotropic material properties, gives the following governing equations, based on conservation of linear momentum and conservation of mass:

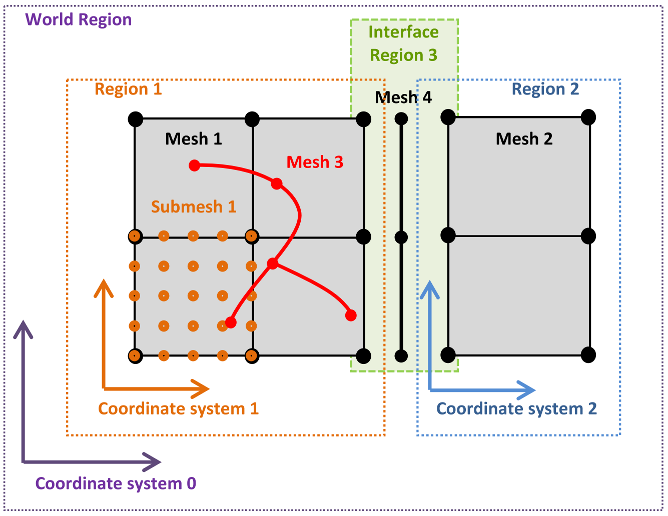


Fig. 1. A geometrically simple example to illustrate the use of regions, coordinate systems and meshes for solving a coupled multi-physics, multi-scale problem in OpenCMISS.

$$\partial_t \rho \boldsymbol{v} + (\boldsymbol{v} - \boldsymbol{w}) \cdot \nabla_{\mathbf{x}} \boldsymbol{v} = \nabla_{\mathbf{x}} \cdot \boldsymbol{\sigma} + \boldsymbol{f} \tag{1}$$

where ρ is material density, $\boldsymbol{v}$ is the material velocity vector, $\boldsymbol{\sigma}$ is the Cauchy stress tensor, $\boldsymbol{f}$ is the force per unit mass, and $\boldsymbol{w} = v$ or $\boldsymbol{w} = \mathbf{0}$ for the Lagrangian or Eulerian forms of the equations, respectively. Note that the Cauchy stress tensor $\boldsymbol{\sigma}$ is related to the 2nd Piola-Kirchhoff stress tensor $\boldsymbol{T}$ by $\boldsymbol{\sigma}=J^{-1}\boldsymbol{FTF}^T$, where $\boldsymbol{F}$ is the deformation gradient tensor and $J = det\boldsymbol{F}$. An additional constraint equation is required to solve Eq. (1). The constraint equation that enforces incompressibility is $J = det\boldsymbol{F} = 1$. Further details are given in Nash and Hunter (Nash and Hunter, 2000). Note also that $\boldsymbol{T}$ is the most appropriate form of stress tensor for defining material properties (being the energy conjugate of the Green-Lagrange strain tensor) and is defined as a function of the strain energy function W and tissue pressure p by

$$\boldsymbol{T} = \frac{1}{2}\left(\frac{\partial W}{\partial E} + \frac{\partial W}{\partial E^T}\right) - p\boldsymbol{C}^{-1} + \boldsymbol{T}_{\text{active}} \tag{2}$$

where $\boldsymbol{E}$ is the Green-Lagrange strain tensor and $\boldsymbol{C}^{-1}=(\boldsymbol{F}^{\boldsymbol{T}}\boldsymbol{F})^{-\mathbf{1}}$ is the inverse of the right Cauchy strain tensor (and is the contravariant metric tensor for the deformed state of the tissue). This constitutive relation is specified in a CellML model, where for soft tissues W is a nonlinear function of the components of $\boldsymbol{E}$ and defines an anisotropic stress–strain relation. For problems in which an actively contracting material is deforming (e.g., a contracting heart) the stress tensor is modified by introducing an active stress term. The active stress comes stress $\boldsymbol{T}_{\text{active}}$ (expressed here as a 2nd Piola-Kirchhoff stress) comes from a CellML encoded model (Hunter et al., 1998; Niederer et al., 2006) in which fibre strain, obtained from the deformation field, is used in a nonlinear integral equation model to update the active stress on the fibre direction.

Typical boundary conditions for large deformation mechanics problems include specifying an appropriate combination of displacements and normal Cauchy stress on a boundary surface. For further details see Nash and Hunter (Nash and Hunter, 2000). Initial conditions must also be specified for the time-dependent formulation given in Eq. (1). For details of the initial conditions see Nordsletten et al. (Nordsletten et al., 2010).

The dependent variable fields are the material displacement vector $\boldsymbol{u}$ and, if the tissue is treated as incompressible, the scalar tissue pressure field p (effectively a Lagrange multiplier associated with the incompressibility constraint).

For further details on the formulation of the finite elasticity (and coupled fluid flow) equations in OpenCMISS see Nordsletten et al. (Nordsletten et al., 2011).

2. ***3D fluid flow*** on mesh 2 of region 2. The 3D Navier–Stokes equations (Batchelor, 2000; Currie, 2002; Nordsletten et al., 2011) are

$$\partial_t \rho \boldsymbol{v} + (\boldsymbol{v} - \boldsymbol{w}) \cdot \nabla_{\mathbf{x}} v = -\nabla_{\mathbf{x}} p + 2\nabla_{\mathrm{x}} \cdot \mu \boldsymbol{D} + \boldsymbol{f}, \tag{3}$$

$$\nabla_{\mathbf{x}} \cdot \boldsymbol{v} = 0 \tag{4}$$

Where $\boldsymbol{D}$ is the Eulerian strain rate tensor and $\boldsymbol{w}$ is the mesh velocity. The dependent variable field components are the fluid velocity vector $\boldsymbol{v}$ and the hydrostatic pressure field p.

Typical boundary conditions for 3D fluid flow include a Dirichlet specification of the fluid velocity at inlet and outlets, specification of zero tangential velocity at walls, Dirichlet specification of fluid pressure at inlets and outlets or of zero pressure for free surface flows. Typical initial conditions are the specification of the fluid velocity in the domain. For further information see Nordsletten et al. (Nordsletten et al., 2010, 2011).

Typical interface conditions for fluid–solid interaction are that the fluid velocity is equal to the rate of change of displacement of the solid and that there is continuity of traction at the fluid–solid interface (Nordsletten et al., 2011).

3. ***1D fluid flow*** on mesh 3 of region 1. The one-dimensional approximation (1D) of the 3D Navier–Stokes equations (Smith et al., 2000, 2002) describe flow within blood vessels that are embedded within the deforming soft tissue.

$$\frac{\partial V}{\partial t} + (2\alpha - 1)V\frac{\partial V}{\partial x} + 2(\alpha - 1)\frac{V^2}{R}\frac{\partial R}{\partial x} + \frac{1}{\rho}\frac{\partial P}{\partial x} = -2\frac{\alpha}{\alpha - 1}\frac{V}{R^2} \tag{5}$$

$$\frac{\partial R}{\partial t} + V\frac{\partial R}{\partial x} + \frac{R}{2}\frac{\partial V}{\partial x} = 0 \tag{6}$$

$$P = \frac{2}{3}\frac{Eh_0}{R_0}\left(\frac{R^2}{R_0^2} - 1\right) + \text{external wall stress term} \tag{7}$$

where α is a parameter defining the assumed velocity profile across the blood vessel of radius R. R_0 is the radius at zero pressure P, where the wall thickness is h_0.

Typical boundary conditions for 1D fluid flow include specification of Dirichlet conditions on the fluid velocity or pressure at the beginning and/or end of the network of vessels. Typical interface conditions for coupled 1D fluid flow in deforming elastic tissue is that the net pressure in the blood vessel is balanced by the external stress acting on the blood vessel wall.

The dependent variable fields are the scalar fluid velocity V (an average over the vessel cross-section) and the hydrostatic pressure field p.

4. ***Porous flow*** on mesh 1 of region 1. The Darcy porous flow equation for poro-elastic problems (Coussy, 2004) is

$$\nabla_{\mathbf{X}} \cdot \left(J\boldsymbol{F}^{-1}\boldsymbol{K}\boldsymbol{F}^{-T}\nabla_{\mathbf{X}} p\right) = 0 \tag{8}$$

where $\nabla_{\mathbf{X}}$ denotes the gradient operator with respect to the undeformed configuration. The additional dependent variable field component for this equation set is the Darcy pressure p. In this strongly coupled poro-elastic model, the equations for the fluid pressure and solid displacements are assembled together. The fluid pressure is solved using Darcy's law (with deformation effects accounted for via $\boldsymbol{F}$), and the solid displacement is solved using the momentum balance equations in (1). The constitutive relation for the integrated tissue representation provides the coupling from the fluid pressure to the deformation and is given by Chapelle (Chapelle et al., 2010)

$$\boldsymbol{T} = \frac{\partial W^{MR}}{\partial E} + \left[K_s(J-1) - (p-p_0)J + \frac{1}{2}\frac{(p-p_0)^2}{M}\frac{f'}{f^2}J\right]\boldsymbol{C}^{-1} \tag{9}$$

where W^{MR} is the Mooney-Rivlin strain energy function, K_s is the bulk modulus, p_0 is a reference fluid pressure, M is the Biot modulus and f is a function of J that provides compatibility with incompressible conditions. In Chapelle (Chapelle et al., 2010) the solid and fluid equations are coupled by iteratively solving each equation in turn. Due to the design of OpenCMISS, we are able to strongly couple the equations for finite elasticity and fluid pressure and build them into a single set of solver equations in order to solve this multi-physics problem simultaneously.

Typical boundary conditions for porous flow are to set the pressure on the boundary as a Dirichlet condition. A Neumann boundary condition can also be set for pressure as means to specify fluid velocity.

5. ***Wavefront propagation (eikonal) equation*** on submesh 1 of region 1, determines the evolution of the activation wavefront throughout the myocardium. Two different eikonal formulations for cardiac activation have been proposed by Keener (Keener, 1991) and Colli Franzone (Colli Franzone et al., 1990). Eikonal equations have been solved using a finite element method (Tomlinson et al., 2003) and the Fast Marching Method (FMM) (Chinchapatnam et al., 2007; Sethian, 1996; Sethian and Vladimirsky, 2000). A variation of the standard cardiac electrophysiological activation model suitable for use with the FMM is the Hamilton–Jacobi equation of eikonal equation form given by

$$\nabla\boldsymbol{\varphi}\boldsymbol{T}\boldsymbol{T}^{\boldsymbol{T}}\nabla\boldsymbol{\varphi}^{\boldsymbol{T}} - 1 = 0 \tag{10}$$

where φ is the activation time and $\boldsymbol{T}$ is the anisotropy tensor defined as $\boldsymbol{T} = \boldsymbol{AF}$, where $\boldsymbol{A}$ is the orthogonal matrix representing the unit vectors along the local myofibril coordinate system and $\boldsymbol{F}$ is a diagonal matrix whose diagonal elements are equal to the components of the conduction velocities along the local myofibril coordinate axes. The Hamilton–Jacobi eikonal equation is used to determine the position of the propagating activation wavefront throughout the myocardium. It assumes a rather simple model for cellular activation but is computationally efficient. It is also sometimes used as a first approximation for the wavefront position and followed by the more biophysically accurate monodomain equations described below.

Typical initial conditions for the Hamilton–Jacobi equation are a specification of an associated state field as inactivated.

The dependent variable field is a state variable from which the activation time φ is derived.

6. ***Electrical activity*** in the deforming soft tissue on mesh/submesh 1. The spread of electrical activity through tissue is, in general, governed by the bidomain equations which relate the spread of electric potential in the intra- and extra-cellular spaces. Under the assumption that the conductivity anisotropies are equal in the intra- and extra-cellular spaces the bidomain equations reduce to the monodomain equation

(Keener and Sneyd, 1998; Pullan et al., 2005). The monodomain equation is given by.

$$\nabla\cdot(\boldsymbol{\sigma}\nabla V_{\mathrm{m}}) = A_{\mathrm{m}}\left(C_{\mathrm{m}}\frac{\partial V_{\mathrm{m}}}{\partial t}+I_{\mathrm{ion}}\right)-I_{\mathrm{s}} \tag{11}$$

Where $\boldsymbol{\sigma}$ is the conductivity tensor, I_{s} is the stimulation current and V_{m} is the transmembrane voltage. The capacitive current $C_{\mathrm{m}}(\partial V_{\mathrm{m}}/\partial t)$ is associated with the membrane capacitance $C_{\mathrm{m}}, A_{\mathrm{m}}$ is the cell surface to volume ratio and the total ion channel current I_{ion} is given by

$$I_{\mathrm{ion}} = I_{\mathrm{Na}}+I_{\mathrm{K_r}}+I_{\mathrm{K_s}}+I_{\mathrm{K_1}}+I_{\mathrm{Ca}}+I_{\mathrm{cl}}+.. \tag{12}$$

Operator splitting techniques (Qu and Garfinkel, 1999; Sundnes et al., 2005) are often used to transform the monodomain equation into two parts. The first part is a set of nonlinear ordinary differential equations (ODEs) which relate the rate of change of the transmembrane voltage to the ionic currents. The second part is a parabolic equation that governs the reaction–diffusion of transmembrane voltage. The set of nonlinear ODEs which model the ionic currents in the cell are described by CellML models.

Typical initial conditions for the monodomain equation are that the transmembrane voltage is at the resting potential and that the state variables for the CellML ionic models are at their steady states. Typical boundary conditions for the monodomain equation are a Neumann condition on the transmembrane potential so that there is no normal current flux from the domain.

The monodomain equation interacts with other equations through the cell model describing the transmembrane current. This cell model can also be used to provide an active stress for coupling to equations of large deformation elasticity, or be used to provide an ionic source term for, say, equations that model the diffusion of ion concentration.

The dependent variable field is the transmembrane voltage V_{m}. An additional dependent field is also required to describe the state variables associated with the CellML models for the ionic current.

Note that the first two problems are defined on their own distinct regions (*Region 1* for the soft tissue and *Region 2* for the fluid) with separate meshes (*Mesh 1* and *Mesh 2*, respectively) and are coupled with a fluid-structure *Interface region* that sets up the appropriate boundary coupling conditions. The third set of 1D flow equations is solved on *Mesh 3* that is embedded within the material coordinate system of *Mesh 1*. The final three sets of porous flow equations, eikonal equations, and monodomain equations are all solved on *Mesh 1* in *Region 1*, or its higher spatial resolution *submeshes,* and are coupled to the equations of large deformation elasticity via the deformation gradient field.

9. Regions, meshes and domain decomposition

A ***region*** provides a name space and container for the various objects that are required to describe a physical problem, including fields, the meshes, and other domains they are defined on, and the coordinate system for geometric fields. In the example of Fig. 1, *Region 1* represents the myocardium and *Region 2* the left and right ventricles. A third *Interface region* allows for the coupling of the first two regions. Finally, these three regions are themselves embedded in a *World Region*.

A ***mesh*** is an OpenCMISS object that includes the topology of the computational mesh (elements and global node numbers) together with the basis functions needed for interpolation of nodal parameters. OpenCMISS is flexible in allowing different forms of interpolation throughout the mesh. Most standard basis families and orders are implemented including linear, quadratic, and cubic Lagrange bases, cubic Hermite bases and linear, quadratic, and cubic simplex bases. Tensor product basis functions may also be used to allow for different polynomial forms in each interpolation direction e.g., cubic Hermite – linear Lagrange basis functions. OpenCMISS also allows for a different basis function in each element of the mesh e.g. simplex triangular bases may be used in one part of the mesh, bilinear Lagrange bases in another part, and bicubic Hermite bases in yet another part. OpenCMISS also allows for different numerical quadrature schemes to be used for each basis function. The different quadrature schemes can be used to allow different numbers of Gauss, or integration, points in each interpolation direction in order to permit optimisations involving higher (or lower) order polynomials. Care, however, must be taken to ensure that the Gauss quadrature scheme is consistent when doing integrations involving a number of different basis functions.

OpenCMISS uses domain decomposition methods (Mathew, 2008; Smith et al., 1996; Toselli and Widlund, 2005) as the main mechanism for distributed parallelism. The decomposition of meshes in *Region 1* onto ***domains*** that correspond to computational nodes, is illustrated in Fig. 2. Note that a computational node could be a single CPU in a distributed memory cluster or a multiprocessor shared memory node in a cluster. A similar domain decomposition exists for *Region 2*.

When choosing a decomposition (or mesh partitioning) for a mesh it is important that computational aspects are considered. It is desirable that each domain has roughly the same computational work load as other domains as this will ensure good computational load balance when running a parallel simulation. A second aspect is that the boundary between domains be as small as possible in order to minimise the amount of communication between the computational domains. To achieve an optimum mesh decomposition, OpenCMISS uses a parallel graph partitioning library (see earlier under Open source libraries). If required, a user can also specify a decomposition for a mesh.

Once a mesh has been decomposed into a number of computational domains, OpenCMISS performs two operations in order to abstract or hide the distributed nature of the problem. The first operation involves computing "ghost" nodes and elements. When using a numerical method in an element or at a node on a decomposed mesh, information is often required from neighbouring elements and nodes. If the node or element is at the boundary of a computational domain, then the neighbouring node or element could be held on a neighbouring computational domain. To avoid having to communicate neighbouring information, a local copy of the information is held on each computational domain. This is achieved by expanding the domain obtained from the partitioning library by one extra layer of elements around the boundary of the domain to provide a degree of overlap between the domains. The elements in the extra layer are known as ghost elements. All nodes in the ghost elements that are not already in the computational domain are also added to the domain as ghost nodes.

The second operation is renumbering. For each distributed numbering scheme (e.g. nodes, elements) OpenCMISS computes a new local numbering scheme that is mapped to the "global" mesh numbering scheme. This renumbering ensures that each scheme now starts at 1 in each computational domain and continues contiguously until the number of items (including ghosts) is reached. In order to allow further optimisations, OpenCMISS also computes which of the numbers are internal to the domain (i.e. not ghosted in any other computational domain), which numbers are on the boundary of the domain (and thus ghosted in other computational domains) and which numbers are ghosts (in this domain). By convention internal and boundary numbers are first in the numbering scheme, and ghosts are placed at the end. This renumbering ensures that a programmer working with local numbers can view each computational domain as just a smaller

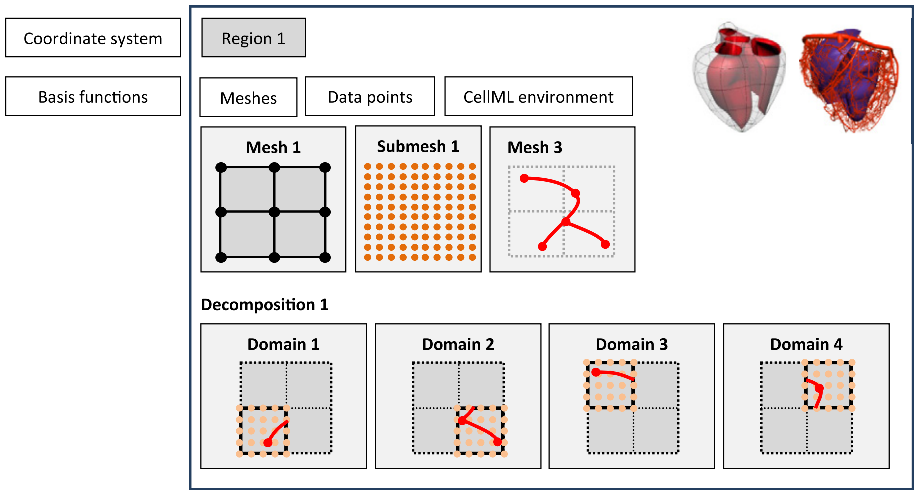


Fig. 2. The meshes and their domain decompositions for *Region 1*. The meshes shown here are designed to illustrate OpenCMISS concepts used in the heart physiome models shown in the top right corner. Note that 'ghost elements' (see text) are shown by the dotted areas in the domain decomposition.

version of the bigger problem, simplifying the programming for the distributed problem.

10. Fields

In OpenCMISS fields are the central mechanism for describing the physical problem and for storing any information required for this description. The extensive use of fields for data storage in OpenCMISS is one of the fundamental linkages to the ideas and concepts of FieldML. OpenCMISS fields have a hierarchy in that a particular OpenCMISS field contains a number of field variables and each field variable may contain any number of field variable components. An OpenCMISS field variable can thus be thought of as a conventional scalar, vector, or tensor field.

In accordance with mathematical and FieldML definitions, a field is defined over a domain. In order to allow for distributed problems, the conceptual domain for a field variable is the entire mesh, but the actual domain is the decomposed computational domain. In OpenCMISS each field variable component is allowed to use a different form of interpolation (i.e., a different set of basis functions in each element of the computational domain) from other field variable components. In addition OpenCMISS allows each component to have different forms of Degree-Of-Freedom (DOF) structure from other components. The DOFs in each component may have the following structures:

- constant structure (i.e., one DOF for the component).
- by element structure (one or more DOFs for each element).
- by node structure (one or more DOFs for each node).
- by Gauss point structure (one or more DOFs for each Gauss or integration point).
- by data point structure (one or more DOFs for each data point).

For each field variable OpenCMISS combines the DOFs for each component as a vector of DOFs. This information is then stored in a distributed vector class (see distributed matrices and vectors) so that the data for each field variable on a computational node are just those that are required for that computational domain. The data in each distributed vector for a field variable is referred to as a parameter set. OpenCMISS allows for multiple parameter sets to be stored for a field variable. Examples of useful parameter sets include the previous values of a field for problems with a time variation, the set of increments to be applied to field variable DOF values, or the set of nonlinear residuals at each DOF.

For the complete coupled multi-physics problem described, the fields, field variables, and their components are listed in Table 1. Note that some of the fields carry anatomical information, some carry tissue material parameters, and some are dependent variable fields that require the solution of partial differential equations for their evaluation.

11. Equation sets and equations

An ***equation set*** is the OpenCMISS object that is used to model a particular physical problem. It is the container object for all the necessary information required to describe the physical equations in a region (which may contain any number of equation sets). The main objects that an equation set contains are all the necessary fields (from the same region) required for the problem description and the mathematical equations that result from applying some solution method (e.g., the finite element method) to the physical problem. The equation set fields can be from the following general field classes: *equation description field*, *geometric field*, *fibre field*, *materials field*, *independent field*, *analytic field, dependent field*, and *source field*.

Another OpenCMISS object is an ***equation*** which contains the matrices and vectors that result from a numerical method being applied to the governing equations of an equation set. The equation matrices and vectors are built using the field variables of an equation set. OpenCMISS abstracts equations into a general form that allows for linear and nonlinear equations, as well as static, quasistatic, and dynamic temporal equations. The general form involves linear matrices $\boldsymbol{A}^i$, mass $\boldsymbol{M}$, damping $\boldsymbol{C}$ and stiffness $\boldsymbol{K}$ matrices of the dynamic sub-system, a nonlinear residual vector $\mathbf{g}(\boldsymbol{u})$, a source vector $\boldsymbol{s}$ and a RHS vector $\boldsymbol{b}$. These are used to form the following discrete equations:

$$\sum_{\mathrm{i}} \boldsymbol{A}^i \mathbf{u}_{\mathrm{i}} + \boldsymbol{M}\ddot{\boldsymbol{u}} + \boldsymbol{C}\dot{\boldsymbol{u}} + \boldsymbol{K}\boldsymbol{u} + \mathbf{g}(\boldsymbol{u}) + \boldsymbol{s} = \boldsymbol{b} \qquad (13)$$

Each matrix or vector is incorporated into the general equation by mapping a field variable onto the matrix or vector. For example, if the governing equation involved a dynamic component, a corresponding field variable $\mathbf{u}$ would be mapped to the dynamic

Table 1
OpenCMISS fields, field variables and their components defined for the regions of Fig. 1. Note that the *dependent field* also includes derived fields, the *equation set* field is used for additional parameters or data required to describe the equations, and the *analytic field* is used for analytic solutions that are used for model verification and convergence checks.

Fields	Field variables	Field variable components				
		Region 1			Region 2	Interface region
		Mesh 1	Submesh 1	Mesh 3	Mesh 2	Mesh 4
Geometric	coordinates $\boldsymbol{X}$	X_1, X_2, X_3		X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3
	embedded coordinates ξ		ξ_1, ξ_2, ξ_3	ξ_1, ξ_2, ξ_3		
Fibre	fibre angles υ	$\upsilon_1, \upsilon_2, \upsilon_3$	$\upsilon_1, \upsilon_2, \upsilon_3$			
Material	material parameters $\boldsymbol{c}$	c_1, c_2, etc.	A_m, C_m, σ_i	ρ, μ	ρ, μ	
Source	gravitational force $\boldsymbol{g}$	g				
Dependent	Displacements $\boldsymbol{u}$ (& p)	u_1, u_2, u_3, p				
	fluid state $\boldsymbol{v}$, p			v_1, p	v_1, v_2, v_3, p	
	Darcy pressure	p				
	membrane voltage		V_m			
	activation state & time		s, φ			
	Lagrange multiplier					λ
Analytic	Not used					
Independent	Mesh velocity $\boldsymbol{w}$	w_1, w_2, w_3			w_1, w_2, w_3	
Equation set	Not used					
CellML	State		V_m, m, n, etc.			
	Intermediate		I_{Na}, I_K, etc.			
	Parameters		g_{Na}, g_K etc.			
	Model		Model index			

matrices and, if the governing equation involved a nonlinear component, a field variable would be mapped to the residual vector.

The equation sets, equations, field variables, and solution matrices and vectors for the coupled equations of Fig. 1 are shown in Table 2.

In order to allow for optimisations it is possible to map a particular field variable to any number of matrices and vectors. As an example of this, the Navier–Stokes equations contain linear and nonlinear (convective acceleration) parts. OpenCMISS permits the same field variable to be mapped to the residual vector and to the linear and dynamic matrices. Integrations that compute the dynamic and linear matrices can then be performed once and stored in those matrices. When the residual vector and Jacobian matrix for the nonlinear part are computed, the linear components can then be incorporated via a computationally efficient matrix vector product, in the case of the residual, or by direct addition of the matrices, in the case of the Jacobian, without the need for further integration. All equations in OpenCMISS use a general distributed matrix and vector class to allow for distributed solutions (see below).

12. Problems and control loops

Problems are the object for the computational steps required to solve a particular coupled model. Similar to regions, problems allow for a name space and serve as a container object for the various objects in a problems workflow.

A control loop is the object in OpenCMISS that allows for control over the workflow in a problem. There are currently four main types of control loop in OpenCMISS. These are *simple* – the work within the control "loop" is executed once; *fixed* – the work within the control loop is executed with a fixed number of times; *time* – the work within the control loop is executed from a start time to a stop time using a (possibly variable) time step; *conditional* – the work within the control loop is executed until a specified condition is met. There are also additional types of control loop for specialised work control. For example, in finite elasticity problems a load increment loop is often required for computational stability to solve the deformation problems in a sequence of small deformations rather than one large deformation.

A particular control loop in OpenCMISS can either contain any number of additional nested sub-loops, or any number of solvers, or even another problem. The workflow then proceeds by executing the first (or root) control loop and, recursively, any sub-loops. Work operations within a control loop with no sub-loops are performed by a number of "solvers" (see next section) each in turn. Complicated workflows for particular problems can be achieved by having a problem customisable pre- and post- control loop routine as well as a problem customisable pre- and post- solve routines.

As an example of a workflow, consider the control loop and solver structure shown in Fig. 3. For this hypothetical workflow we solve the eikonal equations using the fast marching method in order to provide an initial activation sequence for a coupled elasticity, porous flow, Navier–Stokes flow, monodomain problem. We can thus set up a simple loop as a sub-loop of the root control loop. This control loop has one fast marching method solver (Solver 1 in Fig. 3). To solve the remaining coupled problem a time loop is introduced as a sub-loop of the root control loop. This time loop has a "coarse" time step in order to illustrate the ability to solve at different temporal resolutions. Since the coupled problem involves finite

Table 2
The equation sets, equations, dependent field variable components, regions, meshes and equations matrices and vectors involved in the coupled problem. $\boldsymbol{A}$ indicates linear matrices are involved in the equations, $\boldsymbol{C}$ indicates a damping matrix, $\boldsymbol{K}$ indicates a stiffness matrix, $\boldsymbol{g}(\boldsymbol{u})$ indicates a non linear residual vector, $\boldsymbol{s}$ indicates a source vector and $\boldsymbol{b}$ indicates a right hand side vector.

Equation set	Equations	Field variable components	Region	Mesh	Equation vectors and matrices
1	Finite elasticity	u_1, u_2, u_3, p	1	1	$\boldsymbol{g}(\boldsymbol{u}), \boldsymbol{s}, \boldsymbol{b}$
2	3D Navier–Stokes	v_1, v_2, v_3, p	2	2	$\boldsymbol{C}, \boldsymbol{K}, \boldsymbol{g}(\boldsymbol{u}), \boldsymbol{s}, \boldsymbol{b}$
3	1D Navier–Stokes	v_1, p	1	3	$\boldsymbol{C}, \boldsymbol{K}, \boldsymbol{g}(\boldsymbol{u}), \boldsymbol{s}, \boldsymbol{b}$
4	Porous Darcy flow	p	1	1	$\boldsymbol{A}, \boldsymbol{K}, \boldsymbol{g}(\boldsymbol{u}), \boldsymbol{s}, \boldsymbol{b}$
5	Eikonal	s	1	1-sub	None (Fast marching)
6	Monodomain	V_m	1	1-sub	$\boldsymbol{C}, \boldsymbol{K}, \boldsymbol{b}$

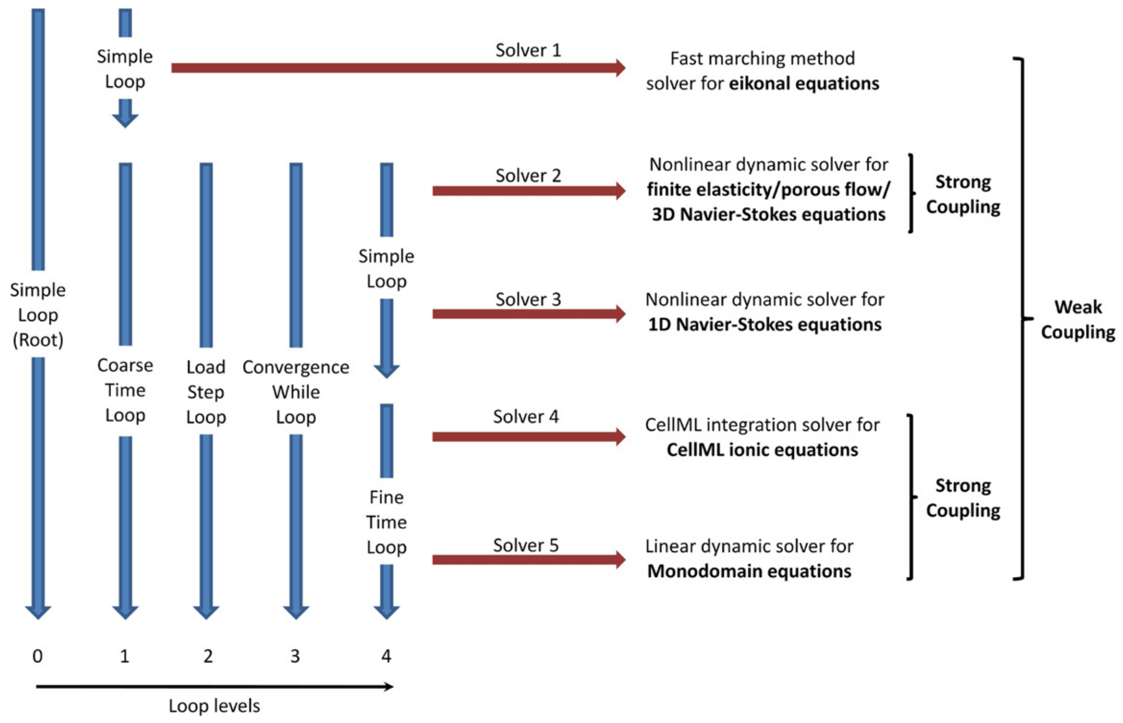

Simple Loop
Solver 1
Fast marching method solver for eikonal equations
Solver 2
Nonlinear dynamic solver for finite elasticity/porous flow/ 3D Navier-Stokes equations
Strong Coupling
Simple Loop
Simple Loop (Root)
Solver 3
Nonlinear dynamic solver for 1D Navier-Stokes equations
Weak Coupling
Coarse Time Loop
Load Step Loop
Convergence While Loop
Solver 4
CellML integration solver for CellML ionic equations
Strong Coupling
Fine Time Loop
Solver 5
Linear dynamic solver for Monodomain equations
0
1
2
3
4
Loop levels

Fig. 3. A hypothetical example of control loops and solvers for the coupled problem of Fig. 1.

elasticity, we can improve the nonlinear convergence behaviour by introducing a load step loop as a sub-loop of the coarse time loop. In order to illustrate weak iterative coupling we introduce a while convergence loop as a sub-loop of the load step loop. In order to have a different time step for the monodomain solution compared to the over-all problem we introduce a simple loop and a time loop as sub-loops of the convergence while loop. For the simple loop we introduce two non-linear dynamic solvers. The first solver in this loop (Solver 2) solves the combined finite elasticity, porous flow, and 3D Navier–Stokes equations. Since we are solving a combined system of equations there is strong (non-iterative) coupling between the equations. The second solver in this loop (Solver 3) solves the 1D Navier–Stokes equation on the deformed geometry computed by Solver 3. For the fine time loop we use, say, a Gudunov operator splitting method to split the monodomain into two parts. The first part introduces a CellML integration solver (Solver 4) and the second part introduces a linear dynamic solver (Solver 5). The iterative solution of the elasticity/porous/3D Navier–Stokes problem with the 1D Navier–Stokes problem and the monodomain problem demonstrates weak iterative coupling.

It should be noted that the ability to construct complicated problem workflows with control loops does not necessarily mean that those workflows will be numerically stable or have desirable numerical properties. There are often important considerations as to the order of solvers, stability of the schemes, and restrictions on the various numerical and solver parameters such as the size of the time step, error, tolerances, etc. Various numerical analysis techniques such as stability and error analysis (Butcher, 2008; Higham, 2002) are often required to investigate the behaviour of algorithms. This is particularly true for complex, coupled systems of equations.

13. Solvers and solver equations

Solvers are the OpenCMISS objects that perform computational work. Solvers are specific to a particular numerical problem or work item and are independent of their data source or equations. As certain solvers are better at solving particular problems than other solvers, OpenCMISS aims to support a number of different numerical solver libraries. The main classes of solver libraries in OpenCMISS include linear solvers (both iterative and direct), non-linear solvers (Newton methods, Broyden–Fletcher–Goldfarb–Shanno (BFGS), sequential quadratic programming (SQP)), dynamic solvers (theta based allowing for different schemes, implicit and explicit, various time polynomial order and degree), differential–algebraic equation (DAE) solvers (including a number of different integrators), eigenproblem solvers, optimisation solvers, state iteration solvers, and CellML model evaluation solvers.

For those solvers that require equations, OpenCMISS allows for a number of solver matrices and vectors. As with the equations from equations sets, solver equations can be classified according to their linearity and temporal nature. Solver equations are formed by adding either equations sets (possibly from different regions) or interface conditions to the solver equations object. Once the adding process has finished OpenCMISS can then form a combined set of equations by looking at the field variables involved and the types of matrices and vectors in the individual equations sets and interface conditions. This flexible approach to the generation of solver equations allows for multiple physical systems in either the same or different regions to be coupled easily.

14. Integration with CellML

CellML is used to enable OpenCMISS users to define, at run-time, custom mathematical models that form parts of a larger model. For example, CellML is used to define cellular electrophysiological models for cardiac electrical models, and to define constitutive relationships for use in finite elasticity modelling. CellML models are also being used to define kinetic models of ion transport proteins that are spatially distributed in a geometric model, and used to provide boundary fluxes in a reaction–diffusion model. CellML support in OpenCMISS is based on the concept of linking field variable components directly to variables in CellML models. This allows for the variable in a CellML model to define the value of each degree-of-freedom in a field variable component. Furthermore, a different CellML model can be used at each degree-of-freedom. This flexibility allows CellML models to be easily plugged into any of the problem types in OpenCMISS with no need for problem specific implementation. The result is a very flexible plug-and-play system for defining custom mathematical models to be incorporated into the standard bioengineering models supported by OpenCMISS. We have defined the 'CellML Environment' type to be the container managing CellML models in OpenCMISS. A given OpenCMISS application is able to use multiple CellML environments, and the preliminary best practice would be to have a CellML environment for each spatial scale in the model requiring CellML. For example, in a cardiac electromechanics model one would have one environment for the cell model(s) and one for the tissue mechanical constitutive relationships.

The first step is to import required CellML models into the environment. After importing models into the environment, the user is able to flag variables from the CellML models to be used by fields outside the CellML environment in which they are defined. CellML model variables are flagged as 'known' if their value will be controlled by a field outside the CellML environment, or as 'wanted' if the variable value computed by the CellML model will be used outside the CellML environment. State variables and the variable of integration are automatically flagged as wanted and known, respectively. Once all required variables have been flagged, the CellML environment has enough information to instantiate each of the CellML models into computable objects. Currently each model will have a simulate-able object generated on each of the computation nodes used in the simulation.

Having been instantiated, the CellML environment is able to define three fields: the state field, consisting of the state variables for each CellML model in the environment; the intermediate field, containing all non-state variables flagged as wanted from all CellML models in the environment; and the parameters field, containing all the variables flagged as known from all the CellML models in the environment. The user is now able to initialise these fields using the generic field methods defined in OpenCMISS (e.g., to set spatially varying cell model parameters not defined or required elsewhere in the model). Furthermore, the user is able to define mappings between the fields within the CellML environment(s) and fields from outside the environment. These mappings will ensure that, whenever the relevant CellML model is to be evaluated and/or integrated, the field values will be transferred in the appropriate direction (known variables in the CellML model will have their value updated from the mapped source field prior to the evaluation; fields mapped to wanted variables will be updated following the evaluation of the CellML model).

The final step in this process is to create an equation set within a defined problem and add the necessary CellML environments. Once the appropriate solver for that problem has been defined, the CellML models contained within these environments are able to be evaluated and/or integrated as appropriate during a general OpenCMISS problem solution.

Currently, the variable flagging and field mappings described above are required to be set up by the user. The plan is that, once FieldML and CellML become more integrated and use common metadata standards, the complete FieldML description of the model will include all this information and such mappings would be performed automatically by OpenCMISS. The ability of the user to override or extend such automatic mappings would be preserved in such future developments.

15. Distributed matrices and vectors

It is important for a distributed computational platform to have good memory scalability as well as good CPU scalability. In order to achieve a good decrease in the memory usage as the number of computational nodes increases, it is important that each computational node only stores that data which is relevant to that node. Indeed, for certain computer architectures, such as symmetric multiple processors or multi-core machines, it is extremely inefficient to store data not required for the computational node as this has the effect of reducing the total amount of memory available by a factor related to the additional data times the number of processors. It is also important for generality reasons that all data be able to be communicated, and for programmability reasons that the types of data objects that are communicated in a distributed environment is minimised. For these reasons OpenCMISS uses only a distributed matrix and vector class as the primary data objects.

In addition to its internal distributed data storage requirements, OpenCMISS also needs to communicate data to other (mainly distributed solver) libraries. In order to allow for multiple current libraries or other future libraries to be used with OpenCMISS, it is important to encapsulate the data requirements of these libraries within the OpenCMISS distributed matrix and vector classes. This allows for solver equations to be formulated independent of the actual solver that may be used to solve a particular numerical problem.

Both the distributed matrix and distributed vector classes in OpenCMISS distribute the rows of the matrix or vector amongst the computational nodes. Each distributed matrix and vector store only the local rows (including ghosts) and columns. In order to allow for data communication each computational node has methods to change the local data values held in the distributed matrix or vector. If a computational node changes the value of a local entry that may be ghosted on other computational domains, then communication must take place so that the other computational nodes can be updated with the new value. OpenCMISS provides a number of routines for the distributed data transfer. The first routine starts the data transfer and returns, the second routine tests whether or not the data transfer has finished or not and returns, and the final routine finishes the data transfer and returns only when the transfer is complete. By using a staged approach to transfer, it is possible to start the transfer and then do additional computations whilst the data transfer is taking place.

16. Example solutions

To provide an illustration of the framework described above we show solutions from three coupled problems. The first is a poro-elastic problem on a cube, the second is an Hamilton–Jacobi eikonal solution on a geometrically simple deforming ventricle, and the third is a monodomain solution on a square 2D domain, coupled to a CellML electrophysiological model.

16.1. Poro-elastic model

In this example model the fluid pressure and solid displacement equations are coupled together with a volume coupling approach. The equations are defined on the same region and are interdependent. The fluid pressure is solved using Darcy's relation on the deformed geometry and the solid displacement is solved using the momentum balance equation. The constitutive law used for the combined material is from Chapelle (Chapelle et al., 2010) as detailed above.

The coupled equations are solved on a 10 mm $\times$ 10 mm $\times$ 10 mm block of myocardium with quadratic Lagrange interpolation for the displacement field and linear Lagrange interpolation for the fluid pressure field. The displacement in the surface normal directions is fixed for the $x = 0$, $y = 0$ and $z = 0$ faces, and the face at $x = 10$ mm in the initial configuration is fixed at $x = 12$ mm. The fluid pressure is constrained to be 0 kPa on the $x = 10$ mm face and 1 kPa at the $x = 0$ mm face. The other faces had zero flux boundary conditions applied. Fig. 4 shows the deformed cube with swelling towards the $x = 0$ mm end due to the increased fluid pressure.

16.2. Electro-elastic model

In the heart there is a well ordered interplay between cellular electrophysiology and cellular force development. Complex feedback mechanisms support this interplay, which can be modelled by coupling anatomical, electrophysiological, deformation and force development models.

As an example of OpenCMISS capability, we present a pipeline to solve the problem of electromechanics in the heart by systems of coupled differential equations. Information is passed forward from the electrophysiology to the active stress and finite elasticity solver, which updates the mesh at each time step. In this example we did not consider ionic cell kinetics for the electrophysiology models, and therefore mechano-electrical feedback was not taken into account by the Hamilton–Jacobi equation (HJE) solver used in these simulations. As a function of the target application, the different modules of the pipeline can solve the problem of cardiac electromechanics by iteratively updating shared variables or by solving them together as a tightly coupled system of equations.

The left ventricular model consisted of 64 tri-cubic hexahedral elements. It was stimulated at time $t = 10$ ms at the endocardium between the apex and the base to trigger depolarisation as shown in Fig. 5(a), where colours correspond to the local activation time produced by the depolarisation wavefront. Dirichlet boundary conditions were applied at all the points of the base to fix movement in z-axis direction. In addition, one point at the epicardium of the base was fixed in all three directions and a point diametrically opposite was fixed in the y-axis and z-axis directions.

Fig. 5 shows the left ventricular geometry at different time points from 50 ms to 550 ms. For this example we ran the Hamilton–Jacobi equation solver, and afterwards we used the activation times as an input to the finite elasticity solver.

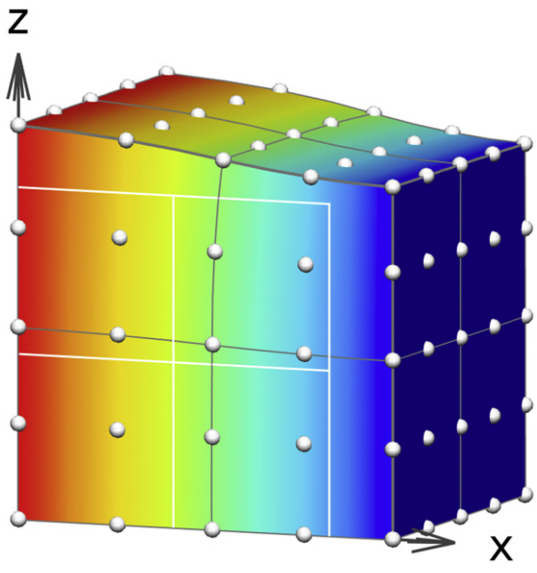


Fig. 4. 3D rendering of the deformed cube with fluid pressure plotted on the surface, with pressure increasing from blue to red. The undeformed geometry is indicated by the white lines. An extension of 1.2 in the x direction has been applied.

16.3. Monodomain with the Noble 98 cell model

To illustrate the use of CellML in an OpenCMISS model simulation we consider solving the monodomain equation (Eq. (6)) on a domain in which the ionic current sources are defined by a Noble 98 cellular model (Noble et al., 1998) that has been described in CellML (Nickerson and Hunter, 2006) and published on the CellML model repository (models.cellml.org/exposure/a40c4434423c/noble_varghese_kohl_noble_1998_b.cellml/view). The Noble 98 model describes the ionic currents in a guinea-pig ventricular cell. A schematic of the model is shown in Fig. 6.

As an exercise to demonstrate the way in which OpenCMISS can model the complex processes that occur at the smaller cellular spatial scale, whilst allowing for variations in those processes at a larger tissue spatial scale, consider the following two dimensional model. The domain in this model is a unit square discretised into a uniform grid of 26×26 nodes with a Noble 98 CellML model at each node. The fast sodium channel conductance, g_{Na}, is defined by a field so that it varies in a radial manner from 100% of its normal value at the lower left corner of the domain to 300% of its normal value in the upper right corner of the domain as shown in Fig. 7(a). The monodomain equation is then solved through time after starting the simulation by stimulating those nodes on the bottom edge of the domain from the lower left hand corner until half way to the lower right hand corner. A plot of the transmembrane voltage immediately after the stimulation pulse is shown in Fig. 7(b).

The effect of the increased fast sodium channel conductance can be seen in Fig. 7. Fig. 7(c) shows the control transmembrane voltage distribution after simulation for a fixed time period with the homogeneous g_{Na} distribution. By contrast, Fig. 7(d) shows that the activation wave for heterogeneous g_{Na} case has progressed further across the domain after the same time period indicating that the activation wave speed had increased.

The OpenCMISS framework for this simulation consists of a single region which contains a high spatial resolution mesh. An equations set for the monodomain equation is formed using a geometric field, an optional fibre field, a materials field containing components for the transmembrane area (A_m in Eq. (6)), the transmembrane capacitance (C_m) and the conductivities (σ_i), and a dependent field with a component for transmembrane voltage (V_m). To incorporate the Noble 98 ionic current models a CellML environment is defined on the region and a Nobel 98 CellML model is imported into the environment from a file on disk. Note that additional CellML models can be imported into the CellML environment if required. To allow for the spatial variation in the fast sodium channel conductance and the stimulation current the CellML variables “fast_sodium_current/g_Na” and “membrane/IStim” are set “as known” to indicate that OpenCMISS will define these values outside CellML via a field. If it was required to return any values from the CellML models (for example to return the CellML intermediate sodium current variable in order to understand how the sodium current field changes throughout time) then additional CellML variables could be set as ‘wanted’.

Once the known or wanted status of each CellML variable has been set, the CellML model is ready to be generated. Upon finishing the creation of the CellML environment in a region OpenCMISS invokes the code generation service of the CellML API. This service automatically generates a C or Fortran function/subroutine from the MathML description of the CellML model. The function or subroutine has a standard interface that allows for the variables that have been set as, known, or, wanted, to be passed in or out. All other CellML variables that are not, known, or, wanted, are set as constants in the function or subroutine with their value defined by the CellML model. After the code is generated it is automatically compiled and dynamically linked into the OpenCMISS library.

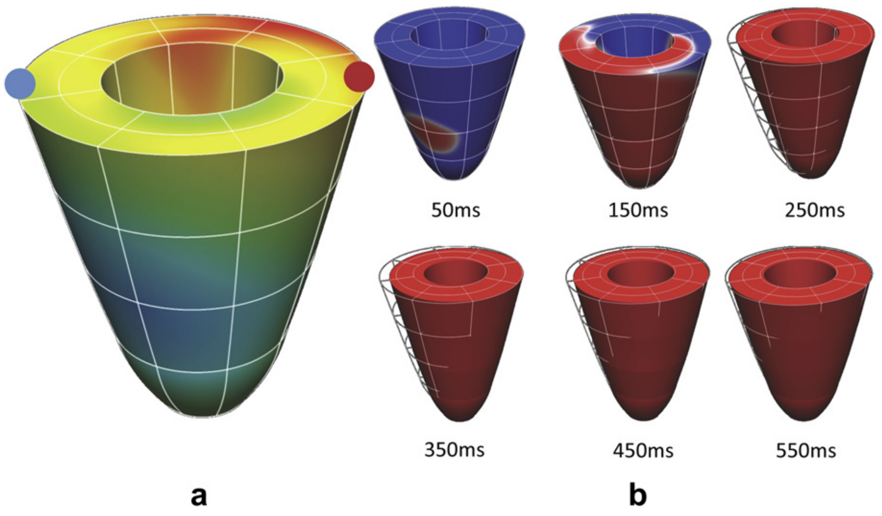


Fig. 5. A geometrically simple left ventricle example to illustrate the application of OpenCMISS in coupling the Hamilton–Jacobi equation (HJE) and finite elasticity solvers in OpenCMISS. (a) Isochrones corresponding to the local activation times of each element obtained from the HJE solver. Blue colour corresponds to 0 ms and red colour to 216 ms. (b) Deformation of the mesh at different time points colour coded by the electrical wavefront propagation.

After the CellML environment has been established, the mapping from OpenCMISS field variable components to CellML, field variable components can be set. For the monodomain problem in question there are two mappings required. The first mapping is from the OpenCMISS dependent field component representing the spatially varying transmembrane voltage, V_m, to the CellML state field variable component for V_m. The second mapping is the reverse of the first mapping. The definition of the field mappings in OpenCMISS sets the number of instances of CellML models in the OpenCMISS library as a CellML model is defined for each degree-of-freedom (DOF) in the mapped OpenCMISS field.

After the field mappings have defined the DOF (or field) structure, OpenCMISS is able to set up CellML fields using the mapped structures. The CellML fields describe the spatial variation of the free CellML variables in the CellML models. The first CellML field is the models field. This is an integer field which allows the OpenCMISS user to specify which imported model index to use at each DOF (OpenCMISS allows for the CellML model to vary spatially). The remaining CellML fields are the *state field*, the *parameters field*, and the *intermediate field*. These fields each have one field variable that contains as many components as there are free variables in the CellML model. For example the Noble 98 model defines 24 state

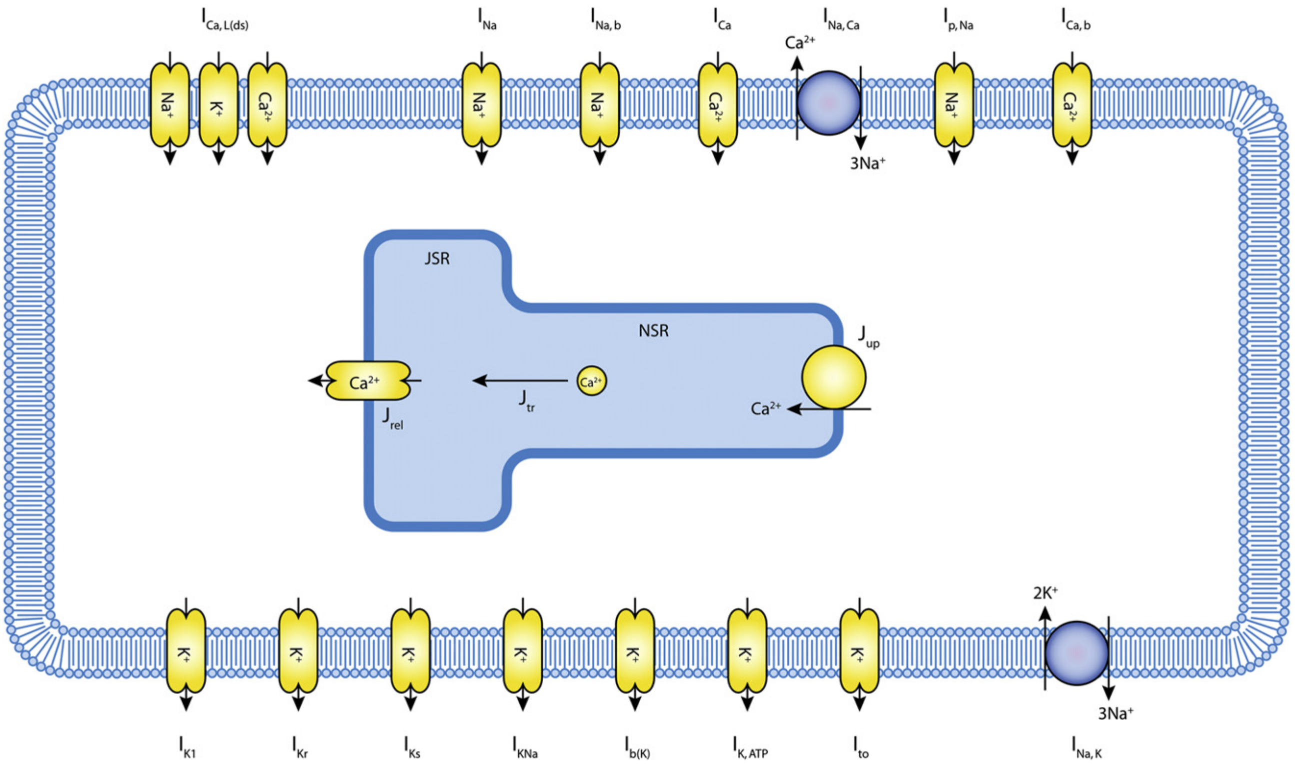


Fig. 6. A schematic diagram of the ion channels in the cell membrane of the Noble 98 model.

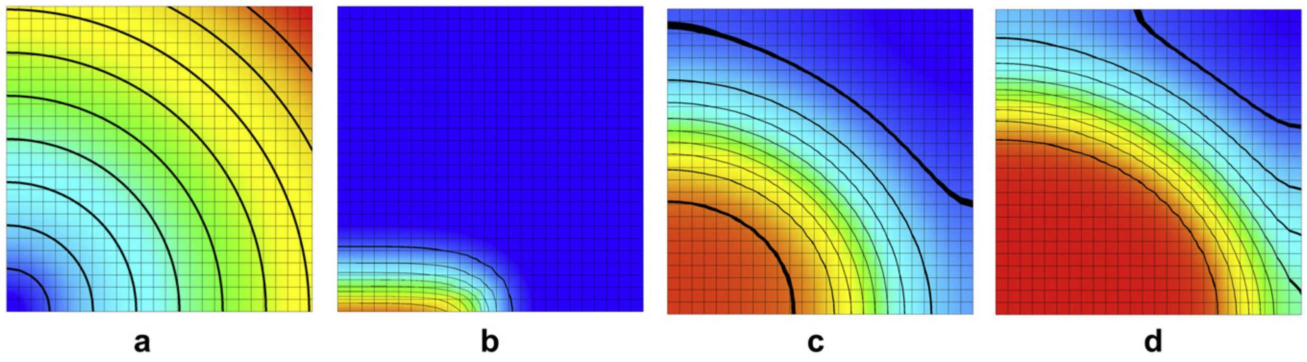


Fig. 7. (a) A plot of the fast sodium channel conductance, g_{Na} throughout the monodomain solution domain. The conductance varies in a radial fashion from 100% of its normal value of 3.855×10^{-5} mS mm^{-2} (blue) to 300% of its normal value (red). (b) A plot of the transmembrane voltage immediately after the stimulation pulse has finished. The following two plots show transmembrane voltage after a fixed time period for (c) a control simulation with a uniform g_{Na} distribution at its normal value, and (d) a simulation in which g_{Na} has been altered to the distribution shown in (a). The transmembrane voltage, V_m, in (b)–(d) varies from −95 mV (blue) to +50 mV (red).

variables, so the CellML state field variable has 24 components. The OpenCMISS simulation defines two parameters as known (g_{Na} and *IStim*), so the CellML parameters field has two components. No CellML intermediate variables were set as wanted, so the CellML intermediate field variable does not contain any field variable components.

The OpenCMISS problem workflow for this monodomain simulation sets up a root time control loop. In order to solve Eq. (6) an operator splitting approach is used. For Godunov type splitting (other splitting schemes are also defined) two solvers are defined in the time loop. The first solver is an ODE integration solver to integrate the CellML ODEs, and the second solver is a linear dynamic solver to solve the parabolic monodomain equation. The problem solution then proceeds by stepping through the time control loop. At each time step the CellML integration solver is invoked first. For this solver the OpenCMISS field to CellML field mappings are used to copy the transmembrane voltage component from the dependent field to the transmembrane voltage component of the CellML state field. The solver then integrates the CellML model ODEs from the current time step to a point in the future as determined by the splitting scheme. The OpenCMISS CellML integrator works by using a user selectable integration scheme to repeatedly call the appropriate dynamically linked CellML routine. Note that the ODE integration time steps can be smaller than the root control time loop steps in order to correctly capture processes that occur on a faster time scale than that of the main problem. Once the integration has finished, the CellML to OpenCMISS field maps are used to copy the transmembrane voltage component from the CellML state field to the OpenCMISS dependent field. The second, dynamic, solver is then invoked to solve the parabolic monodomain equation for the entire domain.

Note that the above ODE integration solver operates on each of the individual CellML models at each dependent field DOF. This process is ideally suited for parallelisation as each CellML model is independent and thus can be integrated in parallel. Further research is currently being undertaken to develop a framework that can take CellML models selected from a model repository and/or file and automatically generate CUDA or OpenCL code that can integrate these CellML models on a GPGPU (Kirk and Hwu, 2010).

17. Discussion and outlook

OpenCMISS is an international open source software project to provide a computational environment for multi-physics, multi-scale physiological modelling in the VPH/Physiome project. It builds on 30 years of experience with the CMISS code, but adds more general data structures, copes with a wider range of computer architectures (distributed memory, GPGPUs, etc) and takes advantage of the markup language environments (CellML and FieldML) developed by the VPH/Physiome project over the last 10 years.

In this paper, we have described the OpenCMISS data objects (*regions*, *meshes*, *decomposed domains*, *fields*, *equation sets*, *equations*, *problems*, *control loops*, *solvers*, *solver equations*) in the context of a geometrically simple multi-region example that couples several physical processes within a single region (large deformation soft tissue mechanics, porous flow, eikonal wave propagation, and monodomain reaction–diffusion equations that incorporate cell electrophysiology) with 1D flow through embedded blood vessels within that region, and physical processes in an adjacent region (3D flow in the ventricles).

Although the examples here are motivated by the Heart Physiome project, the FieldML and CellML based data structures are designed to handle any coupled system of partial differential equations. Processes at the tissue level, described by continuous FieldML fields, are coupled to cellular processes described by the CellML modelling framework.

Despite the successful implementation of a large number of coupled physical processes, OpenCMISS still requires a large amount of further development. The current focus for OpenCMISS development is on optimisation of the code and improved performance and scaling with processor node code, for both memory footprint and solution time. OpenCMISS has also just started to use GPGPUs to accelerate codes. Initial results are promising and OpenCMISS's field based/CellML approach, in which the data parallel vector of field DOFs together with a computational kernel defined by a CellML model, seems to fit well with GPGPU architectures. An important, and often overlooked, part of parallel performance improvement is to increase parallel IO performance. Currently, the FieldML API used for the input and output of field parameters in OpenCMISS is strictly serial. Work is currently underway to integrate HDF5, or other parallel IO libraries, with the FieldML API. This would then allow for the total solution of large, realistic problems to be profiled and optimised.

Another area of development for OpenCMISS and FieldML is the extension of the data structures to cope with more complicated mesh structures. It is planned to allow for and/or extend hierarchical, embedded and adaptive meshes. Hierarchical meshes (and fields) are based on the concept of having multiple levels of (increasing) mesh refinement. A coarse mesh (or element) at one level can be refined to give a finer mesh at another level. The second mesh level can then also be refined etc. in a hierarchical manner to give further levels. A hierarchical mesh is important for solving problems like coupled electromechanics in the same region/mesh as the electrical activation problem requires a much higher resolution mesh than the large deformation mechanics problem. Hierarchical meshes may also be useful for other solution methods e.g., multi-grid (Briggs et al., 2000). Embedded meshes and, more generally, embedded fields occur when the values of a field are interpreted with respect to another field. An example of an embedded field is the when the geometric field of coronary arteries are embedded within the (deforming) geometric field of the myocardium. Adaptive meshes are meshes which can be automatically refined so that larger numbers of field degrees-of-freedom are incorporated around areas of large field gradients. It is also planned to investigate the use of dynamic data structures such as oct trees for use in parallel load balancing with hierarchical or adaptive meshes.

Along with FieldML and CellML the authors are investigating other markup languages for use with OpenCMISS. One promising language is the Simulation Experiment Description Markup Language or SED-ML (sed-ml.org) (Köhn and Le Novère, 2008). It may be possible to use SED-ML to describe workflows which can then be implemented using OpenCMISS problems, control loops and solvers. Some of the ideas in Taverna (www.taverna.org.uk) (Hull et al., 2006) may also be useful in this regard.

The final area of future development for OpenCMISS is to develop suitable graphical user interfaces (GUIs). Whilst OpenCMISS is primarily a library without a specific GUI, we wish to make the use of the library much easier to use for mesh creation and visualisation of computational results. This GUI may be specific to a particular application or it may be more along the lines of a general environment for VPH/Physiome problems. Work has begun to link together CMGUI and/or GIMIAS (www.gimias.org) (Larrabide et al., 2009) with OpenCMISS.

Please see also related communications in this issue by Bordas et al. (Bordas et al., 2011) and Qu et al. (Qu et al., 2011).

Acknowledgements

C. Bradley, D. Paterson and P. Hunter acknowledge the support of the Wellcome Trust for the Cardiac Physiome Project. Work partially funded by the European Commission under the euHeart's Integrated Project (FP7-ICT-2007-224485) and by the Spanish Ministry of Science and Innovation under the cvREMOD CENIT Project (CEN20091044). A. Frangi holds an ICREA-Academia Prize from the Institucio Catalana de Recerca i Estudis Avancats (ICREA). V. Rajagopal acknowledges the support of a RSNZ Marsden Fast Start Grant.

References

Bangerth, W., Hartmann, R., Kanschat, G., 2007. deal.II—a general-purpose object-oriented finite element library. ACM Trans. Math. Software 33, 24/1–24/27.

Batchelor, G.K., 2000. An Introduction to Fluid Dynamics. Cambridge University Press, Cambridge.

Bordas, R., Carpentieri, B., Fotia, G., Maggio, F., Nobes, R., Pitt-Francis, J., Southern, J., 2009. Simulation of cardiac electrophysiology on next-generation high-performance computers. Philos. Transact A Math. Phys. Eng. Sci. 367, 1951–1969.

Bordas, R., Gillow, K., Lou, Q., Efimov, I.R., Gavaghan, D., Kohl, P., Grau, V., Rodriguez, B., 2011. CBG: rabbit-specific ventricular model of cardiac electrophysiological function including specialized conduction system. Prog. Biophys. Mol. Biol. 107, 90–100.

Bradley, C.P., Pullan, A.J., Hunter, P.J., 1997. Geometric modeling of the human torso using cubic hermite elements. Ann. Biomed. Eng. 25, 96–111.

Briggs, W.L., Henson, V.E., McCormick, S.F., 2000. A Multigrid Tutorial. Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. xii, 193 p.

Butcher, J.C., 2008. Numerical Methods for Ordinary Differential Equations. John Wiley & Sons, West Sussex.

Chapelle, D., Gerbeau, J.-F., Sainte-Marie, J., Vignon-Clementel, I.E., 2010. A poroelastic model valid in large strains with applications to perfusion cardiac modeling. Comput. Mech. 46, 91–101.

Chinchapatnam, P.P., Rhode, K.S., King, A., Gao, G., Ma, Y., Schaeffter, T., Hawkes, D., Razavi, R.S., Hill, D.L., Arridge, S., Sermesant, M., 2007. Anisotropic wave propagation and apparent conductivity estimation in a fast electrophysiological model: application to XMR interventional imaging. Med. Image Comput. Comput. Assist. Interv. 10, 575–583.

Christie, G.R., Blackett, S.A., Hunter, P.J., Bullivant, D.P., 2002. Modeling and visualising the heart. Comput. Vis. Sci. 4, 227–235.

Christie, G.R., Nielsen, P.M., Blackett, S.A., Bradley, C.P., Hunter, P.J., 2009. FieldML: concepts and implementation. Philos. Transact A Math. Phys. Eng. Sci. 367, 1869–1884.

Colli Franzone, P., Guerri, L., Tentoni, S., 1990. Mathematical modeling of the excitation process in myocardial tissue: influence of fiber rotation on wavefront propagation and potential field. Math. Biosci. 101, 155–235.

Coussy, O., 2004. Poromechanics. John Wiley & Sons, West Sussex.

Cuellar, A.A., Lloyd, C.M., Nielsen, P.F., Bullivant, D.P., Nickerson, D.P., Hunter, P.J., 2003. An overview of CellML 1.1, a biological model description language. SIMULATION: Trans. Soc. for Model. Simulation Int. 79, 740–747.

Currie, I.G., 2002. Fundamental Mechanics of Fluids. Marcel Dekker, New York, pp. xiv, 525 p.

Higham, N.J., 2002. Accuracy and Stability of Numerical Algorithms. SIAM, Philadelphia.

Holst, M., 2001. Adaptive numerical treatment of elliptic systems on manifolds. Adv. Comput. Math. 15, 139–191.

Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., Oinn, T., 2006. Taverna: a tool fro building and running workflows of services. Nucleic Acids Res. 34, 729–732.

Hunter, P., Nielsen, P., 2005. A strategy for integrative computational physiology. Physiology (Bethesda) 20, 316–325.

Hunter, P., Smaill, B., 1988. The analysis of cardiac function: a continuum approach. Prog. Biophys. Mol. Biol. 52, 101–164.

Hunter, P.J., 2004. The IUPS physiome project: a framework for computational physiology. Prog. Biophys. Mol. Biol. 85, 551–569.

Hunter, P.J., Borg, T.K., 2003. Integration from proteins to organs: the Physiome project. Nat. Rev. Mol. Cell Biol. 4, 237–243.

Hunter, P.J., McCulloch, A.D., ter Keurs, H.E., 1998. Modelling the mechanical properties of cardiac muscle. Prog. Biophys. Mol. Biol. 69, 289–331.

Hunter, P.J., Pullan, A.J., Smaill, B.H., 2003. Modeling total heart function. Annu. Rev. Biomed. Eng. 5, 147–177.

Keener, J.P., 1991. An eikonal-curvature equation for action potential propagation in myocardium. J. Math. Biol. 29, 629–651.

Keener, J.P., Sneyd, J., 1998. Mathematical Physiology. Springer, New York.

Kirk, B.S., Peterson, J.W., Stogner, R.H., Carey, G.F., 2006. libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations. Eng. Comput. 22, 237–254.

Kirk, D.B., Hwu, W.W., 2010. Programming Massively Parallel Processors. Morgan Kaufmann Publishers, Burlington.

Köhn, D., Le Novère, N., 2008. SED-ML – an XML format for the implementation of the MIASE guidelines. Computational Methods Syst. Biol. 5307, 176–190.

Larrabide, I., Omedas, P., Martelli, Y., Planes, X., Nieber, M., Moya, J.A., Butakoff, C., Sebastián, R., Camara, O., De Craene, M., Bijnens, B.H., Frangi, A.F., 2009. GIMIAS: an open source framework for efficient development of research tools and clinical prototypes. In: Ayache, N., Delingette, H., Sermesant, M. (Eds.), Functional Imaging and Modeling of the Heart: Proceedings of the 5th International Conference, vol. 5528. SpringerLink, Nice, France, pp. 417–426.

Lloyd, C.M., Lawson, J.R., Hunter, P.J., Nielsen, P.F., 2008. The CellML model repository. Bioinformatics 24, 2122–2123.

Malvern, L.E., 1969. Introduction to the Mechanics of a Continuous Medium. Prentice-Hall, Englewood Cliffs, N.J., pp. xii, 713 p.

Mathew, T.P.A., 2008. Domain decomposition methods for the numerical solution of partial differential equations. In: Lecture Notes in Computational Science and Engineering. Springer, Berlin, pp. xiii, 764 p.

Miller, A.K., Marsh, J., Reeve, A., Garny, A., Britten, R., Halstead, M., Cooper, J., Nickerson, D.P., Nielsen, P.F., 2010. An overview of the CellML API and its implementation. BMC Bioinform. 11, 178.

Nash, M.P., Hunter, P.J., 2000. Computational mechanics of the heart – from tissue structure to ventricular function. J. Elast. 61, 113–141.

Nickerson, D.P., Hunter, P.J., 2006. The Noble cardiac ventricular electrophysiology models in CellML. Prog. Biophys. Mol. Biol. 90, 346–359.

Niederer, S.A., Hunter, P.J., Smith, N.P., 2006. A quantitative analysis of cardiac myocyte relaxation: a simulation study. Biophys. J. 90, 1697–1722.

Noble, D., Varghese, A., Kohl, P., Noble, P., 1998. Improved guinea-pig ventricular cell model incorporating a diadic space, IKr and IKs, and length- and tension-dependent processes. Can. J. Cardiol. 14, 123–134.

Nordsletten, D., Kay, D., Smith, N., 2010. A non-conforming monolithic finite element method for problems of coupled mechanics. J. Comp. Physiol. 229, 7571–7593.

Nordsletten, D.A., Niederer, S.A., Nash, M.P., Hunter, P.J., Smith, N.P., 2011. Coupling multi-physics models to cardiac mechanics. Prog. Biophys. Mol. Biol. 104, 77–88.

Oden, J.T., 2006. Finite Elements of Nonlinear Continua. Dover Publications.

Patzák, B., Bittnar, Z., 2001. Design of object oriented finite element code. Adv. Eng. Software 32, 759–767.

Patzák, B., Rypl, D., Bittnar, Z., 2001. Parallel explicit finite element dynamics with nonlocal constitutive models. Comput. Struct. 79, 2287–2297.

Pitt-Francis, J., Pathmanathan, J., Bernabeu, M.O., Bordas, R., Cooper, J., Fletcher, A.G., Mirams, G.R., Murray, P., Osbourne, J.M., Walter, A., Chapman, S.D., Garney, A., van Leeuwen, I.M.M., Maini, B., Rodriguez, B., Waters, S.L., Whiteley, J.P., Byrne, H.M., Gavaghan, D.J., 2009. Chaste: a test-driven approach to software development for biological modelling. Comput. Phys. Commun. 180, 2452–2471.

Pullan, A.J., Cheng, L.K., Buist, M.L., 2005. Mathematically Modelling the Electrical Activity of the Heart: From Cell to Body Surface and Back Again. World Scientific Publishing.

Qu, Z., Garfinkel, A., 1999. An advanced algorithm for solving partial differential equation in cardiac conduction. IEEE Trans. Biomed. Eng. 46, 1166–1168.

Qu, Z., Garfinkel, A., Weiss, J.A., Nivalal, M., 2011. Multi-scale modeling in biology: how to bridge the gaps between scales? Prog. Biophys. Mol. Biol. 107, 21–31.

Sethian, J.A., 1996. A fast marching level set method for monotonically advancing fronts. Proc. Natl. Acad. Sci. U S A 93, 1591–1595.

Sethian, J.A., Vladimirsky, A., 2000. Fast methods for the Eikonal and related Hamilton–Jacobi equations on unstructured meshes. Proc. Natl. Acad. Sci. U S A 97, 5699–5703.

Smith, B.F., Bjørstad, P.E., Gropp, W., 1996. Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press, Cambridge; New York, pp. xii, 224 p.

Smith, N.P., Nickerson, D.P., Crampin, E.J., Hunter, P.J., 2004. Multiscale computational modelling of the heart. Acta Numerica 13, 371–431.

Smith, N.P., Pullan, A.J., Hunter, P.J., 2000. Generation of an anatomically based geometric coronary model. Ann. Biomed. Eng. 28, 14–25.

Smith, N.P., Pullan, A.J., Hunter, P.J., 2002. An anatomically based model of transient coronary blood flow in the heart. SIAM J. Appl. Math. 62, 990–1018.

Sundnes, J., Lines, G.T., Tveito, A., 2005. An operator splitting method for solving the bidomain equations coupled to a volume conductor model for the torso. Math. Biosci. 194, 233–248.

Tawhai, M., Pullan, A.J., Hunter, P.J., 2000. Generation of an anatomically based three-dimensional model of the conducting airways. Ann. Biomed. Eng. 28, 793–802.

Tawhai, M.H., Hunter, P., Tschirren, J., Reinhardt, J., McLennan, G., Hoffman, E.A., 2004. CT-based geometry analysis and finite element models of the human and ovine bronchial tree. J. Appl. Phys. 97, 2310–2321.

Tomlinson, K.A., Hunter, P.J., Pullan, A.J., 2003. A finite element method for an eikonal equation model of myocardial excitation wavefront propagation. SIAM J. Appl. Math. 63, 324–350.

Toselli, A., Widlund, O.B., 2005. Domain decomposition methods—algorithms and theory. In: Springer Series in Computational Mathematics. Springer, Berlin, pp. xv, 450 p.

Vigmond, E.J., Hughes, M., Plank, G., Leon, L.J., 2003. Computational tools for modeling electrical activity in cardiac tissue. J. Electrocardiol. 36 (Suppl.), 69–74.