A FRAMEWORK FOR ANNOTATING AND VISUALIZING CellML MODELS

SARALA M. WIMALARATNE

Supervised by: Dr. Matt D. B. Halstead and Associate Professor Poul F. Nielsen

A thesis submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Biomedical Engineering
The University of Auckland, New Zealand
May 2009

To Sajith, mom, dad, and sister for their unconditional love

ABSTRACT

The Physiome Project was established to develop tools for international collaboration and sharing physiological knowledge in the form of biological models and experimental data. The CellML language was developed in response to the need for a high-level language to represent and exchange mathematical models of biological processes.

The language provides a flexible framework for describing the dynamics of biological processes but does not explicitly lend itself to capturing the underlying biological concepts such as the entities and processes that these models represent. The relationship between the biological process and the mathematical model describing the biological process is also often complex. This makes it difficult to see the biological concepts which the CellML structures represent. A framework which supports visualizing the biological concepts and its relationship to the underlying CellML model would provide a very useful toolset for understanding the biological concepts modeled in CellML.

The CellML models need to be annotated with biological concepts in order to provide the machine interpretable data for generating a visual representation. We have developed an ontological framework which can be used to explicitly annotate CellML models with physical and biological concepts, a method to derive a simplified biological view from the annotations, a visual language for representing all biophysical processes captured in the CellML models, and a method to map the visual language to the ontological framework in order to automate the generation of visual representations of a model.

The proposed method of model visualization produces a result that is dependent on the structure of the CellML models which requires modelers to structure the model in a way that best describes the biophysical concepts and abstractions they wish to demonstrate. Our argument is that this leads to a best practice approach to building and organizing models.

As a part of this research, a software tool for visualizing CellML models was developed. This tool combines the visual language and the ontologies to generate visualizations that depict the physical and biological concepts captured in CellML models and enables different communities in diverse disciplines to more easily understand CellML models within the biological domain they represent.

As research continues, with further improvement to the framework it would be possible to visually construct composite CellML models by selecting high level biological concepts.

ACKNOWLEDGEMENT

I express my sincere gratitude towards my supervisors Dr Matt Halstead and Associate Professor Poul Nielsen for their enormous efforts giving guidance and encouragement throughout the course of this project.

I wish to gratefully acknowledge, Professor Peter Hunter, Dr Edmund Crampin, Associate Professor Gill Dobbie, Dr Catherine Lloyd, Dr Mike Cooling, and Dr Vijayaraghavan Rajagopal for their advice and assistance.

I am also thankful to a number of international researchers for fruitful discussions and stimulating suggestions on several aspects of this work; Dr Hiroaki Kitano, Dr Nicolas Le Novère, Dr Michael Hucka, Dr Emek Demir, Ms Yukiko Matsuoka, and Dr Akira Funahashi.

I am grateful for financial assistance from the University of Auckland in the form of a doctoral scholarship. Also to the Education New Zealand Study Abroad Award to Japan and a number of conference and travel grants. Without these financial supports this work would have been impossible.

I wish to thank my fellow postgraduate students and administrative staff at the Auckland Bioengineering Institute for providing a friendly atmosphere.

Finally thanks to my in-laws for their support and encouragement during this study.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AC	Adenylyl cyclase
ATP	Adenosine triphosphate
BioPAX	Biological pathways exchange
cAMP	Cyclic adenosine monophosphate
COPASI	Complex pathway simulator
COR	Cellular open resource
CSML	Cell system markup language
CSO	Cell system ontology
DOM	Document object model
FMA	Foundational model of anatomy ontology
Gd	G Protein with attached guanosine diphosphate
Gi	Inhibitory G protein
GO	Gene ontology
GPCR	G protein-coupled receptor
Gs	Stimulatory G protein
Gt	G protein with attached guanosine triphosphate
GUI	Graphical user interface
IP3	Inositol 1,4,5-trisphosphate
K	Potassium
L	Ligand
MathML	Mathematical markup language
MATLAB	Matrix laboratory
MIM	Molecular interaction maps
Na	Sodium
OBO	Open biological ontology
OWL	Web ontology language
OWL-DL	Web ontology language – descriptive logic
PATIKA	Pathway analysis tools for integration and knowledge acquisition
PCEnv	Physiome CellML environment
PKA	Protein kinase A
PMR	Physiome model repository

R	Receptor
Rg	Receptor-G protein complex
R1	Receptor-ligand complex
R1g	Receptor–ligand–G protein complex
R1gp	Receptor–ligand–G protein phosphorylated complex
RDF	Resource description framework
RDFS	Resource description framework schema
SBML	Systems biology markup language
SBO	Systems biology ontology
SVG	Scalable vector graphics
XLink	XML linking language
XML	Extensible markup language

1 INTRODUCTION

The development of quantitative models to understand the dynamics of complex biological processes is increasing due to the rapid growth of biotechnology and experimental techniques [1]. This knowledge is acquired across different disciplines such as physiology, biomedicine, biophysics, and engineering, generating a need for a consistent method for representing, storing, and exchanging these mathematical models.

There exist many ways for representing and solving mathematical models of biological processes. Most models are published in journals using standard mathematical representation. These may be solved by writing model-specific computer code using tools such as MATLAB [2]. However, the specificity of such code fails to support the sharing, exchange, and reuse of models. Furthermore, to accurately interpret mathematical equations presented in journals, readers and writers should have a common understanding of the terms that are being used. The meanings of most terms are generally captured in the text but this could still lead to misinterpretation of the writer's intension. To reduce such confusion, modelers may use common Extensible Markup Language (XML) [3] standards such as content Mathematical Markup Language (MathML) [4] or OpenMath [5] to represent the mathematics of a model, which can then be solved using tools that support these standards combined with model specific computer code. This allows modelers to represent exchangeable mathematics but

requires the writing of specific software for model simulation. During late 1990s a set of tools were developed to allow modelers to build and simulate models. This tool set includes: Virtual Cell [6] which can be used to develop and simulate detailed biochemical pathways; E-Cell [7] which can be used to build and simulate biochemical and genetic processes; COPASI (Complex Pathway Simulator) [8] which can be used to build and simulate biochemical pathways in cells; and JSim [9] which can be used for building quantitative numeric models and analyzing them with respect to experimental reference data. However, these tools are specific to particular biological domains and integrating models generated from different software is, in general, extremely difficult. In response to the need for a high-level language to represent and exchange mathematical models of biological processes, independent of the different model-building software packages, biological modeling communities started looking at developing tool-independent modeling languages for building biological models.

1.1 MODELING LANGUAGES FOR DESCRIBING MATHEMATICAL MODELS OF BIOLOGY

This section describes a set of modeling languages developed to represent mathematical models of biological concepts that can be used in various computer applications to solve the models.

1.1.1 MARKUP LANGUAGES FOR DESCRIBING MATHEMATICAL MODELS OF BIOLOGY

Extensible Markup Language (XML) is a meta-language published by the World Wide Web Consortium [10]. It provides a flexible way to express data in a computer readable form independent of software applications. The language provides flexibility to represent any kind of data, ranging from simple text, complex mathematics to domain specific knowledge. It is also the backbone of many internet-related standards. Furthermore, XML supports integration with other standards such as Resource Description Framework (RDF) [11], a standard for defining data about data which is often referred to as metadata. These characteristics have lead to the development of a number of markup languages for describing mathematical models of biological process:

- CellML [12] – for describing the mathematics, topology, and metadata of biological processes;

- The Systems Biology Markup Language (SBML) [13] – for representing models in systems biology;
- The Cell System Markup Language (CSML) [14] – for modeling biological pathways;
- The Virtual Cell Markup Language (VCML) [15] – for describing biological and mathematical models and specifications for analysis to be performed on those models.

CellML and SBML are currently the most widely used XML markup languages for describing mathematical models of biological process.

1.1.1.1 CellML

CellML promotes consistency between computational and published models and establishes a flexible framework for the reuse of models and model repositories [16]. The CellML repository contains over 360 models covering a wide range of biological processes including signal transduction pathway, metabolic, electrophysiological, calcium dynamics, immunology, cell cycle, and smooth and skeletal muscle models [17].

CellML models are constructed using a network of interconnected components. A component is the basic unit of a CellML model, containing variables, mathematical equations, which define the relationships between the variables, units, and metadata [18]. Connections specify how variables are shared between the components [18]. CellML supports a feature called encapsulation to hide a set of components from the rest of the model [18]. CellML's import feature can be used to import components, connections and units other models to reuse. This enables users to reuse parts of existing models and provides an efficient framework for model building. The structure of CellML and these features are discussed in detail in Chapter 2.

The CellML language also uses other XML-based standards such as MathML, XML Linking Language (XLink) [19], and RDF, to represent mathematics, import other models, and describe metadata, respectively.

CellML is one of the standard languages used for model representation in the Physiome Project [20]. The Physiome Project is a global public-domain effort to describe the physiology and functional behavior of mammalian physiology. It requires the integration of models over a wide range of spatial and temporal scales, and collaboration between research groups around the world. This effort aims to address the difficulties associated with representing and integrating different levels of biological knowledge ranging from nano-

scale, cells, tissues, to organ level. Markup languages are being defined to encode the models in a consistent form to support their representation, storage, exchange, and simulation. Currently CellML is mainly used to model biological processes occurring at the sub cellular to tissue level scales [1].

In order to support the CellML user community needs, the CellML development team is working on:

- the CellML level 1.2 specification [21] to improve and extend the descriptive capabilities of the CellML language;
- the Physiome Model Repository (PMR) [22] to facilitate model upload, storage, and download;
- model curation to ensure that the mathematics from the publication is accurately represented in the CellML and validating the models with respect to their function and output;
- model annotation to add context to CellML elements, simulation data, graphing data, and to facilitate database searching;
- an Application Programming Interface [23] to provide a standard interface for applications to manipulate and process CellML documents;
- tools such as OpenCell [24] for creating, simulating, analyzing and graphing simulation data.

1.1.1.2 Systems biology markup language (SBML)

SBML is mainly used to model biochemical reactions, signaling pathways, metabolic pathways, and gene regulation networks [13]. Modelers can describe biological components using compartments and species. Their dynamic behavior is described using reactions, events, and mathematical rules. The BioModels database [25] contains a large number of ready-to-use curated SBML models.

SBML also uses other XML-based standards such as MathML to represent mathematics. SBML also defines a method for the association of its components to Systems Biology Ontology (SBO) [26] terms in order to provide a biological context for the model.

SBML is the *de facto* standard for representing systems biology models [27]. Systems biology is a new field of study that focuses on investigating complex interactions in biological systems. In this field, the main emphasis is placed on understanding the structure,

dynamics, control methods, and design methods of gene regulatory networks and biochemical pathways [28]. SBML facilitates qualitative and quantitative modeling, storing, and exchange of models of biological systems.

Currently there is a large community support for the use and improvement of the SBML language. This includes the development of:

- the SBML level 2 [29] specification to improve the SBML language;
- the SBO [30] for annotating SBML models;
- the BioModels database [26] to store, search and retrieve published SBML models;
- Application Programming Interfaces [31] to provide interfaces for applications to work with SBML documents and integrate them with applications such as Mathematica [32] and MATLAB [2];
- over 120 software tools [33] for creating, simulating, analyzing, and visualizing SBML models including graphing simulation data.

1.1.2 ONTOLOGICAL LANGUAGES FOR DESCRIBING MATHEMATICAL MODELS OF BIOLOGY

Recent ontology-based approaches to solving domain specific problems have gained the interest of the scientific community. An ontology is a formal specification of concepts and relations between concepts within a domain of knowledge. It defines a common vocabulary and set of rules to unambiguously represent information [34].

The Web Ontology Language – Descriptive Logic (OWL-DL) [35] and the Open Biological Ontology (OBO) [36] are the two most widely-used ontological representation languages among the biological community. They have different levels of expressivity, rules for capturing knowledge, and computational properties. OWL-DL is a highly expressive language for building knowledge bases and provides powerful reasoning methods. The OBO file format is a subset of OWL-DL and provides a simpler framework for constructing ontologies and reasoning. Details and the usage of the OWL language are further discussed in Chapter 3.

The Cell System Ontology (CSO) [37] is being developed to explore the benefits of using an ontology to represent quantitative and qualitative aspects of biochemical pathways. The ontology supports the storage of modeling, visualization, and simulation data. CSO is

expressed in OWL-DL and uses the reasoning capabilities to check the consistency of biological pathway models.

1.2 IDENTIFYING THE PROBLEM IN DETAIL

Generally journal articles describing mathematical models of biological concepts use schematic diagrams to represent the biological concepts that are being modeled in order to make it easier for readers to comprehend the mechanisms being described. Figure 1.1 shows a schematic diagram of the cAMP/PKA cascade regulating the L-type calcium channel [38].

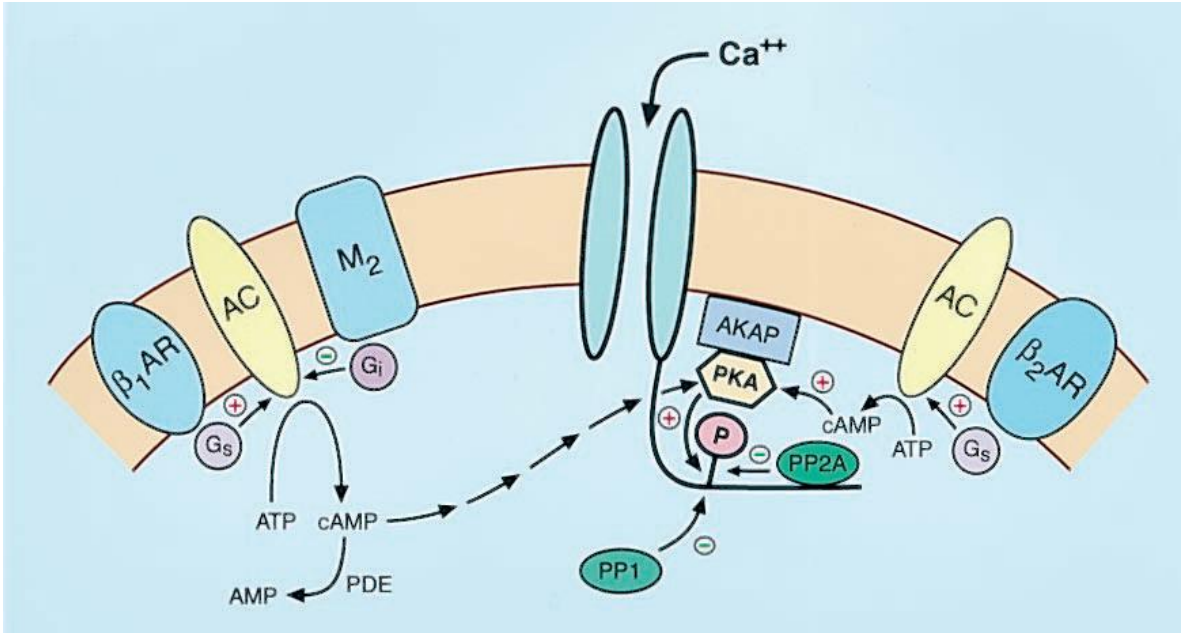


Figure 1.1: A signaling cascade regulating L-type calcium channel
A schematic of the cAMP/PKA cascade regulating L-type calcium channel. Figure derived from [(Kamp and Hell, 2000), Figure 2A].

The diagram in Figure 1.1 clearly conveys the biological entities involved in the pathway but the diagram by itself is insufficient for interpreting all the biological concepts being modeled due to the ambiguous use of icons. For example it does not uniquely identify the different biological processes that occur in the pathway. To interpret the diagram correctly it needs to be supported with text that describe the underlying biological concepts being illustrated. The legend accompanying this figure is:

> *"Stimulation of b1AR or b2AR leads to Gs-mediated activation of AC and increased production of cAMP, which stimulates PKA. PKA can then phosphorylate the channel at multiple potential sites indicated schematically by the single P in the diagram. The PKA phosphorylated site(s) is then sensitive to the phosphatases PP1 and PP2A.*

> *Whereas b1AR regulation causes more global increases in cAMP, b2AR stimulation can result in highly localized cAMP level changes and regulation. Regulatory proteins may be localized to the channel by an AKAP for PKA and by binding of PP2A to the C terminus of the channel. Muscarinic M2 receptors can oppose the bAR upregulation of ICa by acting through Gi to inhibit AC [38]."*

Combined, the figure and the legend allow readers to interpret the biological processes as intended by the writer.

CellML provides sufficient flexibility to describe the above model in a way that closely represents the entities and processes described in the diagram but this is often not done when constructing models. In many models, the biological knowledge is implicit in the names of components and variables. Understanding the relationship between the diagram and the CellML model can be time-consuming and lead to misinterpretation of the models.

The goal of this work is to provide a method to generate detailed biological visualizations directly from CellML models, where the models themselves contain explicit descriptions of biological meaning. The goal is to also establish the use of a well-understood visual terminology that can be used to represent this meaning without a textual explanation.

A prototype tool was developed to understand the problems in detail and identify specific tasks. This involved investigating a method to define the biological concepts modeled in CellML, a visual language to represent these concepts, and programmatically generating visualizations of the biological concepts. The next section describes the development of the prototype tool.

1.2.1 PROTOTYPE DEVELOPMENT

This section discusses a method and software for generating a visual representation of the pathway described in section 1.2; the cAMP/PKA cascade regulating L-type calcium channel. Since the CellML model does not explicitly represent the biological knowledge, an intermediate level is introduced to represent a biological model. The aim of the biological model is to capture biological processes and entities, and the relationships between them. This biological model integrated with a visual language can be used to visualize the biology of the CellML model.

XML is used to structure and store the biological knowledge (Figure 1.2). The entity element describes the biological species. It has three attributes:

- id - specifies a unique identifier for the biological entity;

- name - name of the biological entity;
- type - type of the biological entity.

The process element group the entities involved in a particular biological process. The connection element to state whether the biological entity involved in a particular process is acting as reactants, products, or modifiers. Each connection element has two attributes:

- entityid - refers to an entity;
- type - specifies whether it is a reactant, product, or modifier.

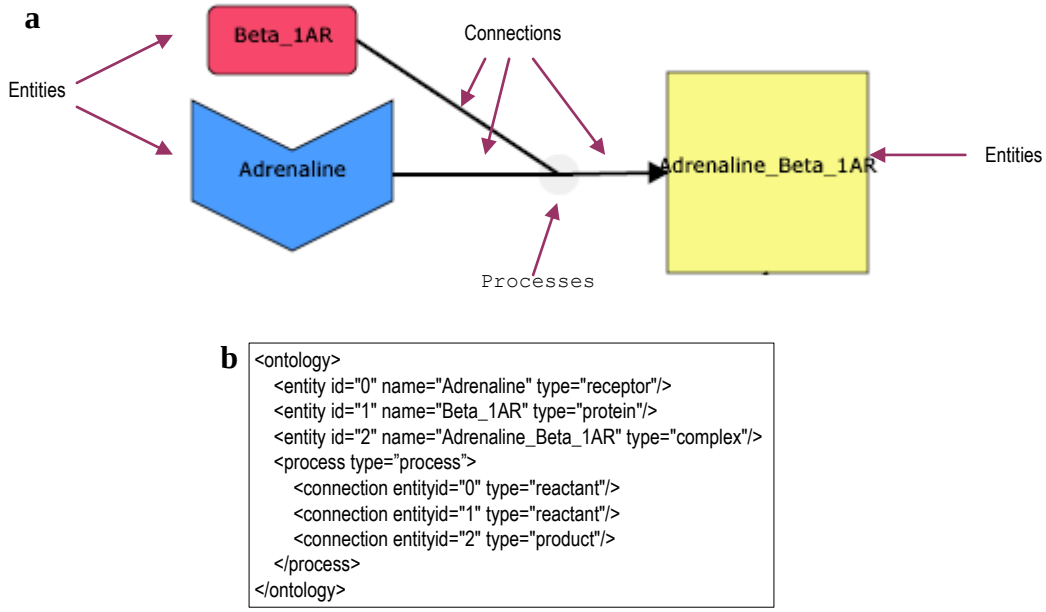


Figure 1.2: Structuring and storing biological knowledge using XML

(a) Visual representation of the way in which the XML schema models the underlying biological process, formation of Adrenaline_Beta_1AR. Beta_1AR, Adrenanline, and Adrenaline_Beta_1AR is encoded using the Entity element. The reaction is encoded using the Process element. Relationships between the entities and the reaction are represented using the Connection element. (b) XML model capturing the underlying biology. Code segment describes the entities, process, and the relationship between the entities represented in the Adrenaline_Beta_1AR reaction model in XML.

A set of visual glyphs were developed to visualize these biological entities, processes, and the relationships captured in the XML file. Various shapes are used to represent protein, receptor, closed ion channel, opened ion channel, small molecule and complex (Figure 1.3). A set of connection glyphs are used to represent the roles activator, catalyst, reactant, product, and inhibitor (Figure 1.3).

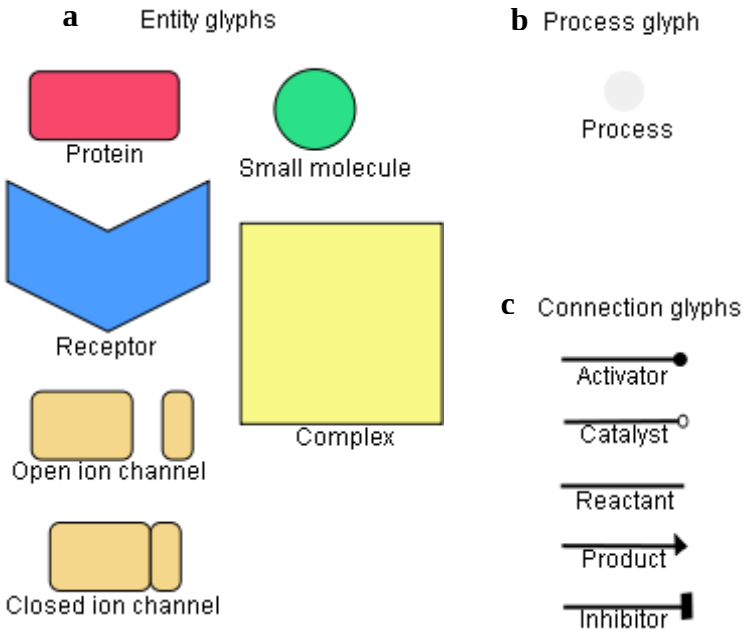


Figure 1.3: Notation for visualizing the underlying biology modeled in CellML

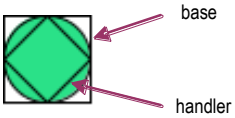
(a) Glyphs for representing the entities. (b) A glyph for representing the processes. (c) Glyphs for representing the connections.

To enable programmatic generation of diagrams, these visual objects need be represented in computer readable form. This is achieved using Scalable Vector Graphics (SVG) [39]. SVG is an XML-based interoperable graphics standard for 2-dimensional graphics and can be used to programmatically render graphics from database data to enable dynamic image updates. Details and the usage of SVG are further discussed in Chapter 4.

To store the graphics shown in Figure 1.3 a SVG file (symbols.svg) was formed. The code segment in Figure 1.4a shows the composition of the graphics for an ion. The circle element specifies the symbol for the ion. The base element specifies the area coverage by the graphic object and the handler element specifies the connection points that can be used to connect glyphs together. Figure 1.4b shows a segment of the graphics for representing connections. Line and arrow coordinates are calculated and set at runtime.

a
```
<g id="ion" name="ion" >
    <circle cx="20" cy="20" r="20" style="fill:#2ae188;stroke:#000000;stroke-width:1.0000000" />
    <rect id="base" x="0" y="0" width="40" height="40" style="fill:white;fill-opacity:0;stroke-width:1;stroke:black" display="none"/>
    <path id="handler" d="M 20,0 L 40,20 L 20,40 L 0,20 Z " style="fill:white;fill-opacity:0;stroke-width:1;stroke:black" display="none"/>
</g>
```

View of the above code



All the components specified in the above code

b
```
<g id="product" name="product">
    <line id="line" x1="100px" y1="100px" x2="150px" y2="150px" stroke="black" stroke-width="2"/>
    <path id="arrow" d="M 0,0 L 50,25 L 100,0 Z" style="fill:#000000;stroke:#000000;stroke-width:1.0000000" />
</g>
```

Figure 1.4: Glyphs represented in SVG

(a) Code segment representing an ion. (b) Code segment representing a product connection.

A prototype tool was developed in Java to generate diagrams by reading the XML file and the SVG graphics. Values of the type attributes encoded in the XML file are used as the key to find the related SVG graphics that needs to be integrated to generate a visualization. The sequence diagram drawn in Figure 1.5 illustrates a detailed breakdown of the behavior and the functions supported by the tool. The steps from 1 to 7 shows the actions executed to generate a visualization for the selected biological model. The tool did not support automated layout of diagrams but it provided a graphical user interface to allow modelers to layout these glyphs to highlight the sequence of biological events.

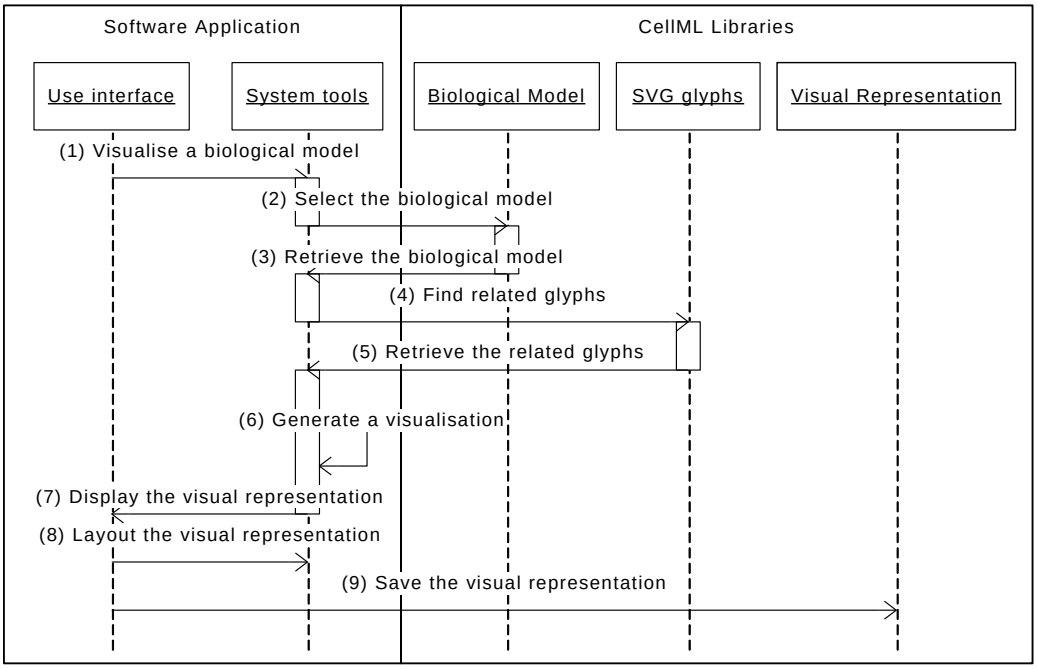


Figure 1.5: Sequence diagram illustrating the functionality supported by the prototype tool

Numbered event sequence shows the steps for generating a visualization for a selected model.

The visualization generated for cAMP/PKA cascade regulating L-type calcium channel using the prototype tool is illustrated in Figure 1.6. This diagram also shows how the visual language identifies the entities, processes, and modifiers. Glyphs with different shapes identify different types of biological entities. Lines with different types of line-ends represent the reactants, products, catalysts, inhibition, and activation.

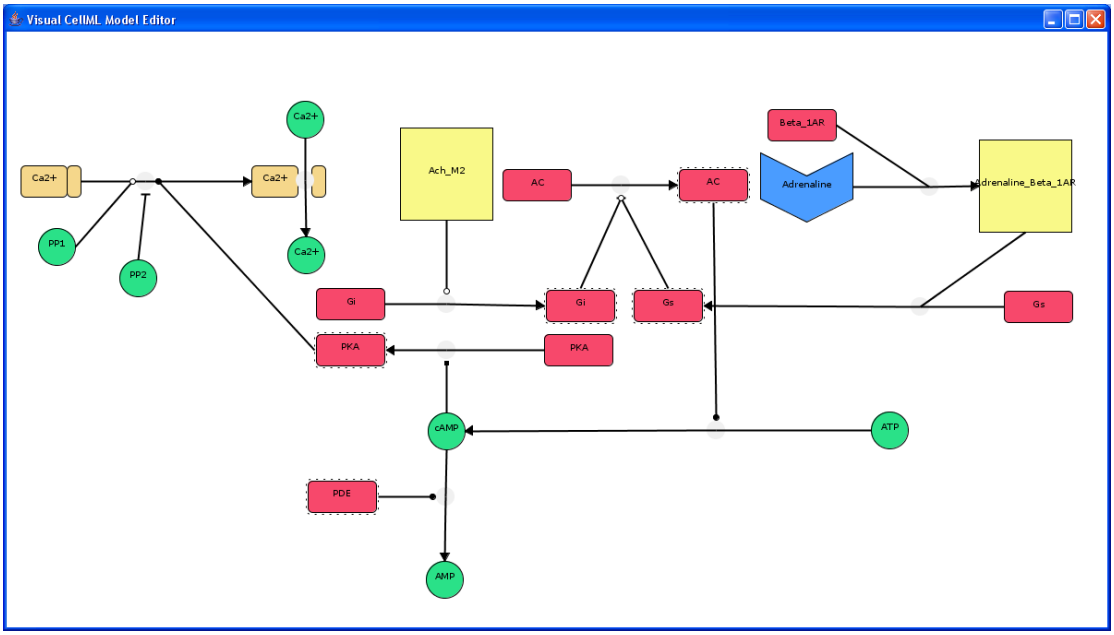


Figure 1.6: Visualization of cAMP/PKA cascade regulating L-type calcium channel generated from the prototype tool

The diagram identifies channels, proteins, small molecules, receptors, and complexes with unique glyphs. Processes are drawn as gray circles. The connections show the relationship between the processes and the entities.

Note that the XML biological model representing the CellML model is not mapped to the original CellML model. In order to visualize a CellML model via this biological model, these need to be linked. To enable this mapping, a modeler will need to restructure the CellML model by grouping or encapsulating components so that it captures the biological model that the modeler wishes to represent.

The prototype illustrated a simple method for creating a biological model, a simple visual language, and a tool that combines these developments, which could be used to generate a visualization of biological concepts captured in a CellML model. It highlighted the challenges that need to be addressed in detail to generate visualizations starting from CellML models depicting the underlying biological concepts. This includes:

- guidelines for structuring a CellML model. In order to visualize and annotate the underlying biology of CellML models, they need to be structured in a way that groups the biological concepts;
- representing the underlying biological concepts in CellML models. A method for capturing the underlying biological concepts which can be used to generate a graphical representation of the processes and entities;

- a visual language for representing the biological concepts modeled in CellML models and a method to associating it to the CellML elements;
- a tool that combines the notation, biology, and CellML to generate visualizations.

The next chapters explore these challenges.

Chapter 2 describes building smaller models that isolate biological concepts, and using these to build complex biological systems. These help to provide a clearer representation of the biophysical processes in the CellML model.

Chapter 3 describes developing an ontological framework representing the underlying physical and biological concepts modeled in CellML and annotating these to the model.

Chapter 4 describes an extension of the ontological framework to combine a visual language together with CellML ontologies and an algorithm for producing visual representations of the biology.

Chapter 5 describes the development of a software tool to combine CellML, ontological framework, and the visual language to support the visualization of the physical and biological concepts and its relationship of the underlying CellML model.

1.3 JOURNAL PUBLICATIONS FROM THIS THESIS

- S. M. Wimalaratne, M. D. B. Halstead, C. M. Lloyd, M. T. Cooling, E. J. Crampin, and P. F. Nielsen. *Facilitating Modularity and Reuse: Guidelines for Structuring CellML 1.1 Models by Isolating Common Biophysical Concepts. Experimental Physiology, 2009, 94, 472-485.*
- S. M. Wimalaratne, M. D. B. Halstead, C. M. Lloyd, E. J. Crampin, and P. F. Nielsen. *Biophysical annotation and representation. Bioinformatics, 2009, 25, 2263–2270*
- S. M. Wimalaratne, M. D. B. Halstead, C. M. Lloyd, M. T. Cooling, E. J. Crampin, and P. F. Nielsen. *A method for visualizing CellML models. Bioinformatics, 2009, 25, 3012-3019.*
- D. A. Beard, R. Britten, M. T. Cooling, A. Garny, M. D.B. Halstead, P. J. Hunter, J. Lawson, C. M. Lloyd, J. Marsh, A. Miller, D. Nickerson, P. M. F. Nielsen, T. Nomura, S. Subramanium, S. M. Wimalaratne, T. Yu. *CellML metadata standards, associated tools and repositories. Philosophical Transactions of the Royal Society A, 2009, 367, 1845-1867.*

- N. L. Novère, S. Moodie, A. Sorokin, M. Hucka, F. Shreiber, H. Mi, E. Demir, K. Wegner, M.Aladjem, S. M. Wimalaratne, F. Bergman, R. Gauges, P. Ghazal, K. Hideya, L. Li, Y. Matsuoka, A. Villeger, M. Courtot, U. Dogrusoz, T. Freeman, A. Funahashi, S. Ghosh, A. Jouraku, S.Kim, F. Kolpakov, A. Luna, S. Sahle, S. Watterson, I. Goryanin, D. Kell, K. Kohn, H. Kitano. *The Systems Biology Graphical Notation. Nature Biotechnology, 2009, 27, 735 – 741.*

2 GUIDELINES FOR STRUCTURING CELLML MODELS

The flexible structure of CellML allows modelers to construct mathematical models of the same biological system in many different ways. However, some modeling styles do not naturally lead to clear abstractions of the biophysical concepts and produce CellML models that are hard to understand and from which it is difficult to isolate parts that may be useful for constructing other models. This chapter advocates building CellML models which represents common biophysical concepts and, using these, to build mathematical models of biological processes that provide a close correspondence between the CellML model and the underlying biological process. Subsequently, models of higher complexity can be constructed by reusing these modularized CellML models in part or in whole. Development of CellML models that best describe the underlying biophysical concepts thus avoids the need to code models from scratch and enhances the extensibility, reusability, consistency, and interpretation of the models.

2.1 INTRODUCTION

CellML provides a flexible structure that is used to represent mathematical models describing a wide range of biological concepts. The CellML specifications describe the rules for constructing models, encoding the mathematics, embedding metadata about models, and

processing models [40]. Below, the elements of a CellML representation that describes a mathematical model of a biological process is briefly introduced. Then the potential pitfalls in some modeling styles are discussed.

CellML consists of 16 elements (Figure 2.1a). The model element is the root of a CellML model. It contains a list of import, units, component, connection, group, and/or Resource Description Framework (RDF) [11] elements. The import element lists imported components or units from other models. The units element is used to define the units that are associated with the model variables. The component element defines the building block of CellML models. Each component consists of a set of variable elements and math elements. The variables have attributes such as an initial value, units, public interface, and private interface. The math elements contain a set of mathematical equations that describe the behavior of the component within the model. These mathematical equations, embedded in a math element, are expressed using content Mathematical Markup Language (MathML) [4]. The connection element links two variables in different components together to allow the values of these variables to be exchanged between components within a model.

The group element can be used to introduce a hierarchy of components to the CellML model. It forms a tree with parent, child and sibling properties. Currently, CellML supports both containment, to describe the physical structure of the model, and encapsulation, to hide information about a set of components from the rest of the model. Each component has variables with public and private interfaces. The parent–child relationship within an encapsulation group is established by connecting the child component's public interface variables to the parent component's private interface variables. Components that are encapsulated by the same parent are called the sibling set. The parent component hides the details of the child components from the rest of the model, but amongst the sibling set they are visible.

Resource Description Framework (RDF) is a standard format for describing metadata. RDF elements are used to store metadata about a model or parts within the model. Several metadata specifications have been developed to represent particular kinds of information. These are as follows. First, the CellML Metadata Specification 1.0 to store information such as bibliographical reference details from which the model was taken and the biological species that are involved in the model. This specification also describes how metadata in general are represented within a model [41]. Second, the Simulation Metadata Specification to store particular simulation information such that specific results can be reproduced via a

simulation tool [42]. Third, the Graph Metadata Specification, which details a method on how to use the model simulation results to provide two-dimensional graphical representations [43].

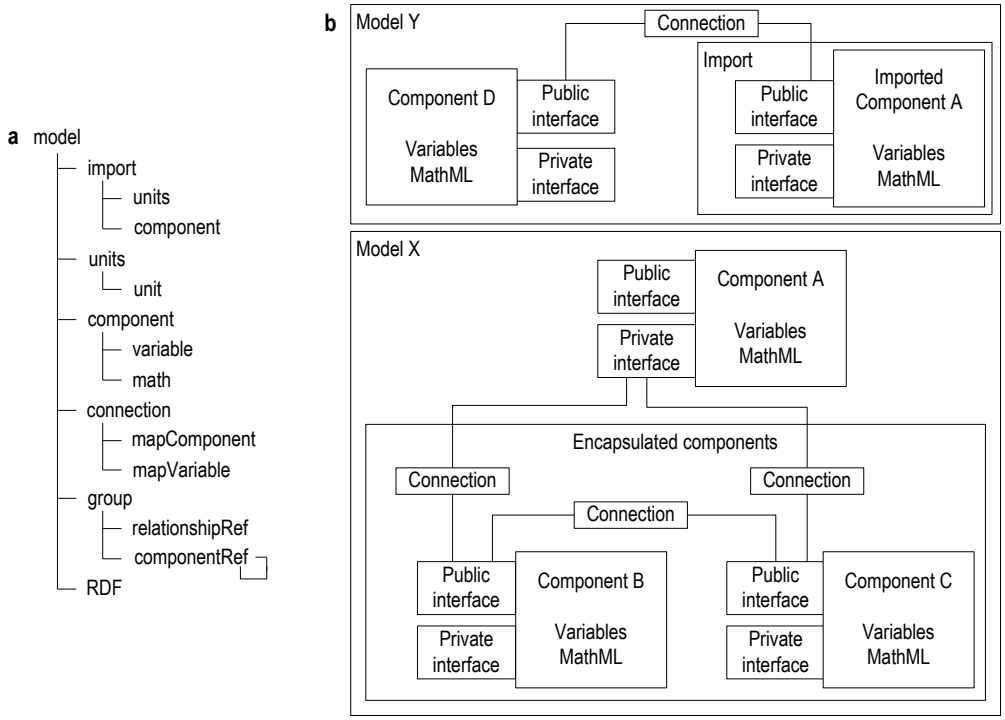


Figure 2.1: CellML structure

(a) The hierarchical structure of the CellML language. (b) Illustration of a CellML model using imports and encapsulation. Model X consists of three components, A, B and C. These are organized inside an encapsulation group, where A is the parent of B and C. Model Y imports component A from model X and contains one component named D. The parent component A in the encapsulation group in model X connects to component D in model Y via public interface variables.

CellML models form a network of interconnected components. Figure 2.1b illustrates a generic example using imports and encapsulation. The parent–child relationship within the encapsulation group in model X is established by connecting the public interface variables of component B to the private interface variables of component A. Components B and C are siblings within the encapsulation group and connect via their public interfaces. Component A is regarded as the parent of the encapsulated group and is part of the top-level component set in the CellML model hierarchy. The top-level components are explicitly imported into other models using the CellML import feature and the encapsulated components are implicitly imported via the encapsulating component. For example, model Y explicitly imports component A. Components B and C, and their connections, are implicitly imported.

The encapsulation feature supported in CellML provides a flexible structure that can be used to model multiple levels of detail. It can be used to plug different representations of a particular biophysical concept into and out of models without having to change the top-level model. Encapsulation can be used to replace components with simple formulations with component hierarchies that implement a particular subsystem in detail, thus reducing the complexity of the top-level model.

The import feature provides a simple method to construct integrated models by reusing existing models, either in part or in whole, within new models. The reuse of models helps to reduce errors and maintain consistency across models. The challenge is to develop a robust model that can be easily manipulated, integrated and extended after the model has been built.

The CellML specifications offer few guidelines for how to use the above elements to produce clear, reusable models. A mathematical model of a biological process can be represented in CellML in many different ways. The structure of a model mainly depends on the individual author's modeling style. CellML models can be constructed such that:

- one component can model multiple processes, which has the effect of hiding the details of the underlying biophysical concepts;
- parts of a concept are distributed over many components, with no clear identification of the underlying biophysical ideas;
- parameters, initial values, constants and approximated constants, which represent biophysical relations between variables, are defined within a component that describes the mathematics for a particular biological process, thus tightly coupling a component with the model-specific values; or
- the same mathematical construct can be structured in many different ways, presenting a problem of consistency across models.

These model structures make it harder for potential users to interpret the underlying biophysical concepts represented in the models. They also make it difficult to combine and reuse a particular biophysical concept because it is hard to isolate the reusable parts of the models.

Clever modularization and reuse of models has been studied in the past. The theoretical work of Ernst-Dieter Gillies and its implementation in ProMot/Diva focuses on modularizing biochemical processes [44, 45]. The methodology follows network theory [46] which promote reusable modeling entities that lead to the development of a modeling library within

the modeling tool ProMot. Modelica [47] is another language that supports modularized modeling of physical systems. It is based on Hilding Elmqvist and Francois E. Cellier work on modeling of physical systems following object oriented paradigm to facilitate the reuse of modeling knowledge via exchange of models and model libraries [48, 49].

Here we explore the process of constructing CellML 1.1 models to capture different levels of abstraction and identify reoccurring patterns that capture specific biophysical concepts in complex biological systems, thus enabling modelers to build models that are easy to interpret, reuse and extend. We refer to this process as 'modularization' of a CellML model. We present three guidelines for enhancing model structure by:

1. isolating biophysical concepts that can be shared between models at the component or whole model level;
2. constructing models combining the components, providing model-specific values and isolated biophysical concepts, which clearly identify the building blocks;
3. using encapsulation to reduce the complexity of models by creating sub-models or to expose points where different implementations of particular details can be swapped in and out.

2.2 METHODS

This section details the process of constructing modularized CellML models following the three guidelines. An example is used to demonstrate this process, starting from a set of equations that are then transformed into a modularized CellML model. Figure 2.2a details the starting set of equations for modeling the following reaction: formation of receptor–ligand–G protein phosphorylated complex, $Rlg \rightarrow Rlgp$.

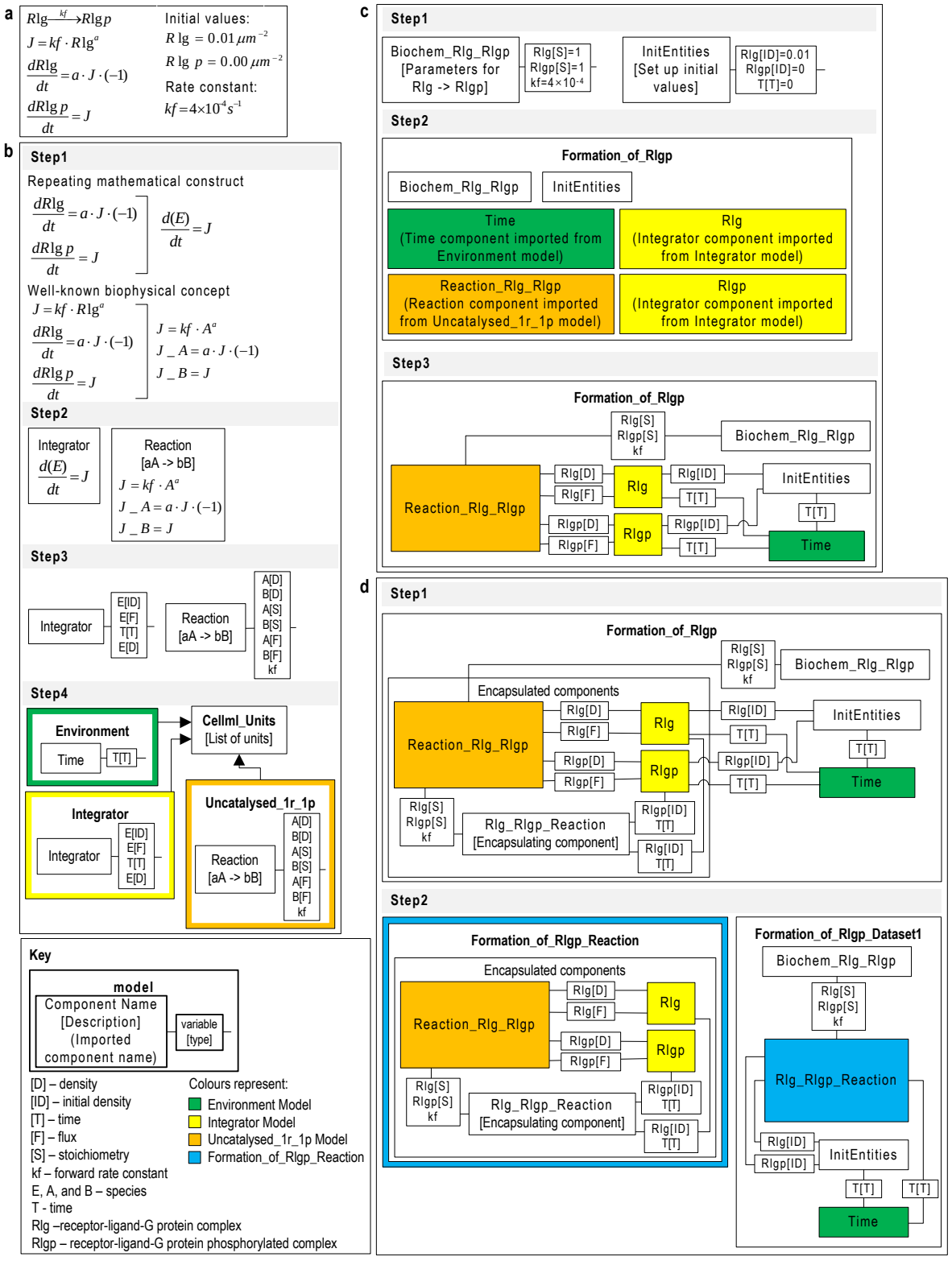


Figure 2.2: The three modularizing steps applied to an example, formation of receptor–ligand–G protein phosphorylated complex

(a) Mathematical equations for modeling the reaction. (b) The construction of reusable models, as follows: Step 1, identify repeating mathematical constructs; Step 2, isolate these mathematical constructs into separate components; Step 3,

generalize the interfaces to components; and Step 4, separate these components into models according to their categories. (c) The reconstruction of the original biological concepts, as follows: Step 1, isolate constants, parameters and initial values into their own components and assign their values; Step 2, import generic components; and Step 3, assign values to the variables in generic components by connecting them to the variables of the model-specific components. (d) the definition of encapsulated components, as follows: Step 1, group components using the encapsulation feature; and Step 2, separate encapsulated groups into individual models.

2.2.1 IDENTIFICATION AND REPRESENTATION OF BIOPHYSICAL CONCEPTS AND COMMON MATHEMATICAL CONSTRUCTS

Most models consist of a small set of mathematical constructs, each of which repeat. These often differ from other similar constructs in their constants and parameters that, in turn, specify their biophysical detail. Examples of these include equations describing rate kinetics to model stoichiometric reactions, Michaelis–Menten kinetics for modeling enzyme–substrate reactions and the Nernst potential equation for describing reversal potentials. Other ubiquitous mathematical constructs may also appear for pure mathematical convenience. Examples of these are the integration of reaction outputs where these are pooled in a model, and summing up a set of fluxes.

CellML 1.1 provides a mechanism by which repeating mathematical constructs can be represented once but specified as many times as required within a target model, each with specific constants or parameters. There are four steps to achieve this, as follows.

Step 1: (a) Identifying ubiquitous mathematical constructs that repeat throughout the model; and (b) identifying mathematical constructs that represent well-known biophysical phenomena.

In Figure 2.2b, Step 1 shows cases of (a) ubiquitous mathematical construct, i.e. the integration of the outputs of Rlg and Rlgp from the reaction; and (b) mathematics identifying a well-known biophysical concept, reaction kinetics for a first order, irreversible, mass action kinetics reaction, in a continuous scheme for one reactant and one product.

Step 2: Isolating the repeating mathematical expressions into separate components. These are referred to as ‘generic’ components and contain mathematical equations that describe a particular biophysical concept.

In Figure 2.2b, Step 2 shows the two components that are defined, Integrator and Reaction, which describe the integrator and the kinetics equation, respectively.

Step 3: Generalizing the interfaces to components. To make these components reusable, the equations and the interfaces of the components need to be generalized by defining variables that depend on other components, to pass in their parameters or constant values. When defining variables in CellML, it is possible to include the parameters or constant values using the initial value attribute, but this leads to components that are specific to modeling a particular biological concept, which restricts the reuse of the components. The variables of the generic components should not have values assigned to them using the initial value attribute.

In Figure 2.2b, Step 3 shows the Integrator and Reaction components associated with variables. The Integrator component defines variables to capture the initial area density (or concentration) of an entity (E[ID]), flux of an entity (E[F]), time variable (T[T]), and the calculated density (or concentration) of an entity (E[D]). The Reaction component defines variables to describe the density (or concentration) of the reactant (A[D]) and product (B[D]), the stoichiometry of the reactant (A[S]) and product (B[S]), the flux of the reactant (A[F]) and product (B[F]), and the forward reaction rate constant (kf). The example model defines initial values for Rlg and Rlgp and states the value of the reaction rate constant, but these initial and constant values for these variables are not assigned within the integrator and reaction components themselves. Since the Integrator component does not have any model-specific values, it can be used to calculate the rate of change of both Rlg and Rlgp entities. Similarly, the Reaction component can be used to model any first-order irreversible mass action kinetics reaction, in a continuous scheme for one reactant and one product.

Step 4: Separating generic components into individual models. The generic components can be categorized and separated into models according to the types of biophysical concepts that are being modeled. This enables modelers to more easily identify the generic components.

In Figure 2.2b, Step 4 shows the Integrator model and the Uncatalyzed_1r_1p model that are defined to categorize the Integrator component and the Reaction component, respectively. The Uncatalyzed_1r_1p model can be used to define components describing irreversible and reversible reaction kinetics involving one reaction and one product. The Environment model contains the Time component. Note that a separate model is introduced to describe the units. This prevents modelers from having to code the units for each of the models. It is also useful

to construct a separate component for defining the time variable because it is shared between models.

2.2.2 RECONSTRUCTION OF THE ORIGINAL BIOLOGICAL CONCEPTS BY COMBINING THE COMPONENTS, PROVIDING MODEL-SPECIFIC VALUES AND USING GENERIC COMPONENTS

Each model is associated with a unique set of measured or calculated values. These include constants, such as rate constants of each reaction, parameters, such as stoichiometry of species of each reaction, and initial values, such as initial concentrations of the species.

CellML provides a method for setting up these model specific values to reconstruct the original biological concepts. There are three steps to achieve this, as follows.

Step 1: Isolating all constants, parameters and initial values into their own components. The generic components do not define model-specific values and force the modeler to build separate components to describe such values. The variables inside the specific components have values assigned to them which are specific to the biological model.

The Biochem_Rlg_Rlgp component shown in Figure 2.2c Step 1 defines the values for the formation of the phosphorylated receptor–ligand–G protein complex (Rlgp), Rlg ->Rlgp. These values include the stoichiometry of Rlg (Rlg[S]) and Rlgp (Rlgp[S]), and the forward rate constant of the reaction (kf). Similarly, the InitEntities component defines the initial densities of Rlg (Rlg[D]) and Rlgp (Rlgp[D]).

Step 2: Importing generic components. Here the generic components are connected with specific components to allow the modeler to assign specific values to the variables. To achieve this, import the relevant generic components that contain the mathematical expressions for modeling the biological processes. Rename these generic components according to the specific biophysical concepts that are being modeled.

The generic components shown in Figure 2.2b Step 4 are imported to model the reaction. The Reaction component from the Uncatalysed_1r_1p model is imported to describe the formation of Rlgp. As illustrated in Figure 2.2c Step 2, this component is identified as Reaction_R1g_R1gp in the Formation_of_Rlgp model. The Integrator component is imported twice from the Integrator model to calculate the rate of change of Rlg and Rlgp. These components are identified as Rlg and Rlgp, respectively, in the Formation_of_Rlgp model.

Step 3: Assigning values to the variables in generic components. The values of the variables of the generic components are set by connecting them to the variables of the model-specific components. Connect the variables in the specific components to the appropriate variables in the generic components.

Figure 2.2c Step 3 illustrates the resulting model once the variables R1g[S], R1gp[S], and kf in Biochem_R1g_R1gp (the specific component) are connected to the variables A[S], B[S], and kf in Reaction_R1g_R1gp (the Reaction generic component). The variables R1g[ID] and R1gp[ID] in InitEntities (the specific component) are connected to E[ID] in Rlg (the Integrator generic component) and E[ID] in Rlgp (the Integrator generic component) to provide the initial densities.

2.2.3 USE OF ENCAPSULATION TO PARTITION THE DETAILS OF A MODEL INTO A HIERARCHY OF COMPONENTS

Most biological systems, and their representation in mathematical models, are very complex. It is possible to reduce this complexity by structuring it into sub-models. Often a group of components, each of which may represent a different biophysical concept, may form a known biological mechanism that is repeated throughout other models.

The CellML encapsulation feature can be used to group components with the following aims: (1) to reduce the complexity of a model by creating sub-models; or (2) to provide mechanisms for plugging in different implementations of a particular detail of a model. The advantage of this method is that all the mappings remain intact, which obviates the need to reconnect all the variables. There are two steps to achieve this, as follows.

Step 1: Grouping components using the encapsulation feature. Group together a set of components that create an assembled biological concept. Introduce a component that would make the encapsulated components visible to the top-level model.

Following the example model, the Reaction_R1g_R1gp, R1g, and R1gp components can be used in another model with a different set of constants, parameters, and initial conditions. The R1g_R1gp_Reaction component defines the R1g[ID], R1gp[ID], R1g[S], R1gp[S], kf, and T[T] variables. These variables allow modelers to set values for the encapsulated components (Figure 2.2d Step 1). It also defines R1g[D] and R1gp[D] variables, allowing modelers to access the values of R1g[D] and R1gp[D].

Step 2: Separating encapsulated groups into individual models. To further reduce the dependency between components, it is useful to move the encapsulated set of components and their related connections into a separate model. To access a set of encapsulated components that reside in a different model, it is necessary to import the encapsulating component into the top-level model.

As illustrated in Figure 2.2 Step 2, the Reaction_R1g_R1gp, R1g, R1gp, and R1g_R1gp_Reaction components and the connections between the encapsulated components can be moved to a different model. This reduces the complexity of the Formation_of_Rlgp model by minimizing the number of components and connections at the top level. The R1g_R1gp_Reaction component can be imported to the Formation_of_Rlgp model to access the encapsulated components Reaction_R1g_R1gp, R1g, and R1gp.

2.3 RESULTS

Several examples are used to illustrate the process of modularization of CellML models to clarify the relationships between the model and the biological processes described by the model, and to allow reuse of components between models.

2.3.1 MODULARIZATION OF THE G PROTEIN-COUPLED RECEPTOR (GPCR) CYCLE

Cooling et al. (2007) [50] described a GPCR cycle in their model of hypertrophic signaling pathways in the heart. Their model consists of a set of biochemical reactions using mass action kinetics. We use this example to illustrate model construction by isolating commonly used reaction kinetics into reusable generic components. The GPCR cycle consists of several biochemical interactions between a set of biological entities. Following Cooling et al. (2007) [50], these entities are as follows:

- R (receptor);
- Rl (receptor–ligand complex);
- Gd (G protein with attached guanosine diphosphate);
- Rg (receptor–G protein complex);
- Rlg (receptor–ligand–G protein complex);
- Rlgp (receptor–ligand–G protein phosphorylated complex);
- Gt (G protein with attached guanosine triphosphate).

There are six reactions that occur between these entities summarized in Figure 2.3, as follows:

- the formation of receptor-ligand complex, R1: $L+R \leftrightarrow Rl$;
- the formation of receptor-G protein complex, R2: $R+Gd \leftrightarrow Rg$;
- the formation of receptor-ligand-G protein complex, R3: $Rl+Gd \leftrightarrow Rlg$;
- the formation of receptor-ligand-G protein complex, R4: $L+Rg \leftrightarrow Rlg$;
- the formation of receptor-ligand-G protein phosphorylated complex, R5: $Rlg \rightarrow Rlgp$;
- the dissociation of receptor-ligand-G protein complex, R6: $Rlg \rightarrow Rl+Gt$.

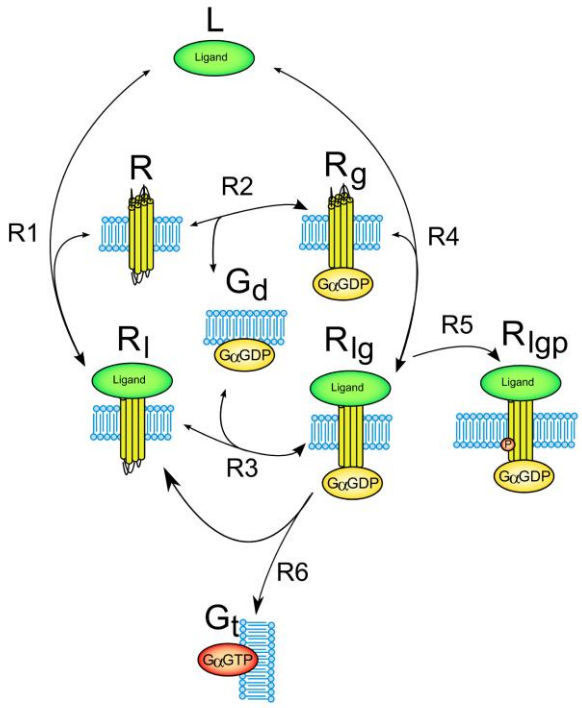


Figure 2.3: Schematic diagram of the GPCR pathway

The arrows indicate the products of the reactions. Bidirectional reactions have arrows pointing to both the reactants and the products. For further explanation, see main text. Figure derived from Figure 2 of Cooling et al. (2008) [51], with permission.

The mathematical equations in the GPCR model describe rate laws and conservation laws. The first step of the modularization process is to identify the biophysical processes and common mathematical formulations. Figure 2.4 illustrates the set of CellML models that were developed to model the components and which capture the generic biophysical equations of rate laws and conservation laws described in the GPCR cycle. These are as follows:

- the Environment model, which contains one component called time containing the variable time;
- the Integrator model, which contains a component with an equation to calculate the rate of change for an entity;
- the Rate_Constant model, which contains a component with an equation to calculate the reverse rate constant;
- the Uncatalysed_2r_1p_r model, which contains a component called Reaction describing the mathematics for a second-order forward reaction with two reactants, a first-order reverse reaction, reversible mass action kinetics and a continuous scheme;
- the Uncatalysed_1r_2p model, which contains one component with the mathematics for first-order irreversible mass action kinetics and a continuous scheme for one reactant and two products;
- the Uncatalysed_1r_1p model, which has one component containing the mathematics for first-order irreversible mass action kinetics and a continuous scheme for one reactant and one product;
- the Sum model, which contains a component for adding together the fluxes of the reactions.

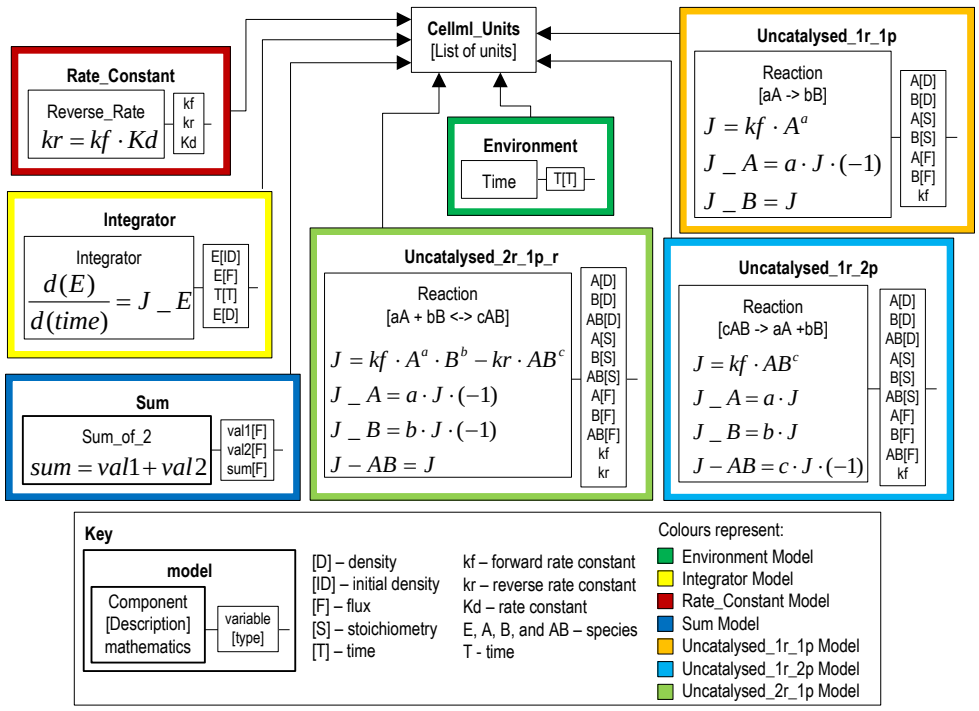


Figure 2.4: The set of CellML models developed to describe the GPCR cycle

These models contain generic components with generalized equations. The colors in these models are used to identify the imported components in Figure 2.5.

Figure 2.5 demonstrates the way in which these generic components can be used to build the final model the GPCR cycle. The Reaction component from the Uncatalyzed_2r_1p_r model is imported four times to model the R1, R2, R3, and R4 reactions. The Reverse_Rate component in the Rate_Constant model is imported three times to calculate the reverse rate constants for the reactions R1, R2, and R4. The Integrator component is imported eight times to model the L, R, Rl, Gd, Rg, Rlg, Rlgp, and Gt entities. It is a requirement of CellML that connecting variables must have the same unit dimensions. This feature is used for model validation, but results in the duplication of components with similar mathematics where the variables are assigned different units.

The Sum_of _2 component from the Sum model is imported four times to add the L, R, Gd, and Rg fluxes. In cases where more than two fluxes are involved, the fluxes are added together by connecting multiple Sum_of_2 components. For example, Rlg requires three Sum_of_2 components to be connected together in order to calculate the total flux.

The white boxes on the left and right of Figure 2.5 list the components that have been defined to provide the reaction kinetic parameters and initial conditions for the imported components.

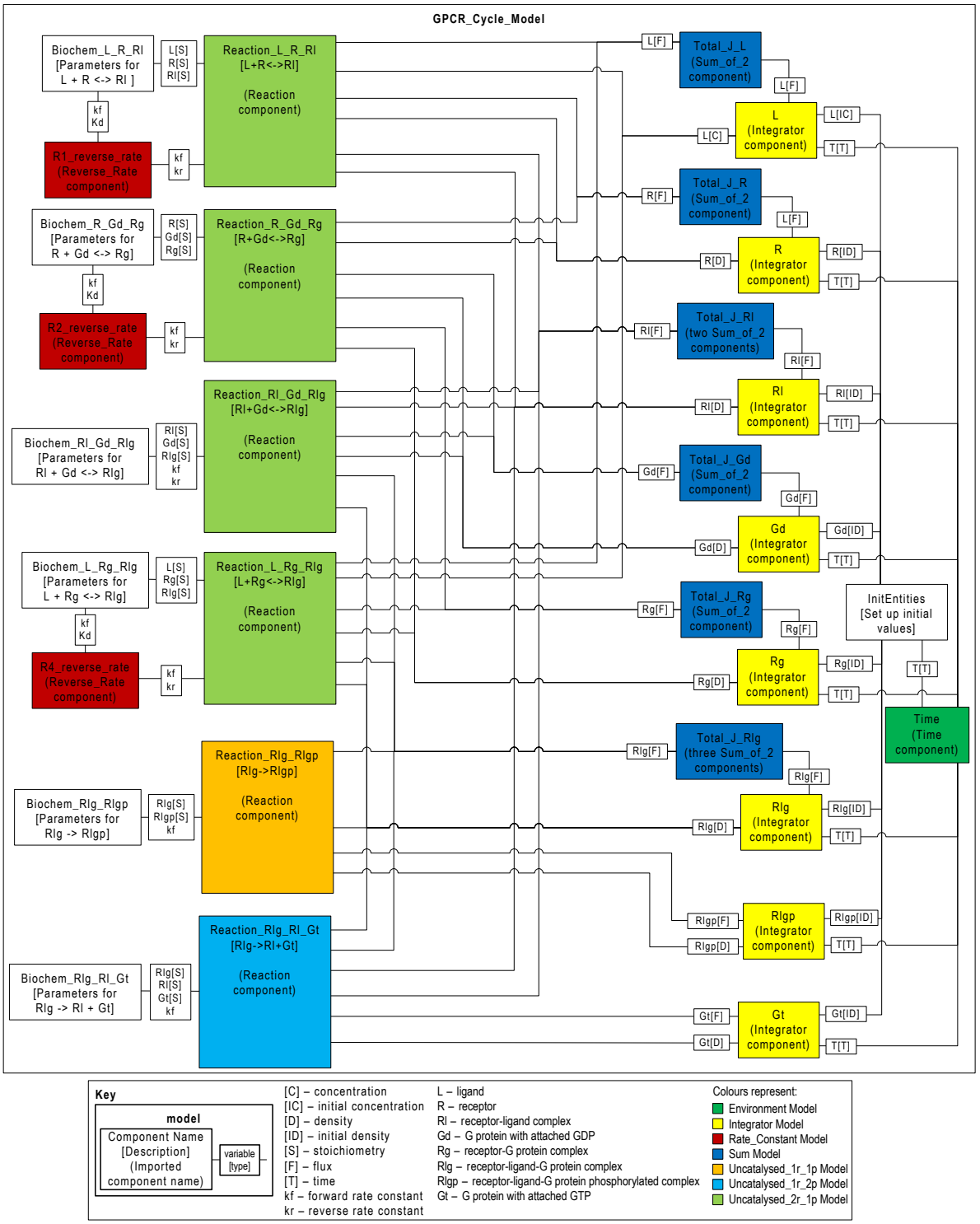


Figure 2.5: The GPCR cycle modeled in CellML

The model has a modular structure to facilitate component reuse. The lines between components are directionless intentionally, since they represent the quantities that are shared between components.

2.3.2 MODULARIZATION OF THE HODGKIN–HUXLEY MODEL

In general, electrophysiological models describe ion flow through channels, exchangers, pumps and membrane leakages. For example, the seminal Hodgkin–Huxley model [52] describes the action potential in squid giant axons. The model includes a description of a sodium current through a gated sodium channel, a potassium current through a potassium channel, and a small leakage current across the cell surface membrane [52]. Most of these channels are reused in other electrophysiological models, many of which are contained in the CellML repository. The identification of each of these as functional units that make up a complete electrophysiological model would make it easier to understand these models and also increase the reusability of these channels.

Figure 2.6, demonstrates the components that contain a set of commonly used equations when modeling voltage-gated channel activity. These include the following:

- the Nernst_Potential model, which contains one component with the mathematics for calculating the reversal potential (EA);
- the Current model, which contains one component for calculating the current flowing through a channel (I A);
- the Gate model, which contains one component with the mathematics describing the gate opening and closing kinetics;
- the Membrane_Potential model, which contains one component with the mathematics for describing the membrane potential;
- the Sum model, which contains one component for summing ionic currents.

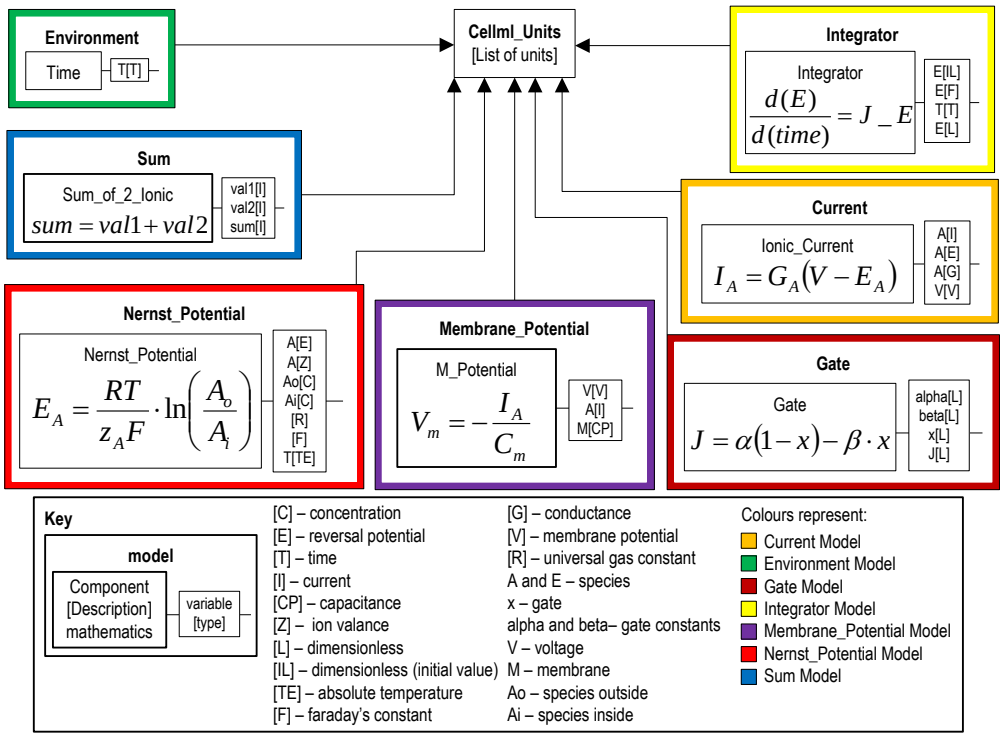


Figure 2.6: A set of models describing several electrophysiological concepts
Note that all the models contain reusable generic components. For this reason 'A' is used to denote the ion (which could be sodium, potassium, calcium or chloride) and 'x' is used to denote the gate (which could be m, h, or j in the case of the sodium channel). The colors in these models are used to identify the imported components in Figure 2.7 and Figure 2.8.

The generic components consist of a set of simplified equations commonly used in the literature when modeling voltage-gated channel activity. These can be reused to calculate model-specific values. For example, the Nernst Potential component can be used to calculate the Nernst potential of any ion.

Figure 2.7a illustrates the way in which the Hodgkin–Huxley model is built using these generic components, multiple levels of imports and encapsulation. One of each of the Ionic_Current and M_Potential generic components is imported to model the leakage current and membrane potential, respectively. The Sum_of_2_Ionic component is imported twice and connected together to add the sodium, potassium, and leakage ionic currents.

Figure 2.7b shows a model created to describe the sodium channel. This model contains the Na_Conductance and generic Ionic_Current components to calculate the sodium conductance and ionic current, respectively. The generic Gate component is imported twice to model the m and h gates. Each gate component is connected to an imported Integrator

component to calculate the rate of change of the gate activity over time. The Fast_m_Gate and Fast_h_Gate components provide parameters for calculating the rates of opening and closing of the m and h gates, respectively. Together, these components model the sodium channel. Since this formulation is frequently used in the literature, the encapsulation feature is used to group these components together to enable reuse of this entire sodium channel. The Na_Channel component makes the encapsulated components visible to the external models. The Hodgkin_Huxley model imports the Na_Channel component from the Na_Ionic_Current model. Similarly, Figure 2.7c illustrates how the K_Channel component is modeled.

The sodium, potassium and leakage current reversal potentials (ENa, EK, and EL, respectively) are not calculated using the Nernst potential equation. Instead, they are calculated in terms of the membrane equilibrium potential (ER). Three components, called Na_Nernst_Potential, K_Nernst_Potential, and L_Nernst_Potential, are introduced to calculate these reversal potentials because they define approximated constants.

The Na_Ionic_Current and K_Ionic_Current models can also be simplified further by abstracting the sodium and potassium gates into two separate models.

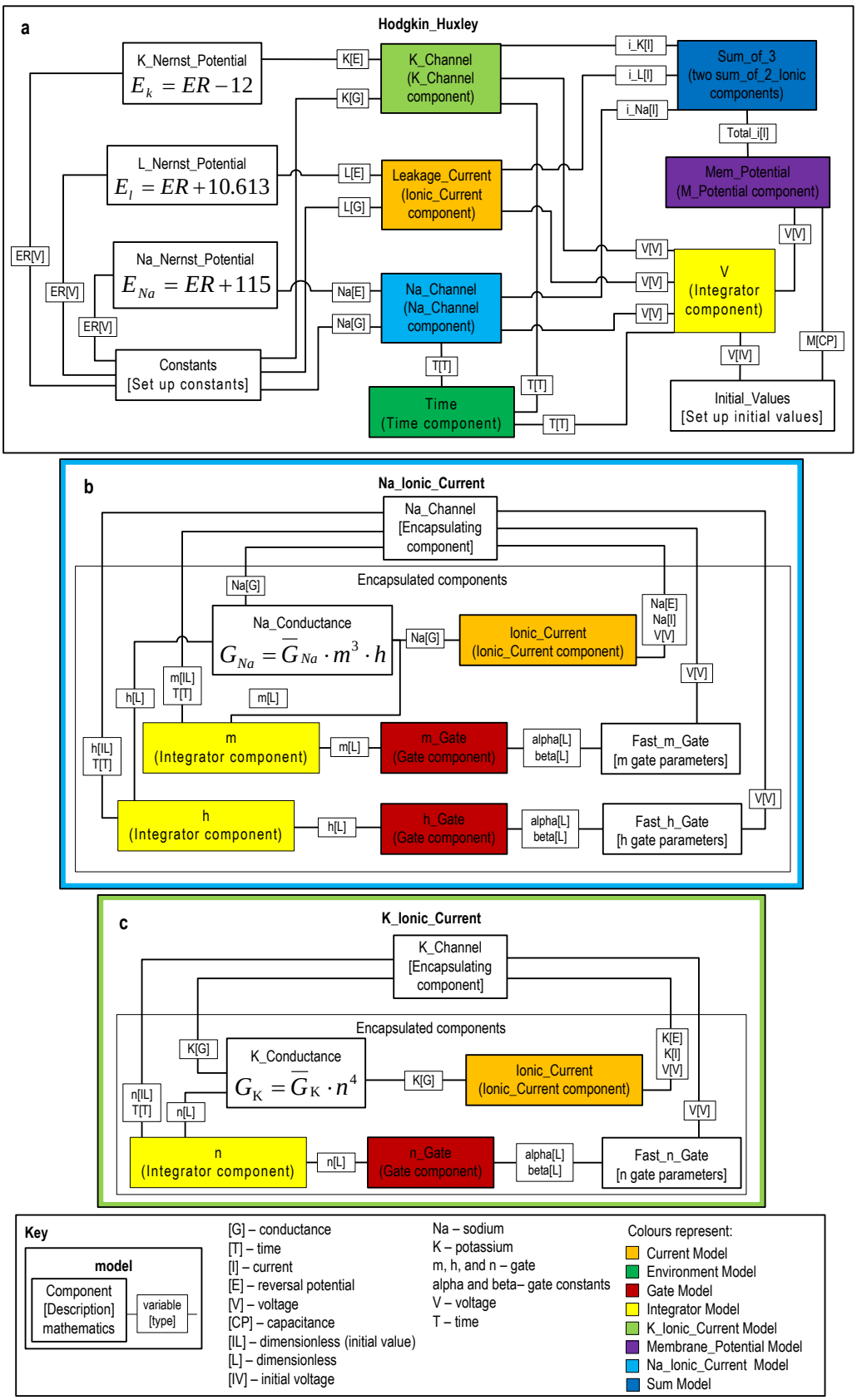


Figure 2.7: Description of the Hodgkin–Huxley model in CellML

(a) A representation of the Hodgkin–Huxley model using the reusable generic components illustrated in Figure 2.6 and encapsulation to hide the calculations for

the sodium channel activity. (b) The structure of the CellML model which encapsulates the sodium ionic current with two gates. (c) The structure of the CellML model which encapsulates the potassium ionic current with one gate. The lines between components are directionless intentionally, since they represent the quantities that are shared between components.

2.3.3 MODULARIZATION OF THE NOBLE MODEL

The Noble model describes the long-lasting action and pacemaker potentials of the Purkinje fibers of the heart [53]. It is developed from the equations of the Hodgkin–Huxley model. The potassium current equations defined in the Noble model identify potassium ion flow through two types of channels in the membrane. The sodium current equations are very similar to those of Hodgkin–Huxley model but use different parameters. Here we describe the construction of the Nobel model from generic components, as well as by reuse of components from the Na_Ionic_Current and K_Ionic_Current models developed for the Hodgkin–Huxley model.

Figure 2.8a illustrates the top-level Noble model. Similar to the Hodgkin–Huxley model, Ionic_Current, M_Potential, and Sum_of_2 Ionic components are imported to model the leakage current, membrane potential, and total ionic currents, respectively. Na_Nernst_Potential, K_Nernst_Potential, L_Nernst_Potential, Constants, and Initial_Values components defined in the Noble model describe the model-specific values.

Figure 2.8 shows the extended Na_Ionic_Current model that is developed to support the sodium ionic current defined in the Noble model. The Fast_m_Gate_Noble and Fast_h_Gate_Noble components are introduced to provide the gate parameters defined in the Noble model. The GNa_Noble component calculates the sodium conductance. The Na_Channel_Noble component defines an encapsulated group, which includes the components needed for modeling the sodium current defined in the Noble model. A set of new connections is introduced to connect the new components with the existing components. The Na_Channel_Noble component is imported into the Noble model. Similarly, Figure 2.8c illustrates the extended K_Ionic_Current model developed to model the flow of potassium ions as described in the Noble model.

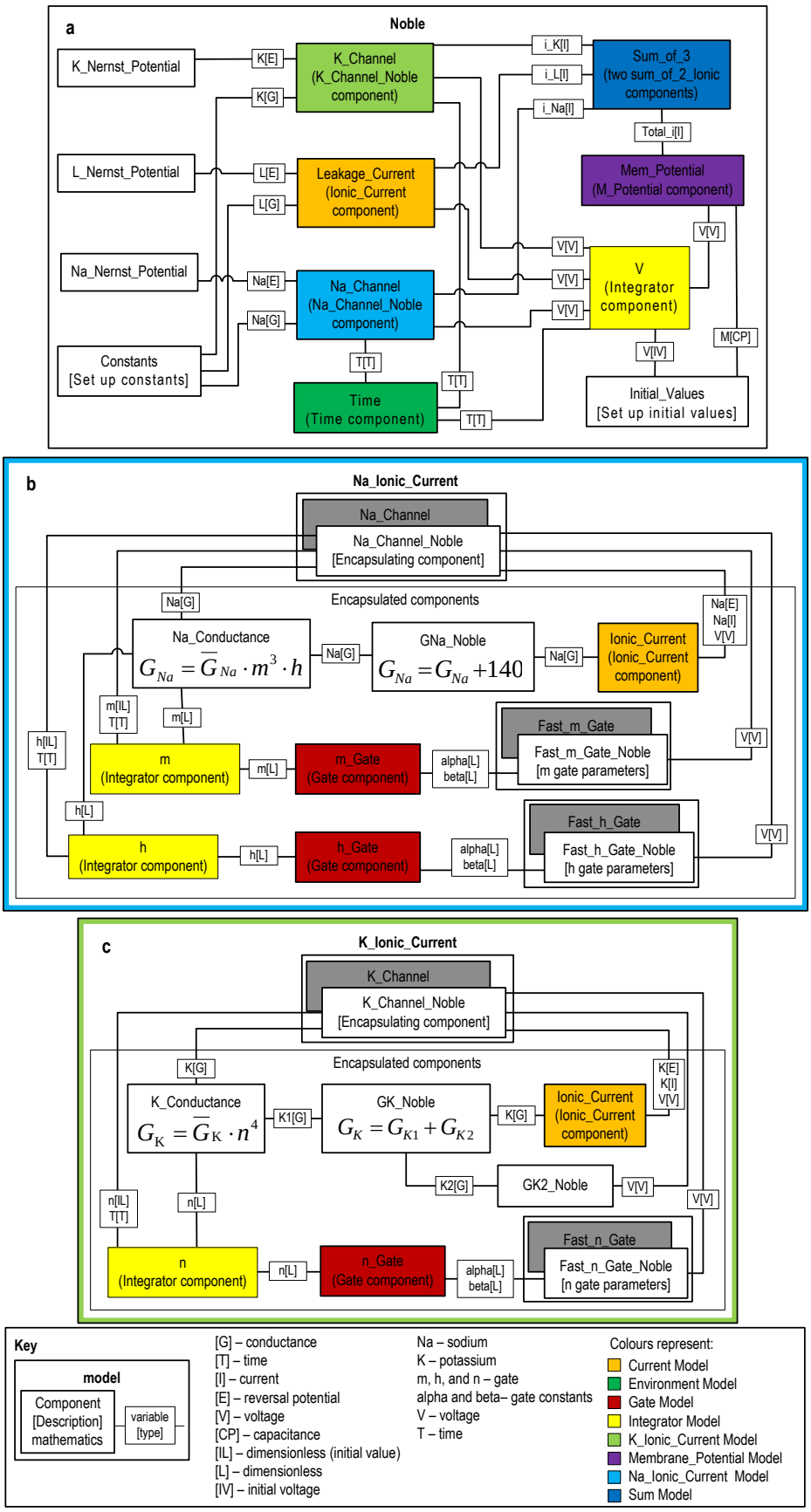


Figure 2.8: Describing the Noble model in CellML

(a) A representation of the Noble model using the reusable generic components illustrated in Figure 2.6. (b) Extended Na_Ionic_Current model to describe the sodium current with new parameters defined in the Noble model. A new encapsulation group is introduced to group the components related to the Noble model. The Na_Channel, Fast_m_Gate, and Fast_j_Gate components are shaded in dark grey because they are not used to model the Noble model. (c) Extended K_Ionic_Current model to describe the potassium current defined in the Noble model. The K_Channel and Fast_n_Gate components shaded in dark grey are not used to model the Noble model. The lines between components are directionless, since they represent the quantities that are shared between components.

2.4 DISCUSSION

The process described here isolates the biophysical concepts represented in a CellML model into logical abstractions, which helps modelers to avoid creating complex representations of biophysical concepts in CellML and provides a better correlation between the CellML model and the underlying biological process it describes.

The development of models with different levels of abstraction is dependent on the personal preferences of the modeler. It is not always useful to force specific levels of abstraction onto the modeler. However, careful attention needs to be given to cases where various components would be well suited to being modular, so that the outcome is a model that is amenable to reuse. The building of reusable generic components is a necessary process to improve the interpretation and reuse of CellML models.

Use of generic components also reduces the complexity of the modeling process. When a model is constructed without imports, the mathematics and variables of each component often need to be repeated. Every component needs to be read in order to understand a model in its entirety. Use of common components simplifies the process of understanding complex models by preventing modelers from having to read the mathematics of every component.

Models can also be abstracted to capture sets of biological concepts and can be encapsulated and reused in more complex models. The GPCR cycle model can be encapsulated and reused to model the Inositol 1,4,5-trisphosphate (IP3) signaling cascade [50]. The conceptual modularization of the IP3 signaling cascade has been previously described by Cooling et al. (2008) [51]. While that study isolated the GPCR cycle, we are

further componentizing modules to identify the reusable components within the GPCR cycle to make it easier to interpret the model and identify the individual processes and entities.

Generic components should be used where possible when building a model, and the approximated constants should only be used when there is insufficient information to use the generic component. Since most of the models are not complete, models will generally have some approximated constants. The advantage of modularization is that the modeler can replace these approximated constant components with the generic components at a later stage when key unknown values become available. For example, the Na_Nernst_Potential component in the Hodgkin–Huxley model can be replaced by the Nernst_Potential component when the internal and external sodium concentrations are known. This illustrates how the isolation of biophysical concepts allows modelers to build reusable and extensible models that are easier to interpret and extend with new information.

The Hodgkin–Huxley example demonstrates how encapsulation may be used to reduce the complexity of a model. The Hodgkin_Huxley model imports the Na_Channel component from the Na_Ionic_Current model, thereby reducing the complexity of the top-level model by hiding calculations of the sodium gate activities and current flow from the Hodgkin_Huxley model. It also shows how the encapsulated groups may be reused across other electrophysiological models by importing the Na_Channel, which would enable visibility to the encapsulated components.

As research continues, models are improved and new parameters are defined. Development of models by isolation of the biophysical concepts also supports better extension and reusability of parts of models. The Noble example illustrates how components of existing models, that were developed to build the Hodgkin–Huxley model, can be reused. This method of reuse is not possible with the existing Hodgkin–Huxley model in the CellML repository, owing to the way it has been structured with tight coupling between components that model different biophysical concepts and model-specific values. In summary, the import and encapsulation features can be used to modularize models, build models from biophysically based components and promote the reuse of CellML models.

While the most commonly used biophysical concepts will be relatively straightforward to define and introduce into a library of models of generic components, we acknowledge that the development of such a library covering all the biophysical concepts in CellML models will be a comparatively lengthy process. Currently, these generic components are identified

by the modeler. A next step is to introduce a method to generate the biophysical concepts modeled in generic components through mathematical normalization. For example: *aA+bB->cC+dD* can be used to generate every kinetic reaction. The aim would be to introduce a meta-model, which captures the normalized mathematical equation, and combine it with a rule set that could be used to manipulate the meta-model in order to generate the generic CellML components that capture elementary biophysical concepts.

The modularity principle of the network theory introduced by Gilles identifies two types of elementary units called components and coupling elements [46, 54]. Components define physical quantities like energy, mass, or momentum and coupling elements describe the fluxes between components. Components and coupling elements can be defined on different hierarchical modeling levels. Components and coupling elements can be aggregated to a single component on a higher level or elementary units on one level may be decomposed into components and coupling units on a lower level [46, 54]. This modularity work described by Gilles was explored later in the project. Extending the modularization guidelines for CellML described here would benefit from further exploration of the interesting modularity concepts described by Gilles.

Reduction of the complexity at the top level increases the consistency and ease of reuse of CellML models. However, this results in a complex organization of components, abstract variables and connections. The editing and creation of such a CellML model at the XML level is difficult. Therefore, it would be useful to provide tools to permit visual creation and viewing of CellML models to support the effective use of generic components and encapsulation.

This study also brought to light a number of opportunities for improving the CellML language. The formation of mathematical operations over arbitrary numbers of inputs is not possible in CellML. It requires modelers to import the same components multiple times, for example, importation of the Sum_of_2_Ionic component twice to add sodium, potassium, and leakage currents in the Hodgkin–Huxley example. This increases the number of components and connections, consequently increasing the complexity of the model. Such mathematical operations are common in modeling biophysical models. Introduction of multiplexers and variables that can carry matrices will help to reduce the number of imported components and connections further. This feature will not enable the reuse of more models but it will reduce the complexity of models.

Systems Biology Modeling Language (SBML) [13] is another mathematical modeling language developed specifically for describing biochemical networks. The structure of SBML itself captures some of the biophysical concepts described in the model. For example, it defines reactions where the kinetic laws are described. In contrast, CellML is abstract, and components do not define what can be included. By modularizing CellML models, we are able to identify and isolate these reactions. The SBML specification does not currently support features to enable reuse of existing models [29]. This forces modelers to construct models from scratch and acts as a barrier to share the models. The advantages of using CellML are that it provides the flexibility required to model different types of biological concepts and it also contains the features necessary to construct reusable models.

The existing CellML Model Repository [17] is composed mostly of CellML 1.0 models. Many of the models form reasonable abstractions of biophysical processes. However, they would benefit from further attention to encapsulation. The CellML 1.0 specification does not support imports, and these models are unable to benefit from the movement of common structures into shared resources. It is the intention of the CellML community to translate the existing CellML 1.0 models into a CellML 1.1 format using the import and encapsulation features to build a library of common and reusable components and models. The aim is to provide a library of generic component models that can be used to build specific biological concepts.

The biophysical concepts isolated in the CellML models are not conveyed in a machine-interpretable manner. This requires the annotation of CellML elements to an ontology defining biophysical concepts. For example, the generic components describing mass action kinetics in the GPCR cycle can have one-to-one mappings to Systems Biology Ontology (SBO) [26] rate law terms. This is discussed in the next chapter.

The building of models by following the modularity principles should enable modelers to develop models that are easier to interpret and reuse. Integration of these with an ontological framework will enable modelers to overcome some of the difficulties associated with building complex biological cell models.

3 BIOPHYSICAL ANNOTATION AND REPRESENTATION OF CELLML MODELS

The focus of CellML is the representation of mathematical formulations of biological processes. The language captures the mathematical and model building constructs well but does not lend itself to capturing the biology these models represent. Such information needs to be represented as metadata. The previous chapter demonstrated how to structure models in a way that it best describes the biophysical concepts and abstractions that the modeler wishes to demonstrate. This chapter describes the development of an ontological framework for annotating CellML models with biophysical concepts. We demonstrate that, by using these ontological mappings, in combination with a set of graph reduction rules, it is possible to represent the underlying biological process described in a CellML model.

3.1 INTRODUCTION

CellML models do not capture biological information explicitly in their model properties. This feature of the CellML language enhances its flexibility, enabling it to describe a wide range of biological processes without the need to include a large number of domain-specific language constructs. The result of this, though, is that biological information, such as the

entities and processes described by the model, are not represented or only weakly implied in the variable and component names. Physical data, such as units of measurement and type of mathematical formulation, are captured in the units of the variables and the structure of the mathematical equations. Complicated relationships often exist between the biological process and the mathematical model describing the biological process. A biological concept may be represented using several mathematical equations and variables, spread across multiple components, to form a complex CellML structure. It is a difficult task to pull together the relevant information to discover the underlying biology.

A parallel specification to the CellML/XML language specification, the CellML Metadata specification [41], provides a method for attributing extra information to CellML elements. The CellML metadata specification uses the Resource Description Framework (RDF)[11] and RDF Schema (RDFS) [55], which are standard formats based on XML, for describing metadata. A CellML-specific element, <cmeta:bio_entity>, is used to define biological entity metadata. A biological entity can refer to a human readable name, database identifier, or both. This provides a simple method for annotating CellML elements with biological data.

This kind of annotation only provides a simple labeling mechanism and fails to capture the relationships between biological processes and entities. One could argue that information about these relationships is captured implicitly in the 'connection' structures of CellML that represent the transfer of quantities between mathematical operations. But transforming such information into meaningful biological descriptions is very complex; it also misses the intention here which is to help the author to accurately describe the biological knowledge or intention of the mathematical model they are presenting.

An ontology is a formal representation of a set of concepts, and the relations between those concepts, within a specific domain of knowledge. It defines a common vocabulary and set of rules to unambiguously represent information [34]. Formalizing data in an ontological format involves clearly identifying concepts, defining the characteristics of these concepts, providing specific instances of the concepts, and describing ways in which concepts and instances can be related.

Biological ontologies are being used to define a standard representation of biological models [56]. There exist many biological ontologies which capture different domains of

biology. Ontologies that are particularly relevant to the annotation of physical and biological concepts modeled in CellML include:

1. BioPAX ontology – which describes metabolic pathway data, molecular binding interactions, hierarchical pathways, and some signal transduction pathway and gene regulatory network concepts [57];
2. Foundational model of anatomy ontology (FMA) – which captures the structural relationships between the organs and tissues of the human body [58];
3. Gene ontology (GO) – which represents genes, gene products, and gene sequences of a variety of plant, animal and microbial genomes [59];
4. NCI thesaurus – which provides a reference terminology for describing cancers, drugs, therapies, anatomy, genes, pathways, cellular and subcellular processes, proteins, and experimental organisms [60];
5. Systems biology ontology (SBO) – which addresses biological concepts related to computational modeling [30].

A controlled vocabulary is an organized list of terms that are used to annotate data so that they can be easily retrieved. The ontologies listed from 2 to 5 provide controlled vocabularies which could be used for annotating CellML structures representing biological entities and processes or, as in the case of SBO, can also be used to annotate the physical concepts such as quantitative parameters and mathematical expressions modeled in CellML. Such an annotated CellML model can often result in an overcomplicated representation of the underlying biophysical concepts. The notion of 'views' can be useful here to refer to simplified representations of the complex annotated CellML models. One such view would be a 'biological view' that explicitly shows the biological processes and entities, and the relationships between them. However, these ontologies alone do not provide the means for creating a biological view which can be used to show the underlying biological concepts and relationships captured in a CellML model. Whereas BioPAX describes biological concepts in such a way that a biological view can be built. BioPAX provides a set of classes that can be used to model a large number of the biological process and entity types described in CellML models. The process types include transport, covalent, non-covalent, and modulation interactions. The entity types include protein, complex, small molecule, RNA, and DNA. Supplementary Material A describes the initial research carried out to compare the ontologies BioPAX, SBO, and GO.

Figure 3.1 shows the way in which BioPAX can be used to build an abstract model of the formation of cyclic adenosine monophosphate (cAMP) [38].

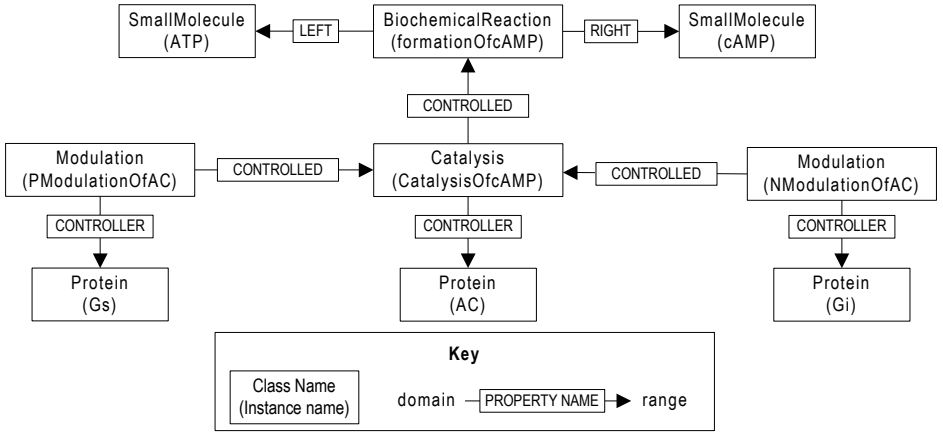


Figure 3.1: Modeling the formation of cyclic adenosine monophosphate (cAMP) in BioPAX

In BioPAX concepts such as proteins or reaction types are defined using Classes, objects such as specific proteins are expressed as instances, and the relationship between the objects are defined using properties. Adenylyl cyclase (AC) catalyses this reaction, stimulatory G protein (Gs) positively modulates the catalysis process, and inhibitory G protein (Gi) negatively modulates the catalysis process. The process of the formation of cAMP is modeled using the BiochemicalReaction class. AC, stimulatory G protein (Gs) and inhibitory G protein (Gi) entities are represented using the Protein class. Adenosine triphosphate (ATP) and cAMP are represented using the SmallMolecule class. The interaction between AC and the biochemical process is identified as a Catalysis process. The interactions between the G proteins and the Catalysis process are identified as Modulation processes.

A limitation of BioPAX at present is that, while it captures the concepts and relations surrounding biochemical pathways, it does not provide an equally rich set for electrophysiological concepts. For example, BioPAX cannot be used to represent a transport process via a voltage activated gated channel. Even though BioPAX can be extended to support the modeling of electrophysiological processes, the difficulty is that these concepts have to be recast in the form of biochemical mechanisms, which is often not the relevant interpretation required by the modeler at the point of annotation.

Although it is possible to add biological meaning to CellML elements by mapping each component to a BioPAX process instance and each variable to a BioPAX entity instance, the semantic differences between BioPAX and CellML models make it harder to establish a one-

to-one relationship with BioPAX instances and CellML elements. For example, the components in a CellML model that do not have any biological meaning (i.e. components calculating rate constants) cannot be mapped to a BioPAX instance. Furthermore, without an explicit mapping between CellML and BioPAX, it makes it harder to understand the relationship between the CellML elements and BioPAX instances. These limitations make it difficult to adopt BioPAX as a standard to annotate the biological and physical concepts modeled in CellML. Initial efforts to use BioPAX to annotate CellML models are further described in Supplementary Material B section 2.

Hence, in the absence of an existing ontology which is capable of fully annotating a CellML model with physical and biological information, we were compelled to develop our own ontological framework. Here we describe the process of developing such a framework which is designed to structure the biophysical concepts captured in CellML models and allow modelers to explicitly annotate a CellML model with physical and biological information. The annotated information is used to help users to clearly indentify the underlying physical concepts captured in the CellML model, without the need to go through all the individual CellML elements. Similarly, the same information can also be used to construct a biological view of the CellML model.

3.2 METHODS

Our ontological framework is modeled using the Web Ontology Language (OWL) [35]. OWL is a knowledge representation language which is based on RDF/RDFS. It provides additional modeling concepts along with formal semantics when compared with XML and RDF. OWL can be used to represent concepts, relations between them, and identify members of these concepts. The OWL properties describe the relationships between instances or constraints defining necessary and sufficient conditions for being classified as a member of a class. This class-based knowledge representation enables automated reasoning to check the consistency of models. OWL also provides the basis for efficient querying mechanisms and, furthermore, it can easily be integrated with other ontologies.

OWL has three sublanguages OWL-Lite, OWL-DL, and OWL-Full. OWL-Lite can be used to construct a classification hierarchy and simple constraints with limited expressiveness. OWL-DL is conceptually based on descriptive logic and supports maximum expressiveness while maintaining computational completeness and decidability. OWL-Full supports maximum expressiveness with no computational guarantees. We use OWL-DL to

construct our ontologies to take advantage of its ability to model incomplete and irregular knowledge, which is well suited for modeling biological facts [61].

The process we describe here for the construction of a biological view of a CellML model involves three steps:

1. transformation of the CellML/XML model into an OWL format (CellML/OWL). This uses an ontology for representing CellML models and a method for binding elements of these back to the CellML/XML model;
2. annotation of the CellML/OWL model to an OWL model of biophysical concepts. This involves developing an ontology that represents the physical and biological concepts that are described in CellML models (CellMLBiophysical/OWL);
3. simplification of the CellMLBiophysical/OWL model using the ontological mappings, in combination with a set of graph reducing rules, to represent the underlying biological view of the CellML model.

3.2.1 TRANSFORMATION OF A CELLML/XML MODEL INTO A CELLML/OWL MODEL

The CellML/OWL ontology represents a CellML/XML model in OWL (Figure 3.2). CellML/OWL models are created by a programmed transform that for each element in the CellML/XML model, creates the analogous representation in the CellML/OWL model.

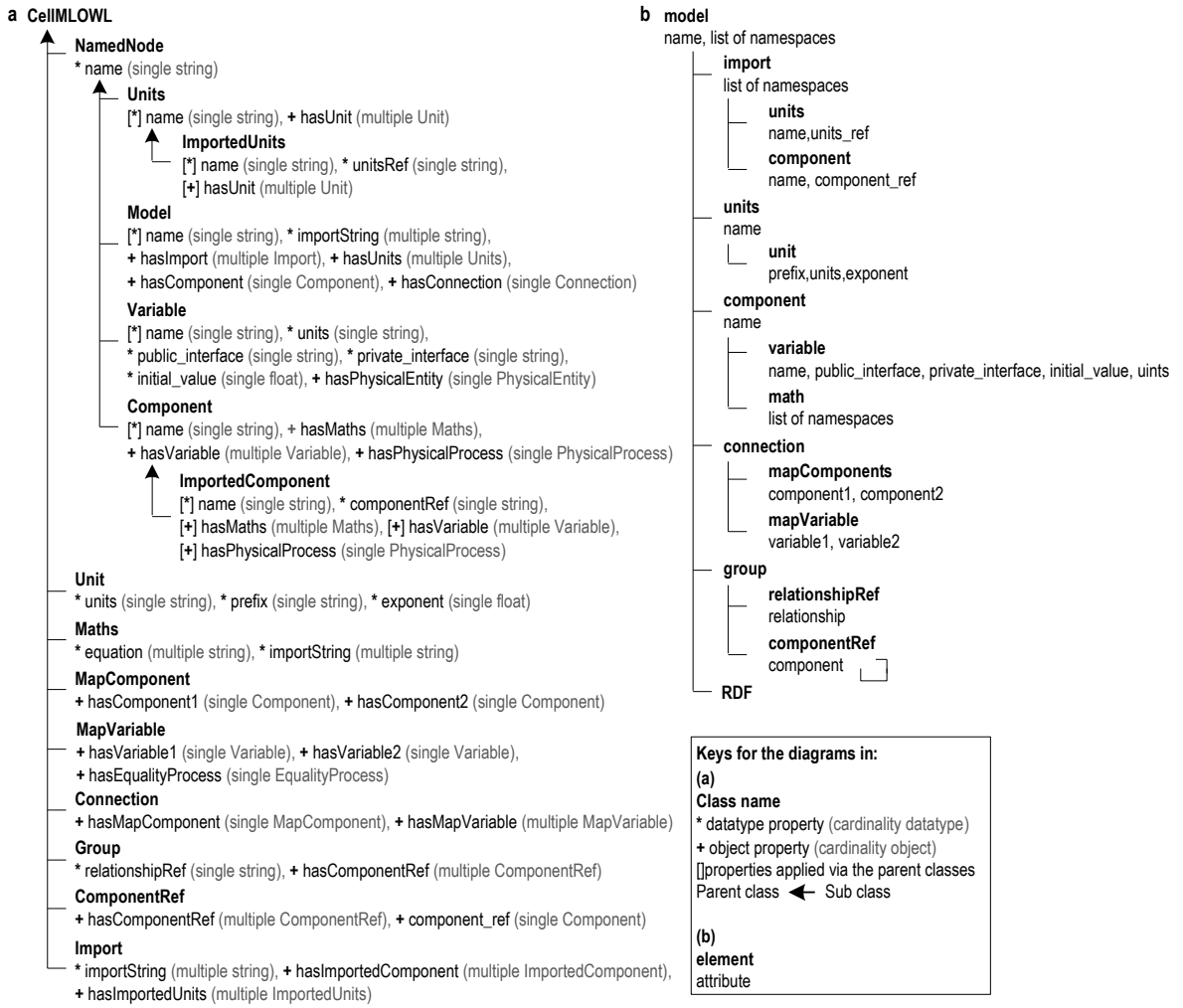


Figure 3.2: CellML structures

(a) The CellML/OWL ontology class structure. The CellML/OWL classes capture the CellML/XML elements while the CellML/OWL properties capture the CellML attributes and their relationships. (b) The CellML/XML hierarchical structure.

CellML/XML imports declare relations to other models and references components intended to be imported when the model is instantiated in a simulation/processing software [62]. If these declarations are mirrored in the CellML/OWL model, we would have a list of imported models and components. The imported components can be annotated using the Import elements, but it is impossible to annotate separate instances of variables defined within the imported components. As a result the CellML/OWL model represents the full instantiation such that all import references are resolved into explicit instances of components within the model. Using this approach it is possible to annotate the same component with different biophysical annotations at the top level. For example: GPCR model imports the integrator component multiple times, each of which is annotated with different biological

concepts. It is important to note that the implicit imports supported via encapsulated groups are not expressed as explicit instances in the CellML/OWL model.

The Import class in the CellML/OWL ontology should not be confused with owl:import statements in the OWL language. OWL refers to imports as importing concepts and relations from other resources. In CellML/OWL the import class represents the result of importing components in models referenced in the CellML/XML import declarations. The CellML/OWL model stores instances of imported components inside the current model allowing users to annotate variable metadata and imported components with respect to the importing model.

In order to express biophysical concepts that the CellML Component, Connection, and Variable elements represent, we define the property hasPhysicalProcess, hasMathematicalEquality, and hasPhysicalEntity, respectively in these classes. The process of defining values for these is explained in step 2.

The transformation of CellML/XML into CellML/OWL does not currently include metadata elements. Including these could provide some value and would make the transformation more complete

Instances of every OWL class require a unique identifier which is declared as an rdf:ID in the OWL/XML syntax. The CellML metadata specification defines a cmeta:id attribute to uniquely identify CellML elements [41]. The values of cmeta:id can thus be used as the rdf:ID in the CellML/OWL model instances. It is important to note that the imported components have cmeta:ids associated to them within the imported statement which is different to any existing cmeta:id that may exist on the component in the original model. As imported variable and math elements are not uniquely identified within a particular CellML model description, additional processing is required to create rdf:ID values for these elements. The formula for creating the rdf:ID values for imported variable and math elements is: rdf:ID of imported variable/math = {cmeta:id of imported component}+ "_"+{cmeta:id of variable/math defined in the imported model}.

CellML/OWL models are generated from CellML/XML models by traversing through the XML Document Object Model (DOM) and creating CellML/OWL instances. The generated CellML/OWL model instances are mapped to the original CellML model using the RDF. The RDF statement explicitly maps a CellML cmeta:id to a CellML/OWL rdf:ID. The RDF statement has:

- a subject: http://www.sarala.bioeng.auckland.ac.nz/example.cellml#A_A;
- a predicate:http://www.sarala.bioeng.auckland.ac.nz/cellmlowlbinding.rdf #cbinding;
- an object: http://www.sarala.bioeng.auckland.ac.nz/example.owl#A_A.

This maps CellML variable with cmeta:id A_A to CellML/OWL instance with rdf:ID A_A (Figure 3.3). The CellML/OWL model is stored in a separate OWL file and the RDF references are written back to the CellML/XML file.

a

```
<model…>
  <component…>
    <variable cmeta:id="A_A" name="A" private_interface="none" public_interface="out" units="micromolar" initial_value="1"/>
            …
  </component>
</model>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:cob ="http://www.sarala.bioeng.auckland.ac.nz/cellmlowlbinding/">
<rdf:Description rdf:about="#A_A">
   <cob:cbinding rdf:resource="http://www.sarala.bioeng.auckland.ac.nz/example.owl#A_A"/>
</rdf:Description>
…
</rdf:RDF>
```

b

```
…
<co:Variable rdf:ID="A_A">
   <co:hasPhysicalEntity rdf:resource="exampleBio.owl#A_A"/>
   <co:units>micromolar</co:units>
   <co:private_interface>none</co:private_interface>
   <co:public_interface>out</co:public_interface>
   <co:initial_value>1</co:initial_value>
   <co:name>A</co:name>
 </co:Variable>
…
```

Figure 3.3: Example of a metadata definition

(a) Fragment of a CellML/XML model highlighting the RDF mapping for variable A. (b) Fragment of the CellML/OWL model capturing the CellML/XML variable A description in OWL.

3.2.2 ANNOTATION OF A CELLML/OWL MODEL TO A CELLMLBIOPHYSICAL/OWL MODEL

Now that we have the model represented in CellML/OWL, the next step is to define the physical and biological information the elements of the model represent. To support this, CellMLBiophysical/OWL ontology was developed. It includes two ontologies, Physical and Biological. The Physical ontology captures the physical quantitative information and concepts captured in mathematical expressions. The Biological ontology captures the biological entities and processes. The CellMLBiophysical/OWL model defines the concepts and relations both within and between these ontologies (Figure 3.4).

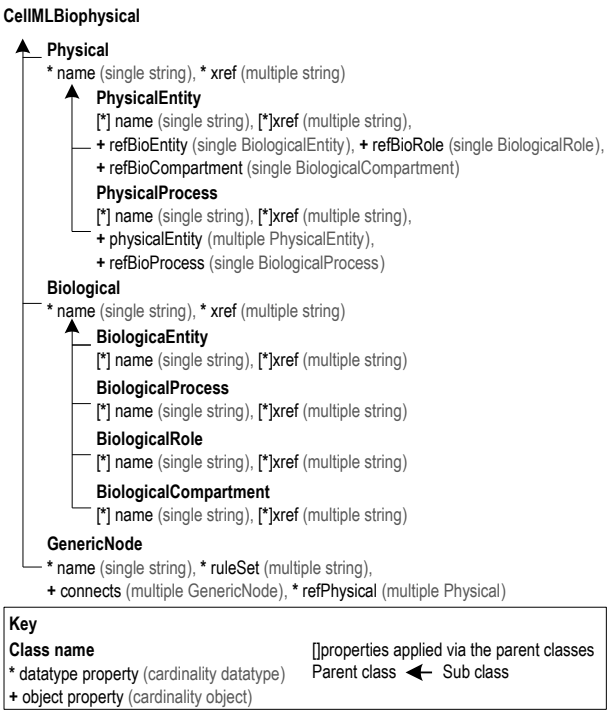


Figure 3.4: CellMLBiophysical/OWL ontology top-level class structure

The CellMLBiophysical/OWL ontology consists of the Physical, Biological, and GenericNode class structures and defines properties which link the instances of these classes.

The process of attributing physical and biological concepts to the CellML elements in CellML/OWL is a two step process of:

1. attributing physical meaning to the variables and components in the CellML/OWL model
2. attributing biological meaning (where relevant) to the physical attributes defined in 1.

3.2.2.1 Annotating physical information

Physical concepts are mapped in a one-to-one relationship between the CellML elements (components, variables, and connections) in the CellML/OWL model and the physical concepts defined in the Physical subclass tree. Here we describe the physical concepts modeled in CellML models explicitly by treating:

- every variable as a physical entity to capture the quantitative data defined in units of the variables;
- every component as a physical process to capture the type of mathematical formulations defined within components.

These concepts are represented by the PhysicalEntity and PhysicalProcess classes. Instances of these classes form the values of the hasPhysicalEntity and hasPhysicalProcess properties

within CellML/OWL models. In order to support the complete transformation of all the information in a CellML model, the connections are also treated as a process of equivalence (MathematicalEquality).

The Physical subclass tree also captures the relationships between variables, components, and connections. Every component defines a set of variables and every connection contains mappings between variables in two components. These are captured by the physicalEntity property defined in the PhysicalProcess class. This effectively creates a process-entity graph (Figure 3.5).

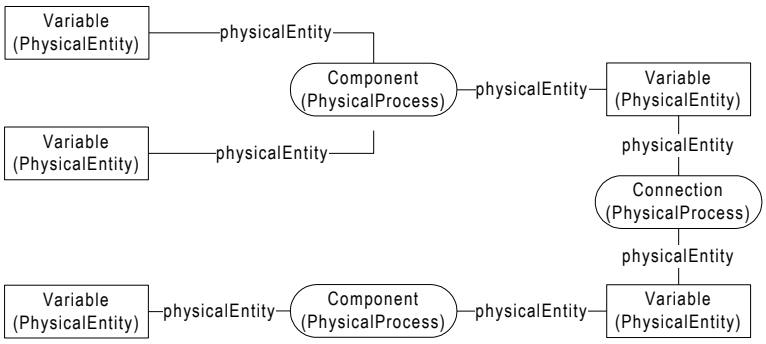


Figure 3.5: A physical-entity graph generated for a CellML model

A schematic diagram showing the relationship between CellML elements (components, variables and connections) and the underlying physical instances and relationships.

The subclasses of the PhysicalProcess class are intended to define physical processes commonly modeled in CellML. These include:

- MassActionKinetics - components describing mass action kinetics;
- EnzymeKinetics - components describing enzyme kinetics;
- Pooling - components with integrators;
- IonicCurrent - components calculating ionic currents;
- NernstPotential - components calculating Nernst potentials;
- PotentialDifference - components calculating potential differences;
- RateConstant: components calculating rate constants;
- ConversionFactor - components calculating conversion factors;
- Parameter - components providing parameters;
- MathematicalEquality - CellML connections.

The subclasses of the PhysicalEntity class define terms representing the physical dimensions of the variables in the CellML model. These include: Area, Capacitance,

Concentration, Conductance, Constant, Current, Dimensionless, Flux, Stoichiometry, Time, Voltage, and Volume.

A CellMLBiophysical/OWL model is programmatically generated, which consists of instances of PhysicalProcess and PhysicalEntity classes, and the relationship between them. These instances need to be specialized to the relevant subclasses manually using the classes supported by the Physical subclass tree. Model developers can use external tools, such as Protégé [63] or SWOOP [64], to complete this task.

3.2.2.2 Annotating biological information

The result of the previous step is a physical process entity graph that needs to be annotated with biological information. Biological concepts are mapped in a one-to-many relationship to instances of PhysicalEntity and PhysicalProcess, i.e. multiple physical entities can point to the same biological concept. The biological subclass tree of the CellMLBiophysical/OWL ontology has been developed to capture the biological concepts covered in the example models used in this work. This includes biological processes (BiologicalProcess) that occur between entities (BiologicalEntity), function of participation (BiologicalRole) of a biological entity in relation to a particular process, and a specific location (BiologicalCompartment) of the entity in a biological system.

The PhysicalProcess instances that have biological significance are annotated with BiologicalProcess instances via the refBioProcess property (Figure 3.6). The BiologicalProcess class is subdivided into:

- BiochemicalReaction - to capture reactions where one or more biological entities undergo covalent changes to form one or more different biological entities;
- Transport - to describe the transport of biological entities from one compartment to another compartment;
- ComplexAssembly - to capture the reactions where complexes are formed via non-covalent interactions.

If a particular PhysicalProcess instance is mapped to a biological process instance, then this instance must have at least one PhysicalEntity instance which, in turn, has mappings to both BiologicalEntity and BiologicalRole instances (Figure 3.6). This is important as the aim is to develop a biological view that represents the underlying relationships between biological processes and entities. It is also important to note that the biological view is “connected” by way of the connectivity of the physical model.

The PhysicalEntity instances that have biological significance are annotated with BiologicalEntity instances via the refBioEntity property. The BiologicalEntity class is subdivided into:

- Complex - to capture the biological entities bound together by non-covalent links;
- Protein - to describe the biological entities with a sequence of amino acids;
- SmallMolecule - to capture bioactive molecules that are not peptides;
- PhysicalFactor - which captures physical factors such as voltage.

In cases where these PhysicalEntity instances are mapped to PhysicalProcesses with a biological significance, they may also be annotated with a BiologicalRole instance (Figure 3.6). The BiologicalRole class is subdivided into:

- Modifier - which captures the modification types such as activator, catalyst and inhibitor;
- Reactant - which describes reactants;
- Product - which captures the reaction products.

These PhysicalEntity instances may also be annotated with a BiologicalComparment instance via the refBioComparment property to capture the location of the biological entity.

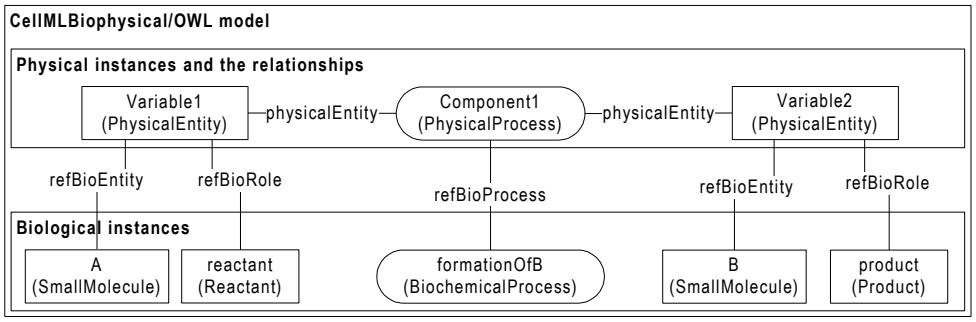


Figure 3.6: The mapping between Physical instances and Biological instances
The PhysicalEntity instance Variable1 is mapped to the BiologicalEntity instance A and the Role instance reactant. The PhysicalEntity instance Variable2 is mapped to the BiologicalEntity instance B and the Role instance product. PhysicalProcess instance component1 is mapped to the BiochecmicalProcess instance formationOfB.

The choice of physical and biological subclasses here serve only to reflect those necessary for the examples in this work and additional examples such as G protein-coupled receptor (GPCR) cycle described by Cooling *et al*. [50]. These examples can be downloaded at http://www.cellml.org/tools/downloads/cellml-viewer. In practice these classes would be

expanded, where the goal is to reflect all those concepts relevant to all models in the repository.

3.2.3 SIMPLIFICATION OF A CELLMLBIOPHYSICAL/OWL MODEL

The result of the previous step provides a complete representation of all physical entities, processes, and interpretations of these into biological entities, processes, and roles. For even simple CellML models the result can be a very large and complex looking biophysical representation. Here we describe a method to reduce this complexity to produce simplified views by implementing a graph reduction algorithm.

The aim is to collapse the process-entity graph by:

- simplifying the graph consolidating sub graphs or groups of nodes;
- ensuring there is no loss of information in this process.

We have identified four collapsing rules which depend on the relationships between entities and processes. The meaningfulness and validity of applying these to a particular instance is based on the ontological information.

The four collapsing rules and the cardinality constraints are:

1. entity(1*)-process(=1)-entity(1*) can be collapsed into an entity;
2. process(1*)-entity(=1)-process(1*) can be collapsed into a process;
3. terminal_entity(1*)-process(=1) can be collapsed into a process;
4. terminal_process(1*)-entity(=1) can be collapsed into an entity.

[(1*) - one-to-many, (1=) - exactly one, terminal entities refer to entities that are connected to only one process, and terminal processes refer to processes that are connected to only one entity]

These rules are then applied to a selected set of nodes in the process-entity graph on the basis of their ontological properties. We have identified a set of specific cases that are used in the example models:

1. a MathematicalEquality process (rule 1);
2. a processes which has entities mapped to the same biological term such as Pooling processes which capture integration of N fluxes of a particular biological entity to produce its concentration. (rule 1);
3. entities connected to processes which map to the same biological process (rule 2) because these processes together describe a particular biological term;

4. entities and processes which are not associated with biological meaning such as time (rule 3 & rule 4).

The application of the rules is selective and manual, allowing a user to highlight particular details and hide others. Applying these rules in the same order provides a consistent output. However, applying these rules in a different sequence can result in different outputs, implying that a particular CellMLBiophysical model can have multiple biological views.

Applying the reduction process to the annotated CellMLBiophysical/OWL model involves three steps:

1. creating a generic node graph with a reference to the original physical instances. This is illustrated by the dotted lines between the CellMLBiophysical/OWL model and generic model in Figure 3.7;
2. applying the rules recursively to collapse the generic node graph. The starting node of the graph is arbitrary but the node type depends on the rule. For example when applying rule 1, the starting node can be any generic node that references the MathematicalEquality (case 1);
3. each iteration creates a new generic node graph to store the output from each step.

When the generic nodes are collapsed, the references to the physical entity and process instances are accumulated in the new generic node. The generic node thus retains enough information to build a conceptual representation of what entities and processes (physical or biological) make up a node.

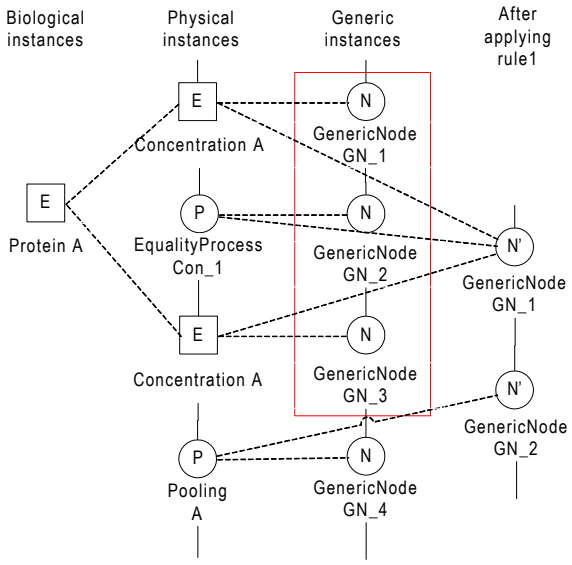


Figure 3.7: Application of the reducing rules

Applying rule 1 to collapse physical entity instances that are mapped to a common biological instance (case 2).

A GenericNode class has been introduced to the CellMLBiophysical/OWL ontology to store the generic node graph (Figure 3.4). It has the following set of properties: name: which refers to the name of the generic node; connects: which references connected generic nodes; and refPhysical: which refers to the accumulated physical nodes. Each of these physical nodes points to a particular biological concept. Each generic node graph is saved in a separate OWL file which imports the CellML/OWL model.

3.3 RESULTS

Here we illustrate the process for annotating CellML models and applying the reducing rules to generate the underlying biological views using a simple example. Below we focus on annotating and representing the flow of potassium ions (K_Ionic_Flow) modeled in the Hodgkin–Huxley model [52]. Then the annotated Hodgkin–Huxley model is illustrated.

3.3.1 REPRESENTING THE K_IONIC_FLOW MODEL IN CELLML

The model is first represented in CellML by structuring it in such a way that best describes the biophysical concepts and abstractions (Figure 3.8) [65] (Chapter 2). It is built using generic components, multiple levels of imports, and encapsulation. The top level K_Ionic_Flow model defines a set of components which describe:

- potassium Nernst potentials (K_Nernst_Potential);
- potassium currents through a potassium channel (K_Channel);
- membrane potential and the rate of change of voltage (Mem_Potential and V);
- constant values, initial values, and time (Constants, Initial_Values, and Time).

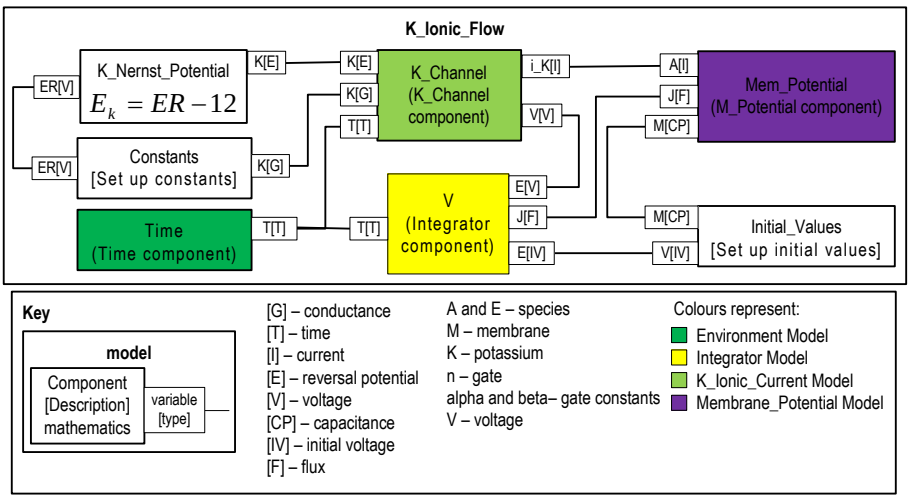


Figure 3.8: Modeling the potassium ionic current described in Hodgkin-Huxley model

The model reuses some of the generic components described in Figure 2.6 to calculate model specific values. The M_Potential generic component is imported to model the membrane potential. Mem_Potential component is connected to an imported Integrator component to calculate the rate of change of the voltage over time. The K_Ionic_Flow model imports the K_Channel component from the K_Ionic_Current model (Figure 2.7c). K_Nernst_Potential component calculate the reversal potential in terms of the membrane equilibrium potential (ER). The arcs are directionless intentionally, as they represent the quantities that are shared between components.

3.3.2 TRANSLATING THE K_IONIC_FLOW CELLML MODEL INTO A CELLML/OWL MODEL

The resulting CellML/OWL model defines an OWL instance for each CellML element described in the top level K_Ionic_Flow model. It also defines instances for the explicitly imported components, variables, and mathematics such as the K_Channel. It does not define instances for implicitly imported components, variables, mathematics, and connections such as the components grouped using encapsulation.

3.3.3 ANNOTATING THE K_IONIC_FLOW CELLML/OWL INSTANCES TO CELLMLBIOPHYSICAL/OWL INSTANCES

The annotated CellMLBiophysical/OWL model created for the K_Ionic_Flow model is shown in Figure 3.9a. Every CellML variable is represented as a specific PhysicalEntity instance. For example, variables associated with currents (i_K[I] and A[I]) are identified as a

Current instance. Similarly, every CellML component is represented as a specific PhysicalProcess instance. For example, the components describing the currents such as K_Current are identified as IonicCurrent instances. All the connections are represented as MathematicalEquality (Con[EP]) instances.

Figure 3.9a also shows the biological annotations for the K_Ionic_Flow model. The gate *n* is identified as a protein state. The potassium current is identified as a transport process. The Ek instance mapped to the K ionic current is also mapped to the K (BiologicalEntity), reactant (BiologicalRole), and IC (compartment) instances.

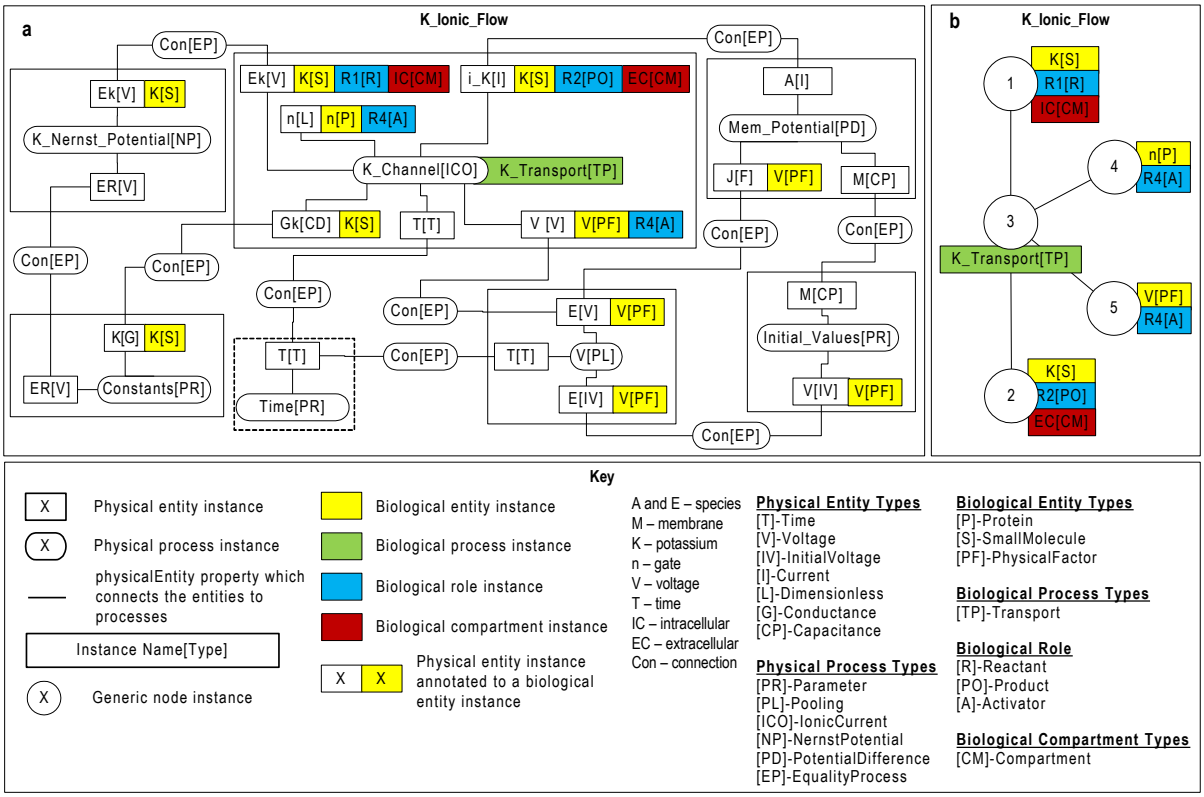


Figure 3.9: The K_Ionic_Flow model

(a) The CellMLBiophysical/OWL model generated for the K_Ionic_Flow model.

(b) The biological view showing the biological mappings that can be retrieved via the annotations.

3.3.4 SIMPLIFYING THE K_IONIC_FLOW CELLMLBIOPHYSICAL/OWL MODEL TO SHOW THE BIOLOGICAL VIEW

The CellMLBiophysical/OWL model is collapsed by applying the reducing rules. Figure 1b shows the resulting biological view that is generated for the K_Ionic_Flow example. Each node points to a biological annotation via the biophysical relationships. For example node 3

in the diagram references physical instances that are mapped to the K_Transport biological instance. This biological view essentially captures the underlying biological concepts described in the K_Ionic_Flow model. The order of specific cases that are used to collapse this model includes:

- collapsing all the MathematicalEquality processes (Con[EP]) (case 1);
- collapsing the Pooling process V[PL] which have entities mapped to the same biological term V (case 1);
- collapsing the entities T[T] and processes K_Nernst_Potential[NP] Constants[P], Mem_Potential[PD], Initial_Values[P], and Time[T] which have no biological meaning (case 4).

To clarify the annotation process further, a more detailed diagram which focuses on annotation of the K_Channel component is shown in Figure 3.10.

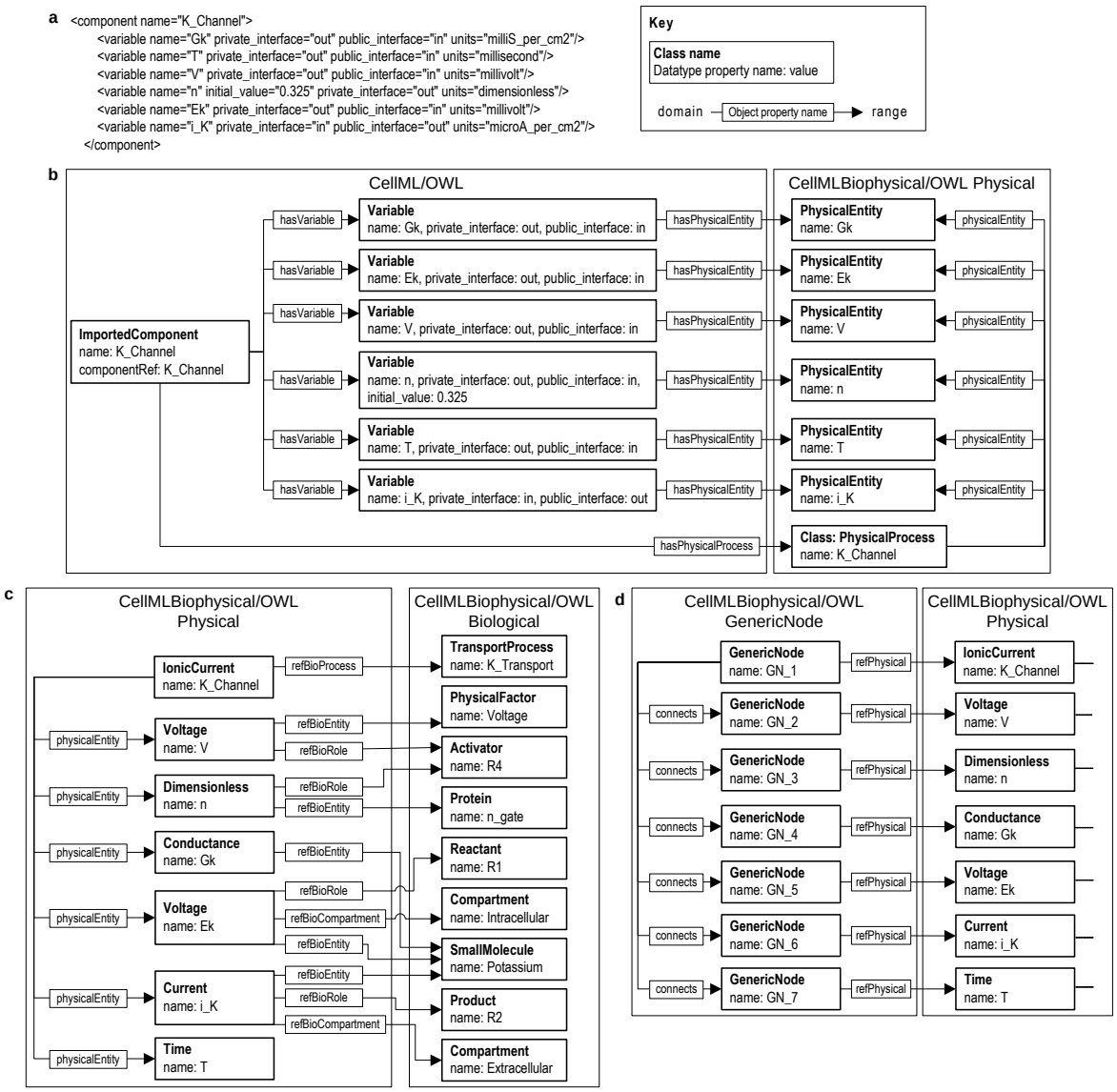


Figure 3.10: Annotating the imported K_Channel shown in Figure 3.9

(a) CellML code of the K_Channel component that is imported from the K_Ionic_Current model. The K_Channel component contains five variables Gk, Ek, I_k, h, V, and T to define potassium conductance, equilibrium potential of the potassium ion, potassium current, gate activity, membrane potential, and time, respectively. (b) A part of the CellML/OWL model generated for the imported component K_Channel. The ImportedComponent class is used to represent the K_Channel component. The Variable class is used to capture the variables Gk, Ek, I_k, h, V, and T. The CellML/OWL instances are annotated to CellMLBiophysical/OWL Physical instances. (c) Annotated CellMLBiophysical/OWL instances. The K_Channel is identified as a IonicCurrent. The Gk, Ek, I_k, h, V, and T physical instances are typed according to their units as Conductance, Voltage, Current, Dimensionless, Voltage, and time, respectively. The physical entities that can be related to biological concepts

are mapped to biological instances. For example Gk represents potassium conductance therefore it is mapped to the Potassium biological instance. (d) The starting set of GenericNode instances generated from the CellMLBiophysical/OWL instances. This is used to apply the reducing rules.

The process of annotating and representing the complete Hodgkin-Huxley model is illustrated in Figure 3.11. Every CellML variable is represented as a specific PhysicalEntity instance (Figure 3.11a). For example, variables associated with a current such as i_Na, i_K, and i_L are identified as Current instances. Similarly, every CellML component is represented as a specific PhysicalProcess instance. For example, the components describing the currents such as Leakage_Current are identified as IonicCurrent instances. All the connections are represented as MathematicalEquality instances. The gates *m*, *h*, and *n* are identified as protein states. The potassium current, sodium current, and the leakage current are identified as transport processes. The Ek instance mapped to the K ionic current is also mapped to the K (BiologicalEntity), reactant (BiologicalRole), and IC (compartment) instances (Figure 3.11a).

A biological view showing the biological mappings that can be retrieved via the annotations (Figure 3.11b). Node 1, 2, and 3 reference physical instances that are mapped to the Na_Transport, L_Transport, and K_Transport biological process instances, respectively. The rest of the nodes reference physical instances that are mapped to biological entity, role, and compartment instances. This biological view essentially captures the underlying biological concepts described in the Hodgkin–Huxley model. The order of specific cases that are used to collapse this model includes: collapsing all the MathematicalEquality processes (Con[EP]) (case 1); collapsing the Pooling process V[PL] which have entities mapped to the same biological term V (case 1); collapsing the entities T[T] and processes K_Nernst_Potential[NP], L_Nernst_Potential[NP], Na_Nernst_Potential[NP], Constants[P], Sum_of_2[PL], Sum_of_3[PL], Mem_Potential[PD], Initial_Values[P], and Time[T] which has no biological meaning (case 4).

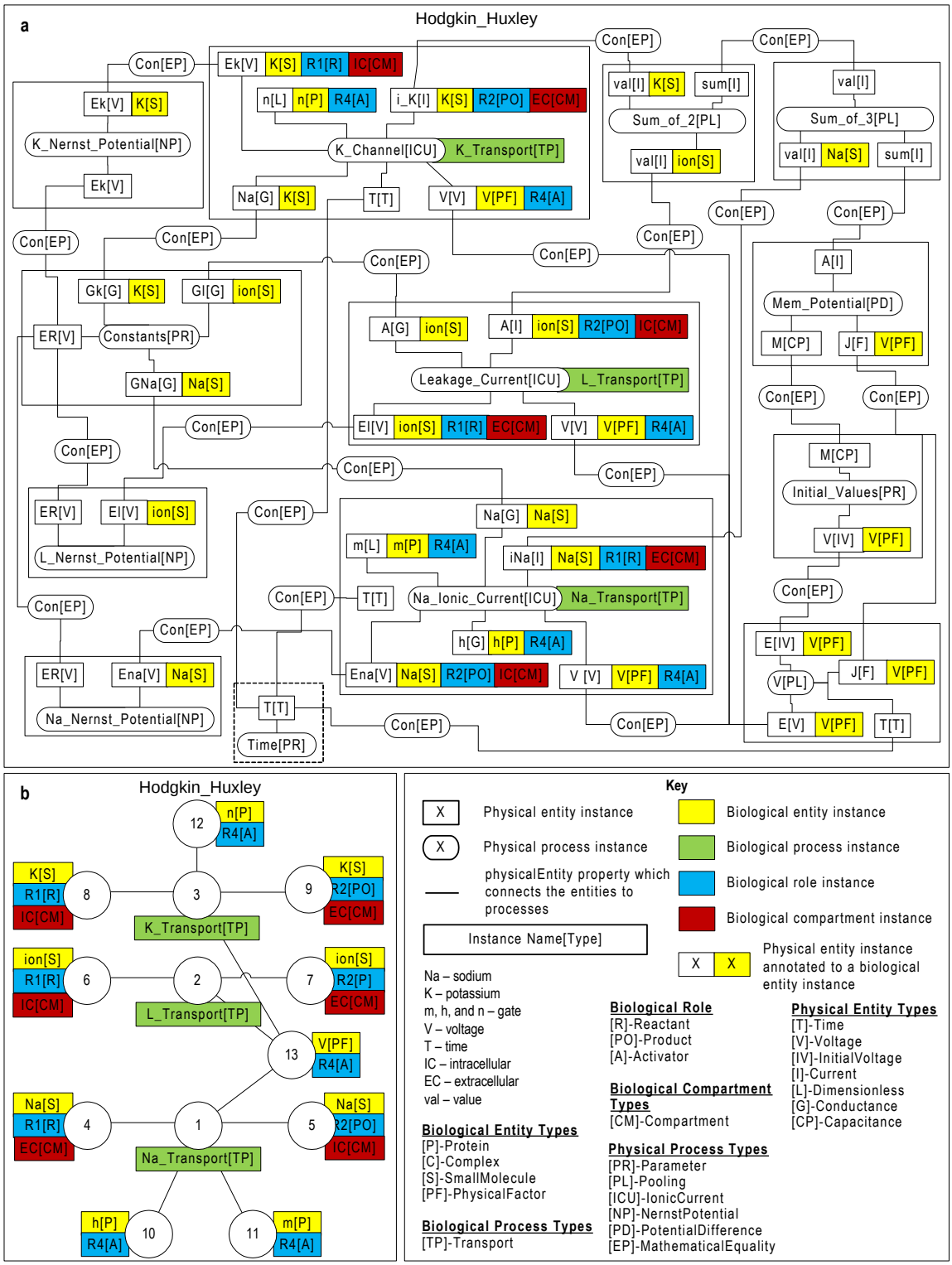


Figure 3.11: Annotated Hodgkin-Huxley model

(a) The CellMLBiophysical/OWL model generated for the Hodgkin–Huxley model. (b) The biological view showing the biological mappings that can be retrieved via the annotations.

3.4 DISCUSSION

Here we have presented a method for the incorporation of ontologies with CellML that can be used to annotate the biophysics of a model, and generate ontological representations, to highlight the underlying biological view.

Our ontological framework provides a fully integrated system, allowing modelers to traverse between models. A CellML/OWL model provides the necessary layer for integrating a CellML model with a CellMLBiophysical/OWL model which explicitly captures the physical and biological concepts. The CellMLBiophysical/OWL model is then collapsed to generate a biological view which is much easier to interpret. The biological view contains links to the collapsed biophysical instances which have one-to-one relationships with the CellML/OWL component, variable, and connection instances. This allows modelers to trace back to the original CellML elements from a particular biological view.

The CellMLBiophysical/OWL model explicitly identifies the physical concepts in the CellML model. This makes it easier for readers to identify the computational modeling constructs used in the model. For example, the K_channel component is typed as an IonicCurrent identifying that the component is responsible for calculating a current generated by an ion flow. The K_channel component does not contain any mathematics itself, but instead it encapsulates a set of components that are involved in calculating the current flow through the potassium channel.

Even though the K_Ionic_Flow CellMLBiophysical/OWL model (Figure 3.9a) captures the biological concepts it is difficult for readers to interpret the underlying processes. The graph reduction rules reduce the complexity of the annotated K_Ionic_Flow model by collapsing together the physical instances that point to identical biological concepts. The resulting biological view (Figure 3.9b) shows a simplified representation of the underlying biological concepts by highlighting the transport processes, gated channels, and the relationships between them.

The ontology based rules for applying the generic rules that we have identified can be used to apply the reduction rules to collapse most of the signal transduction pathway and electrophysiological models. These can fail when applied to other types of models described in the CellML model repository. However, more specific cases can be identified by applying it to different types of models. Due to the wide range of models in the CellML model

repository, it may not be possible to come up with a generic set of cases to collapse all models but a set of specific cases depending on the model category.

Note that the rules we have identified are used to simplify the process-entity graph using the biological annotations in this work. The advantage of this rule set is that these can be applied to simplify the process-entity graph using a different context by identifying specific cases. As research continues, it is one our goals to annotate the CellMLBiophysical/OWL models with mathematical constructs which can be used to reduce the complexity of a CellML model.

The biological and physical annotations depend on the modeler's interpretation of the model. The reducing rules depend on the biophysical annotations. Having different annotations may result in different biological views. This allows modelers to change the annotations to emphasize or hide details. Modelers can also choose the order of graph reduction rules applied. By changing the order of rules, it is possible to generate different biological views for some models. A particular CellML model may thus have multiple biological views.

The topology of the biological view also depends on the topology of the CellML model. The CellML model structure has a direct effect on the topology and complexity of the CellMLBiophysical/OWL model. If a CellML model is incomplete or invalid, the generated CellMLBiophysical/OWL model might be difficult to annotate with biological concepts, resulting in an uninteresting collapsed model. When building the topology of the CellML model, the modeler thus needs to reflect the biophysical abstractions that she is trying to communicate [65]. Note that this process also helps modelers to construct modular CellML models by clearly separating out the underlying biophysical concepts.

The K_Ionic_Flow model example shown here demonstrates the way in which the CellML models can be collapsed to represent the underlying biological concepts without any loss of information. The CellML models are structured to clearly separate the biophysical concepts. This is reflected in the resulting CellMLBiophysical/OWL model, as highlighted in the dotted areas (Figure 3.9a). Note that the CellMLBiophysical/OWL model does not capture implicitly imported encapsulated concepts such as the details of calculations of the gated channel. However, it does identify the explicitly imported variables such as *n*-gate, which is typed as a Dimensionless instance in the physical domain and Protein in the biological domain.

The CellMLBiophysical/OWL and BioPAX both can be used to develop biological views but the way in which the views are represented is different. BioPAX represents a biological process as having properties directly specifying participants. In contrast, in CellMLBiophysical/OWL the biological participants of a process are reached indirectly through the physical entity instances. Even though this makes querying of biological data more complex, it supports querying of physical concepts captured in CellML models.

The CellMLBiophysical/OWL ontology does not provide a complete ontology for annotating all the CellML models in the repository. It supports most of the biophysical concepts covered in signal transduction pathway and electrophysiological models. However, it can also be extended to support additional biophysical concepts and relationships by introducing new classes, properties, and restrictions. The CellMLBiophysical/OWL models can also be annotated against external ontologies and vocabularies to take advantage of existing knowledge definitions. Every class within the CellMLBiophysical/OWL ontology has an xref property to enable the model developer to map the class to an external resource such as terms in GO or SBO. For example annotating a BiologicalEntity instance against the voltage-gated sodium channel complex in the cellular component ontology in GO provides information about the BiologicalEntity instance, such as it is a protein complex and part of the plasma membrane. SBO can be used to annotate the physical entity and process instances with quantitative parameter concepts and mathematical expression concepts, respectively, to provide physical significance. The biological entity and process instances can be annotated with SBO participant concepts and event concepts, respectively, to provide additional biological meaning.

One of the advantages of the CellML import element is the ability to reuse previously defined models in new model descriptions. As mentioned in the section 3.2.1 OWL:Import cannot be used to achieve the same semantic results. It is feasible to achieve the same modular approach in OWL without using a domain specific CellML import concept by introducing meta-classes/meta-models which could then be used to build OWL models. However, this was not addressed in this work as the intension of our work was to reflect the CellML/XML in OWL to provide a method to attribute properties to the CellML elements.

The workflow discussed in this chapter involves programmatic and manual manipulation of the models. A software package has been developed to programmatically generate the ontological representations of a CellML model. The package is developed in Java to run on any platform. Currently it has only been tested on Widows but as research continues this tool

will be tested in other environments. This tool is freely available at http://www.cellml.org/tools/downloads/cellml-viewer. The manual process includes the construction of the CellML model and its subsequent annotation. The generated CellMLBiophysical/OWL model needs to be manually annotated with specific physical and biological concepts using external OWL editors such as Protégé [63].

The error-prone and time-consuming manual annotation process can be effectively simplified by reusing annotated CellML models. It is our intension to replace these manual processes with formal or automated methods in order to support the effective programmatic construction of biological views. Errors that may occur during the processes of annotation of biological concepts can be reduced by introducing a rigor peer-reviewed workflow similar to that of publication in journals. The annotation of physical concepts can be improved by implementing a method for variable typing, and unit and process translation by using an ontological representation of mathematical equations. The construction of CellML models can be further improved by introducing a library of annotated reusable components and models [65]. Such a library could then be used to automatically link components and integrate models to create more complex annotated models. The generic node model which holds bindings through to all relevant components could be used to generate or formulate a function of all relevant pieces of specific annotation.

The outcome of this research offers an extensible ontological framework which allows modelers to:

- build CellML models by abstracting biophysical concepts;
- annotate the CellML models with physical and biological concepts;
- reduce the complexity of the underlying biophysical model by generating a simplified biological view;
- integrate CellML models with external controlled vocabularies and other modeling standards.

4 A METHOD FOR VISUALIZING CELLML MODELS

CellML models can be very complicated, making it difficult to interpret the underlying physical and biological concepts and relationships captured/described in the mathematical model. The previous chapter described a set of ontologies that were developed to explicitly annotate the biophysical concepts represented in the CellML models. This chapter presents a framework that combines a visual language, together with CellML ontologies, to support the visualization of the underlying physical and biological concepts described by the mathematical model and also their relationships with the CellML model. Automated CellML model visualization assists in the interpretation of model concepts and facilitates model communication and exchange between different communities.

4.1 INTRODUCTION

The creation and simulation of CellML models are supported by a set of software tools, including OpenCell and Cellular Open Resource (COR) [62]. The main features supported by these tools include text-based editing, tree views, graphical views, running simulations, and graphing of simulation results of CellML models [62, 66, 67]. Modelers are encouraged to share their models with the community by adding them to the CellML Model Repository [17]. The Physiome Model Repository (PMR) [22] is a software product designed to facilitate

the upload, storage, curation, download, and viewing of the models in the CellML model repository.

CellML models comprise networks of interconnected components. The language supports a mechanism for importing components from one model into another. This feature facilitates the building of complex, composite models of function, it encapsulates hierarchies of detailed mechanisms (therefore hiding much of the model complexity). Imports can be used to easily build upon existing models (without the need to start from scratch), or they can be utilized in libraries of pre-defined functional components.

However, the result of such language features and flexibility is that the specific details of biological and physical concepts can end up distributed over a complex interconnected network. This makes it difficult to identify what elements of a model contribute to a concept. It also makes it harder for a modeler to maintain a clear conceptual picture of the underlying biological mechanisms described by the model, especially when that model is large, complex and/or composite.

Existing software tools have attempted to address some of these issues. Each model entry in the PMR has an option to view a rendered list of the mathematical equations described in that particular model. OpenCell, a tool for editing and simulating CellML models, provides a tree view of the CellML model, allowing the user to more easily view the units, components, variables, connections and mathematical equations in a particular model. Further, physical quantitative information about the model can be derived via the defined units of the variables.

To highlight the biological concepts described in CellML models, almost all the model entries in the PMR include a schematic diagram of the underlying biological concepts being described by that particular CellML model. These same schematic diagrams can be included in OpenCell session files as Scalable Vector Graphics (SVG) [39] with annotations to the CellML variables and components. These clickable diagrams allow the user to trace the different behaviors of each variable in the model over the time course of the simulation.

Currently these schematic diagrams are created manually, to reflect the diagrams often found in the published papers from which the CellML model has been derived. However, diagram creation is time-consuming and can be an error-prone process. Further, the *ad hoc* nature of these diagrams often requires textual support, usually in the form of a figure legend that describes the underlying biological concepts being illustrated. We suggest that such diagram generation would benefit from a visual language that could be used to provide

consistent graphical notations. A visual language is a way of expressing the semantics of a specific domain in a multi-dimensional space using visual symbolic notations [68]. A good visual language should be easily recognizable and simple to learn.

The embedded SVG diagrams in OpenCell are manually annotated to their corresponding variables in the CellML model. This is achieved by mapping each SVG glyph to a single CellML variable via the scripting language Javascript. Javascript enables programmatic access to objects in OpenCell and allows modelers to interact with these SVG diagrams within the OpenCell graphical user interface. Modelers can then select glyphs in the diagram to watch the simulation graph of the mapped variable.

There are several problems associated with this method of building schematic diagrams and the binding of variables from models to diagrams:

- the schematic diagrams do not capture the composite nature that can exist in the underlying CellML models.
- there is no existing method to build new visual representations of models that are composites of existing models. The glyphs in the diagram can be annotated to variables in the composite model as well as imported components;
- the manual creation of diagrams can be an error-prone process, especially for the larger, more complex models. In particular, there are no methods in place to ensure that all the biological elements described in the CellML models are represented in the diagram;
- this manual process is labor-intensive and time consuming.

The aim of the work described here is to find a visual language suitable for representing the underlying biophysical concepts captured in a CellML model. We also define a formal method to consistently apply this visual language to diagram generation.

Many alternative visual languages have been developed to visualize biological models. These include, BioD [69], Molecular Interaction Maps (MIM) [70], the state transition diagram notation supported in CellDesigner [71], BioUML [72], Pathway Analysis Tool for Integration and Knowledge Acquisition (PATIKA) [73, 74], Cell Illustrator [75], PathSys [76], BioPath [77], Edinburgh Pathway Notation (EPN) [78], and insilicoIDE [79]. A matrix developed to compare the notations supported in CellDesigner, PATIKA, MIM, and EPN is illustrated in Supplementary Material C. Despite their considerable number, identifying a suitable visual language for a particular type of model can present a challenge, as they are often domain specific and can have different underlying schemas of how biological meaning

should be modeled and displayed. To address these issues, there has been a consortium approach to developing a common visual notation: the Systems Biology Graphical Notation (SBGN) [80]. This aims to draw on the experiences of existing visual languages to formulate a single, standard graphical notation of biology that is vital for the interpretation and sharing of biological knowledge between different research communities.

The SBGN community has identified three main types of visual representations:

1. process diagrams which represent a sequence of interactions between biochemical entities;
2. entity-relationship diagrams which depict interactions which occur if the relevant entities are present;
3. activity flow diagrams which represent the influences between entities.

The models in the PMR encompass a wide range of molecular processes. The SBGN process diagram can represent all the molecular processes and interactions that occur between biological entities, and their outcomes. This particular type of visual representation provides a promising option for visualizing CellML models. However, since SBGN is currently in a state of rapid evolution, we have also been developing our own visual language in parallel to this effort. That said, we are contributing to the discussions and development of SBGN and we have found guidance for the development of our own visual language through this collaboration.

The generation of diagrams for CellML models using a visual language requires the language to be mapped to the CellML elements. In turn, this mapping process requires the CellML language to explicitly identify the underlying biophysical concepts being modeled. To achieve this, a biophysical ontology was developed to explicitly capture the physical and biological concepts in a CellML model [81]. The goal here is to develop a method to construct and map a visual language to the biophysical ontology which in turn can be used to visualize the physical and biological concepts described in CellML models.

In this chapter we address these issues by describing:

- a visual language for representing the physical and biological concepts modeled in CellML;
- a method to map the visual language to the concepts defined in the biophysical ontology;
- an algorithm that combines the visual language with the ontologies in order to facilitate the automated generation of visual representations of the CellML models.

In the following section we describe the ontologies and strategy developed to map physical and biological concepts in a CellML model to a visual representation of the model, and illustrate the approach with a simple example.

4.2 METHODS

The previous chapter described an ontology to annotate and represent CellML models, which we will use as the basis for developing our approach. The biophysical ontology, which is referred to as CellMLBiophysical/OWL [81], is based on the Web Ontology Language (OWL) [35]. It captures the underlying physical and biological concepts and relationships described in a CellML model. The CellMLBiophysical/OWL ontology is made up of two distinct ontologies; the Physical and the Biological. The Physical ontology captures the physical quantitative information and mathematical expressions in a CellML model. It creates a process-entity graph where the process represents a CellML component or a connection while the entity represents a variable (Figure 4.1). The Biological ontology captures the biological processes and entities described in a CellML model. The CellMLBiophysical/OWL ontology defines the concepts and relationships both within and between these ontologies.

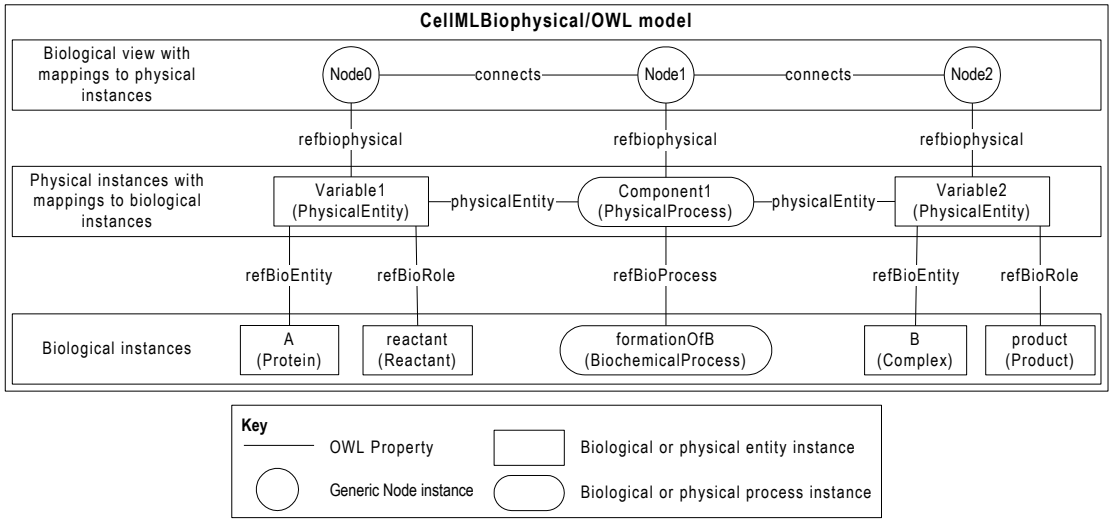


Figure 4.1: Schematic diagram illustrating a CellMLBiophysical/OWL model
The figure illustrates an annotated CellMLBiophysical/OWL model generated for a CellML model with one component and two variables. In the physical domain, components (Component1) is typed as a Process and variables (Variable1 and Variable2) are typed as PhysicalEntities. These physical entities are annotated to Biological instances via properties (Variable1 PhysicalEntity instance is mapped to Protein A biological instance via the refBioEntity property). GenericNode

instances are generated by reducing the physical-entity graph according to the biological annotations. These GenericNode instances maps to the Physical instances which, in turn, have mappings to Biological instances.

An instantiation of this ontology, which we refer to as a CellMLBiophysical/OWL model capturing the biophysical concepts modeled in a particular CellML model, results in a relatively complicated representation. A simplified view of these representations is generated by applying a graph reducing algorithm to the CellMLBiophysical/OWL model. This simplified view is a node-edge graph with mappings to physical instances which, in turn, have mappings to biological instances (Figure 4.1). This simplified view, together with biological mappings, can be used to highlight the underlying biological concepts and relationships described by the CellML model [81] (Chapter 2).

The framework for generating visual representations for CellML models involves four steps:

1. the development of a standardized visual language for representing the physical and biological processes captured in CellML models. This involves designing a set of visual glyphs which have one-to-one mappings with the physical and biological concepts represented in the CellMLBiophysical/OWL models;
2. the representation of the visual language in a computer readable form;
3. the mapping of the visual notation to the CellMLBiophysical/OWL ontology. This involves developing an ontology (VisualTemplate/OWL) that maps to the visual language which, in turn, is mapped to concepts in the CellMLBiophysical/OWL ontology;
4. the development of an algorithm for generating visualizations of CellML models based on these ontological mappings.

4.2.1 THE DEVELOPMENT OF A STANDARDIZED VISUAL LANGUAGE FOR REPRESENTING THE PHYSICAL AND BIOLOGICAL PROCESSES CAPTURED IN THE CELLML MODELS

The goal here is to develop visual elements to represent the physical and biological concepts captured in the CellMLBiophysical/OWL ontology.

4.2.1.1 A visual language for representing physical concepts

A visualization should ensure that the detail of the underlying physical information described by the model is correctly and fully represented. A CellMLBiophysical/OWL model instance

captures the relationships between all the CellML elements (components, variables, and connections) as a process-entity graph. A detailed visualization can be constructed by identifying each of these processes and entities with a one-to-one mapping, i.e. every process, entity, and connection can be identified by a unique glyph.

There are often many physical process and entity concepts associated with a CellMLBiophysical/OWL model that need to be visualized [81] (Chapter 2). The visualization should also uniquely identify these physical processes and entities. One solution would be to provide a separate glyph for each type of process and entity. However, this would not necessarily be very helpful to the modeler as it requires them to remember the meaning of many different symbols. We have thus chosen to use text labels, in addition to a reasonably small, distinct set of glyphs, to support the visualization of CellMLBiophysical/OWL models (Figure 4.2). The type of the entity or process, which is captured in CellMLBiophysical Class name, is written in italics, whilst the name of the entity or process is written in standard text.

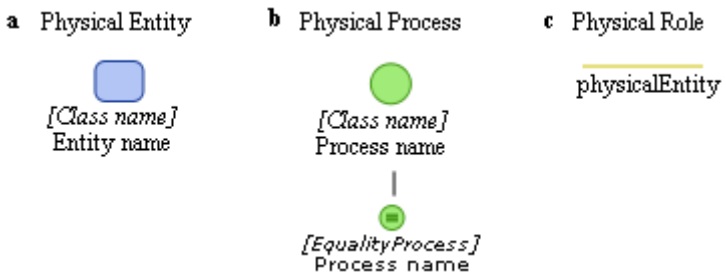


Figure 4.2: A notation for visualizing physical concepts

(a) A glyph for visualizing physical entities. (b) Glyph set for visualizing physical processes. A separate (EqualityProcess) is used to represent the MathematicalEquality class instances in the CellMLBiophysical/OWL ontology which intern represents the CellML connections. (c) A glyph for visualizing the relationship between the physical entities and processes.

The rules for constructing a visual representation of a CellML model are based on the underlying process-entity graph. There is a unique glyph for each PhysicalEntity, PhysicalProcess, and the relationship between the entities (physicalEntity).

4.2.1.2 A visual language for representing the biological concepts

Here we aim to develop the visual language elements required for representing the biological processes (BiologicalProcess), entities (BiologicalEntity), and the function of participation (BiologicalRole) of a biological entity in relation to a particular process.

A set of glyphs has been developed to uniquely identify:

- biological entities such as small molecules, proteins, complexes, physical factors, and null elements (Figure 4.3a);
- biological processes such non-enzyme catalyzed covalent reaction (BiochemicalProcess), non-covalent reaction (ComplexProcess), enzyme catalyzed reaction (EnzymeProcess), movement of biological entities from one location to another (TransportProcess), and the process of degrading a physical entity (DegratationProcess) (Figure 4.3b);
- biological roles such as catalysis, activation, inhibition, reactant, and product (Figure 4.3c).

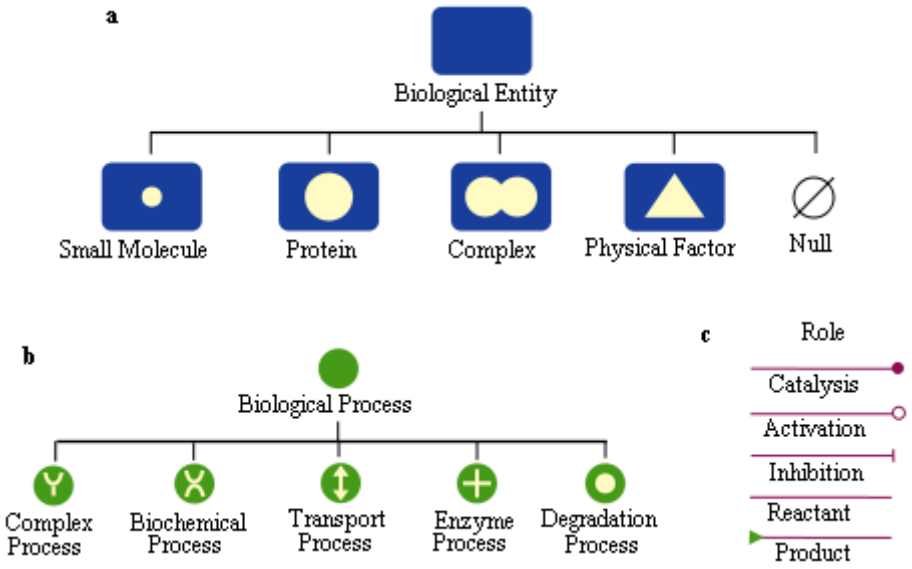


Figure 4.3: A notation for representing biological concepts

(a) Glyph set for visualizing biological entities. (b) Glyph set for visualizing biological processes. (c) Glyph set for visualizing biological roles.

The rules for constructing a biological visual representation are based on the simplified view which captures the underlying biological concepts in a CellML model. A biological visual representation is generated by forming a one-to-one mapping between the biological concepts and the visual elements, or glyphs. i.e. each GenericNode is mapped to either a BiologicalProcess glyph or a BiologicalEntity glyph; whilst the Role glyph is used to represent the relationship between a biological entity and a biological process.

4.2.2 THE REPRESENTATION OF THE VISUAL LANGUAGE IN COMPUTER READABLE FORM

To support the computer based manipulation of a visual language, it is important to find a standard format for representing the visual language in a computer readable form. To be suited for the visualization of CellML models, such a representation would have to support two dimensional graphics for representing the visual language, dynamic image updates to automatically construct visual representations from the CellML models, in addition to an interactive environment to track the user's mouse activity on the graphics.

SVG is an XML-based graphics standard for representing two-dimensional graphics. SVG supports vector graphics shapes, images, and text. Graphical objects can be grouped, styled, transformed, and composited into previously rendered objects. The feature set includes nested transformations, clipping paths, alpha masks, filter effects, and template objects. Use of XML enhances the searchability and the accessibility of the SVG graphics, and also allows the use of Java to generate SVG code to render graphics from database content to enable dynamic image updates. Interactive SVG content can also be executed using mouse events, and it allows users to initiate hyperlinks to traverse to new web pages and layout the objects. These features make SVG a well suited visual representation format for our particular needs.

SVG consists of large number of elements. The svg element is the root of a SVG document. For the work described here we use the basic types of drawing elements such as text (text), shapes (rect, line, and cir), and paths (path). Each of these elements can be associated with attributes to define properties such as dimensions (width and height), unique identifier (id), coordinates (x and y), transformation (transform), and style (style). The g element is used to group the drawing elements together. It can also be associated with attributes such as id (id), coordinates (x and y), and transformation (transform).

Each visual element shown in Figure 4.2 and Figure 4.3 has been coded up in SVG, allowing these glyphs to be reused when generating the visual representation of a particular CellML model. The remainder of this chapter refers to these glyphs as templates. Three SVG files are used to store the entity, process, and role visual elements.

The template for representing a generic entity is illustrated in Figure 4.4a. The rect element is used to draw the base rectangle of a glyph. It specifies the size of the area covered by the graphic object. The handler element specifies the points that are used to connect a

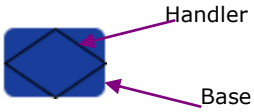
glyph to other glyphs such as role glyphs. Note that in Figure 4.3a each visual element has a specific shape to identify whether it represents a small molecule, protein, complex, or a physical factor. These shapes inside the base rectangle are represented using the cir element for drawing circles and the path element for drawing different shapes. The g element is used to group these elements together to form a specific template. A text element is added to display the type and name of a particular entity. The id attribute of the g element contains a unique identifier to recognize each glyph in a model and the transformation attribute specifies the position of the glyph in a particular coordinate system.

The template for representing a generic process is illustrated in Figure 4.4b. Similar to the entity template, the process template also defines a base and a handler. The base for the process template is a circle which is described using cir element. The shapes inside the base circle which identify each of the different processes are represented by path elements.

The template for representing a generic role is illustrated in Figure 4.4c. Every role type (Figure 4.3c) consists of a line and another shape drawn at one end of each line. The line element is used to draw the line of each glyph. The additional shapes associated with each of the visual elements can be drawn using the line, cir, or path SVG elements.

a

View of the code below

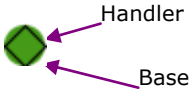


All elements specified in the be code below

```
<g id="genericEntity" name=" genericEntity " transform="translate(0.0,100.0) scale(1.0,1.0)">
   <rect id="base" x="0" y="0"  rx="5" ry="5" width="50" height="34" style="fill:#183d9a;stroke:none;stroke-width:1.0000000" />
   (SVG elements describing the shapes inside the base)
   <path id="handler" d="M 25,0 L 50,17 L 25,34 L 0,17 Z " style="fill:white;fill-opacity:0;stroke-width:1;stroke:black" display="none"/>
</g>
```

b

View of the code below



All elements specified in the code below

```
<g id="genericProcess" transform="translate(30.0,0.0)" >
      <circle id = "base" r="10" cx="10" cy="10" style="fill:#459a18;stroke-width:1;stroke:none"/>
  (SVG elements describing the shapes inside the base)
   <path id="handler" d="M 10,0 L 20,10 L 10,20 L 0,10 Z " style="fill:white;fill-opacity:0;stroke-width:1;stroke:black" display="none" />
</g>
```

c


```
<g id="genericRole" name=" genericRole">
   <line id="line" x1="100px" y1="100px" x2="150px" y2="150px" stroke="#8B0A50" stroke-width="1"/>
      (SVG elements describing the additional shapes attached to the line)
</g>
```

Figure 4.4: Template codes describing the visual language

(a) SVG graphic and the template code for representing the generic entity template (b) SVG graphic and the template code for defining the generic process (c) SVG graphic and the template code to define the generic role

4.2.3 THE MAPPING OF THE VISUAL LANGUAGE TO THE CELLMLBIOPHYSICAL/OWL ONTOLOGY

The visual language represents the concepts captured in CellMLBiophysical/OWL ontology which, in turn, captures the biophysical concepts modeled in CellML. The goal here is to map the visual elements to the biophysical concepts described in the CellMLBiophysical/OWL ontology. This can be achieved by developing an intermediate OWL ontology (VisualTemplate/OWL) with mappings to the SVG visual elements. This provides two loosely coupled ontologies which are then free to evolve independently. Currently the VisualTemplate/OWL class structure mirrors the top level CellMLBiophysical/OWL class structure, thus making mapping between the two ontologies a relatively straightforward process.

The VisualTemplate/OWL ontology class structure is illustrated in Figure 4.5. The top level VisualTemplate class has two properties: a url to store the URLs of the SVG files that contain the glyphs; and an svgId to store the value of the id attribute of the g element. It has two subclasses; a Biological class and a Physical class. These two classes are used to represent the biological glyphs and the physical glyphs, respectively. The Biological class has three subclasses:

1. a BiologicalEntity class to reference to biological entity glyphs (Figure 4.3a);
2. a BiologicalProcess class to reference to biological processes glyphs (Figure 4.3b);
3. a BiologicalRole class to reference to biological role glyphs (Figure 4.3c).

Similarly the Physical class can also be further divided into three subclasses:

1. a PhysicalEntity class to reference to physical entity glyphs (Figure 4.2a);
2. a PhysicalProcess class to reference to physical process glyphs (Figure 4.2b);
3. a PhysicalRole class to reference to physical role glyphs (Figure 4.2c).

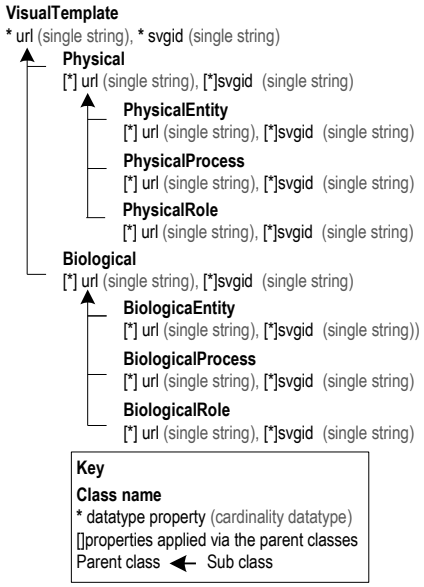


Figure 4.5: Class structure of the VisualTemplate/OWL ontology

An instance in a CellMLBiophysical/OWL model needs to reference an instance of a VisualTemplate/OWL model. This mapping is achieved by introducing a property called visualTemplate to the top class of the CellMLBiophysical/OWL ontology, CellMLBiophysical. The set of rules for model annotation includes mapping each CellMLBiophysical/OWL:

- BiologicalEntity instance to a VisualTemplate/OWL BiologicalEntity instance;
- BiologicalProcess instance to a VisualTemplate/OWL BiologicalProcess instance;
- BiologicalRole instance to a VisualTemplate/OWL BiologicalRole instance;
- PhysicalEntity instance to a VisualTemplate/OWL PhysicalEntity instance and a PhysicalRole instance;
- PhysicalProcess instance to a VisualTemplate/OWL PhysicalProcess instance.

Note that the CellMLBiophysical/OWL PhysicalEntity instance is mapped to both a VisualTemplate/OWL PhysicalEntity instance and a PhysicalRole instance. The relationship between a CellMLBiophysi-cal/OWL PhysicalEntity and a PhysicalProcess is captured via the physicalEntity property. OWL properties describe relationships between OWL class instances. Here we want to express the relationship between the physicalEntity property and the PhysicalRole class instance which cannot be expressed in OWL.

In summary, there is a many-to-one relationship between CellMLBio-physical/OWL and VisualTemplate/OWL instances. i.e. many instances of a CellMLBiophysical/OWL (Physical/Biological) can be mapped to a single VisualTemplate/OWL (Physical/Biological) instance. These mappings between each CellMLBiophysical/OWL class instance and each

VisualTemplate/OWL class instance allow us to obtain defined references to the SVG visual element templates shown in Figure 4.2 and Figure 4.3.

4.2.4 THE DEVELOPMENT OF AN ALGORITHM FOR GENERATING VISUALIZATIONS OF CELLML MODELS

Here we describe the process of generating a visualization once a CellMLBiophysical/OWL model has been mapped to the VisualTemplate/OWL instances. The following section describes the algorithm for generating the physical and biological visual representations of a CellML model.

4.2.4.1 Generating visualizations of the physical concepts in a CellML model

There is a one-to-one mapping between a CellMLBiophysical/OWL model, representing the physical concepts in a CellML model, and the visual elements in a schematic diagram. Every PhysicalProcess and PhysicalEntity is represented using the PhysicalProcess and PhysicalEntity glyph, respectively. The relationship between the PhysicalProcess and PhysicalEntity captured via the physicalEntity property is captured using the PhysicalRole glyph. This one-to-one association should not be confused with the explicit many-to-one mapping that exists in the CellMLBiophysical/OWL and the VisualTemplate/OWL discussed in the previous section.

The algorithm for visualizing physical concepts is thus relatively simple:

1. create a SVG document to describe the visualization;
2. for each CellMLBiophysical/OWL (PhysicalEntity and PhysicalProcess) instance retrieve the mapped VisualTemplate/OWL (PhysicalEntity and PhysicalProcess) instance (Figure 4.6). Then retrieve the mapped SVG template, copy it to the created SVG document and set the label (type and name);
3. for each CellMLBiophysical/OWL PhysicalEntity instance retrieve a VisualTemplate/OWL PhysicalRole instance (Figure 4.6). Then retrieve the mapped SVG template, copy it to the created SVG file and set the coordinates of the lines so that they form the connections between the underlying CellMLBiophysical/OWL PhysicalEntity and PhysicalProcess;
4. generate the coordinates for the PhysicalEntity and PhysicalProcess glyphs. A simple algorithm has been written to control the layout of the process and entity glyphs, and this ensures they do not overlap.

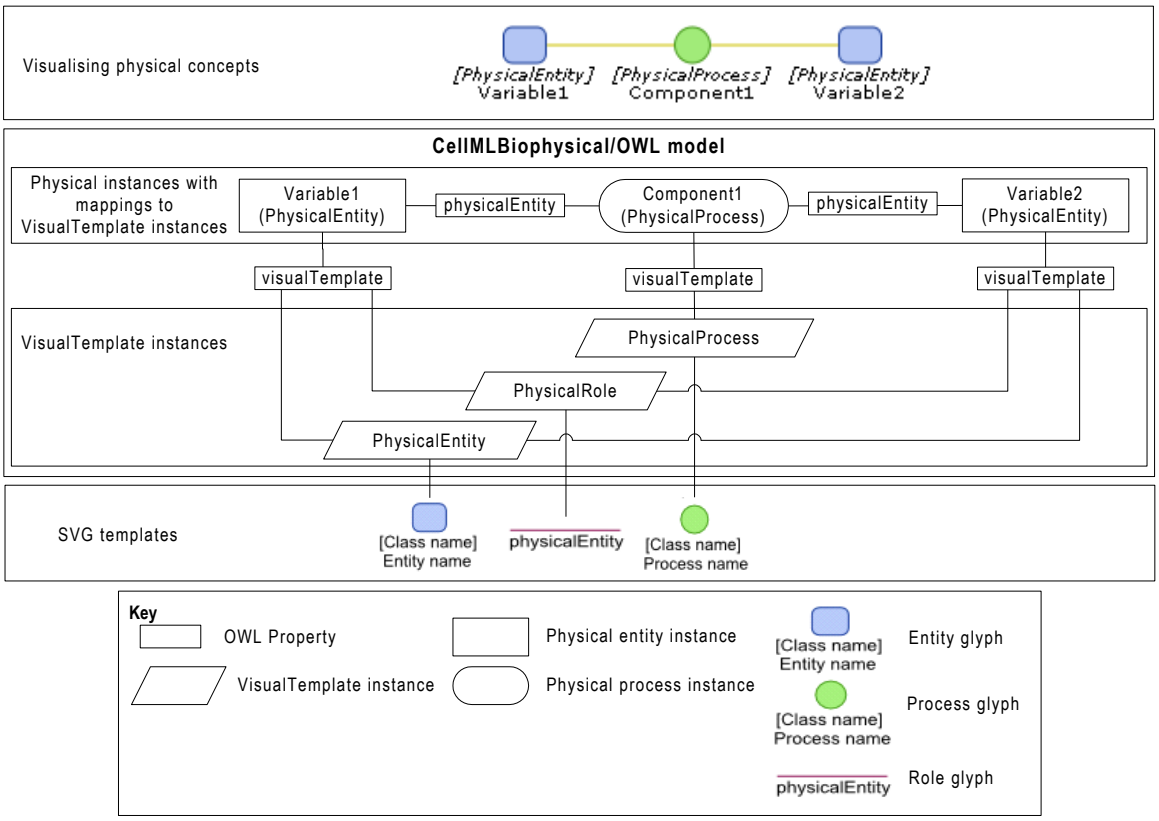


Figure 4.6: Mappings for generating the physical view

Component1 PhysicalProcess instance is mapped to PhysicalProcess VisualTemplate/OWL instance which in turn references the PhysicalProcess glyph. Variable1 and Variable2 PhysicalEntity instances are mapped to PhysicalEntity VisualTemplate/OWL instance which in turn references the PhysicalEntity glyph. These also map to PhysicalRole VisualTemplate/OWL instance which in turn maps to the Role glyph. These mappings are then used to generate a physical view.

4.2.4.2 Generating visualizations of the biological concepts in a CellML model

There is a one-to-one mapping between a collapsed, simplified view of a CellML model and its biological visualization. Retrieving biological visual elements for the simplified view of the model requires traversing from physical mappings to biological mappings to visual template mappings. Figure 4.7 illustrates these mappings. The algorithm for visualizing the biological concepts in a CellML model has five steps:

1. create a SVG file to describe the visualization;
2. for each GenericNode retrieve the common biological concept mapping via the CellMLBiophysical/OWL (PhysicalEntity and PhysicalProcess) instances;

3. if the mapped biological concept is a CellMLBiophysical/OWL (BiologicalProcess or BiologicalEntity) instance, then retrieve the mapped SVG template via the VisualTemplate/OWL (BiologicalProcess or BiologicalEntity) instance, copy it to the created SVG file and set the label;
4. for each mapped CellMLBiophysical/OWL BiologicalRole instance, retrieve the mapped SVG template, copy it to the created SVG file and define the coordinates of the lines such that they form the connections between the underlying GenericNodes;
5. generate the coordinates for the BiologicalEntity and BiologicalProcess glyphs such that they do not overlap.

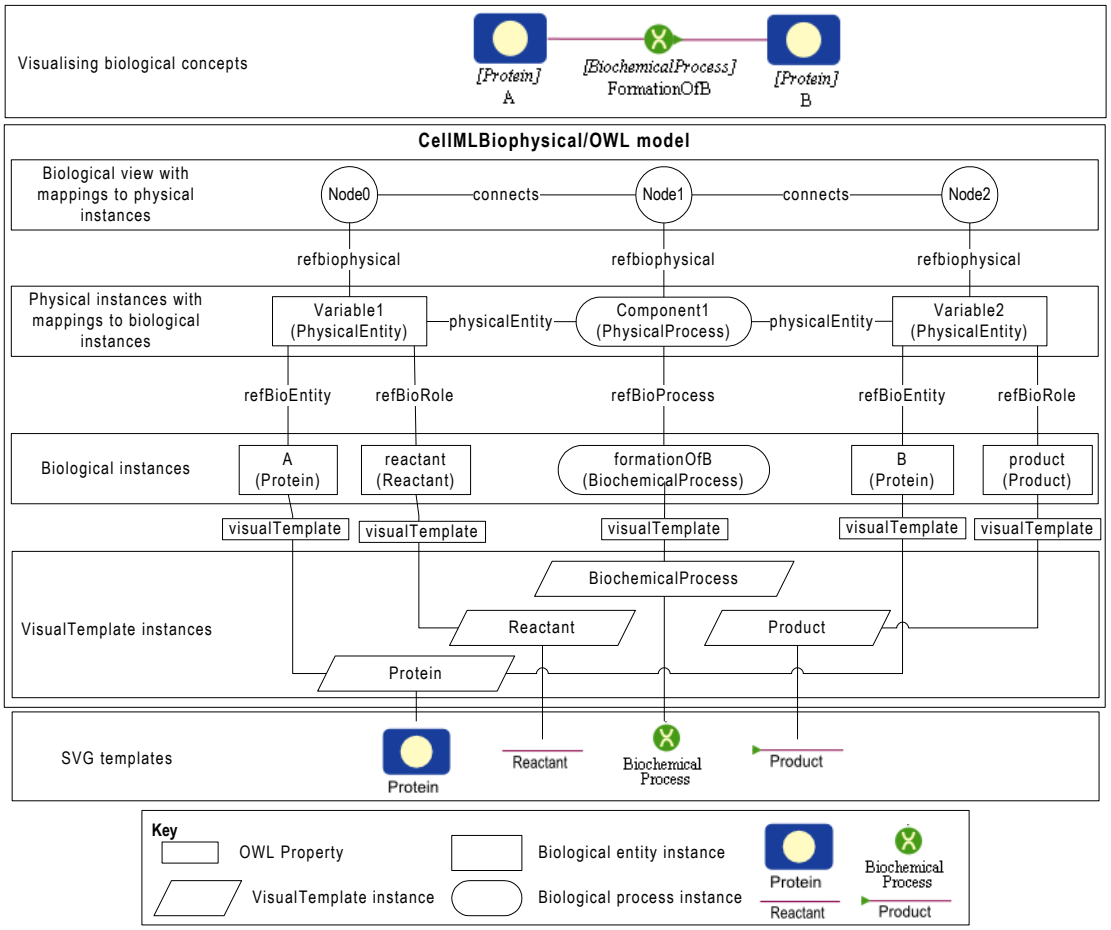


Figure 4.7: Mappings for generating the biological view

The GenericNode edge graph is generated by reducing the physical-entity graph according to the Biological annotations to generate the biological view. These GenericNode instances map to the Physical instances which in turn maps to Biological instances. These Biological instances are mapped to VisualTemplate/OWL instances which in turn references SVG glyphs. For example: Variable1 PhysicalEntity instance is mapped to Protein A Bio-logical entity which maps to Protein VisualTemplate/OWL instance. The Protein

VisualTemplate/OWL instance is mapped to the Protein SVG glyph. These mappings are then used to generate a visualization of the biological model

4.3 RESULTS

Here we use a simple example to illustrate the application of this methodology to create a visual representation. Below we focus on representing a part of the Hodgkin–Huxley model; the flow of potassium ions (K_Ionic_Flow). Then the complete visualization of the Hodgkin–Huxley model is illustrated.

First, the CellMLBiophysical/OWL model is annotated with physical and biological concepts [81]. Figure 3.9a shows the K_Ionic_Flow CellMLBiophysical/OWL model which identifies each physical entity and process type and the relationships described in the model. This diagram essentially shows the CellML model structure with annotated physical information. The PhysicalEntity instances depict CellML variables. For example, the Current instance potassium (i_K) is represented as a variable in the CellML. Similarly, the PhysicalProcess instances depict CellML components. For example, the Pooling instance V is represented as a component in CellML. MathematicalEquality instances show the CellML connections. Figure 3.8 illustrates the K_Ionic_Flow CellML model.

The model's biological annotations are also illustrated in Figure 3.9a. The potassium channel gate n is identified as a protein state. The K_Channel instance is identified as transport process. The Ek instance mapped to the K ionic current is also mapped to the K (BiologicalEntity), reactant (BiologicalRole), and IC (compartment) instances.

The annotated CellMLBiophysical/OWL model is then used to generate the visual representation of the physical concepts by applying the algorithm described in the above method section. Figure 4.8a shows the diagram which is generated when the visual language is used to represent the physical concepts captured in the K_Ionic_Flow model shown in Figure 3.9a. The PhysicalProcess instances K_Nernst_Potential, K_Channel, Mem_Potential, V, Constants, Initial_Values, and Time in the model are represented using the PhysicalProcess glyph. The CellMLBiophysical/OWL MathematicalEquality instances (Con) are represented using the EqualityProcess glyph, while all the PhysicalEntity instances are represented using the PhysicalEntity glyph.

The generic node-edge graph with the biological annotations that can be retrieved from the ontological mappings is shown in Figure 3.9b. Node 3 references physical instances that

are mapped to the K_Transport biological process instance. The rest of the nodes reference physical instances that are mapped to biological entity, role, and compartment instances. This biological view captures the underlying biological concepts described in the K_Ionic_Flow model.

Figure 4.8b shows the diagram which is generated when the visual language is used to represent the biological concepts captured in the K_Ionic_Flow model. Each GenericNode node instance is represented using a glyph.

The relationships between the nodes in Figure 3.9b and Figure 4.8b are listed in Table 4.1.

GenericNode instances	Glyph type
3	TransportProcess glyph
1 and 2	SmallMolecule glyph
4	Protein glyph
5	PhysicalFactor glyph

Table 4.1: Relationship between the GenericNode instances and the glyph types used in the K_Ionic_Flow model

The relationships between the GenericNode nodes are represented using Role glyphs (namely; reactant, product or activator). The relationships between the connections presented in Figure 3.9b and Figure 4.8b are listed in Table 4.2.

Connections	Glyph type
3 – 4 and 3 – 5	Activator glyph
3 – 1	Reactant glyph
3 – 2	Product glyph

Table 4.2: Relationship between the connections and the glyph types used in the K_Ionic_Flow model

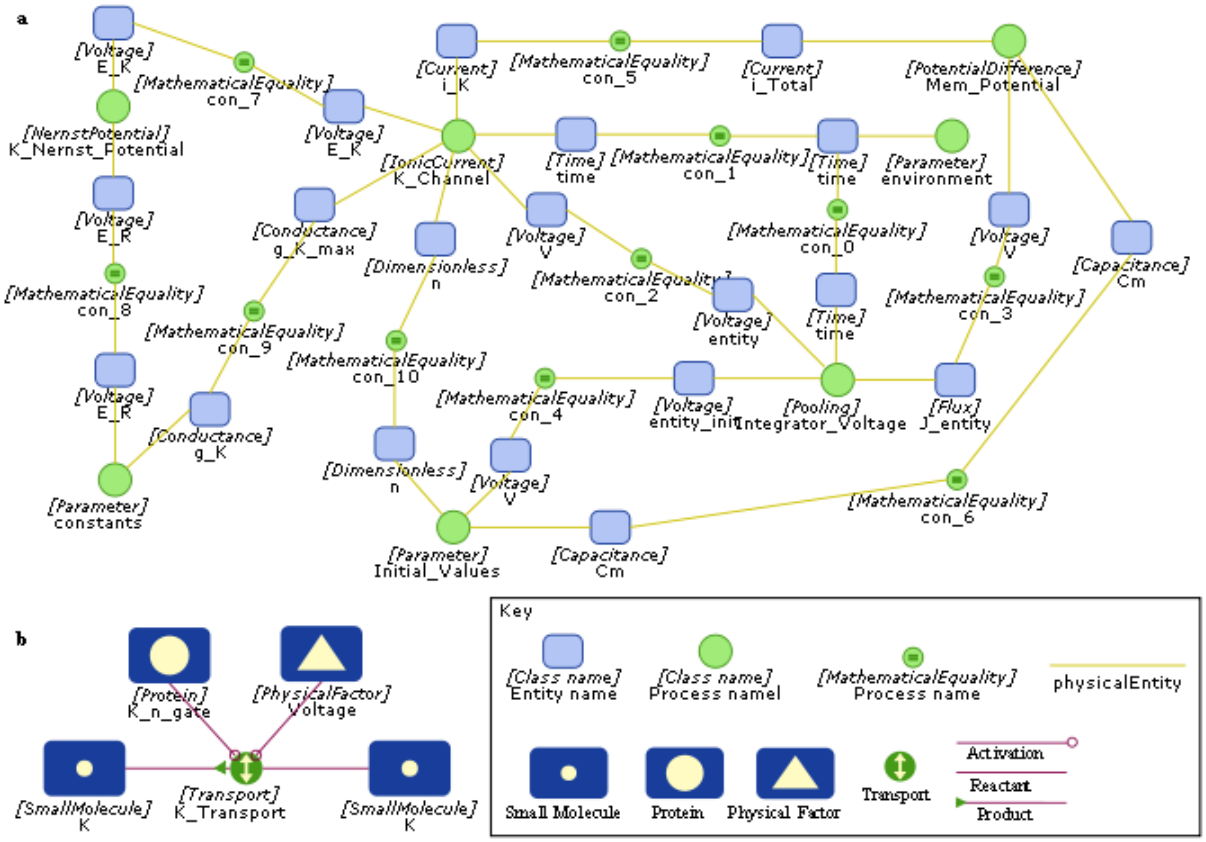


Figure 4.8: Visualizing the K_Ionic_Flow model

(a) Physical view generated for the K_Ionic_Flow model. (b) Visual representation depicting the biological view.

The visualizations generated for the complete Hodgkin–Huxley model is illustrated in Figure 4.9 and Figure 4.10. The annotated Hodgkin–Huxley CellMLBiophysical/OWL model (Figure 3.11a) is used to generate the visual representation of the physical concepts. The PhysicalProcess instances Na_Nernst_Potential, K_Nernst_Potential, L_Nernst_Potential, Na_Channel, K_Channel, Leakage_Current, Sum_of_3, Mem_Potential, V, Constants, Initial_Values, and Time in the Hodgkin–Huxley CellMLBiophysical/OWL model are represented using the PhysicalProcess glyph. The CellMLBiophysical/OWL MathematicalEquality instances (Con) are represented using the EqualityProcess glyph, while all the PhysicalEntity instances are represented using the PhysicalEntity glyph.

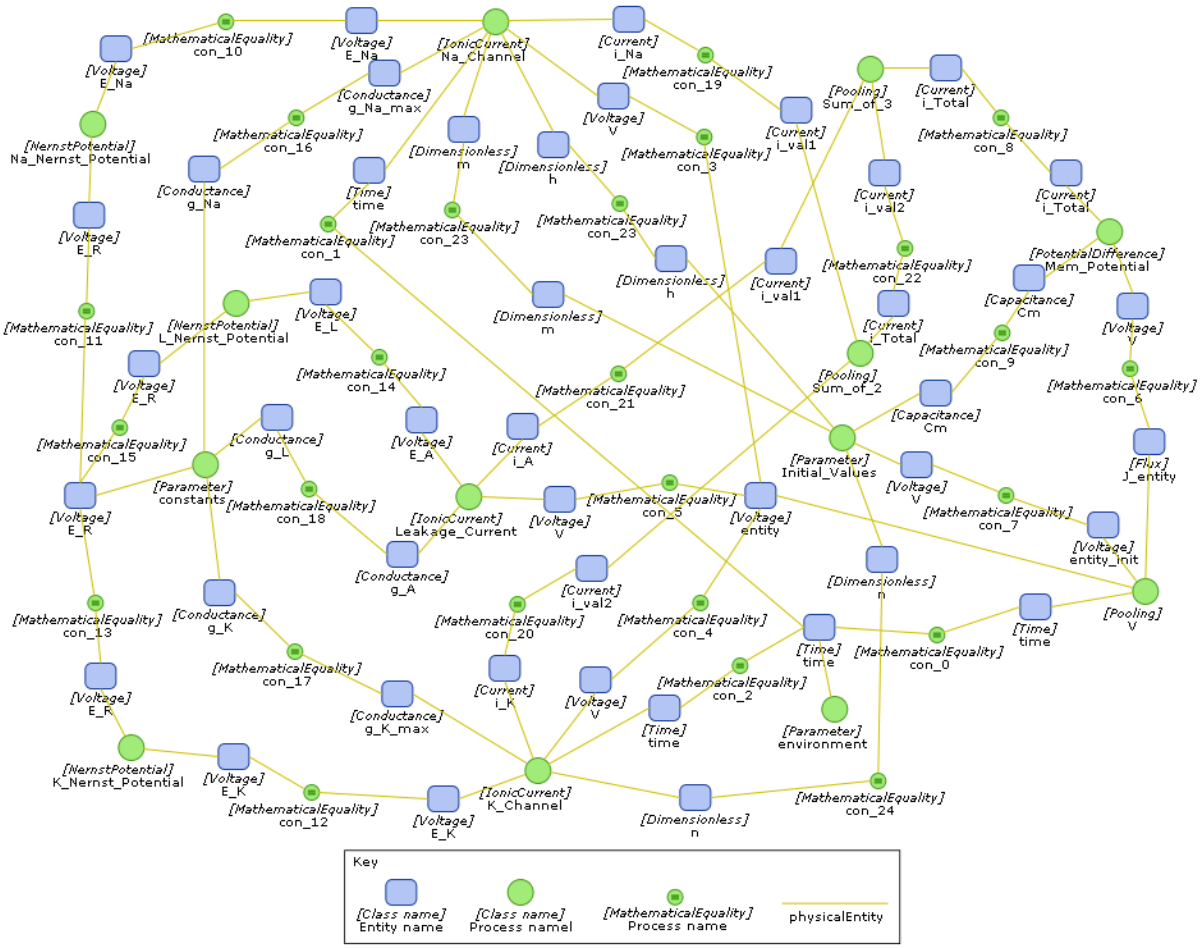


Figure 4.9: Physical view generated for the Hodgkin–Huxley model

Figure 4.10 shows the diagram which is generated when the visual language is used to represent the biological concepts captured in the Hodgkin-Huxley model.

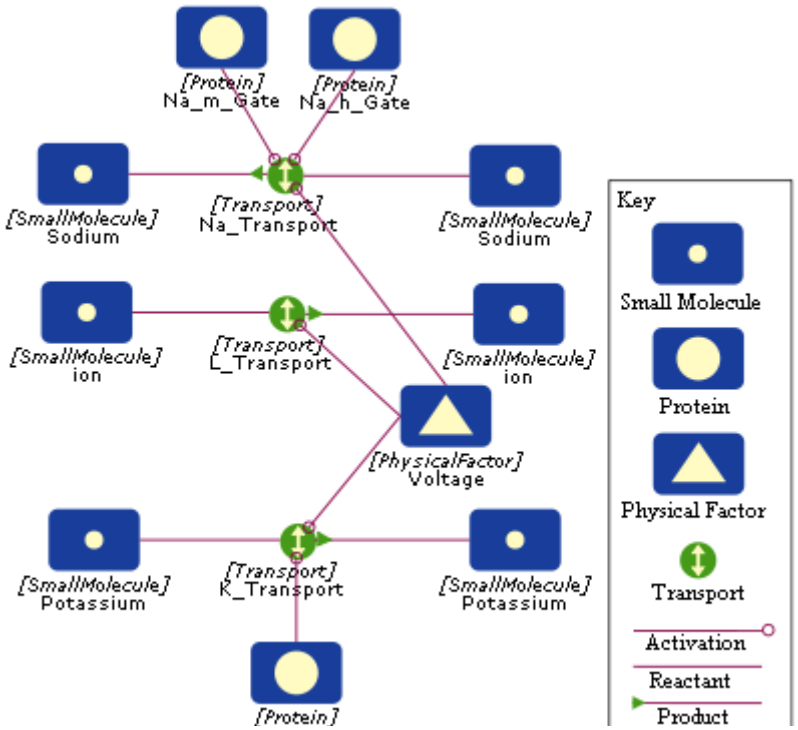


Figure 4.10: Biological view generated for the Hodgkin-Huxley model

The relationships between the nodes in Figure 3.11b and Figure 4.10 are listed in Table 4.3.

GenericNode instances	Glyph type
1, 2, and 3	TransportProcess glyph
4, 5, 6, 7, 8, and 9	SmallMolecule glyph
10, 11, and 12	Protein glyph
13	PhysicalFactor glyph

Table 4.3: Relationship between the GenericNode instances and the glyph types used in the Hodgkin-Huxley model

The relationships between the connections presented in Figure 3.11b and Figure 4.10 are listed in Table 4.4.

Connections	Glyph type
10 – 1, 11 – 1, 12– 3, 13 – 1, 13 – 2, and 13 – 3	Activator glyph
4 – 1, 6 – 2, and 8 – 3	Reactant glyph
5 – 1, 7 – 2, and 9 – 3	Product glyph

Table 4.4: Relationship between the connections and the glyph types used in the Hodgkin-Huxley model

4.4 DISCUSSION

The method described above facilitates the process of automatically generating visual representations of CellML models, allowing modelers to more easily see all the underlying physical and biological concepts captured in the mathematical description.

Two different notation sets were developed to visualize the underlying physical and biological concepts modeled in the CellML models. The visual representation of the physical concepts has a one-to-one mapping with the underlying CellML model. This visualization is very detailed and has a limited set of nodes with textual descriptions, making it easier to see the physical concepts captured by the model. In contrast, the visualization of the biological concepts has a many-to-one mapping with the underlying CellML model. This visualization is relatively simple, using a small set of unique glyphs which results in the creation of schematic diagrams in which the underlying biology can easily be interpreted.

The final appearances of both the physical and biological visualizations depend heavily on how the CellML model has been annotated. Different coding styles also generate different visualizations. Any mistakes created at the annotation level will flow through to the final visual representations. Only a properly annotated CellML model will accurately generate the correct physical and biological visualizations.

The flexible structure of CellML allows modelers to construct mathematical models of the same biological system in many different ways. The different CellML model structures also generate different visualizations since the annotations, and the application of the reduction rules to generate biological views of models, depend on the structure of the model. When structuring the CellML model, the modeler thus needs to reflect the biophysical abstractions that he or she is trying to communicate [65]. Unforeseen outcome is an indication of an unstructured model thus this process also helps modelers to construct modular CellML models.

Interpreting the underlying physical relationships and the biological views of a CellML model from a CellMLBiophysical/OWL representation can be a challenging task. The diagrammatic representations generated using the method described in this chapter clearly show the underlying physical and biological concepts captured in the CellML model. For the Hodgkin-Huxley model example, Figure 4.9 identifies and labels the physical processes and entities. Similarly, Figure 4.10 identifies the sodium, potassium, and leakage ion transport

reactions and the gated channels. These diagrams are much easier to interpret than having to read through either the raw XML of the CellML model or its ontological representation.

InsilicoIDE is a tool which is capable of visualizing, editing, and creating CellML models [79]. The visualizations generated by insilicoIDE focus on representing the components, variables, and connections in a model. InsilioIDE supports a collapsing feature which can be used to simplify visualizations of complex CellML models. The models are simplified according to their encapsulation hierarchy in contrast to using higher level biological information. InsilicoIDE currently does not focus on visually representing the biophysical concepts modeled in CellML models.

The development of modularized CellML models allows modelers to construct larger models using smaller models, resulting in compact top-level models. Even for a small modularized CellML model such as the Hodgkin-Huxley model, the physical view can be relatively complex. It is our aim to allow modelers to drill down or expand the imported models. This will result in rather complex diagrams. As research continues we will be looking at improving the editor, or using external editors such as Donnart [82], to support the visualization of such complex models.

Such improved tool support could also be used to enhance the visualization of large biological models. The biological visualization generated for the reasonably complex IP3 model [50] can be downloaded at http://www.cellml.org/tools/downloads/cellml-viewer/releases/1.0rc2. This model can be separated into three simpler functional modules GPCR, PLC-beta, and IP3 in CellML [51]. Such a modularized model can be used to generate a simpler top-level biological view.

In the biological visualization of the K_Ionic_Flow model (Figure 4.8b), it should be noted that the glyph used to represent ion channel is different to the conventional ion channel representation supported in text books and publications, which tend to focus on the movement of ions through the channel. In contrast, our representation of a channel and a voltage as an activator in a biochemical reaction is unusual, and this may be the basis for another operation of visual notation. SBGN supports a similar notation for visualizing ion transport. One potential solution would be to collapse the detailed process of ion transport through a membrane via a channel protein into a more conventional style diagram which is well known in the electrophysiological modeling community.

Currently the glyph set for visualizing the biological representation (Figure 4.3) of a CellML model only contains the visual symbols that are needed to visualize signal transduction pathways and electrophysiological models. To support the wide range of model types which are currently in the CellML Model Repository, this glyph set will have to be extended. It is also our intention to adopt SBGN in future for visualizing the underlying biological concepts in CellML models. One possible solution we are currently exploring is using the VisualTemplate/OWL ontology to map to a SBGN process diagram representation [80]. The entity glyphs we describe here are similar to SBGN entity pool nodes, while the process glyphs are similar to SBGN process nodes, and the role glyphs are similar to SBGN connecting arcs. By representing SBGN notation in SVG, and subsequently linking the VisualTemplate/OWL ontology to these SVG elements, users will be able to visualize biological views of the CellML model using SBGN.

The development of the VisualTemplate/OWL ontology, as a separate ontology to store references to the visual templates, also provides a flexible ontological framework that could be used to visualize other biological models, particularly if the CellML-specific biological ontology is replaced by an external ontology such as BioPAX. For example, the BioPAX ontology can be extended to annotate to the instances the VisualTemplate/OWL ontology, and these mappings can be used to generate visual representations. The generation of such visual diagrams depends on the underlying ontological representation of the model. For example BioPAX identifies catalysis as a separate process and defines a concept of modulators which act on the catalysis process. This scenario cannot currently be visualized using our visual language or SBGN, as it does not support the concept of processes acting on processes. The generation of such a diagram would thus require a different visual language (Supplementary Material B Section 3).

Further, there may be problems in visualizing CellML models which have a large number of nodes. We acknowledge that the work described here does not address diagram layout issues. However, for large models it is time consuming to manually control the layout the diagrams. As research continues it is vital for us to develop an algorithm which automatically controls the layout of the diagrams in order to allow modelers to easily see the sequence of events described by a model.

Finally, we are aware that the biological visualization of the K_Ionic_Flow model shown in Figure 4.8b does not show the compartment information captured in the ontological representation of the model. It is possible to define compartments in the

VisualTemplate/OWL ontology by introducing a Compartment class. We are proposing that the visual language can be extended to support compartments as rectangular boxes. To interpret such a diagram correctly, we would need to illustrate the relationship between the entities and a compartment. One possible way to achieve this would be to arrange the entities inside the compartment. This would involve the development of an algorithm to control the layout of the entities and processes inside the compartment.

In conclusion, the main outcomes of this research include:

- a visual language that is simple to interpret but is sufficiently expressive to represent the underlying physical and biological concepts captured in CellML models;
- a specification for building visual templates that support this visual language;
- a set of rules for binding the visual templates to biological concepts within the CellMLBiophysical/OWL ontology via the VisualTemplate/OWL ontology; and
- an algorithm that combines the visual language and the ontologies to automatically generate the schematic diagrammatic representations of the CellML models.

5 A SOFTWARE TOOL FOR VISUALIZING CELLML MODELS

A number of tools have been developed to create and simulate CellML models but there is limited support for visualizing the underlying biophysical concepts captured in the mathematical models. This chapter describes the implementation and application of a software tool called CellMLViewer that integrates CellML models with the CellML ontological framework and the visual language in order to automatically generate visual representations of CellML models. CellMLViewer provides a user interface for visualizing the biophysical concepts modeled in CellML. Users can manipulate the layout of these dynamic representations to create diagrams highlighting the sequence of events. These diagrams can also be exchanged over the web without the use of the tool.

5.1 INTRODUCTION

A range of tools has been developed to support working with CellML models [83]. These include OpenCell [24], COR [66], JSim [9], Virtual Cell [6], and insilicoIDE [84]. These existing tools have contributed greatly to the creation, validation, and simulation of CellML models. In addition, they also support the graphical analysis of the simulated results.

The CellML Model Repository contains over 380 models representing a wide range of biological processes [17]. In order to understand the physical and biological concepts captured in these CellML models, modelers need to read the XML-based CellML code, refer back to the relevant peer-reviewed publication from which the model was derived, and often contact the original authors of the mathematical model for clarification. Combined, these steps frequently constitute a time consuming process for a modeler who is looking to implement, or further develop, existing models. Facilitating the automated visualization of the physical and biological concepts captured in a CellML model would enable modelers to easily identify the models of interest and allow them to quickly understand the processes being described.

There exist many tools which have been developed to visualize molecular interaction networks, including CellDesigner [85], Cytoscape [86], Virtual Cell [6] and PATIKA [73]. These tools provide interfaces to visually create, edit, and manipulate the layout of large biological models. Cell Illustrator and Virtual Cell can be used to import CellML models. Visualizations are generated by translating the CellML representation into their internal representation. As the CellML models do not explicitly capture the biophysical data, the diagrams are not focused on representing the physical and biological concepts modeled in CellML. Instead, the models are often visualized as networks of interconnected components. Another limitation is that these tools only support the import of CellML 1.0 models.

The CellML editing and simulation tool OpenCell has started to address the issue of model visualization by displaying schematic diagrams summarizing the mathematical model. These diagrams are often based on figures from the original publication, and currently they are created by hand. There are no methods in place to validate the diagrams to ensure that all the biological concepts captured in a model are displayed. The ad hoc nature of these diagrams often requires readers to refer to the original publication in order to correctly interpret the biological concepts being displayed.

To resolve the issue of automated model visualization, we have developed

- an ontological framework which can be used to represent the underlying biophysical concepts captured in the CellML models [81] (Chapter 3);
- a visual language that consistently describes the underlying biophysical concepts in the CellML models [87] (Chapter 4);
- a method to integrate the visual language to the ontological framework [87] (Chapter 4).

Here we describe a stand-alone software tool (CellMLViewer) which uses this ontological framework to visualize CellML models.

This chapter addresses the specific steps that a modeler has to follow to create visualizations for CellML models. These include:

1. visualizing CellML models without annotated biophysical data;
2. annotating CellML models with biophysical concepts and visual language data which can be used to generate more informative visualizations;
3. visualizing the annotated CellML models capturing the underlying physical and biological concepts;
4. layout the diagrams to highlight relevant biological concepts;
5. storing visual representations for reuse or publish them online.

5.2 IMPLEMENTATION

Chapters 3 and 4 described an ontological framework for visualizing CellML models, which we will use as the basis for developing our approach. Our ontological framework is constructed using the Web Ontology Language (OWL) [35]. It consists of multiple ontologies:

- CellML/OWL: which captures CellML/XML model in OWL format [81]. It provides an intermediate layer that allows us to map a CellML/XML model to other OWL based ontologies (Chapter 3);
- CellMLBiophysical/OWL: which is constructed using Physical and Biological ontologies for capturing the physical and biological concepts modeled in CellML models [81]. The CellMLBiophysical/OWL model defines the concepts and relations both within and between Physical and Biological ontologies (Chapter 3);
- VisualTemplate/OWL: which stores references to the visual language that has been developed to represent CellML models. Each CellMLBiophysical/OWL instance is mapped to a VisualTemplate/OWL instance [87] (Chapter 4).

A set of algorithms are also defined to generate these intermediate ontological representations and mapping between them. These include:

- an algorithm for transforming an CellML/XML model into an CellML/OWL model. This involves generating a CellML/OWL instance for each CellML/XML element including the explicitly imported components and the variables and math elements within them.

Each CellML/XML element is mapped to a CellML/OWL instance using the CellML metadata specification [41, 81] (Chapter 3);

- an algorithm for creating the initial CellMLBiophysical/OWL model and annotation to the CellML/OWL model [81]. This involves generating a CellMLBiophysical/OWL PhysicalEntity, PhysicalProcess, and MathematicalEquality class instance for each CellML/OWL Variable, Component, and Connection class instance, respectively. (Chapter 3);
- an algorithm for simplifying the CellMLBiophysical/OWL model using the biological ontological mappings, in combination with a set of graph reducing rules, to represent the underlying biological view of the CellML model [81] (Chapter 3);
- an algorithm for generating physical and biological visualizations of CellML models using the ontological mappings. The physical view is created using the CellMLBiophysical/OWL physical instances and retrieving the relevant SVG glyphs via the VisualTemplate/OWL mappings. Similarly, the biological view is created using simplified CellMLBiophysical/OWL model and retrieving the relevant SVG glyphs via the VisualTemplate/OWL mappings [87] (Chapter 4).

The next section discusses the application of the CellMLViewer to generate a visualization for a CellML model. We then outline the architecture of the tool, which has been developed to facilitate the generation of the intermediate models and mappings during the process of visualizing the biophysical concepts captured in the CellML models.

5.2.1 APPLICATION OF THE CELLMLVIEWER

The CellMLViewer provides a simple interactive interface to represent and layout the visualizations but does not provide support for manual annotation of the CellMLBiophysical/OWL models. OWL editors such as Protégé can be used to annotate these models. Here we describe each of the steps to illustrate how a CellML model can be annotated and visualized. We also briefly describe the behavior of the tools following the actions of the modeler (Figure 5.1).

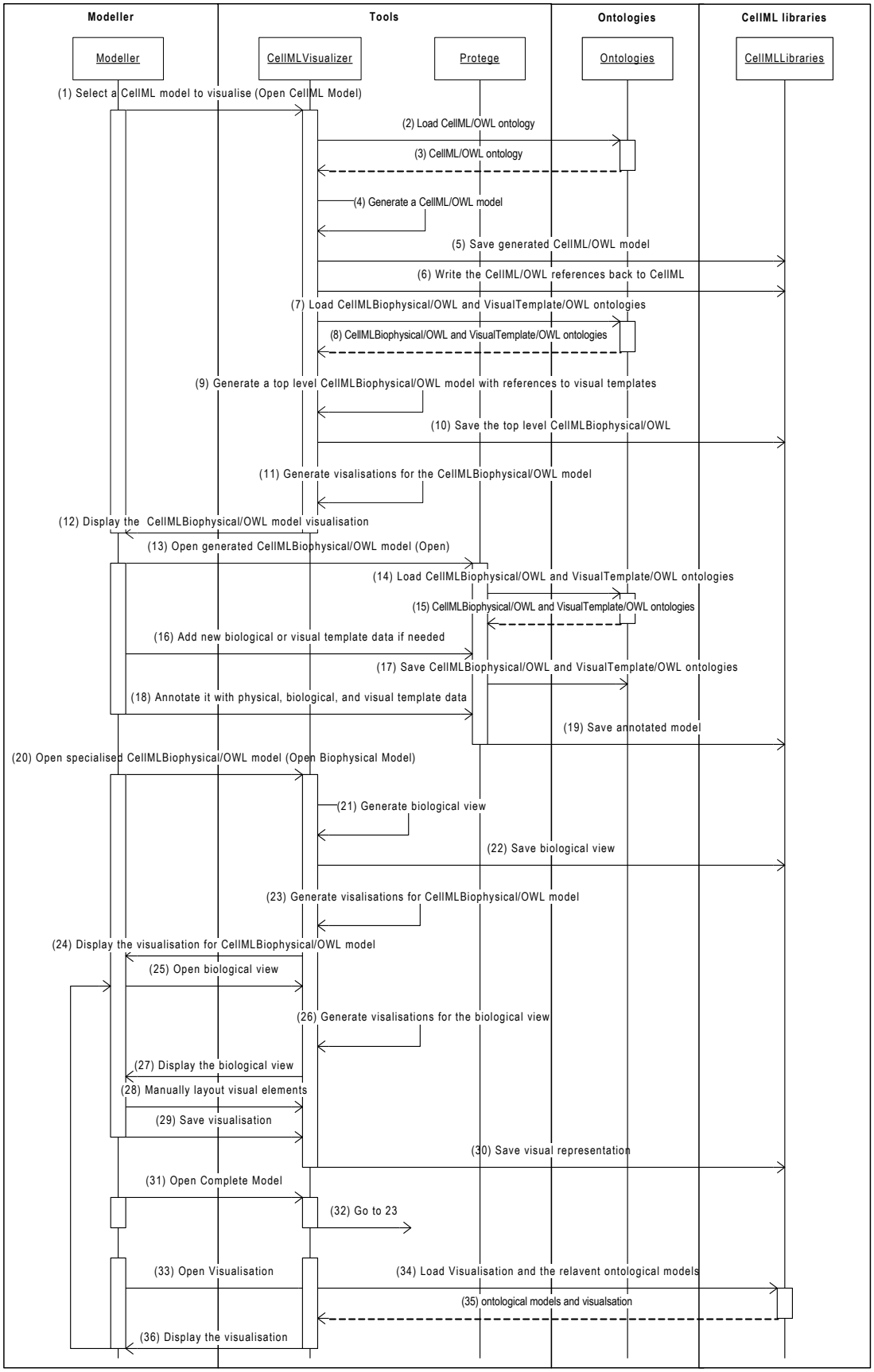


Figure 5.1: Sequence diagram for user activity flow for visualizing and annotating CellML models

5.2.1.1 Visualizing CellML models without annotated biophysical data

The CellML models without biophysical annotations can be loaded onto the CellMLViewer by selecting 'Open CellML Model' option. In order to generate a visualization, the CellMLViewer executes a series of steps. The OWL ontology is used to generate a CellML/OWL representation of the CellML/XML model. The CellML/OWL model is used to generate a CellMLBiophysical/OWL model with mappings to the VisualTemplate/OWL ontology (Figure 5.1 steps 1-12). The CellMLBiophysical/OWL model is then used to generate a diagram which identifies CellML components as physical processes, variables as physical entities, and connections as mathematical equality processes (Figure 5.2). The intermediate model representations are saved in the local file system.

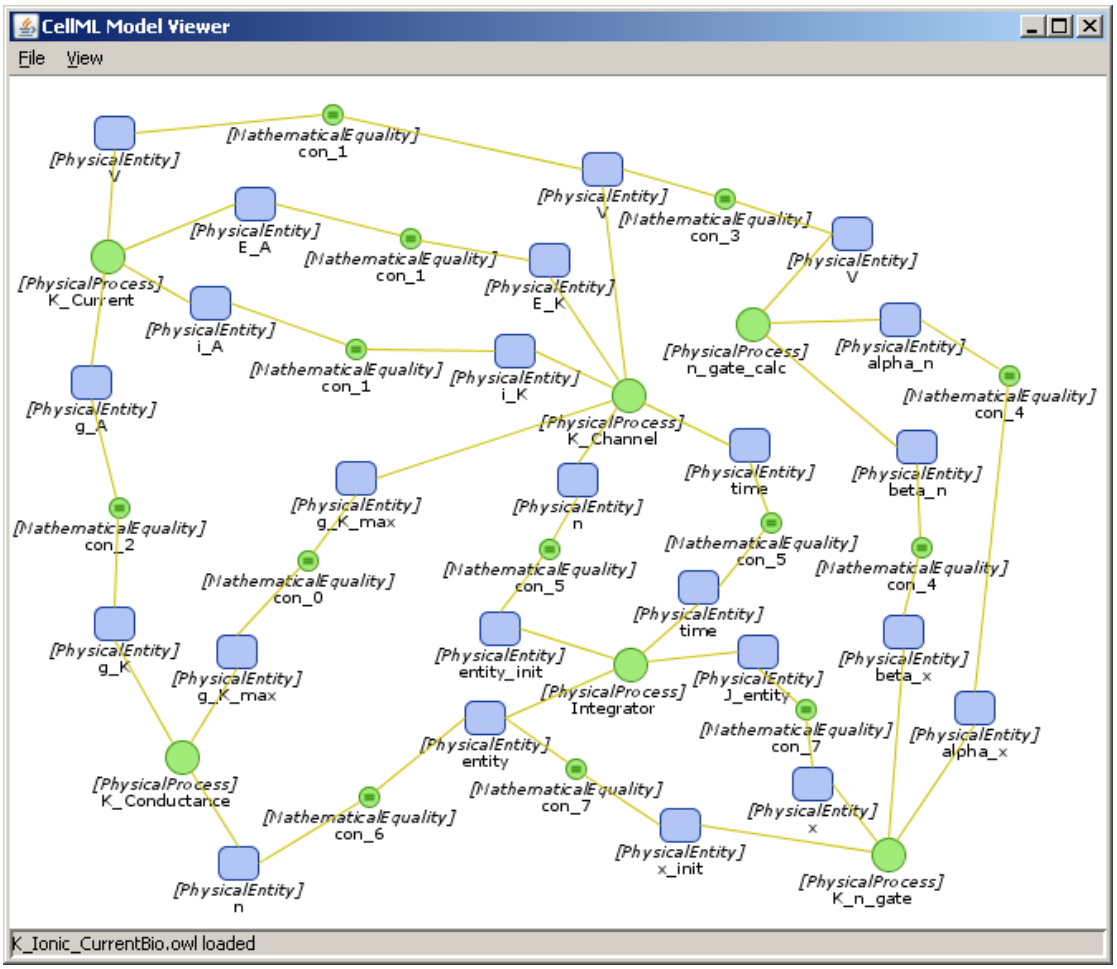


Figure 5.2: A visualization generated for a CellML model without annotations

5.2.1.2 Annotating CellML models with biophysical concepts and visual language data

The CellMLBiophysical/OWL model generated from the previous action consists of instances of PhysicalProcess and PhysicalEntity classes, and the relationship between them. These instances can be specialized to the relevant subclasses using the classes supported by the Physical ontology. The Physical instances can be further annotated with biological concepts using the classes supported by the Biological ontology. This manual annotation process is carried out using Protégé.

Protégé provides a graphical user interface for working with OWL models. Figure 5.3 shows a CellMLBiophysical/OWL model loaded on to Protégé (Figure 5.1 steps 13-19). The *INDIVIDUALS* tab is used to annotate the instances. The *CLASS BROWSER* panel list the classes captured in the CellMLBiophysical/OWL ontology. Individuals of a particular class are listed in the *INSTANCE BROWSER* panel. The CellMLBiophysical/OWL Physical instances can be specialized by dragging the instances into the related classes. Properties of the instances such as biological and visual template data can be set using the *INDIVIDUAL EDITOR* panel.

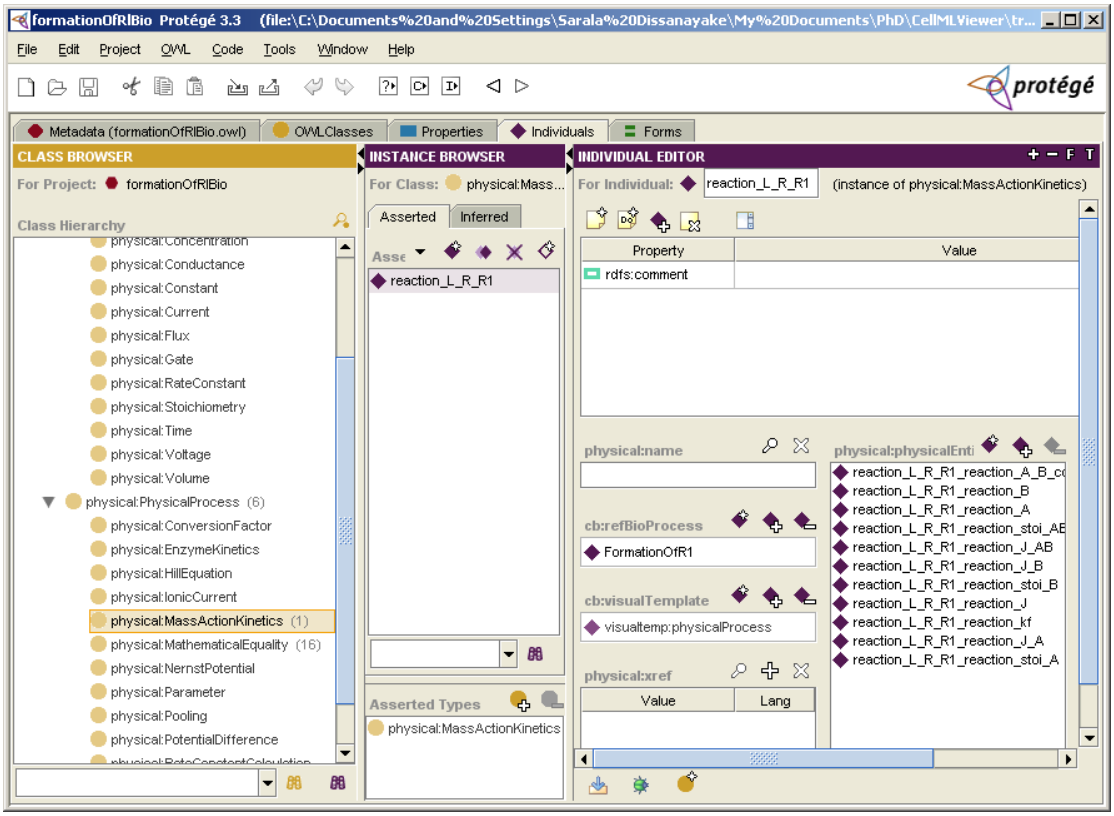


Figure 5.3: A CellMLBiophysical/OWL model loaded in Protégé

Protégé can also be used to extend the CellMLBiophysical/OWL and VisualTemplate/OWL class structure to add new concepts. New Classes can be added by navigating to the *OWL Classes* tab.

5.2.1.3 Visualizing the annotated CellML models capturing the underlying physical and biological concepts

The annotated CellMLBiophysical/OWL model can be loaded to the CellMLViewer by selecting 'Open Biophysical Model' option. This action will execute the algorithm for generating visualizations using the ontological mappings. The output diagram of the annotated CellMLBiophysical/OWL model identifies the physical processes and entities according to their specialized types (Figure 5.1 steps 20-25).

The 'Open Biophysical Model' action also executes the reducing algorithm to generate the condensed, simplified view (Figure 5.1 step 24). The 'View' menu allows users to swap between the physical and biological views (Figure 5.4).

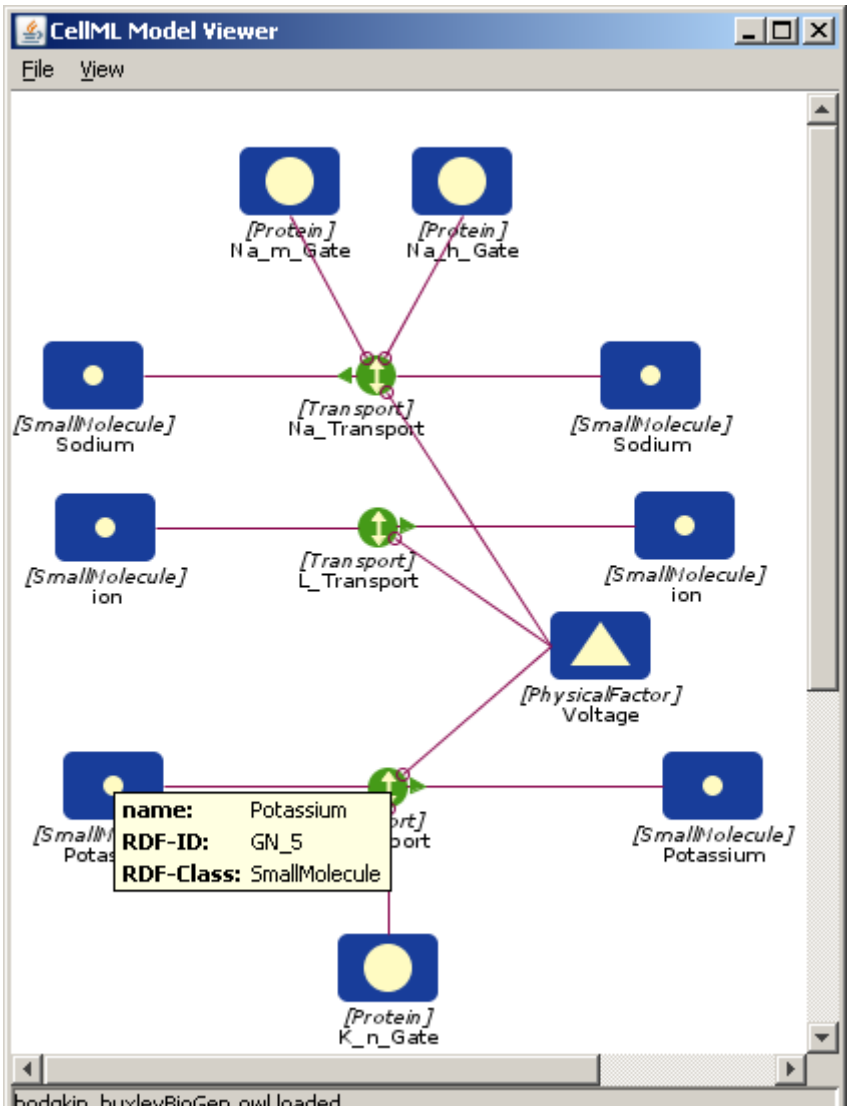


Figure 5.4: Biological view generated from an annotated CellMLBiophysical/OWL model

5.2.1.4 Layout the diagrams to illustrate the sequence of biological interactions

The CellMLViewer provides an interactive graphical user interface. Modelers can move the visual objects around the canvas to highlight relevant biological concepts (Figure 5.1 step 30). Moving over objects will prompt the user with a tooltip which contains more information about the object (Figure 5.4).

5.2.1.5 Storing visual representations

The visualizations can be saved into locations specified by the user (Figure 5.1 steps 31-32). It is also possible to re-open a saved SVG representation in the CellMLViewer. These

diagrams retain the layout information that has previously been defined (Figure 5.1 steps 33-36). This action also loads the OWL models associated with it.

5.2.2 SYSTEM ARCHITECTURE

The CellMLViewer is developed in Java to allow the application to run on multiple different platforms. It also allowed us to use existing Java APIs to work with CellML models, SVG representations, and OWL models.

The overall architecture of the system consists of three tiers: a graphical user interface (GUI) tier which provides the interface for users to carryout tasks; a logic tier which implements the algorithms that were listed above; and a repository where CellML, OWL, and SVG documents are stored (Figure 5.5).

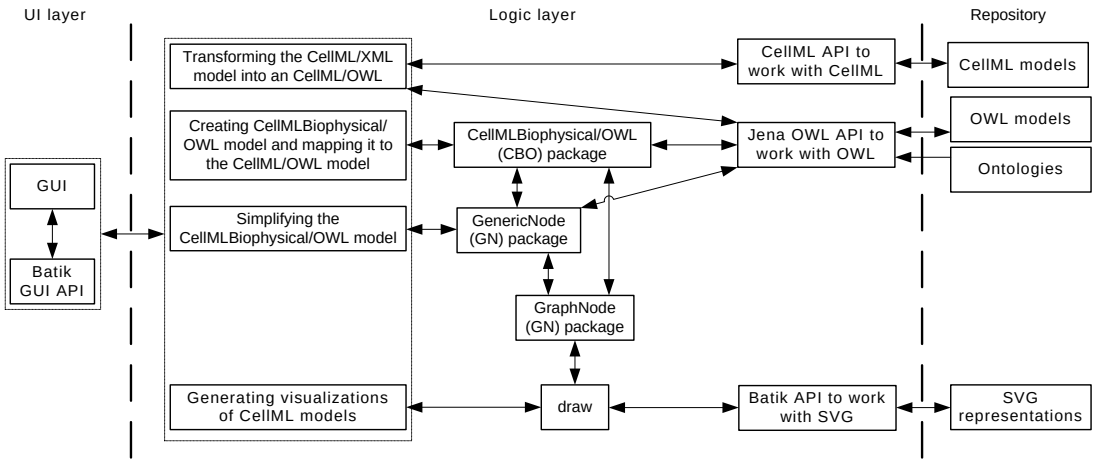


Figure 5.5: Schematic diagram of the system architecture underlying the model visualization tool CellMLViewer

The CellMLViewer GUI is integrated with the Batik API [88]to display the visual representations. Batik is an open source Java-based toolkit for working with Scalable Vector Graphics (SVG) format. It can be used to display, generate or manipulate SVG diagrams. A UI component, Batik JSVGCanvas is used to display SVG graphics and also support the interactive use of the visual content such as graphic selection and the movement of objects to generate a biologically realistic model network diagram.

The logic layer separates the algorithms used for generating model visualizations from the UI layer and the repository layer. The external java packages: Jena API [89], CellML API, and Batik API are used to create, read, write, and modify the OWL, CellML, and SVG resources, respectively. In addition to these external resources we have developed our own

set of packages to instantiate the intermediate models and to draw the application specific graphics. These include:

- *cbo* – to instantiate the CellMLBiophysical/OWL model;
- *gn* – to instantiate the simplified view generated (GenericNode/OWL model) by reducing the CellMLBiophysical/OWL model;
- *gm* – creates a graph model which separates the ontological models from the visualization. It is used as the underlying model for visualizing both the physical and the biological views;
- *draw* – to draw application specific shapes by changing SVG Document Object Model (SVG DOM).

The process of generating visualizations for CellML models creates several intermediate models. These models explicitly reference each other. CellML/XML models have references to CellML/OWL models. These CellML/OWL models have references to CellMLBiophysical/OWL models. It is possible to have many GenericNode/OWL models to a particular CellMLBiophysical/OWL model depending on the order of rules applied to reduce the model. Therefore a separate OWL file is created to save the GenericNode/OWL models. These models reference back to the original CellMLBiophysical/OWL models. Each visual representation has a reference to the simplified view and the CellMLBiophysical/OWL model. The associations between these models are summarized in Figure 5.6.

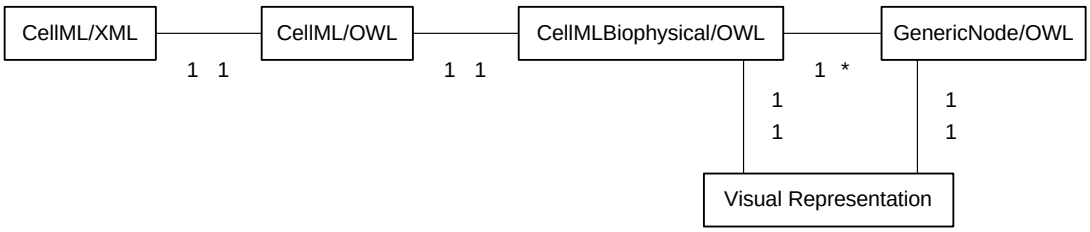


Figure 5.6: Model associations

The lines describe the cardinality relationship between the instances. 1 denotes to one and * denotes to many.

5.3 DISCUSSION

Here we have presented a tool that can be used to:

- generate CellML/OWL models;
- generate initial CellMLBiophysical/OWL models which are ready for further specialization;

- provide an interface to visualize the biological and physical concepts captured in CellML models;
- provide an interface to layout the diagrams to represent the sequence of interactions between the biological entities.

The SVG graphics generated by CellMLViewer can be exchanged via the web and can be viewed with any web browsers with SVG support. Even though viewing the diagrams through web browsers does not allow modelers to interact with the model, it is a simple way of communicating the underlying biological concepts. With such a goal in mind, we intend to upload these schematic diagrams on the CellML Model Repository website together with the model descriptions.

Although the CellMLViewer is a vast improvement on the current system of hand-generating the schematic diagrams of the CellML models, we acknowledge that the tool does have its limitations. For example, it can become slow when handling large models containing more than 200 nodes. When the number of elements on the canvas increases, the support operations become slow. As research continues this feature needs to be enhanced by improving the algorithm implementations as well as the batik SVG renderer. Faster rendering of SVG has been the subject of several recent discussions on the Batik mailing list [88].

A further limitation of CellMLViewer is lack of a built-in model annotation interface. Currently users have to look to external software tools, such as Protégé, to annotate the CellMLBiophysical/OWL models. Protégé is a fairly advanced and complex tool, and it would be advantageous to have an interface which would facilitate the annotation of the CellMLBiophysical/OWL models within the CellMLViewer itself.

Currently the CellMLViewer does not support the automated layout of diagrams. Users are required to manipulate the layout of the diagrams manually by moving the objects around. This feature allows modelers to arrange the diagram to highlight the biological concepts of interest. However, when working with large models this can be a time consuming task. In these situations it would be useful to provide a basic automated layout, which can be subsequently rearranged by the user if required.

Currently the SVG diagrams, CellML/OWL and CellMLBiophysical/OWL models are saved as individual files in the local file system. These need to be moved to PMR to allow public access to the CellML users. The CellMLViewer tool needs to be able to connect to such a repository to retrieve and store models and visual representations.

The CellMLViewer tool is developed following best practice methods to allow developers to easily extend the system. The three tier architecture is used to logically separate the processes. A set of loosely coupled packages are developed to work with intermediate model representations. Each algorithm is tested by writing unit tests. As the tool evolves, it is our intention to carry out unit testing, integration testing, and user evolutions.

It is also our intention to enhance the tool to visualize the composite nature of the CellML models. The CellML encapsulation feature allows modelers to hide information about a set of components from rest of the model. The user interface will be extended to provide controls to hide and expose details of models captured via the CellML encapsulation hierarchy.

Finally, we intend to integrate the CellMLViewer with the editing and simulation environment OpenCell, in order to provide a complete CellML modeling framework.

5.4 AVAILABILITY AND REQUIREMENTS

Project name: CellMLViewer
Project home page: http://www.cellml.org/tools/downloads/cellml-viewer
Operating system(s): Platform independent
Programming language: Java
Other requirements: Java 1.5 or higher
License: GNU GPL

6 CONCLUSIONS

The aim of this work was to provide a software framework to visualize the underlying biology of the CellML models. While focusing on visualization of CellML models, this thesis presents solutions for a number of challenges to the CellML modeling community. These include:

- constructing models that clearly identifies the biophysical processes;
- a method to capture the biophysical concepts modeled in CellML models and explicitly annotating CellML elements with the biophysical concepts;
- a visual language for representing the biophysical concepts and a method to bind them to the CellML elements;
- a software tool that combines the visual language and the biophysical concept to generate visualizations for CellML models.

Chapter 2 discusses a set of guidelines for modularizing a CellML model in a way that best describes the biophysical concepts and abstractions the modeler wish to demonstrate. This method helps modelers to construct models that can easily be reused and extended to build more complex models. Developing modularized models also increases the consistency across models and improves interpretation of the complex CellML models.

The modularization process identifies a structured way to build new CellML models but modularizing the 360 models that currently exist in the CellML repository remain as a challenge. This problem can be partially resolved by starting with modeling commonly used biophysical concepts (such as reaction kinetics) and basic models (such as Hodgkin-Huxley model) that have been extensively extended over time. Modelers can then reuse and extend these smaller models to build more complex models.

A formal justification for the guidelines was not carried out during this work as it was not our intension. The guidelines were developed following a set of examples which provided the best arrangement for generating diagrams. As research continues, it is our intension to explore the theories behind modularization and reuse of models, such as network theory [46], to validate the modularized concepts captured in CellML models.

Chapter 3 discusses annotating CellML models with physical and biological meaning. The physical annotations enable modelers to interpret the CellML model in terms of the mathematical functions and physical qualitative data, without the need to go through all the individual CellML elements. The annotated biophysical model is used to construct a simplified view that highlights the underlying biological concepts which is easier to interpret.

The biophysical ontology developed during this work only reflects the concepts found in the example models. As new models are annotated, the ontology needs to be extended to support the additional concepts. The intension is to also map instances of biological and physical terms to existing external ontologies and controlled vocabularies such as SBO [30], OPB [90], GO [59], ChEBI [91], and BioPAX [92]. This allows modelers to take advantage of existing knowledge definitions.

The construction of simplified views uses a reducing algorithm which consists of a generic and specific set of rules. The generic rule set does not require further improvements as it can be applied on any instance of a biophysical model. The specific rule set is dependent on the interpretation of the biophysical annotations. It can be improved to support collapsing of different types of CellML models in the repository as needed.

The CellML repository needs to be improved to facilitate storage and querying of ontological models. Use of ontologies enhances the querying of CellML models. Biophysical annotations can be used to query physical and biological details of a CellML model. A goal would be to populate the Biological ontology with entities and processes used in CellML models and annotate the physical instances against the common biological concepts. This will

enable modelers to query the CellML repository to find models that interest them according to the underlying biological annotations. For example it would possible find all the models which uses a particular ion channel. It is also our intension to extend this framework to generate composite models using OWL reasoner.

Chapter 4 describes a visual language developed to represent the biophysical concepts modeled in CellML and a method to map it to the ontological framework. These annotations are then used to generate visualizations for CellML models to help modelers to easily recognize the biophysical concepts modeled in CellML without having to go through the complex ontological representation.

Two separate visual representations are generated that represent the physical and biological concepts. The physical view represents the structure of a CellML model in terms of Components, Variables, and Connections with the physical annotations. The biological view uniquely identifies biological entities, processes, and roles. Together these views can be used to not only to interpret the biophysical concepts but also understand the structure of a CellML model.

Extending the biophysical ontology to support additional physical concepts does not require extending the glyph set as the different physical types are represented using text. In contrast, additional biological concepts supported by extending the biophysical ontology or integrated with external ontologies or controlled vocabularies will require additional glyphs if the new concepts need to be uniquely identified in a diagram.

The development of the ontological framework focuses on generating the underlying biological view of the CellML model. Therefore, the biophysical annotations, application of the reducing rules, and visual language are all developed focusing on generating biological visualizations. However, it provides an extensible ontological framework for annotating the CellML elements against other ontologies, use the reducing rules to collapse the CellML model focusing on different ontological properties, and a visual language to visualize these new concepts. For example the CellMLBiophysical/OWL model can be annotated with an ontology defining mathematical constructs; a set of specific rules can be developed to reduce the complexity of a CellML by simplifying the mathematical constructs, and the simplified CellML model can be visualized by creating a set of new glyphs.

The biological visualizations are generated from the reduced model which in turn depends on the annotations and the structure of the CellML model. Therefore a particular

CellML model can have multiple visualizations. The visualizations can be used as an aid to dictate the structure of CellML model and its annotations. This helps modelers to build modularized CellML models grouping the underlying biological concepts.

Chapter 5 describes the CellMLViewer tool that combines CellML, the ontological framework, and the visual language to generate visualizations. It provides a simple graphical user interface for modelers to interact with the visualizations. The tool can be used to generate a visual representation and layout the visualization to highlight sequence of biological interactions. This image can be saved as a SVG file which can then be published on the Physiome Model Repository.

While this work has provided significant contribution towards building and visualizing modularized annotated CellML models, it provides limited tool support for efficient construction of these models. The tasks of constructing modularized models, annotating biophysical information, and laying out visualizations are all manual steps which can be time consuming and error prone. As research continues, the tool support needs to be improved to provide interfaces to visually construct modularized CellML models and laying out visual elements, programmatically annotating physical concepts by interpreting the units and mathematics, and providing intelligent suggestions for biological annotations. It is also the intention to enhance the tool support to allow modelers to visually traverse between the models and complement annotations to generate different visualizations.

We also acknowledge that it is important to visualize the dynamics of the modeled processes as they are dynamic by nature. This work focuses on the initial attempt to visualize CellML models by only looking at visualizing the static biological concepts modeled in CellML. The final diagrams have references to CellML variables or processes and the values of those variables can be reflected in the final diagram. The workflow for generating visualizations does not lose information during the process. As research continues, we will be looking at ways to represent this dynamic information. Use of SVG will also help to extend this work to support dynamic changing of visualizations.

While this work focuses on CellML, it is a generic solution which can be extended to visualize other model representations, including SBML models. The CellML/OWL ontology is specific to CellML as it captures the structure of a CellML model. The CellMLBiophysical/OWL ontology captures biophysical concepts and can be used to annotate SBML models. The annotated models can then be visualized discussed this thesis.

The method of using OWL to integrate information can also be used to visualize ontology based models such as BioPAX. As research continues, we will be looking at using this workflow for visualizing SBML and BioPAX models.

Biologists make extensive use of diagrams, a form of pictures, when communicating information. The diagrammatic representations created by this framework explicitly preserve the information about the topological and geometric relations among the biological entities and processes. The diagrams are unambiguously defined, contain sufficient information and are based on well-defined notation which allows domain experts to interpret complex biological diagrams in the same way. Such diagrams can not only be used for communicating information, but also further enhance understanding of the complex nature of biological systems and discovery of new knowledge.

The outcome of this research offers an extensible and constructive software framework for working with CellML models. It helps modelers by promoting construction of:

- CellML models that are easier to interpret and reuse;
- annotated CellML models that clearly identify the biophysical concepts;
- visualizations that help to understand the biophysical concepts captured in their complex structures.

REFERENCES

1. Hunter, P.J. and T.K. Borg, *Integration from proteins to organs: the Physiome Project.* Nature Reviews Molecular Cell Biology, 2003. **4**.
2. MATLAB, *The MathWorks.* [Available online from http://www.mathworks.com/], 2009.
3. Bray, T., et al., *Extensible Markup Language (XML) 1.0 (Fifth Edition).* [Available online from http://www.w3.org/TR/2008/REC-xml-20081126/], 2008.
4. Ausbrooks, R., et al., *Mathematical Markup Language (MathML) Version 2.0.* [Available online from http://www.w3.org/TR/MathML2/], 2003.
5. S.Buswell, et al., *The OpenMath Standard V2.* [Available online from http://www.openmath.org/], 2004.
6. Loew, L.M. and J.C. Schaff, *The Virtual Cell: a software environment for computational cell.* TRENDS in Biotechnology, 2001. **19**.
7. Tomita, M., et al., *E-Cell: software environment for whole-cell simulation.* Bioinformatics, 1999. **15**(1): p. 72-84.
8. Hoops, S., et al., *COPASI - a COmplex PAthway SImulator.* Bioinformatics, 2006. **22**(24): p. 3067-3074.
9. Bassignthwaighte, J., *JSim.* [Available online from http://www.physiome.org/jsim/], 2009.
10. W3C, *World Wide Web Consortium.* [Available online from http://www.w3.org/], 2009.
11. Herman, I., R. Swick, and D. Brickley, *Resource Description Framework (RDF).* [Available online from http://www.w3.org/RDF/], 2004.
12. Lloyd, C.M., M.D.B. Halstead, and P.F. Nielsen, *CellML: its future, present and past.* Progress in Biophysics & Molecular Biology, 2004. **85**: p. 433-450.
13. Hucka, M., et al., *The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models.* Bioinformatics, 2003. **19**: p. 524–531.
14. Nagasaki, M., et al., *Cell System Markup Language (CSML).* [Available online from http://www.csml.org/documentation/csml-30-specification/], 2007.

15. VCell, *The VCML Specification Version 0.4.* [Available online from http://ntcnp.org/twiki/bin/view/VCell/VcmlSpec], 2009.

16. Hedley, W.J., et al., *A short introduction to CellML.* Philosophical Transactions of the Royal Society, 2001(359): p. 1073-1089.

17. Lloyd, C.M., et al., *The CellML Model Repository.* Bioinformatics, 2008. **24**(18): p. 2122–2123.

18. Cuellar, A.A., et al., *An overview of CellML 1.1, a biological model description language.* Simulation, 2003. **79**(12): p. 740-747.

19. DeRose, S., E. Maler, and D. Orchard, *XML Linking Language (XLink) Version 1.0.* [Available online from http://www.w3.org/TR/xlink/], 2001.

20. Hunter, P.J., *The IUPS Physiome Project: a framework for computational physiology.* Progress in Biophysics & Molecular Biology, 2004. **85**: p. 551–569.

21. CellML, *The CellML specifications.* [Available online from http://www.cellml.org/specifications], 2009.

22. PMR, *Physiome Model Repository.* [Available online from http://www.cellml.org/tools/pmr/], 2009.

23. CellMLAPI, *CellML Application Programming Interface.* [Available online from http://www.cellml.org/tools/api/], 2009.

24. OpenCell, *An environment for working with CellML models.* [Available online from http://www.cellml.org/tools/opencell/], 2009.

25. Novère, N.L., *BioModels.* [Available online from http://www.ebi.ac.uk/biomodels/], 2009.

26. SBO, *Systems Biology Ontology*. [Available online from http://www.ebi.ac.uk/sbo/], 2006.

27. Hucka, M., et al., *Evolving a lingua franca and associated software infrastructure for computational systems biology: the Systems Biology Markup Language(SBML) project.* Systems Biology, 2004. **1**(1).

28. Kitano, H., *Systems Biology: A Brief Overview.* Science, 2002. **295**(5560): p. 1662-1664.

29. Hucka, M., et al., *Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions.* [Available online from http://sbml.org/Documents/Specifications], 2007.

30. Novère, N.L., *Model storage, exchange and integration.* BMC Neuroscience, 2006. **7**: p. 1471-2202.

31. Keating, S., et al., *libSBML.* [Available online from http://sbml.org/Software/libSBML], 2009.

32. Mathematica, *Wolfram Mathematica.* [Available online from http://www.wolfram.com/], 2009.

33. Hucka, M., *SBML Software Guide.* [Available online from http://sbml.org/SBML_Software_Guide], 2009.

34. Noy, N.F. and D.L. McGuinness, *Ontology Development 101: A Guide to Creating Your First Ontology.* 2001.

35. McGuinness, D.L. and F.v. Harmelen, *OWL Web Ontology Language Overview.* [Available online from http://www.w3.org/TR/owl-features/], 2004.

36. OBO, *The Open Biomedical Ontologies.* [Available online from http://www.obofoundry.org/], 2009.

37. Jeong, E., et al., *Cell System Ontology: Representation for Modeling, Visualizing, and Simulating Biological Pathways.* In Silico Biology, 2007. **7**: p. 623–638.

38. Kamp, T.J. and J.W. Hell, *Regulation of Cardiac L-Type Calcium Channels by Protein Kinase A and Protein Kinase C.* Circulation Research, 2000. **87**: p. 1095-1102.

39. Ferraiolo, J., F. Jun, and D. Jackson, *Scalable Vector Graphics (SVG) 1.1 Specification.* [Available online from http://www.w3.org/TR/SVG/], 2003.

40. Cuellar, A., et al., *CellML 1.1 Specification.* [Available online from http://dev.cellml.org/specifications/cellml_1.1/], 2005.

41. Cuellar, A.A., M. Nelson, and W. Hedley, *CellML Metadata 1.0 Specification.* [Available online from http://www.cellml.org/specifications/metadata/cellml_metadata_1.0], 2006.

42. Miller, A., *Simulation Metadata Specification.* [Available online from http://www.cellml.org/specifications/metadata/simulations], 2007.

43. Nickerson, D. and A. Miller, *CellML Graph Metadata Specification - 02*. [Available online from http://www.cellml.org/specifications/metadata/graphs], 2007.

44. Mirschel, S., et al., *PROMOT: modular modeling for systems biology*. Bioinformatics, 2009. **25**(5): p. 687–689.

45. Ginkel, M., et al., *Modular modeling of cellular systems with ProMoT/Diva.* Bioinformatics, 2003. **19**(9): p. 1169–1176.

46. Gilles, E.D., *Network Theory for Chemical Processes.* Chemical Engineering Technology, 1998. **21**(2): p. 121-132.

47. Modellica, *Modeling of Complex Physical Systems.* [Available online from http://www.modelica.org/], 2009.

48. Elmqvist, H., F.E. Cellier, and M. Otter, *Determining Models*, in *The Control Handbook*, W.S. Levine, Editor. 1995, IEEE Press. p. 99-112.

49. Elmqvist, H., F.E. Cellier, and M. Otter. *Object Oriented Modeling of Hybrid Systems*. in *European Simulation Symposium*. 1993.

50. Cooling, M., P. Hunter, and E.J. Crampin, *Modeling Hypertrophic IP3 Transients in the Cardiac Myocyte.* Biophysical Journal, 2007. **93**: p. 3421-3433.

51. Cooling, M.T., P.J. Hunter, and E.J. Crampin, *Modelling biological modularity with CellML*. IET Systems Biology, 2008. **2**(2): p. 73-79.

52. Hodgkin, A.L. and A.F. Huxley, *A quantitative description of membrane current and its application to conductance and excitation in nerve.* The Journal of Physiology, 1952. **117**(4): p. 500-544.

53. Noble, D., *A modification of the Hodgkin-Huxley equations applicable to Purkinje fibre action and place-maker potentials.* Journal of Physiology, 1962. **160**: p. 317-352.

54. Mangold, M., S. Motz, and E.D. Gilles, *A network theory for the structured modelling of chemical processes.* Chemical Engineering Science, 2002. **57**(19): p. 4099-4116.

55. Brickley, D. and R.V. Guha, *RDF Vocabulary Description Language 1.0: RDF Schema.* [Available online from http://www.w3.org/TR/rdf-schema/], 2004.

56. Stevens, R., C.A. Goble, and S. Bechhofer, *Ontology-based knowledge representation for bioinformatics.* Briefings in Bioinformatics, 2000. **1**(4): p. 398-414.

57. BioPAX, *BioPAX - Biological Pathway Exchange Language.* [Available online from http://www.biopax.org/], 2005.

58. Rosse, C. and J.L.V. Mejino, *A reference ontology for biomedical informatics: the Foundational Model of Anatomy.* Biomedical Informatics, 2003. **36**: p. 478-500.

59. Ashburner, M., et al., *Gene Ontology: tool for the unification of biology.* Nature Genetics, 2000. **25**: p. 25 - 29.

60. Sioutos, N., et al., *NCI Thesaurus: A semantic model integrating cancer-related clinical and molecular information.* Biomedical Informatics, 2007. **40**: p. 30–43.

61. Stevens, R., et al., *Using OWL to model biological knowledge.* Human-Computer Studies, 2007. **65**: p. 583-594.

62. Garny, A., et al., *CellML and associated tools and techniques.* Philosophical Transactions of the Royal Society A, 2008. **366**(1878): p. 3017-3043.

63. Protege, *Protégé: ontology editor and knowledge-base framework.* [Available online from http://protege.stanford.edu/], 2009.

64. SWOOP, *Semantic Web Ontology Editor.* [Available online from http://www.mindswap.org/2004/SWOOP/], 2009.

65. Wimalaratne, S.M., et al., *Facilitating Modularity and Reuse: Guidelines for Structuring CellML 1.1 Models by Isolating Common Biophysical Concepts.* Experimental Physiology, 2008. **94**: p. 472-485.

66. Garny, A., et al., *Cellular Open Resource (COR): current status and future directions.* Philosophical Transactions of the Royal Society A, 2009. **367**(1895): p. 1885-1905.

67. Beard, D.A., et al., *CellML metadata standards, associated tools and repositories.* Philosophical Transactions of the Royal Society A, 2009. **367**(1895): p. 1845-1867.

68. Kremer, R. *Visual Languages for Knowledge Representation*. in *Eleventh Workshop on Knowledge Acquisition*. 1998. Canada: http://ksi.cpsc.ucalgary.ca/KAW/KAW98/kremer/.

69. Cook, D.L., J.F. Farley, and S.J. Tapscott, *A basis for a visual language for describing, archiving and analysing functional models of complex biologial systems.* Genome Biology, 2001. **2**: p. 0012.1–0012.10.

70. Kohn, K.W., et al., *Molecular Interaction Maps of Bioregulatory Networks: A General Rubric for Systems Biology.* Molecular Biology of the Cell, 2005. **17**(1): p. 1-13.

71. Kitano, H., et al., *Using process diagrams for the graphical representation of biological networks.* Nature Biotechnology, 2005. **23**(8): p. 961-966.

72. Kolpakov, F.A. *BioUML – Framework for visual modeling and simulation biological systems*. in *Proc. Int. Conf. Bioinf. of Genome Regulation and Structure*. 2002.

73. Demir, E., et al., *Patika: an integrated visual environment for collaborative construction and analysis of cellular pathways.* Bioinformatics, 2002. **18**: p. 996-1003.

74. Demir, E., et al., *An ontology for collaborative construction and analysis of cellular pathways.* Bioinformatics, 2004. **20**: p. 349-356.

75. Doi, A., et al., *Constructing Biological Pathway Models with Hibrid Functional Petri Nets.* In Silico Biology, 2004. **4**: p. 271-291.

76. Baitaluk, M., et al., *PathSys: integrating molecular interaction graphs for systems biology.* BMC Bioinformatics, 2006. **7**(55): p. doi: 10.1186/1471-2105-7-55.

77. Schreiber, F., *High Quality Visualization of Biochemical Pathways in BioPath.* In Silico Biology, 2002. **2**(2): p. 59-73.

78. Moodie, S., et al., *A Graphical Notation to Describe the Logical Interactions of Biological Pathways.* Journal of Integrative Bioinformatics, 2006. **3**(2): p. 36–46.

79. Suzuki, Y., et al. *A Platform for in silico Modeling of Physiological Systems II. CellML Compatibility and Other Extended Capabilities*. in *30th Annual International IEEE EMBS Conference*. 2008. Vancouver, British Columbia, Canada. 573 - 576.

80. Novère, N.L., et al., *The Systems Biology Graphical Notation.* Nature Biotechnology, 2009. **27**: p. 735 - 741.

81. Wimalaratne, S.M., et al., *Biophysical annotation and representation of CellML models.* Bioinformatics, 2009. **25**: p. 2263-2270.

82. Dunnart, *Constraint-Based Diagram Editor.* [Available online from http://www.csse.monash.edu.au/~mwybrow/dunnart/], 2009.

83. Garny, A., et al., *CellML and associated tools and techniques.* Philosophical Transactions of the Royal Society A, 2008. **366**(1878): p. 3017-3043.

84. insilicoIDE, *Integrated Development Environment.* [Available online from http://www.physiome.jp/downloads/], 2009.

85. Funahashi, A., M. Morohashi, and H. Kitano, *CellDesigner: a process diagram editor for gene-regulatory and biochemical networks.* Biosilico, 2003. **1**: p. 159-162.

86. Shannon, P., et al., *Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks.* Genome Research, 2003. **13**: p. 2498-2504.

87. Wimalaratne, S.M., et al., *A method for visualizing CellML models.* Bioinformatics, 2009. **25**(22): p. 3012-3019.

88. Batik, *Batik SVG Toolkit.* [Available online from http://xmlgraphics.apache.org/batik/], 2009.

89. Jena, *Jena 2 Ontology API.* [Available online from http://jena.sourceforge.net/ontology/index.html], 2009.

90. Cook, D.L., et al. *Bridging Biological Ontologies and Biosimulation: The Ontology of Physics for Biology*. in *AMIA 2008 Symposium Proceedings*. 2008.

91. Degtyarenko, K., et al., *ChEBI: a database and ontology for chemical entities of biological interest.* Nucleic Acids Res., 2008. **36**: p. D344–D350.

92. Luciano, J.S. and R.D. Stevens, *e-Science and biological pathway semantics.* BMC Bioinformatics, 2007. **8**: p. 1471-2105.