# An Overview of CellML 1.1, a Biological Model Description Language

**Autumn A. Cuellar**
**Catherine M. Lloyd**
**Poul F. Nielsen**
**David P. Bullivant**
**David P. Nickerson**
**Peter J. Hunter**
The Bioengineering Institute
University of Auckland
Private Bag 92019
Auckland, New Zealand
*a.cuellar@auckland.ac.nz*

CellML is an XML-based exchange format developed by the University of Auckland in collaboration with Physiome Sciences, Inc. CellML 1.1 has a component-based architecture allowing a modeller to build complex systems of models that expand and reuse previously published models. CellML Metadata is a format for encoding contextual information for a model. CellML 1.1 can be used in conjunction with CellML Metadata to provide a complete description of the structure and underlying mathematics of biological models. A repository of over 200 electrophysiological, mechanical, signal transduction, and metabolic pathway models is available at www.cellml.org.

**Keywords:** Biological model, XML, markup language, mathematical model

## 1. Introduction

CellML is a standard format for defining and exchanging biological models. CellML is based on the eXtensible Markup Language (XML) [1], a structured document format that can be read by both humans and machines. Documents written in XML are plain text and, therefore, can be created and edited in any basic text editor. XML documents can be easily transmitted over the Internet. Because of its acceptance by the Web community as a standard, vast amounts of software that read, write, manipulate, and process XML documents are both freely and commercially available through the Web.

CellML differs distinctly in scope from other biological markup languages such as MageML, ChemML, and NeuroML. CellML's closest relative is the Systems Biology Markup Language (SBML) [2]. SBML is "meant to support basic biochemical network models." In comparison, CellML is an equally rigorous description language covering a more general field of application. The flexibility of the CellML language is demonstrated by the variety of models available on the CellML Web site, including electrophysiological and mechanical models as well as biochemical pathway models. In addition to being broader in scope than SBML, CellML has a modular structure that facilitates the description of complex interconnected cell models.

CellML takes advantage of the extensibility of the language on which it is based. CellML already incorporates multiple XML-based standards, including MathML, XLink, and Resource Description Framework (RDF). CellML elements are used to define model structure, but other information is incorporated into the model document using existing standards, which ensures that parts of the CellML model may still be understood by XML processors that are not CellML compliant. MathML [3] is used to encode the mathematics of the model, XLink [4] is used to establish the connection between the original model and the importing model, and background information, or metadata, is included via RDF [5]. Furthermore, modellers have the option of using features of other biological markup languages in conjunction with CellML. For instance, if a biologist wanted to give the molecular description of a species participating in a reaction, he or she could include ChemML constructs in the ChemML namespace in a CellML component.

The specification of the CellML language is contained in two separate documents. The first, the CellML 1.1 specification [6], demonstrates how a model author may specify the underlying mathematics of the model without

*SIMULATION,* Vol. 79, Issue 12, December 2003 740-747

DOI: 10.1177/0037549703040939

including simulation-specific information. The second, the CellML Metadata specification [7], allows an author to include other pertinent information at his or her discretion. The same CellML document can be processed by rendering software to generate equations suitable for publishing, or it can be processed by solution software to generate computer code that can then be compiled and executed to perform simulations, ensuring that the executable model and the published equations are consistent.

Since an in-depth description of CellML 1.0 has already been published [8], this article only gives a brief overview of the CellML structure, including metadata, before describing the recent introduction of the import and reuse capabilities with CellML 1.1 and explaining how biologists can use CellML with their favorite simulation package.

## 2. CellML Model Structure

### 2.1 Models and Components

CellML models are represented by a network of interconnected components. A component is the smallest functional unit of a CellML model. Each component may contain variables, and it may also contain mathematical equations to describe how that component behaves within the model. These equations are expressed using the content markup portion of MathML 2.0 [3], an XML-based language that is embedded within the CellML framework.

These features of the CellML language can be illustrated with an example of a metabolic pathway model such as glycolysis [9] (see Fig. 1). Biochemical reactions between substrates are organized into components that represent the reactants and products of the reactions, the reactions themselves, and the enzymes or inhibitors that influence the reaction rates. The properties of a reaction—such as its reactants, products, enzymes, and inhibitors—and its reaction kinetics are all captured by the variables and the mathematical equations of a component.

### 2.2 Variables

A CellML variable is a named entity that belongs to a single component. That is, a variable is declared, or defined, within one component only. When a variable is being declared, it must be associated with *units*. It has an optional *initial_value*, and it may or may not have *public_* or *private_interface*s. See section 2.5 for the definitions and explanations of these terms.

### 2.3 Connections

Although a variable can only belong to a single component, it is possible to map its value to other components in the model; however, a variable's value cannot be altered by these importing components. Due to the complex, integrated nature of many biological models, a large number of mappings usually arise. To cope with the potential complexity of all these mappings, mappings between components are organized into sets between any two given components. These sets of mappings between two given components are known as *connections*. There can only be one connection between any two components in a model, so all mappings between these two components have to be listed in the one connection element.

The CellML fragment below shows the connection made between the component representing the species `ATP` and a reaction in which the species participates, `reaction10`, in the glycolysis pathway shown in Figure 1. The variables `ATP` and `delta_ATP_rxn10` are being mapped between the two components. `ATP` represents the concentrations of the substrate adenosine triphosphate, and `delta_ATP_rxn10` represents the change in concentration in ATP in reaction 10. In both components, the variables are referenced by the same name, although this is not always the case:

```
<connection>
  <map_components
    component_1="ATP"
    component_2="reaction10"/>
  <map_variables
    variable_1="ATP"
    variable_2="ATP"/>
  <map_variables
    variable_1="delta_ATP_rxn10"
    variable_2="delta_ATP_rxn10"/>
</connection>
```

### 2.4 Units

One essential feature of the CellML language is that all variables and numbers must be declared with units. Units declarations ensure the robustness and reusability of the CellML components and models as they allow components and models containing variables with different units to be connected. However, it should be emphasized that variables that are mapped between components and models must be the same class of units. That is, it is possible to map two variables with units of mass, even if one is in imperial pounds and the other in metric kilograms, but modellers are unable to map two variables if one has units of length, such as meters, while the other has units of mass, such as grams.

CellML provides a dictionary of standard units that may be used when declaring a variable or when including pure numbers in equations. This dictionary contains the International System of Units (SI) [10, 11] as a base, with the addition of derived units, such as coulomb, joule, and volt, and other units, such as gram and liter, that are frequently used in the types of biological models described by CellML. New units may be expressed in terms of the units already defined in the dictionary. Using this method, it is possible to create complex units made up from the combination of simple units, define imperial units (which are expressed as

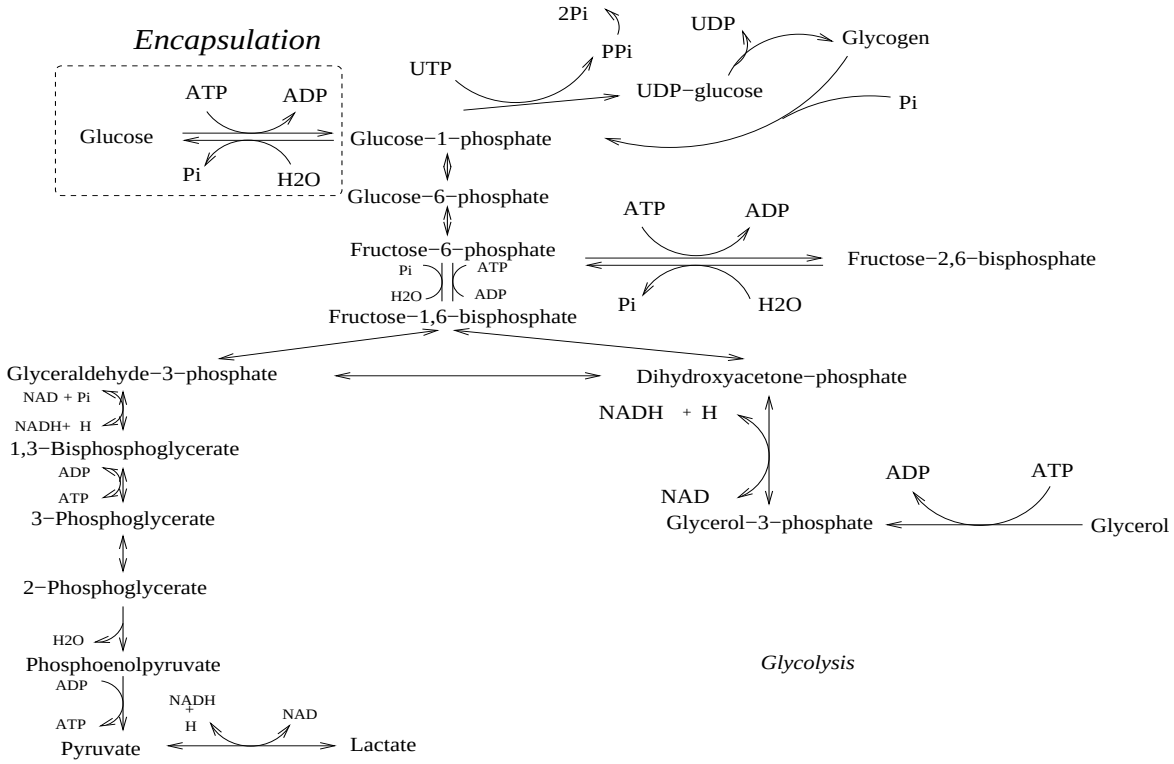


**Figure 1.** The glycolysis pathway. Through a series of enzyme- catalyzed reactions, one molecule of glucose is converted into two molecules of pyruvate, which are then fed into the tricarboxylic acid (TCA) cycle.

a scaled, and possibly offset, version of an SI unit), and even define base units that have no simple relation to SI units (such as pH). Thus, model authors may work with their units of choice while ensuring that their models can still be integrated with models that use other units.

The following CellML fragment demonstrates how a complex unit can be built from SI units:

```
<units name="millimolar">
  <unit prefix="milli" units="mole"/>
  <unit units="liter" exponent="-1"/>
</units>
```

### 2.5 Groups

Grouping adds structure to a model by defining named relationships between components. Two types of grouping relationships are predefined by CellML: *encapsulation* and *containment*. In addition, if required, modellers may define their own types of relationships between components.

Encapsulation simplifies the model structure by restricting variable exchange between particular sets of components. In effect, encapsulation hides a group of components from the rest of the model. A single component is used as an interface through which all variables' values exchanged between the hidden subnetwork and the rest of the model components have to be passed.

The components to which any given component can connect can be divided into four different sets:

1. The set of components encapsulated by the current component is referred to as the *encapsulated set*.
2. If the current component is encapsulated, then the encapsulating component is referred to as the *parent*.
3. All the other components encapsulated by the same parent are known as the *sibling set*.
4. All other components in the model, which cannot be directly connected to the current component, are called the *hidden set*.

Using the glycolysis pathway model as an example, the two components representing the reactions between `glucose` and `glucose-1-phosphate` (`reaction5` and `reaction6`) are encapsulated within the `glucose-1-phosphate` component. They make up the *encapsulated set* of their *parent* `glucose-1-phosphate` component. Relative to each other, the two encapsulated components represent a *sibling set*. The `glucose-`

1-phosphate itself is not encapsulated and has no parent. Its sibling set consists of all the other components in the model. The encapsulated components are unable to make direct connections with components in their parent's sibling set, the *hidden set*; instead, variables coming from (or going to) the hidden set must be mapped through the parent glucose-1-phosphate:

```
<group>
<relationship_ref
    relationship="encapsulation"/>
  <component_ref
    component="Glucose_1_phosphate">
  <component_ref component="reaction5"/>
  <component_ref component="reaction6"/>
  </component_ref>
</group>
```

When variable values are mapped between components, each variable must have an interface. Encapsulation introduces the need for two different types of interfaces: *public* and *private*. A public interface refers to the variable exchange interface exposed to components in the parent and sibling sets. A value of *out* or *in* implies the direction in the exchange of the variable value (i.e., a variable interface value of out denotes value export from the component, while an interface value of in represents variable value import into the current component). A private interface refers to the interface exposed to components in the encapsulated set. Like the public interface, it can have a value of either out or in.

In the following CellML fragment, the change in concentration of inorganic phosphate, Pi, is declared and defined in the encapsulated reaction5 component and passed out through a public interface of out to the parent Glucose_1_phosphate, where it is received through a private interface of in:

```
<component name="Glucose_1_phosphate">
  <variable name="delta_Pi_rxn5"
private_interface="in" units ="flux" />
  ...
</component>
<component name="reaction5">
  <variable name="delta_Pi_rxn5"
public_interface="out" units="flux"/>
  ...
</component>
```

The containment relationship is used to describe the physical organization of a model. That is, the containment relationship allows a modeller to specify that one component is physically nested inside another one. For example, a channel component can be physically embedded within a membrane component by specifying a containment relationship, as shown in the CellML fragment below. This information may then be used by rendering software to create a visual diagram of the model:

```
<group>
<relationship_ref
   relationship="containment"/>
   <component_ref
    component="membrane">
    <component_ref component="channel"/>
  <component_ref component="pump"/>
</component_ref>
</group>
```

### 2.6 Metadata

*Metadata* are "data about data." Metadata are included in CellML to provide context for models and to facilitate searches of collections of models and model components. They provide a means for modellers to include structured descriptive information about their models, which can help other modellers determine whether they can incorporate the model into their own work. Metadata are defined in a separate CellML specification [7] and include constructs such as model author, literature reference, copyright, model creation date, and various elements intended to place the model into a meaningful biological context.

CellML Metadata are composed of a combination of several existing metadata standards. The syntax developed for embedding metadata within CellML documents is based on the RDF [5] specification developed and maintained by the World Wide Web Consortium (W3C). RDF consists of sets of triples: a *resource* has a *property* of some *value*. For instance, a CellML model, the resource, may have a model author property with a person as the property value. The property value may itself be another resource described by properties. In this way, one can break down large chunks of contextual information for CellML models.

The information stored within the RDF metadata framework is based on the Dublin Core, vCard, and bibliographic query service (BQS) public standards. In addition, the CellML Metadata specification defines several attributes that cover the biological categories that the above standards do not.

The Dublin Core Metadata Initiative (DCMI) group, composed of members from the library science and knowledge management communities, identified 15 "common" properties across a large range of resources. The Dublin Core Metadata Element Set [12, 13] includes title, creator, subject, date, format, and language properties. The DCMI has also recognized two classes of qualifiers [14, 15] for the metadata elements, *element refinement* and *encoding scheme* qualifiers. The element refinement qualifiers restrict the scope of the element. Valid refinement qualifiers for the date element, for example, are "created, valid, available, issued, and modified." Encoding scheme qualifiers indicate how the content of the element is encoded. For instance, if a date element has an encoding scheme of W3C-DTF, the content has a format of YYYY-MM-DD.

The vCard data model [16] has several elements necessary to reference people. These elements include name constructs; contact information in the form of mailing addresses, e-mail addresses, and telephone numbers; and affiliation information.

No bibliographic standards currently exist within RDF/XML. The CellML Metadata specification proposes an RDF serialization of the **DsLSRBibObjects** module defined in the Object Management Group's BQS specification [17]. The RDF serialization of the BQS data model makes use of the Dublin Core Metadata Element Set and vCard elements wherever possible, and it includes methods to reference journal articles, books, and Web resources and to cite databases of references such as PubMed. For example, the XML fragment below shows how publication information can be included in CellML Metadata. The `rdf`, `bqs`, `dc`, `dcterms`, and `vCard` prefixes indicate that elements are in the RDF, BQS, Dublin Core, Dublin Core Qualifier, and vCard namespaces, respectively:

```
<bqs:Book rdf:parseType="Resource">
  <dc:creator
      rdf:parseType="Resource">
    <bqs:Person
        rdf:parseType="Resource">
      <vCard:N
         rdf:parseType="Resource">
        <vCard:Family>
         Bronk
        </vCard:Family>
        <vCard:Given>J</vCard:Given>
        <vCard:Other>
         Ramsey
         </vCard:Other>
        </vCard:N>
      </bqs:Person>
    </dc:creator>
    <dcterms:issued
        rdf:parseType="Resource>
      <dcterms:W3CDTF>
         1999
      </dcterms:W3CDTF>
    </dcterms:issued>
    <dc:title>Human Metabolism
    </dc:title>
</bqs:Book>
```

In addition, biological metadata are necessary to describe the models encoded in CellML. The CellML Metadata specification includes several newly defined elements for this purpose, such as species, sex, and biological entity.

Modellers are free to define additional metadata within their own RDF schema. However, CellML processing software is not required to recognize any metadata other than that defined in the CellML Metadata specification.

Metadata are optional in a CellML document. A model without metadata is a valid CellML model. However, the CellML specification strongly recommends that the modeller provide as much metadata as possible, particularly his or her name and contact information and a reference for a paper that describes the development of the model.

## 3. CellML 1.1: Import and Reuse

All of the features discussed so far are common to both CellML versions 1.0 and 1.1. However, the recently developed CellML 1.1 has the additional feature of allowing unit, variable, and component exchanges between different models.

A repository of more than 200 electrophysiological, mechanical, signal transduction, and metabolic pathway models suggests that CellML 1.0 [18] is sufficiently flexible to describe a variety of biological models. However, one limitation of CellML 1.0 is the isolation of the models: each model must contain all components in a single file. As models become larger and more complex, requiring all the information in a single document becomes inconvenient and unwieldy. When dealing with a complex system, it is essential to break it down into manageable pieces, enabling each piece to be considered independently.

CellML 1.1 [6] provides additional features to take advantage of the modular structure of CellML 1.0 and provide authors with the ability to reuse parts of other models. A model in CellML 1.1 may import components and units from other models. The <`cellml:import`> element is used to locate the model that will be imported. The CellML import feature makes use of the W3C hyperlink standard, XLink [4], to identify the link between the current model and the model being imported. Each <`cellml:import`> element must have an `xlink:href` attribute. The `xlink:href` attribute has a value equal to the uniform resource identifier that identifies the location of the imported cell model.

Imported units and components are treated as templates from which instances are created in the importing model. Each <`cellml:import`> element contains a list of units and components that the model author would like to make available to the importing model. Each <`cellml:units`> element inside a <`cellml:import`> element has two attributes: `units_ref`, which locates the units' definition in the original model, and `name`, which reassigns the name by which the units will be referred in the current model. When imported units are used within an importing model, a processor is expected to resolve the units from the original model so that units that depend on other user-defined units can still be understood.

The <`cellml:component`> element inside a <`cellml:import`> element also has two attributes: `component_ref`, which identifies the component from the original model, and `name`, which gives a unique name by which the component will be referred in the current model. Each component in the import list represents an instance of its *subtree*. The component subtree consists of the

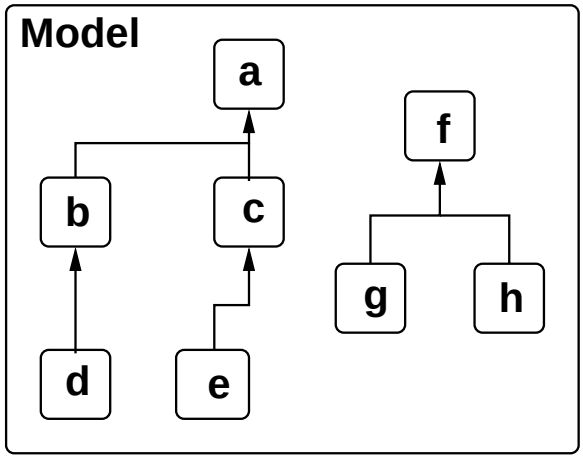


**Figure 2.** Encapsulation hierarchy

component, its encapsulated subset, any components that are encapsulated by the encapsulated subset, and so on. Figure 2, for example, shows a model that contains eight components. The arrows between the components represent the encapsulation hierarchy: **d** is encapsulated by **b**, **b** is encapsulated by **a**, and **g** and **h** are encapsulated by **f**. Component **a**'s subtree consists of components **a**, **b**, **c**, **d**, and **e**. Therefore, if the imported component encapsulates and depends on other components in the original model, the importing model can still get the full meaning from the hidden complexity of the original encapsulation hierarchy.

All connections between the components listed in the <cellml:import> element are maintained in the importing model. If, for some reason, a model author wishes to make use of several components from a single model individually, the model may be identified in several <cellml:import> elements so that components are imported separately.

CellML 1.1 is best illustrated by use of an example. Metabolism is a highly integrated process: individual metabolic pathways are connected with complex networks through common, shared intermediate substrates. An example of this is in the pathway of carbohydrate metabolism. Pyruvate is a product of the glycolytic pathway, which gets fed into the tricarboxylic acid (TCA) cycle [9] (see Figs. 1 and 3).

In CellML 1.1, the model of the TCA cycle can import the parts of the glycolysis model that a processor would need in order to calculate the initial concentration of pyruvate. Since the pyruvate concentration in the glycolysis model depends on the rest of the glycolysis pathway, the list of components that the TCA cycle model needs to import consists of every component in the glycolysis model (except reaction5 and reaction6, described in Section 2.5, which are encapsulated by Glucose_1_phosphate). The pyruvate component from the glycolysis model is renamed initial_pyruvate in the TCA cycle model. The TCA cycle model also defines a new component representing the pyruvate species. The initial value of the concentration is passed in from the imported component. The full description of the TCA cycle is available in the examples section of the *cellml.org* Web site. A fragment of the CellML code is shown below (the xlink prefix indicates that the attributes are located in the XLink namespace):

```
<import
    xlink:href="glycolysis_model.xml">
  <units
      units_ref="micromolar"
      name="uM"/>
  <component>
      component_ref="glycogen"
      name="glycogen"/>
  <component
      component_ref="pyruvate"
      name="initial_pyruvate"/>
<component
      component_ref="reaction0"
      name="reaction0"/>
  ...
</import>
<component name="pyruvate>
  <variable
      name="pyruvate"
      public_interface="out"
      initial_value="pyruvate_init"
      units="uM"/>
  <variable
      name="pyruvate_init"
      public_interface="in"
      units="uM"/>
  ...
</component>
<connection>
  <map_components
      component_1="pyruvate"
      component_2="initial_pyruvate"/>
  <map_variables
      variable_1="pyruvate_init"
      variable_2="pyruvate"/>
</connection>
```

In addition to providing a mechanism with which metabolic pathway models can be connected into a network, the model import feature of CellML 1.1 is useful in allowing mathematical modellers to build more complex models upon previously published, simpler models, without having to include all the components of the original model.

New models often incorporate components of previously published models. This is illustrated by the example of the Sachse et al. [19] model of stretch-activated ion

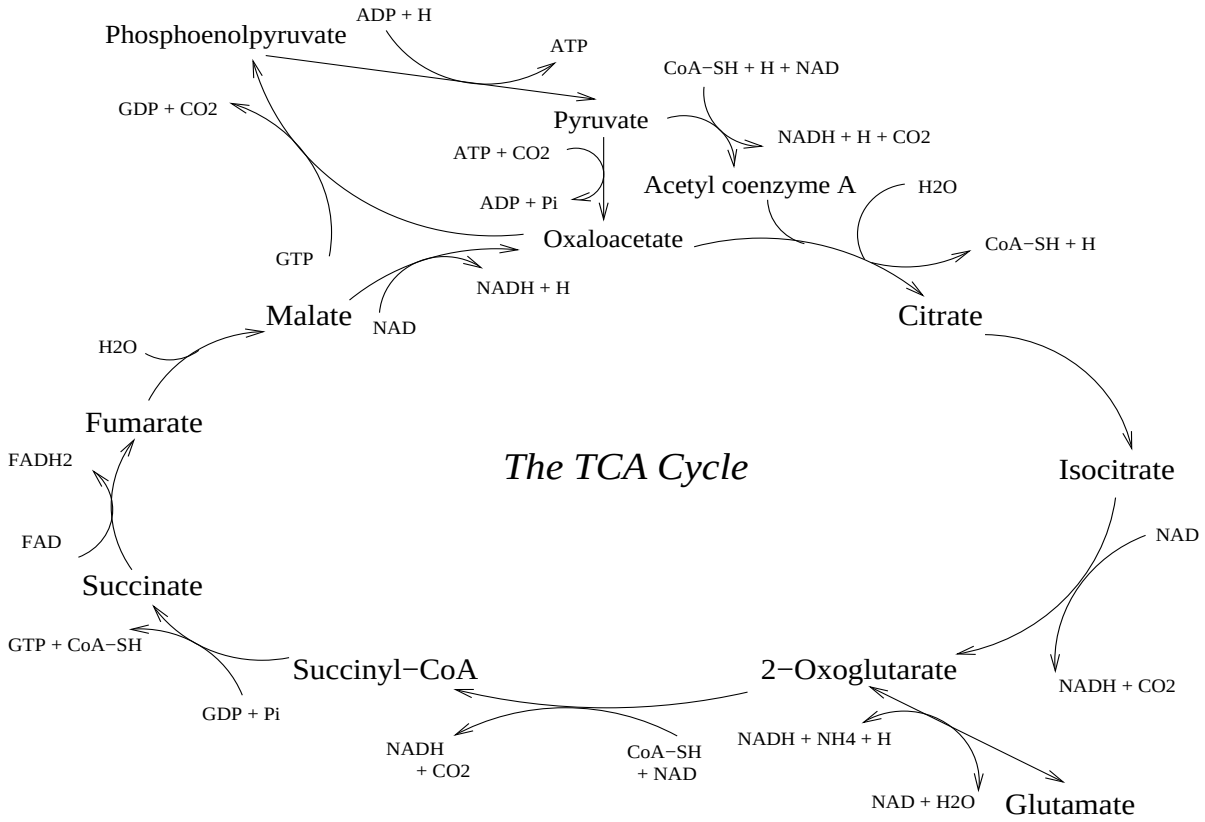


**Figure 3.** The tricarboxylic acid (TCA) cycle

channels in ventricular myocytes. This model uses the previously published Noble et al. [20] mathematical model as a framework in which the new stretch current calculated by the Sachse et al. model is implemented. This example is ideal for illustrating the use of CellML version 1.1, and a full description can be found on the CellML Web site.

Another example in which the model reuse and import features of CellML 1.1 could be applied is if a mathematical biologist is modelling interactions between multiple cells of the same type but with slightly different parameter specifications. Apart from their initial values, the models for each cell would remain the same. Rather than developing a new model for every cell, the same basic model could be imported but with different initial values specified.

## 4. Interfacing to Simulation Packages

CellML is a static description language and makes a distinction between the mathematical representation of a model and the implementation of the model. However, a model is of limited use to a biologist if it cannot be run. To facilitate reading and writing CellML models from and to simulation applications, a CellML application programming interface (API) has been written, based on the W3C DOM standard (www.w3c.org/DOM) and the W3C MathML DOM [21]. The CellML API and its C++ implementation, which are available at cellml.sourceforge.net, currently work with the subset of CellML 1.0 relevant to electrophysiological and mechanical models but will shortly be expanded to cover the entire CellML 1.1 language.

The C++ implementation of the CellML API is used to incorporate CellML into CMISS (www.cmiss.org), a multiplatform interactive computer program for continuum mechanics, image analysis, signal processing, and system identification. The CMISS application is now capable of importing individual cell models described by CellML and using the models in tissue and whole-organ simulations [22].

There are simulation packages that do not make use of the CellML API but do import and run CellML. These include Physiome Sciences' PathwayPrism software and the National Resource for Cell Analysis and Modelling's (NRCAM's) remote user modelling and simulation environment, Virtual Cell (www.nrcam.uchc.edu).

## 5. Conclusion

CellML 1.1 can be used in conjunction with CellML Metadata to provide a complete description of a wide class of biological models. With XML syntax as the basis for the language, CellML ensures that the model can be easily exchanged between applications. The component-based architecture and import features of CellML 1.1 allow a modeller to describe complex models. Furthermore, the open-source C++ implementation of the CellML API is available, allowing biologists to reap the benefits of using CellML with the simulation package of their choice.

One limitation of CellML is the shortage of tools available that write and edit CellML. Current work is focusing on the development of such tools. Future development of CellML will include the addition of typing mechanisms to link ontologies into model descriptions.

## 6. References


[1] Bray, T., J. Paoli, C. M. Sperberg-McQueen, and E. Maler. 2000. Extensible Markup Language (XML) 1.0. 2d ed. W3C recommendation, 6 October [Online]. Available: http://www.w3.org/TR/REC-xml

[2] Hucka, M., A. Finney, H. M. Sauro, H. Bolouri, et al. 2003. The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics* 19:524-31.

[3] Ausbrooks, R., S. Buswell, S. Dalmas, S. Devitt, A. Diaz, R. Hunter, B. Smith, N. Soiffer, R. Sutor, and S. Watt. 2001. Mathematical Markup Language (MathML) version 2.0. W3C recommendation, 21 February [Online]. Available: http://www.w3.org/TR/MathML2/

[4] DeRose, S., E. Maler, and D. Orchard. 2001. XML Linking Language (XLink) version 1.0. W3C recommendation, 27 June [Online]. Available: http://www.w3.org/TR/xlink/

[5] Lassila, O., and R. R. Swick. 1999. Resource Description Framework (RDF) model &syntax specification. W3C recommendation, 22 February [Online]. Available: http://www.w3.org/TR/1999/REC-rdf-syntax-1999

[6] Cuellar, A. A., P. Nielsen, M. Halstead, D. Bullivant, D. Nickerson, W. Hedley, M. Nelson, and C. Lloyd. 2003. CellML specification 1.1. Draft 30, September [Online]. Available: http://www.cellml.org/public/specification/20030930/cellml_specification.html

[7] Cuellar, A. A., M. Nelson, W. Hedley, D. Bullivant, F. Livingston, C. Lloyd, and P. Nielsen. 2002. CellML Metadata 1.0 specification. Working draft, 16 January [Online]. Available: http://www.cellml.org/public/metadata/cellml_metadata_specification.html

[8] Hedley, W. J., M. R. Nelson, D. P. Bullivant, and P. F. Nielsen. 2001. A short introduction to CellML. *Philosophical Transactions of the Royal Society, Series A* 359:1073-89.

[9] Bronk, J. R. 1999. *Human metabolism: Functional diversity and integration*. Harlow, UK: Addison Wesley Longman.

[10] Bureau International des Poids et Mesures. 1998. The International System of Units (SI) [Online]. Available: http://www.bipm.fr/pdf/si-brochure.pdf

[11] Bureau International des Poids et Mesures. 2000. The International System of Units. Supplement 2000: addenda and corrigenda to the 7th ed. (1998) [Online]. Available: http://www.bipm.fr/pdf/si-supplement2000.pdf

[12] Dublin Core Metadata Initiative. 1999. Dublin Core Metadata Element Set, version 1.1: Reference description [Online]. Available: http://purl.org/dc/documents/rec-dces-19990702.htm

[13] Beckett, D., E. Miller, and D. Brickley. 2001. Expressing Simple Dublin Core in RDF/XML. DCMI proposed recommendation, 28 November [Online]. Available: http://dublincore.org/documents/2001/11/28/dcmes-xml/

[14] Dublin Core Metadata Initiative. 2000. Dublin Core qualifiers [Online]. Available: http://purl.org/dc/documents/rec/dcmes-qualifiers-20000711.htm

[15] Kokkelink, S., and R. Schwanzl. 2001. Expressing qualified Dublin Core in RDF/XML. DCMI proposed recommendation, 30 November [Online]. Available: http://dublincore.org/documents/2001/11/30/dcq-rdf-xml/

[16] Iannella, R. 2001. Representing vCard objects in RDF/XML. W3C note, 22 February [Online]. Available: http://www.w3.org/TR/vcard-rdf

[17] Object Management Group. 2001. Bibliographic query service specification [Online]. Available: http://cgi.omg.org/docs/formal/02-05-03.pdf

[18] Hedley, W., M. Nelson, D. Bullivant, A. A. Cuellar, Y. Ge, M. Grehlinger, K. Jim, S. Lett, D. Nickerson, P. Nielsen, and H. Yu. 2001. CellML 1.0 specification. Recommendation, 10 August [Online]. Available: http://www.cellml.org/public/specification/20010810/cellml_specification.html

[19] Sachse, F. B., C. Riedel, G. Seemann, and C. D. Werner. 2001. Stretch activated ion channels in myocytes: Parameter estimation, simulations and phenomena. In *Proceedings of the 23rd Annual International Conference, IEEE Engineering in Medicine and Biology Society*, pp. 52-5.

[20] Noble, D., A. Varghese, P. Kohl, and P. Noble. 1998. Improved guinea-pig ventricular cell model incorporating a diadic space, IKr and IKs, and length- and tension-dependent processes. *Canadian Journal of Cardiology* 14:123-34.

[21] World Wide Web Consortium (W3C). 2001. Document object model for MathML [Online]. Available: http://www.w3.org/Math/DOM/mathml2/appendixd.html

[22] Nickerson, D. P., and P. J. Hunter. 2003. Modelling cardiac electromechanics: From CellML to the whole heart. In *Proceedings of the 5th IFAC Symposium on Modelling and Control in Biomedical Systems*, pp. 47-50.



***Autumn A. Cuellar*** *is a research fellow at The Bioengineering Institute, University of Auckland, Auckland, New Zealand.*

***Catherine M. Lloyd*** *is a research fellow at The Bioengineering Institute, University of Auckland, Auckland, New Zealand.*

***Poul F. Nielsen*** *is a senior lecturer at The Bioengineering Institute, University of Auckland, Auckland, New Zealand.*

***David P. Bullivant*** *is a software development engineer at The Bioengineering Institute, University of Auckland, Auckland, New Zealand.*

***David P. Nickerson*** *is a PhD student at The Bioengineering Institute, University of Auckland, Auckland, New Zealand.*

***Peter J. Hunter*** *is Director of the Bioengineering Institute, University of Auckland, Auckland, New Zealand.*