J. Physiol. Sci. Vol. 58, No. 7; Dec. 2008; pp. 447–458
Online Nov. 6, 2008; **doi:10.2170/physiolsci.RP013308**

Specifications of *insilico*ML 1.0: A Multilevel Biophysical Model Description Language

Yoshiyuki ASAI[1], Yasuyuki SUZUKI[2], Yoshiyuki KIDO[1], Hideki OKA[3], Eric HEIEN[4], Masao NAKANISHI[2], Takahito URAI[5], Kenichi HAGIHARA[1,4], Yoshihisa KURACHI[1,6], and Taishin NOMURA[1,2]

[1]The Center for Advanced Medical Engineering and Informatics, Osaka University, Japan; [2]Department of Mechanical Science and Bioengineering, Graduate School of Engineering Science, Osaka University, Japan; [3]Fujitsu Limited, Japan; [4]Department of Computer Science, Graduate School of Information Science and Technology, Osaka University, Japan; [5]Intasect Communications, Japan; and [6]Department of Pharmacology, Graduate School of Medicine, Osaka University, Japan

Abstract: An extensible markup language format, *insilico*ML (ISML), version 0.1, describing multilevel biophysical models has been developed and is available in the public domain. ISML is fully compatible with CellML 1.0, a model description standard developed by the IUPS Physiome Project, for enhancing knowledge integration and model sharing. This article illustrates the new specifications of ISML 1.0 that largely extend the capability of ISML 0.1. ISML 1.0 can describe various types of mathematical models, including ordinary/partial differential/difference equations representing the dynamics of physiological functions and the geometry of living organisms underlying the functions. ISML 1.0 describes a model using a set of functional elements (modules), each of which can specify mathematical expressions of the module functions. Structural and logical relationships between any two modules are specified by edges, which allow modular, hierarchical, and/or network representations of the model. The role of edge relationships is enriched by key words in order for use in constructing a physiological ontology. The ontology is further improved by the traceability of history of the model's development and by linking between different ISML models stored in the model's database using meta-information. ISML 1.0 is designed to operate with a model database and integrated environments for model development and simulations for knowledge integration and discovery.

Key words: model descriptive language, multiscale modeling, model sharing, physiome.

Physiome and systems biology are emerging research fields aimed at integrating vast stores of knowledge on human physiological and pathological functions at multiple levels and scales in time and space, from molecules and cells to individual organisms [1, 2]. Mathematical models, in particular dynamic system models of physiological functions, play a key role for integration, since they are capable of describing time evolution of biological system states quantitatively based on physical and chemical principles or phenomenological logic governing system behavior. However, the number of mathematical models of biological functions published in peer-reviewed journals and the complexity of each of these models rapidly increase as computational performance increases. This raises difficulties in reproducing simulated behaviors of the published models and in reuse of the models by third parties, thus hindering the promotion of sciences and knowledge integration.

Systems biology markup language (SBML) [3] and CellML [4] have promoted pioneering efforts to overcome this problem. These are extensible markup language (XML)-based formats aimed at describing mathematical models of biological functions, such as gene expression/regulation and electrical activities of cell membranes. Models written in SBML and CellML formats can be downloaded from their model repositories. They include all information necessary to reproduce simulated results described in the corresponding publication about each model. In parallel with these efforts, we have developed *insilico*ML (ISML) 0.1 to describe multilevel biophysical models. ISML 0.1 is available in the public domain through a model database at www.physiome.jp [5]. The format of ISML is defined according to W3C XML specifications [6]. ISML is fully compatible with CellML 1.0 and possesses several features complementary to CellML for enhancing knowledge integration and model sharing.

This article illustrates the new specifications of ISML 1.0, which largely extend the capabilities of ISML 0.1.

Received on Aug 28, 2008; accepted on Nov 1, 2008; released online on Nov 6, 2008; doi:10.2170/physiolsci.RP013308
Correspondence should be addressed to: Taishin Nomura, Department of Mechanical Science and Bioengineering, Graduate School of Engineering Science, Osaka University, Machikaneyama 1-3, Toyonaka, Osaka, 560-8531 Japan. Tel: +81-6-6850-6532, Fax: +81-6-6850-6557, E-mail: taishin@bpe.es.osaka-u.ac.jp

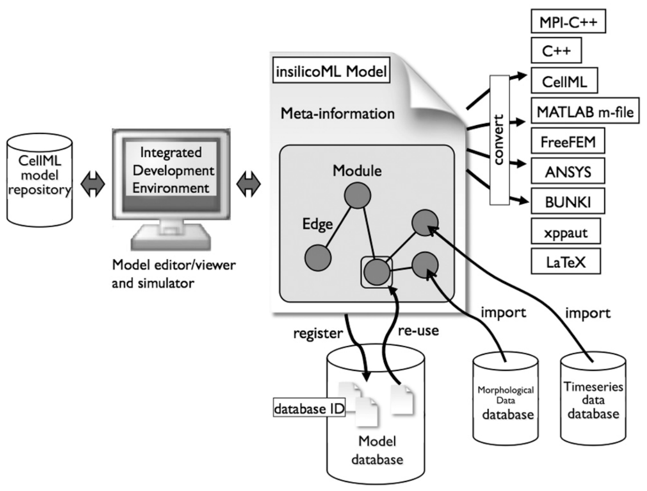


Fig. 1. Typical use of ISML models. ISML cooperates with other model-development and simulation environments such as model databases, databases of morphological and time series data, and other computer languages used for model simulations. ISML can be converted to other formats such as C++, CellML, SBML, xppaut ode file, and LaTeX. ISML models can be parsed, newly constructed, modified, and simulated using integrated development environments. An ISML model is composed of several modules representing functional elements (modules) of the model. The modules are linked to each other by edges that represent structural, logical, and functional relationships among modules. They can include morphological and time series data to describe a model that is characterized by its geometry and prescribed time-depenedent data series.

ISML 1.0 can describe various types of mathematical models, including ordinary differential equations (ODEs), partial differential equations (PDEs), difference equations, and agent-based simulation models that utilize IF-THEN rules, among others. These models represent the dynamics of physiological functions and the geometry, i.e., morphometrics, of living organisms underlying the functions. In ISML, a model is described by a set of functional elements (modules), each of which specifies mathematical expressions of the module functions. In this article we focus on several types of specifications that characterize ISML 1.0 and offer examples of each. They include (1) specifications that allow the user to construct biophysical models with modular, hierarchical, and/or network representations; (2) specifications of how the morphology or geometry of biological entities are incorporated with dynamic system models such as PDEs and agent-based models; (3) specifications to provide a basis for model reuse and physiological model-ontology that support easy model construction, large-scale model construction and simulation, and knowledge discovery from the multilevel complex networks of the physiological functions.

First we illustrate the overall picture of ISML and related tools, applications, and databases before getting into the details of ISML 1.0. As in ISML 0.1, ISML 1.0 is designed to cooperate with model databases and integrated environments for model development and simulations (Fig. 1). The *insilico*IDE (Integrated Development Environment; ISIDE) provided by our physiome project is such an environment, which is currently the most promising tool for the use of ISML. As shown in Fig. 1, a model written in ISML 1.0 format can be constructed using a model development and simulation environment provided by ISIDE. As later described in detail, ISML 1.0 describes a model using a set of functional elements referred to as "modules," each of which can specify mathematical expressions of the module functions. Structural and logical relationships between modules are specified by the "edges," which allow modular, hierarchical, and network representations of the model. Models in ISML are registered in our model database in the public domain for reuse. Another way to obtain ISML models is to convert models from the CellML 1.0 model repository by using ISIDE or other APIs. ISML models that require morphological and/or time series data such as PDEs and agent-based models cooperate with the databases of morphological models and time series data. Since ISML models are written in an XML-based format, they can be easily parsed and converted to other formats, such as CellML, SBML, C++, and MPI-C++, for simulations, i.e., numerical integration with respect to time and space.

For bifurcation analysis of nonlinear dynamical system models, ISML models can be converted to model description formats of existing application software such as xppaut [7] and BUNKI [8]. Several use-cases of ISML 1.0 and future plans for our platform with the use of ISML 1.0 are described and discussed for knowledge integration and discovery at the end of this article.

1. Overview of model representation in ISML

We consider a target biological system as an aggregate of elements referred to as "modules." These modules are characterized by a name, physical quantities representing dynamic or static states with their mathematical implementations. These implementations specify the dynamics of how the states evolve in time and space and also the model geometry, such as morphological model (shape), posture, and position, among others. As illustrated in Fig. 2, modules can functionally affect each other by transmitting values of their physical quantities. The value of a module's physical quantity goes out through an output "port" of the module and is transmitted to an input "port" of other modules. This functional relationship can be defined between any two modules with its direction, and it is specified by its departure (head) and destination (tail) of a functional "edge" linking two modules. The

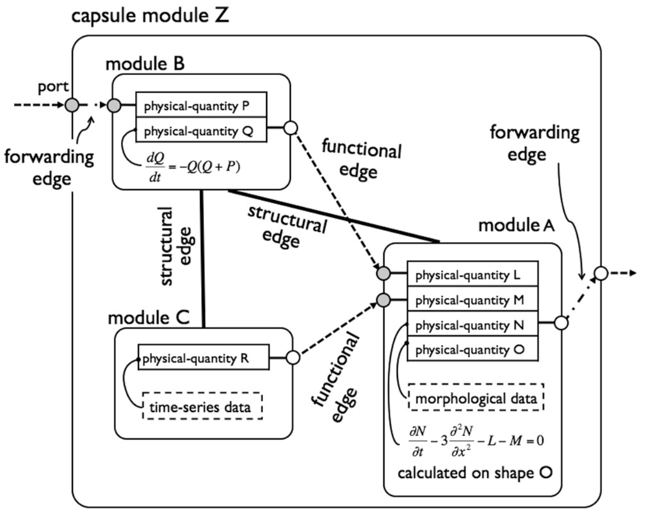


Fig. 2. ISML 1.0 model representation. This figure includes modules, edges linking modules, and the encapsulation of modules. For example, the value of physical-quantity Q in module B goes out through a port and is transmitted to module A as indicated by the edge. The value is input to physical-quantity L through a port and utilized within module A. In this example, modules A, B, and C are encapsulated by a capsule module Z, which means that modules A, B, and C are tightly packed and cannot directly access to or be accessed by modules exterior to capsule module Z. The paths through the ports of capsule module Z are the only ways for interactions to occur between modules inside and outside the capsule module.

edges can also represent structural and logical relationships between two modules. The edges with functional, structural, and logical (also forwarding and capsular) types allow the user to construct biophysical models with modular, hierarchical, and network representations. The modules can also import morphological data (numerical and analytical) and time-series data acquired by experiments in the real world and/or model simulations and can utilize them in the model. With this feature, ISML 1.0 allows the user to combine experimental, theoretical, and model-based research.

Another important idea employed in ISML 1.0 is called "capsulation." A set of modules can be tightly packed by declaring capsulation and contained by a capsule module as a representative of the modules within the capsule (Fig. 2). The capsule module can have input and output ports that provide the sole interfaces for access to and from the modules inside the capsule. Input ports of the capsule module are associated with input ports of encapsulated modules by another type of edge: "forwarding," designated so because of similarity to the concept of port forwarding used in computer networks. Similarly, output ports of modules inside the capsule module may be connected to output ports of the capsule module by "forwarding" edges. A capsule module can be one of the modules encapsulated by another capsule module, which means that the capsulation can nest, leading to a hierarchical representation of a model. A module encapsulation covers all modules within the capsule module at the most interior (lowest) level. In many cases, a capsule module simply symbolizes a certain physiological function and can be easily reused as an element of other models.

ISML 1.0 is compatible with CellML. Let us briefly summarize the similarities between them. Both languages describe physical units for values used in models. A basic element comprising a model is called a "component" in CellML, and its counterpart is the "module" in ISML 1.0. Physical variables called "variables" in CellML can be read as "physical-quantity" in ISML 1.0. A physical-quantity for ISML 1.0 can be specified as one of several types, such as "state," "dynamic-parameter," "static-parameter," and "func-expression," among others. In particular, a major difference from CellML is that the physical-quantity of ISML 1.0 has a type of "morphology" and "time series" to deal with, numerically and analytically defined morphology and time series data acquired by real world biological experiments and/or numerical simulations. The morphology-type physical-quantity in ISML 1.0 is required, for example, in a problem with PDEs that uses morphology to define the shape of domain and boundary of the problem, or in a problem with ODEs that has constraint conditions defining a nontrivial manifold (e.g., equations of motion of a mass constrained on a surface with a complicated shape).

Ports to export/import values to/from modules outside the module under consideration specified in ISML 1.0 have no clear correspondence in CellML. This concept is indirectly expressed in CellML as public-/private-interfaces of variables. The "connections" in CellML are used to express equivalence between different variables and components, and this roughly corresponds to the edges defined in ISML. Relationships indicated by edges and connections contribute to the construction of physiological ontology based on a large set of models. There are several differences between CellML connections and ISML edges. The edges in ISML represent directional relationships among modules, whereas CellML connections are nondirectional. That is, each edge in ISML possesses "head" and "tail" modules to be linked. Moreover, each edge can have an operation-type "meaning" to which several words are assigned to specify how two modules are affected by each other.

In the following sections, we focus mainly on typical ISML 1.0 features that are not found in CellML or others.

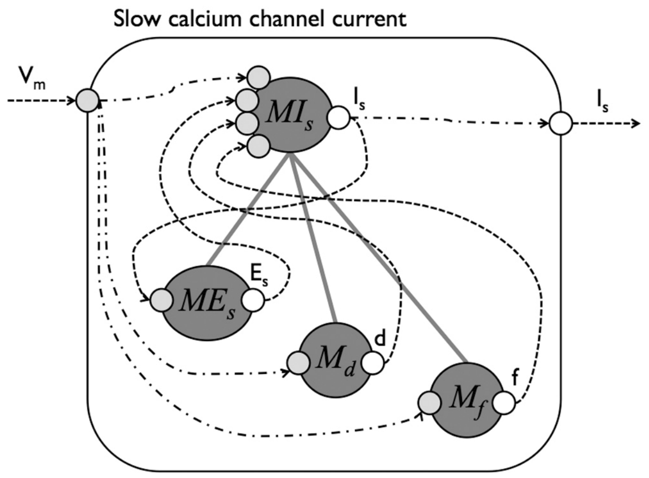


Fig. 3. A schema of the insilico model of the slow calcium channel current proposed in the Beeler-Reuter model. Dark gray circles and a square frame represent functional-unit modules and a capsule module, respectively. Solid, dashed, and chain lines represent structural, functional, and forwarding edges, respectively. Small circles on the circle modules represent ports (light gray, input; white, output). The module MI_s defines the electrical current $i_s = \overline{g}_s \cdot d \cdot f \cdot (V_m - E_s)$ receiving the membrane potential V_m from the outside of the module, a reversal potential E_s from the module ME_s, and two gate variables d and f, respectively, from the modules M_d and M_f. $\overline{g}_s$ is a constant conductance defined in MI_s.

See our Web site http://www.physiome.jp/documents/HTMLs/isml_ver1.html for more details and other specifications that are not described in this article.

2. Expressions used in ISML 1.0

In this text, the term "insilico model," "ISML model," or simply "model" refers to a single model described by an entire ISML document, which includes metainformation, definition of physical units, morphological data, and time series data as well as mathematical expressions representing dynamics of the physical functions under consideration.

2.1. Modularity and linkage among modules

Here we describe the Beeler-Reuter (BR) model [9] using ISML 1.0 to illustrate the modularity and linkage among modules. The BR model can simulate electrical excitation of a cardiac ventricular cell membrane. The dynamics of the membrane potential V are determined by four ionic channel currents, i.e., time-independent potassium outward current, time-activated outward current, fast inward sodium current, and slow inward calcium current, and another current as an external stimulus. The dynamics of each ionic current are determined by the state of the corresponding ion channel and the membrane potential. For example, the slow inward calcium current i_s can be described as

$$i_s = \overline{g}_s \cdot d \cdot f \cdot (V_m - E_s) \quad (1)$$

where V_m is the membrane potential and E_s the reversal potential of i_s; d and f are the gate variables whose dynamics depend on the membrane potential.

Let us consider a model representing the slow calcium channel current i_s designed as in Fig. 3. The module MI_s defining the current i_s is designed with three submodules, which are graphically represented by the three ball-like modules connected by solid lines (representing the structural edges) in Fig. 3. In this case, the variables $d, f,$ and E_s on the right-hand side of Eq. 1 are specified as physical-quantities in the modules M_d, M_f, and ME_s, respectively. To define i_s in the module MI_s, these three physicalquantities are imported through the functional edges indicated by the dashed curves in Fig. 3. V_m is obtained externally via the input port of this model and is used by the modules MI_s, M_d, and M_f as indicated by the chain curves (representing the forwarding edges).

2.1.1. Edge. The edges shown in Fig. 3 are listed in an ISML document in the **edge-set** section. This section specifies the type of each edge, its departure (**head**) and destination (**tail**), and **operation** describing the meaning of its functional relationship with a verb or a verb phrase.

```
<edge-set>
    <edge type="functional" edge-id="CBDFC06C-07D8-...-C30D2AE48D3A">
        <head module-id="E72OOPSA-EE46-...-0FAC08E14333" port-id="3"/>
        <tail module-id="A6CD8604-2F61-...-DBD7C92378E5" port-id="2"/>
        <operation> excitation </operation>
    </edge>
    <edge="structure" -id="OD4JC011A-MSC3-...-CAUE2AE4334A">
        <head module-id="A6CD8604-2F61-...-DBD7C92378E5" port-id="0"/>
        <tail module-id="R12ZQPSA-31D6-...-BVA3LME173E3" port-id="0"/>
        <operation> constitute </operation>
    </edge>
        ⋮
</edge-set>
```

The head and tail of each edge of functional and forwarding types specify both module-id and port-id to uniquely identify a port within the model. In the case of structural, logical, and capsular edges, port-id = 0 is used as a dummy port since these edges define the relationship between modules and do not use ports.

2.1.2. Module. A description of the module MI_s starts with **<module** module-id = "A6CD8604-2F61-...-DBD7C92378E5" type = "functional-unit"/>, where the module-id is specified as a 16-byte universally unique identifier (UUID), which is unique across all insilico models. The type of the module

MI_s is specified as *functional-unit*, meaning that this module possesses particular features such as **physical-quantities** necessary for modeling physiological functions.

ISML 1.0 offers several types of modules other than *functional-unit*. These include *container*, *capsule,* and *template*. A module of *container* type does not possess **physical-quantities** and is supposed to represent a conceptual box to put several modules together, e.g., a *container*-type module with a name "ganglion" may be composed of many modules representing nerve cells. A *capsule*-type module acts as a symbol of a physiological function that is modeled by a set of modules. A capsule module itself thus does not possess any **physical-quantities**, but may have only input/output ports as its interfaces. A *template*-type module is similar to the concept of a class in C++, which can realize an object instantiation. Dynamic construction and destruction of instances of a template module can be managed during the run time of numerical simulations, and rules necessary for instance management can be described in ISML 1.0. This is particularly important for constructing and simulating agent-base models. A **module** is also characterized by the following four children tags: **property**, **port-set**, **physical-quantity-set** and **morphology-set**. The **property** describes basic properties of a module, such as name, keywords, track, and so on. The tags **keywords** and **track** are described in the next section.

2.1.3. Port. Returning to the module MI_s example: it has four input ports and one output port. An ISML document enumerates them in the **port-set** subsection of the **module** section.

```
<port-set>
    <port direction="out" port-id="1">
        <name> I_s </name>
        <reference physical-quantity-id="1"/>
        <description> calcium channel current </description>
    </port>
    <port direction="in" port-id="2">
        <name> V_m </name>
        <description> potential </description>
    </port>
        ⋮
</port-set>
```

In this example only two of the five ports in module MI_s are described for brevity. Each port has a direction (in or out) and is given a port-id as a sequential number unique within a module. A port with direction = "in" is called an input port, and "out" an output port. A **physical-quantity** that goes out through this output port must be specified by its ID (physical-quantity-id) in a child tag **reference** of the port. A value of a **physical-quantity** delivered from an external module through the input port is set as a **physical-quantity** of the module and used in the module. In this case, physical-quantity-id is not specified. Instead, a **physical-quantity** that receives a value from the input port specifies its port-id.

2.1.4. Physical-quantity. In the ISML paragraph described above, the **physical-quantity** with the output port of physical-quantity-id = "1" corresponds to the slow calcium current and can be defined as a *variable-parameter* in the **physical-quantity-set** section of the module as follows

```
<physical-quantity type="variable-parameter" physical-quantity-id="1">
    <name> I_s </name>
    <precision> double </precision>
    <unit unit-id="1"/>
    <dimension type="scalar"/>
    <implementation> ... </implementation>
</physical-quantity>
```

A type and physical-quantity-id must be given first when a **physical-quantity** is defined. In this example, I_s is declared as the *variable-parameter*, which expresses values that vary during simulation, such as a computed value of a static (nondynamic) mathematical function of dynamic variables (state). The values obtained by this module are defined in the **implementation** with mathematical formulae. ISML 1.0 defines other types of **physical-quantity**, i.e. *state*, *variable-parameter*, *func-expression*, *nominal*, *morphology*, and *timeseries*. In the example above, the **precision** is set to *double* (other possible candidates are *int*, *char*, and *bool*). The **unit** specifies a unit-id as a user-defined combination of the fundamental seven base units (i.e., meter, kilo-gram,second, ampere, kelvin, candela, mole), together with prefixes such as kilo-, milli-, and micro-. The **dimension** is set to *scalar* in this example. ISML 1.0 supports vector and matrix values for a **physical-quantity** to use in equations.

The **implementation** section is used to describe concrete contents of the **physical-quantity**, such as values for *variable-* and *static-parameters* and mathematical formulae for *states*. Specific definition is described in a child tag **definition**. For example, the *variable-parameter* I_s is defined as

```
<implementation>
    <definition type="ae" format="mathml">
        <math>
            <apply> <eq/>
                <ci> I_s </ci>
                <apply> <times/>
                    ⋮
                Here is the MathML expression for i_s = ḡ_s · d · f · (V_m − E_s)
                    ⋮
        </math>
    </definition>
</implementation>
```

The contents described in the **definition** are categorized by specifying one of the following types: *ode* (ordinary differential equations), *pde* (partial differential equations), *dde* (delay differential equations), *sde* (stochastic differential equations), *de* (difference equations), *ae* (algebraic equations), *assign*, *func-expression*, and *conditional*. The user can specify arithmetic, algebraic, and statistical expressions including vectors and matrices in equations and inequalities described by MathML in the **definition** tag.

2.1.5. Capsulation. The module MI_s and its submodules are encapsulated in the example above. The capsule itself is represented as a module with the *capsule* type depicted as a square frame in Fig. 3. A module at the top level (referred to as the root) of a local tree structure can be encapsulated by a capsule module if all edges (except forwarding edges) are self-contained. The module MI_s satisfies this condition and can be encapsulated. The capsule module is linked by a capsular edge (which is not illustrated in the figure). Once the root module is encapsulated, input and output ports of the modules under the root module are linked to the input and output ports of the capsule module by edges of the *forwarding* type. Whether or not the module is encapsulated is specified in the **property** tag of the module as

```
<capsulation state="true">
    <capsulated-by module-id="C0EA5AE5-16DD-...-8A5848ACF217"/>
</capsulation>
```

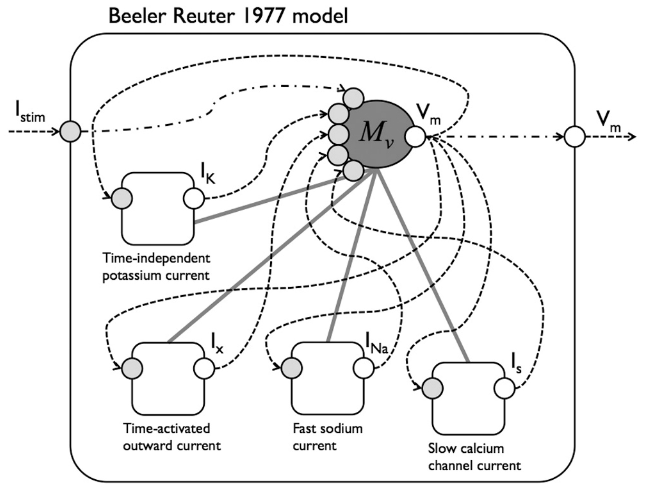
in which the module-id of the capsule module is specified.

Figure 4 shows the whole BR model composed of the module (M_v) modeling the membrane and four encapsulated modules modeling the channel currents (one of them is the slow calcium current module shown in Fig. 3). The BR model is encapsulated by the capsule module named “Beeler Reuter 1977 model,” which has an input port to receive the external stimulus current and an output to export the membrane potential. Using this model, the user can create, for example, coupled BR models by linking several individual BR models.

2.2. Morphological data and mathematical expressions

Let us consider another example, a reaction-diffusion dynamics with the FitzHugh-Nagumo model. This model represents the excitation conduction on a two-dimensional medium as a model of cardiac tissue analyzed by Hall and Glass (1999) [10]. They investigated spatio-temporal changes in the propagating action potential on a two-dimensional sheet paved with excitable cell models. The reaction diffusion equations that they used are as follows:

$$\frac{\partial v}{\partial t} = \frac{1}{\varepsilon}\left(v - \frac{1}{3}v^3 - w\right) + D\nabla^2 v + I_{loc} + I_{stim}(t) \qquad (2)$$



Beeler Reuter 1977 model
I_stim
M_v
V_m
V_m
I_K
Time-independent potassium current
I_x
I_Na
I_s
Time-activated outward current
Fast sodium current
Slow calcium channel current

Fig. 4. A schema of the BR model [9] composed of a membrane and four channel currents, which are modeled using *insilico*ML 1.0 and encapsulated for easy reuse. The capsule module of slow calcium current at the bottom right of “Beeler Reuter 1997 model” is shown in Fig. 3.

$$\frac{\partial w}{\partial t} = \varepsilon\,(v + \beta - \gamma w)g(v) \qquad (3)$$

where $g(v)$ is a sigmoidal function controlling the rate of the pacemaker

$$g(v) = \frac{w_H - w_L}{1 + exp(-kv)} + w_L \qquad (4)$$

v and w are the excitation and recovery variables, respectively. I_{loc} is a constant current applied to a localized region at the center of the sheet. $I_{stim}(t)$ is a pulsatile stimulation current used for resetting applied to selected points at selected timings.

We begin by designing the corresponding insilico model using ISML 1.0 (Fig. 5). The PDE with FitzHugh-Nagumo is modeled here by two modules representing excitation variable v and recovery variable w, respectively, which are encapsulated to consolidate model independence. The modules M_v and M_w are mutually linked by functional edges. The module M_v receives two external electric current stimuli I_{stim} and I_{loc} via the input ports of the capsule module. The module M_{area} provides a morphology of the tissue on which the reaction diffusion problem is solved. Here the module M_{area} is linked by a structural (constituent) edge to the capsule module, which contains the modules representing the current stimulations as well as the PDE of the FitzHugh-Nagumo model, since all of them are solved on the morphology provided by module M_{area}.

2.2.1 Morphology. The module M_{area} is implemented as a *functional-unit* type that has several **physical-quantities**, one of which is the *morphology* type. ISML 1.0 can handle numerically defined morphology, i.e., using morphological data that is an assembly of points (a set of two or three dimensional coordinates), and also morphology

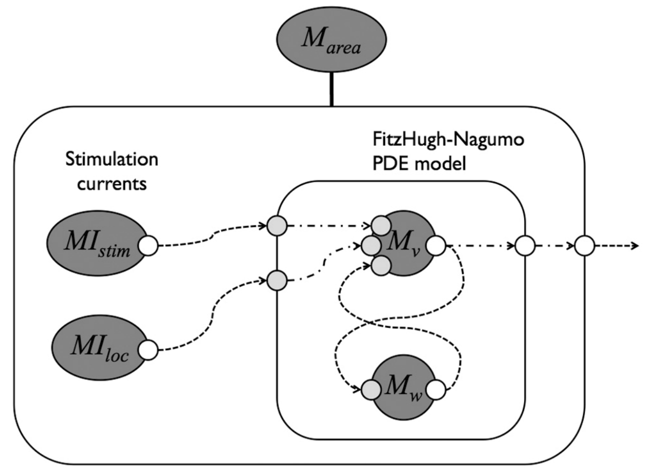


Fig. 5. A schema of the reaction-diffusion PDE with the FitzHugh-Nagumo model receiving two external stimulation currents. See the caption of Fig. 3 for representations.

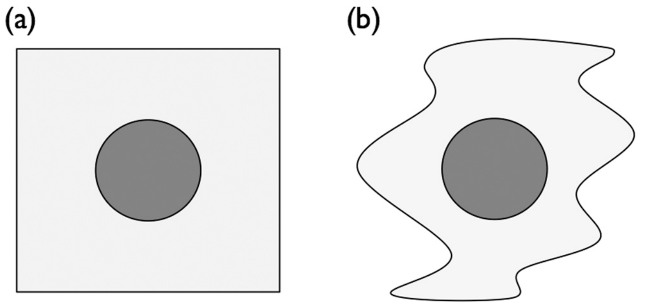


Fig. 6. (**a**) A simple two-dimensional square sheet (tissue) used in the reaction-diffusion system with the FitzHugh-Nagumo model. (**b**) An example of the complicated two-dimensional tissue that may be defined using numerical morphological data.

defined analytically (mathematically). In the former case, the data can be kept as an independent file either on a hard disk at a local machine or in a remote database. For the latter case, mathematical expressions are written in the insilico model. Morphological models used in a module are described in the **morphology-set** section of the module. The simple two-dimensional square sheet used in this example (shown in Fig. 6a) can be described using MathML as follows:

```
<morphology-set>
    <morphology type="internal" morphology-id="1">
        <definition format="mathml">
            <math>
                <apply> <and/>
                    <apply> <geq/>
                        <ci> x </ci>
                        <cn> 10 </cn>
                    </apply>
                        ⋮
                    Here are MathML expressions for
                        −10 ≤ x ≤ 10 & −10 ≤ y ≤ 10
                        ⋮
            </math>
        </definition>
    </morphology>
</morphology-set>
```

A sequential unique ID number within the module is assigned to each fragment defining the morphology listed here. A type of the **morphology** tag specifies whether the morphology is defined within the insilico model file (type = “internal”) or outside as an independent file (“external”). The morphology defined in the module M_{area} can be more complicated, such as shown in Fig. 6b, defined numerically as morphological data. In that case, the description of ISML is as follows:

```
<morphology-set>
    <morphology type="external" morphology-id="1">
        <definition format="numeric" iref="./data/morphology/sample1.wrl"/>
    </morphology>
</morphology-set>
```

Note that if the resource file defining the morphology is at a remote machine, the attribute xref for cross-reference must specify the remote address.

Morphological data used in an insilico model should be cataloged in the **morphology-set** section. Each data set must be bound to a **physical-quantity** with the *morphology* type. The **physical-quantity** with the *morphology* type can be assigned morphological data (numerical data or mathematically defined morphology), and the data may be used to solve PDEs numerically. An example of **implementation** for a *morphology* type **physical-quantity** is as follows:

```
<implementation>
    <definition type="assign" format="morphology">
        <reference morphology-id="1"/>
    </definition>
</implementation>
```

where the morphology-id is specified at **reference** regardless of the type of morphology, i.e., numerical data or mathematical expressions.

ISML 1.0 defines a special type of **physical-quantity** named *shape*, which is one of the ISML-predefined **physical-quantities**, i.e., all modules possess this **physical-quantity** by default. It is a special case of the *morphology* type and can be used for binding morphological data. The **physical-quantity** *shape* represents the shape of a module. The value of this **physical-quantity** can be referenced by any children modules that are linked by structural or logical edges to the module possessing this **physical-quantity** *shape*. In the example (Fig. 5), the modules M_v, M_w, MI_{stim}, and MI_{loc}, which are encapsulated by the outer capsule module, can use the morphological data defined in the module M_{area} through its *shape*. For example, assume that the module M_v has a morphology-type **physical-quantity** with ID = 4. In its definition in the implementation, the **physical-quantity** is assigned morphological data by inheriting the **physical-quantity** *shape* of the parent module M_{area}. This is done by specifying the module-id of the parent module as follows:

```
<implementation>
    <definition type="assign" format="morphology">
        <reference module-id="A6CD8604-2F61-...-DBD7C92378E5"/>
    </definition>
</implementation>
```

Notice that the parent module-id is specified instead of the morphology-id at the **reference** tag.

2.2.2. Partial differential equations. The module M_v includes a PDE as the governing equation of the dynamics of *v*, which is a *state*-type **physical-quantity**. For the *state*-type **physical-quantity**, unlike the *variable-parameter* type, **argument-set**, **domain**, and **problem-condition** must be defined as well. By declaring the argument variables in **argument-set**, it becomes clear that the *state*-type **physical-quantity** used in PDEs is a function of those argument variables. For example, *state*-type **physical-quantity** *v* of the FitzHugh-Nagumo model PDE is a function of time and space (*x* and *y*). The arguments are also used for the *func-expression* type **physical-quantity**, as shown later. The **domain** specifies the domain on which the function or differential equations are evaluated or solved. The following is a section of ISML 1.0 describing **argument-set** and **domain** for our PDE example:

```
<argument-set>
    <argument physical-quantity-id="2"/>
    <argument physical-quantity-id="3"/>
</argument-set>
<domain>
    <definition format="morphology">
        <reference physical-quantity-id="4"/>
    </definition>
</domain>
```

In the example, we are dealing with the reaction-diffusion equations defined on a two-dimensional sheet. The two arguments declared in this ISML fragment correspond to the spatial coordinate variables *x* and *y*, which are independently defined elsewhere as the *nominal*-type **physical-quantities** with ID = 2 and 3. The *nominal*-type **physical-quantity** does not have its **implementation** in the definition, only the specifications of **name**, **unit**, **precision**, and **dimension**. The domain is defined by specifying a morphological **physical-quantity** (ID = 4) assigning morphological data from the parent module as mentioned above. The **domain** can also be defined mathematically in MathML, in which the argument variables define functions.

In the **implementation** of the *state* type **physical-quantity** *v* for example, the PDE is described in MathML format. All terms of the equation are written on the left-hand side of the equation, and the right-hand side is always 0. Only for PDE definition, the **definition** tag has a subtype attribute to specify the type of PDE as one of *elliptic*, *parabolic*, *hyperbolic*, and *others*, and a form attribute specifying if the PDE is represented as *weak*, using variational formulation, or *strong*, using PDE with derivatives.

```
<implementation>
  <definition type="pde" sub-type="parabolic" format="mathml">
    <math>
      <apply><eq/>
        <apply><plus/>
          <apply><partialdiff/>
            <bvar><ci> t </ci></bvar>
            <ci> v </ci>
          </apply>
          <apply><minus/> <apply><times/>
            <ci> D </ci>
            <apply><partialdiff/>
              <bvar><ci> x </ci> <degree><cn> 2 </cn></degree></bvar>
              <ci> v </ci>
            </apply>
          </apply> </apply>
            ⋮
```

Here are MathML expressions for

$$\frac{\partial v}{\partial t} - D\frac{\partial^2 v}{\partial x^2} - D\frac{\partial^2 v}{\partial y^2} - \frac{1}{\varepsilon}\left(v - \frac{1}{3}v^3 - w\right) - I_{loc} - I_{stim}(t) = 0$$

```
            ⋮
    </math>
  </definition>
</implementation>
```

In the example of the FitzHugh-Nagumo model, zero-flux boundary conditions are applied to the four borders of the square area, which is equivalent to the zero Neumann condition at each boundary. These conditions must be described in the model. In general, ISML 1.0 specifies conditions accompanying the equations to be solved, such as boundary conditions, in the **problem-condition** section. The *constraint*, *material*, and *initial* conditions, as well as the *boundary* conditions, can be defined. For definitions of the boundary conditions, the user must specify a type of either *neumann*, *dirichlet*, *robin*, or *other*. If necessary (for example, in the cases of force and flow), the direction can be defined in the **direction** tag by specifying argument variables, such as x or y, or explicitly by a word, either *perpendicular* or *parallel*, indicating the direction with respect to the tangential plane (or line) at every point on the boundary. Notice that *perpendicular* is therefore the direction normal to the boundary. The notation *Neumann*(·) is used to describe the Neumann condition. For example, if the Neumann condition of state v is 0 on a segment of the boundary (specified by a segment-id described later), it is written as $Neumann(v) = 0$, as shown in the example ISML document below:

```
<problem-condition-set>
    <problem-condition type="boundary">
        <configured-at segment-id="2"/>
        <direction> perpendicular </direction>
        <definition type="neumann" format="mathml">
            <math>
                <apply> <eq/>
                    <apply> <ci type="function"> Neumann </ci>
                        <ci > v </ci>
                    </apply>
                    <cn> 0 </cn>
                </apply>
            </math>
        </definition>
    </problem-condition>
</problem-condition-set>
```

Similarly *Dirichlet*(·) is for defining the Dirichlet condition. The user can define other types of conditions using general mathematical formulae.

For constraint conditions, a space (usually a manifold) M_x, in which the target *physical-quantity* x is constrained, is defined in the definition section. M_x can be described either in a mathematical format, such as $\{x|f(x) = 0\}$ with format = "mathml," or by a set of points represented by morphological data for digitized, numerically defined cases with format = "morphology." To specify material conditions, the values of physical material properties (such as Young's modulus), density, and viscosity are set in an equation format.

For those conditions to be complete, it is necessary to specify spatial positions where each condition is applied. Particular segments constituting the morphology, i.e., subsets of points of numerical morphological data or pieces of geometric objects such as an arc or a portion of a rectangle, are defined in the **segment-set** section in the definition of the **morphology** and are labeled by a sequential ID (segment-id) to be specified uniquely within the module. Each segment is also categorized either by *point*, *line*, *area*, or *volume*. An appropriate condition is then applied to each segment specified by its segment-id in the **configured-at** tag. Here is an example of a definition of the segments.

```
<morphology type="external" morphology-id="1">
    <definition iref="./data/morphology/sample1.wrl"/>
    <segment-set>
        <segment segment-id="1" type="line">
            <point no="1"/>
            <point no="3"/>
            <point no="5"/>
        </segment>
    </segment-set>
</morphology>
```

A dummy variable is used to mathematically define segments of a given morphology. For example, geometric figures can be given as $x = \cos(2\pi s)$, $y = \sin(2\pi s)$ where $s \in [0:0.6]$ is a dummy variable. For a morphology defined by numerical data, this can be done by listing points specified by their number.

In the **problem-condition**, conditions such as *mesh*, *solver*, and *post* can be defined. They are related to numerical methods and processing performed during and after simulations of the model. The *mesh* condition specifies the number of nodal points involved in the specified structure, and it is used by a mesh-generator. Mesh-node information is described in a separate file with a certain format. Note that instead of specifying the number of nodes in the structure, it is also possible to import external node files. The *solver* condition specifies a solver used for the finite element method to solve PDE with morphology. ISML 1.0 offers at least six choices: LU, Cholesky, Crout, CG, GMRES, and UMFPACK. Convergence criteria must be specified. The *post* condition is used to define the postprocessors. The postprocessor type determines the file format to which simulation results are saved, such as gnuplot and medit.

In our example of the FitzHugh-Nagumo model, the module MI_{loc} generates the constant current applied to a localized region at the center of the sheet (indicated as a gray disk in Fig. 6, a and b) either to make the media in the corresponding region oscillate or to depress excitability. The current can be described mathematically as a function of spatial coordinates x and y: $I_{loc}(x, y) = C_{low}$ when $x^2 + y^2 \leq r^2$; otherwise $I_{loc}(x, y) = C_{high}$, where C_{low} and C_{high} are constants. Thus the **physical-quantity** transmitted from MI_{loc} to the module M_v is not a scalar value, but a set of values as the function of the space. To describe this sort of **physical-quantity**, the *func-expression* type **physical-quantity** is available. This type does not provide values, but instead a set of mathematical formulae of the function as it is (as text descriptions). The *func-expression* type **physical-quantity** also requires specification of the argument variables for use in the definition of the **imple-**

mentation of the *func-expression*. The definition of the *func-expression* type **physical-quantity** begins with **<physical-quantity** type="func-expression" -quantity-id="1"**>**. It is characterized by **name**, **precision**, **dimension**, **unit**, **argument**, **domain**, and **implementation**. Note that to define the **dimension**, in this case the row and/or col must be specified as *discretization-dependent* (i.e., **<dimension** type="matrix"**> <col>** discretization-dependent **</col> <row>** discretization-dependent **</row> </dimension>**), since the dimension of such a variable cannot be specified until a solver discretizes the domain of the function defined in the *func-expression* for numerical computations. The example code of the **implementation** for module MI_{loc} is as follows:

```
<implementation>
    <definition type="conditional">
        <case-set case-set-id="1">
            <case case-id="1">
                <condition format="mathml">
                    <math>
                      ⋮
                    expression for x² + y² ≤ 4
                      ⋮
                    </math>
                </condition>
                <definition type="func-expression" format="mathml">
                    <math>
                        <apply><eq/>
                            <apply><ci type="function"> I_loc </ci>
                                <ci> x </ci>
                                <ci> y </ci>
                            </apply>
                            <cn > 0 </cn>
                        </apply>
                    </math>
                </definition>
            </case>
            <case case-id="2">
                <definition type="func-expression" format="mathml">
                    <math>
                      ⋮
                     MathML expression for I_loc (x, y) = 10
                     This is the same as the above MathML expression,
                     except that the right-hand side is 10, not 0.
                      ⋮
                    </math>
                </definition>
            </case>
        </case-set>
    </definition>
</implementation>
```

In the **implementation**, the current I_{loc}, which is the function of spatial coordinates x and y, is defined using conditional branches, since the value of I_{loc} depends on the position, for example, IF $x^2 + y^2 \leq r^2$, THEN $I_{loc} = C_{low}$, ELSE $I_{loc} = C_{high}$. The conditional branches are described by the **definition** tag with type = "conditional", which takes **case-set** and **case** tags as children. The condition is described in MathML format, and the definition is evaluated if the condition is satisfied. These are described in the **case** section labeled by case-id. The **case**s in the same **case-set** indicate that the **case**s are in the same sequel, which is similar to the "IF - ELSE : IF - ELSE" syntax in computer languages. The last **case** in the example code above, which has no **condition** section, corresponds to the "ELSE" phrase.

3. Databasing, tracking, and relating models

Models created by the user are stored in a model database (ISML database) if the models satisfy certain criteria, such as publication in peer-reviewed journals. Models in the database are uniquely identified by a database ID (db-id), which is assigned by the database server when the model is registered in it. Users can download models from the database and reuse them to develop new models. ISML 1.0 is capable of tracking any module in terms of the history of its reuse if the module is registered in the database. This is done using the **property** section, where an ISML document of each module records the db-id of the models that involve (reuse) the module. The reuse history of modules can provide another basis for the model-based ontology of physiological functions as well as the metainformation assigned to the modules and the edges of **operation**-type with "meaning" as mentioned above.

Let us take the BR model again as an example for illustrating how the reuse history is recorded in each module. When creating the BR model, we first create the model of the slow inward calcium channel current (Fig. 3) as an element of the BR model. During development of the calcium channel model until it is completed and registered in the ISML model database, the model ISML document has no database ID (db-id). In this period, when a module such as the module MI_s is created in the calcium channel model, module-tracking information, such as "module MI_s is involved in this calcium channel model," is written as a property of the module MI_s as follows:

```
<track>
    <involved db-id="this" date="2007-06-06"/>
</track>
```

where "this" corresponds to the calcium channel model. This indicates that the module MI_s is used as an element of the calcium channel model. When the development of the calcium channel model is completed and it is uploaded to the ISML database, the database server assigns a unique db-id to the calcium channel model. More precisely, the db-id is given to the capsule module encapsulating the model of the calcium channel. Let us assume that the server has given database ID 41 to the calcium current model. Then the value "this" in the **track** tag for the module MI_s is replaced by the newly assigned db-id at the server, and the tracking record is rewritten as **<involved** db-id="41" date="2008-06-06"/>.

In the subsequent development process of the entire BR model (Fig. 4), let us assume that the slow calcium current model has already been registered in the database and that we can download it from the database for reuse as one of the modules composing the BR model. When the calcium channel model is reused in the BR model, a new **track** record is added to the module MI_s, which has been a submodule of the calcium channel model as follows:

```
<track>
    <involved db-id="this" date="2008-03-22"/>
    <involved db-id="41" date="2007-06-06"/>
</track>
```

In this way, the reuse history of the module MI_s is accumulated and logged.

The Luo-Rudy (LR) model [11] is another representative model describing the action potential generation of the ventricular cardiac myocyte. The LR model can be considered as an extended or modified version of the BR model. Luo and Rudy modified the model by adding several new modules, including representations of extracellular ion concentrations, a novel potassium channel, and a plateau potassium current. The slow calcium current model was used in the LR model the same way as formulated in the BR model. Let us consider ISML model creation according to this actual history of the model development. That is, we reuse the slow calcium channel model of the BR model for the creation of the LR model using ISML 1.0. Upon registration of the newly created LR model, tracking for module MI_s is revised as

```
<track>
    <involved db-id="131" date="2008-08-14"/>
    <involved db-id="75" date="2008-03-22"/>
    <involved db-id="41" date="2007-06-06"/>
</track>
```

where ISML models with bd-id = 41, 75, and 131 correspond to the slow calcium channel model, the BR model, and the LR model, respectively. Once the module is created and the model including that module is registered, this history track is maintained as long as the module exists even if the module is modified (deletion or edit of some part of the module). Because of this, modifications made to the module cannot be identified clearly from the **track** records, though one can try to compare the module after repeated modifications with the original module, which is kept unchanged in the database. Nevertheless, because it is expected that the essential nature of the module will remain even after modification, it is worthwhile to provide the history-tracking capability.

Based on the tracking information recorded in each module, the origin of each module involved in a model can be revealed, and, with appropriate analytical tools for ISML documents on the database, the user can obtain a tree diagram of the module (model) development for an analysis of module phylogeny. In our simple example, the user can discover that the BR model and the LR model use the same or similar slow calcium channel model. Moreover, the track information enables the user to define distance or similarity between two different models. The larger the number of common modules shared in the two models, the smaller the distance between the models, leading to a model-based construction of the physiological ontology.

4. Discussion

This article illustrated the progress of the development in the description language *insilico*ML (ISML) 1.0 for multiscale and multilevel models of physiological functions. ISML 1.0 and integrated development environment tools, such as the ISML editor, model viewers, and simulators, are expected to be a pillar of promotion of physiome and systems biology by which researchers in the interdisciplinary fields among physiology, engineering, medicine, and informatics can make concerted efforts toward quantitative and integrated understanding of the human physiology [1]. The key concept here is to describe the physiological functions using common languages that can be shared worldwide.

The development of ISML 1.0 aims to enhance model sharing among researchers, both experimental and theoretical, by cooperating with other major model description languages (including CellML and SBML). Individual researchers constructing and simulating physiological models often use different languages and different environments, making it difficult to exchange models and ideas. ISML 1.0 and its tools will provide a common platform on which many researchers can cooperate in modeling, simulating, and analyzing their own models of particular physiological functions.

ISML 1.0 is designed to describe in a consistent manner various physiological phenomena at any spatiotemporal scale, such as the dynamics of ion movement, proteins, cells, tissues, organs, and the whole body along with relevant morphology. If unrelated researchers develop models for different targets or at different levels using a common framework such as ISML 1.0, those models can easily be combined to construct models with multiple scales and levels. Moreover, users can easily examine changes in the model dynamics when particular parts of the model are replaced by other models with the aid of the encapsulation used in ISML.

An important aspect provided by the development of ISML 1.0 and its database is a novel basis necessary for construction of an ontology of the physiological functions. Our idea proposed in this article for the physiological ontology has its basis in several types of relationships between mathematical models of the physiological functions. There have been several developed ontologies, such as the gene ontology and the protein ontology. However, construction of the ontology for physiological functions may not be straightforward, since the subjects have much wider varieties of nomenclature than for genomes or proteomes. If a physiological function is described using the ISML model, it can be decomposed into fundamental elements based on the modules included in the model. Each module can represent a noun by a name of the specific physiological function modeled by the module. Functional relationships between modules are represented by the edges in ISML, and each of them can describe a meaning of the relationship between any related modules in the ISML model. The functional relationship can correspond to a verb, i.e., how the state of one module can affect the state of another module through the functional relationship. A large number of nouns and verbs stored in the ISML database contribute to constructing the ontology. The history of module reuse is used to define similarities and differences between models, and can also contribute to the construction of the ontology. By quantifying those relationships, the ontology proposed in this study can determine what physiological functions exist as the elements of the ontology, what relationships are formed among them, and how these elements can be categorized according to the similarities and differences between models. Such ontology can support research activities in various ways, including (1) model search from a large number of models stored in the database, and (2) semi-automated construction of new models and large-scale models by combining individual models and modules using the ontology, among others.

For the model sharing and constructing ontology to be productive, a sufficient number of ISML models are required in a model database (ISML database) that provides storage, search, view, and distribution (download) functions. There is already a database (as of August 21, 2008) for ISML 0.1 (http://www.physiome.jp) that stores more than 100 models written in ISML 0.1. ISML 1.0 has complete backward compatibility with ISML 0.1; thus the models in the current database will function with ISML 1.0. The ISML database will be able to work in cooperation with other databases, such as gene and protein databases and SBML and CellML model repositories. Collaboration between the ISML database and other databases will also be supported by the ontology.

This work is supported in part by the Global COE program, "an In Silico Medicine," at Osaka University and Grants-in-Aid (Nos. 20650012 and 19650072) from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

REFERENCES

1. Bassingthwaighte JB. Strategies for the physiome project. Ann Biomed Eng. 2000;28:1043-58.
2. Hunter PJ, Borg TK. Integration from proteins to organs: the Physiome Project. Nat Rev Mol Cell Biol. 2003;4:237-43.
3. Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H, Arkin AP, Bornstein BJ, Bray D, Cornish-Bowden A, Cuellar AA, Dronov S, Gilles ED, Ginkel M, Gor V, Goryanin II, Hedley WJ, Hodgman TC, Hofmeyr JH, Hunter PJ, Juty NS, Kasberger JL, Kremling A, Kummer U, Le Novère N, Loew LM, Lucio D, Mendes P, Minch E, Mjolsness ED, Nakayama Y, Nelson MR, Nielsen PF, Sakurada T, Schaff JC, Shapiro BE, Shimizu TS, Spence HD, Stelling J, Takahashi K, Tomita M, Wagner J, Wang J; SBML Forum. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics. 2003;19:524-31.
4. Cuellar AA, Lloyd CM, Nielsen PF, Bullivant DP, Nickerson DP, Hunter PJ. An overview of CellML 1.1, a biological model description language. Simulation. 2003;79:740-7.
5. Kawazu T, Nakanishi M, Suzuki Y, Odai S, Nomura T. A platform for in silico modeling of physiological systems. Conf Proc IEEE Eng Med Biol Soc. 2007;1394-7.
6. http://www.w3.org/TR/xml/
7. http://www.math.pitt.edu/~bard/xpp/xpp.html
8. http://bunki.sat.iis.u-tokyo.ac.jp/
9. Beeler GW, Reuter H. Reconstruction of the action potential of ventricular myocardial fibres. J Physiol. 1977;268:177-210.
10. Hall K, Glass L. How to tell a target from a spiral: the two probe problem. Phys Rev Lett. 1999;82:5164-7.
11. Luo CH, Rudy Y. A model of the ventricular cardiac action potential. Depolarization, repolarization, and their interaction. Circ Res. 1991;68:1501-26.