34th Annual International Conference of the IEEE EMBS
San Diego, California USA, 28 August - 1 September, 2012

# A Program Code Generator for Multiphysics Biological Simulation using Markup Languages

Akira Amano, Masanari Kawabata, Yoshiharu Yamashita, Florencio Rusty Punzalan,
Takao Shimayoshi, Hiroaki Kuwabara, Yoshitoshi Kunieda

*Abstract*— **To cope with the complexity of the biological function simulation models, model representation with description language is becoming popular. However, simulation software itself becomes complex in these environment, thus, it is difficult to modify the simulation conditions, target computation resources or calculation methods. In the complex biological function simulation software, there are 1) model equations, 2) boundary conditions and 3) calculation schemes. Use of description model file is useful for first point and partly second point, however, third point is difficult to handle for various calculation schemes which is required for simulation models constructed from two or more elementary models. We introduce a simulation software generation system which use description language based description of coupling calculation scheme together with cell model description file. By using this software, we can easily generate biological simulation code with variety of coupling calculation schemes. To show the efficiency of our system, example of coupling calculation scheme with three elementary models are shown.**

## I. Introduction

Simulation models needs to have flexible declarative description to allow changes in boundary conditions or optimization of parameters in different computational environments. This flexible description is important in coupling simulations where two or more models are used. Thus, a declarative way of describing procedural information of the coupled mathematical models in coupling of multiphysics phenomena is useful. This declarative design allows flexible representation of coupling simulations and may simplify using various hardwares.

To realize declarative description of simulation models, some description language based modeling was proposed and many models are written in such languages. One of the most famous one is CellML [1] which is intended to be used for describing cellular function models. There is CellML repository [2] and many models are already accumulated. PHML[3] (formerly known as isML) is another solution to describe cellular level models and also wide range of models in biological phenomena. SBML[4] is description language for metabolic model description which is now considered standard in systems biology area. Another important area is field description where FieldML[5] is now being developed.

These languages give both functional and physiological simulation capability. To use model written with a description language, some simulation software which can handle such file is necessary [6][7][8][9]. However, in order to fully simulate complex models composed of two or more elementary models, numerical solutions for ordinary differential equations (ODE) and a coupling calculation scheme needs to be included in the simulation code. The coupling calculation scheme enable the integration of different phenomena such as cardiac cell contraction and hemodynamics. Note that if we consider constructing a large scale simulation model from several elementary models, it will be difficult to ensure that the model is correct since the constructor may not be a professional researcher in all the elementary model areas. To ensure the correctness of the constructed model, it is important to use each elementary model as is.

In some case, researchers want to evaluate more sophisticated ODE solving scheme for speedup or stabilizing the calculation. In other case, researchers want to use parallel computation resources for calculation of large scale simulations. In another case, the biological simulation model is now becoming very complex and in some case, coupling simulation between different phenomena such as structural simulation and fluid simulation are necessary. In all of these case, it requires special knowledge about numerical calculation or programming and also time consuming process to modify and debug the simulation program. This is quite difficult for biological or medical researchers.

Accordingly, the primary goal of this study is to define an XML-based description language that can represent model coupling schemes in multiple phenomena simulation. We have recently proposed a description language to describe complex ODE solving scheme for single biological model i.e. single set of ordinary differential equation which can only produce program with single loop structure [10]. However, coupling calculation of multiple models involves multiple temporal streams in the calculation scheme. Therefore, the proposed description language uses recurrence relation formula to describe the relationship between different biological models. This coupling schemes description has applications in simulations where the output of one model calculation serves as the input of another model, which in turn gives variable values necessary in the calculation of the first model. The relationship between a cardiac muscle model with sarcomere dynamics [11] and the hemodynamics of the cardiovascular system [12] serves as an example of such coupling.

The coupling scheme described in a description language serves as the primary system input, together with CellML

A. Amano, M. Kawabata, F. Punzalan is with Department of Bioinformatics, Ritsumeikan Univerisity, Shiga-ken, 525-8577, Japan (phone:+81-77-561-2584; e-mail:a-amano@fc.ritsumei.ac.jp)
Y. Yamashita, H. Kuwabara, Y. Kunieda is with Department of Informatics, Ritsumeikan Univerisity, Shiga-ken, 525-8577, Japan)
T. Shimayoshi is with ASTEM RI, Kyoto, 600-8813, Japan

files for describing physiological models, to generate biological simulation codes in different programming languages. Possible output codes include parallel programs for GPU (graphics processing unit) computing.

In this paper, we propose a system which can automatically generate complex simulation program by using coupling calculation schemes together with declarative description models.

## II. Method

### A. Declarative Description of Multiphysics Simulation Model

We propose a declarative description language of coupling calculation schemes by extending the previously proposed Time Evolution Calculation Markup Language (TecML) [10]. In the previous TecML, the variables at time $t_n$ and $t_{n+1}$ are implicitly defined as input and output variables, however, in the extended description, these are explicitly declared as indexed variables of certain calculation step indices.

A time evolution calculation procedure can be described as a recurrence relation between variables at time $t_n$ and $t_{n+1}$. Together with an initial value and a termination condition of the recurrence relation, we can construct a *while* loop for the recurrence relation. Consequently, we based the coupling calculation scheme description language on the recurrence relation.

Consider an equation $dx/dt = k = -x$ as a model, we can describe a time evolution calculation by the Euler method using recurrence relation as follows.

$$x_0 = 1.0 \tag{1}$$
$$k_n = -x_n \tag{2}$$
$$x_{n+1} = x_n + k_n \times \delta \tag{3}$$
$$y = x_n \text{ if } (n = 100) \tag{4}$$

Note that the above description is a declarative description, thus it is flexible in combining with various boundary conditions or other model equations. From this relation, we can construct a calculation procedure as follows.

```
x[0] = 1.0;
n = 0;
do {
  k[n] = -x[n];
  x[n+1] = x[n] + k[n] * dt;
  n = n + 1;
} while ((n-1) != 100);
y = x[n];
```

If we consider two variables calculated with nested double while loop, outer loop variable can be represented by a single index variable (e.g. $x_n$), while inner loop variable can be represented by a double index variable (e.g. $y_{n,m}$). By analyzing the dependencies among included variables in a recurrence relation equations, we can determine the loop structure.

TABLE I
TecML variable types

| Variable Type | Definition |
|---|---|
| recurvar: | Variables calculated by recurrence relations |
| arithvar | Arithmetic variables |
| constvar | Constants |
| timevar | Time variable |
| step | Step variable for recurrence relation |
| condition | Condition for terminating recurrence loop |

### B. Coupling Simulation Code Generation System

The TecML file serves as one of the inputs in the code generation system to create the simulation program. Two other inputs are necessary for generating code, namely, the CellML and RelML (Relation Markup Language) files [10]. CellML allows the mathematical description of biological models. This includes information such as cell structure, equations for underlying processes and boundary conditions. RelML is a language for describing the correspondence between variables in the CellML model file and the variable types in an ODE numerical solution or a coupling calculation scheme described in the TecML file. Table I lists the variable types found in the TecML file and connected by RelML to the physiological model variables.

The code generation system uses the TecML and RelML files to configure the sets of recurrence relation equations describing the time evolution of the variables in the biological models. These sets of equations include the conditions for terminating the recurrence relations described by RelML. The *while* loop in the generated program code calculates the variables in a CellML model until the termination condition of the calculation is reached.

Once the sets of equations are generated, the loop structure is analyzed by variable dependency analysis. For a single loop structure corresponding to a certain time index, related equations are classified into 6 groups, and given attributes to the equations. They are *pre*, *initial*, *inner*, *loopcondition*, *final* and *post*, where *pre* and *post* are equations to be calculated before and after the loop, *initial* is initialization of loop related variables, *inner* is inner equations of the loop, *loopcondition* is a terminating condition and *final* is variable assignment equations of the resulting loop variables.

If there are two or more loop structures in a scheme, we can determine which index structures a loop inside or parallel to the other loop. In the case of double nested loop, equations whose attribute is *inner* for the index 1 can have above 6 types of attributes for index 2.

This dependency analysis is based on the algorithm proposed by Pantelides [13] which analyze the calculation order from a set of simultaneous equation set. The algorithm first generates a bipartite graph where the equation number in the left hand side is linked with the variables in the right hand side which appear in the equation. By calculating the maximum matching from left hand side to right hand side, we can determine which variable should be calculated by which equation by applying the Tarjan's algorithm [14].

The code generation system creates the executable code according to the dependency analysis result. By looking up the attributes of each equation together with the analyzed

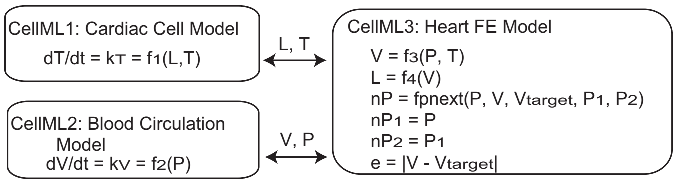

Fig. 1. The coupling of three CellML models. CellML1 contains the cardiac contraction model while CellML2 contains the hemodynamic model. CellML3, which is coupled with the other CellML models, describes the relationship between left ventricular volume, pressure, and cell state.

**TecML1**
Variable definition
output: $\{\xi_{end}, \xi_{rend}, \iota_{rend}\}$
recurvar: $\{\xi[n], \xi[n], \xi[n+1], \xi_r[n][m], \xi_{r1}[n][m], \xi_r[n][m+1]\}$
arithvar: $\{\kappa[n], \iota[n], \iota_r[n][m]\}$
constvar:$\{\xi_{init}, \tau_{init}, \zeta, \zeta_r, \delta\}$
recurequ: $\{\phi(\xi[n], \iota[n], \tau[n], \zeta), \phi_r(\xi_r[n][m], \iota_r[n][m], \zeta_r\}$
aritheq u: $\{\gamma(\xi[n], \iota[n], \tau[n], \zeta), \gamma_r(\xi_r[n][m], \iota_r[n][m], \zeta_r\}$
step:$\{n, m\}$
condition: $\{cond, cond_r\}$
timevar: $\{\tau[n], \tau[n+1]\}$
Equations
$\xi[0] = \xi_{init}$
$\tau[0] = \tau_{init}$
$\iota[n] = \gamma(\xi[n], \iota[n], \tau[n], \zeta)$
$\kappa[n] = \phi(\xi[n], \iota[n], \tau[n], \zeta)$
$\xi[n+1] = \xi[n] + (\kappa[n] \times \delta)$
$\tau[n+1] = \tau[n] + \delta$
$\xi_{end} = \xi[n]$ if $(cond)$
$\xi_r[n][0] = \xi_{rinit}$
$\iota_r[n][m] = \gamma_r(\xi_r[n][m], \iota_r[n][m], \zeta_r)$
$\xi_{r1}[n][m] = \phi_r(\xi_r[n][m], \iota_r[n][m], \zeta_r)$
$\xi_r[n][m+1] = \xi_{r1}[n][m]$
$\xi_{rend} = \xi_r[n][m]$ if $(cond_r)$
$\iota_{rend} = \iota_r[n][m]$ if $(cond_r)$

Fig. 2. The variable types, recurrence relation equations and a coupling scheme constructed from an ODE solver (Euler method) and a convergence loop described in the TecML file.

dependency structure. If certain part of the equations construct simultaneous equation system, the code generation system automatically generates root finding solver code for corresponding equations which is still a simple algorithm in the current implementation. Note that the recurrence relation generated by our system and the corresponding program code are not optimized, thus it requires large memory space and computation time in the original form. However, since the recurrence relations are declarative description, we can use many optimization techniques, and also we can use HPC optimization techniques for optimizing our generated program codes.

## III. Coupling Simulation Example

Here, we show a coupling simulation example that uses three model (CellML) files together with a coupling calculation scheme (TecML) and a relation (RelML) file.

The example CellML files are based on the coupling calculation of heart motion where the contained equations are

**RelML**
cellml: cellml1 = ’CellML1’, cellml2 = ’CellML2’,
cellml3 = ’CellML3’
tecml: ’TecML1’
link: cellml1:t = cellml2:t, cellml2:t = cellml3:t,
cellml1:$T$ = cellml3:$T$
timevar: $\tau$ = {cellml1:t}
recurvar: $\xi$ = {cellml1:$V$, cellml2:$T$},
$\xi_r$ = {cellml3:$V_{target}$, cellml3:$P$, cellml3:$P_1$, cellml3:$P_2$},
$\xi_{r1}$ = {cellml3:$V_{target}$, cellml3:${}_nP$, cellml3:${}_nP_1$,
cellml3:${}_nP_2$}
arithvar: $\kappa$ = {cellml1:$dT/dt$, cellml2:$dV/dt$},
$\iota$ = {*null*}, $\iota_r$ = {cellml3:$V$, cellml3:$L$, cellml3:$e$}
constvar: $\zeta$ = {*null*}, $\zeta_r$ = {*null*}, $\tau_{init}$ = {0.0},
$\delta$ = {0.001}, $\xi_{init}$ = {*Vinit*, *Tinit*},
$\xi_{rinit}$ = {cellml2:$V$, cellml2:$P[n-1]$,
cellml2:$P[n-1]$, cellml2:$P[n-1]$}
condition: $cond$ ={tecml:$n$ = 1000},
$cond_r$ ={cellml3: $e[n][m] < \epsilon$}
output: $\xi_{rend}$ = {*null*, cellml3:$P[n]$, *null*, *null*},
$\iota_{rend}$ = {*null*, cellml3:$L[n]$, *null*}

Fig. 3. RelML file information showing the variable correspondence in the variables of the TecML file and the CellML models. The initial and termination (boundary) conditions are also specified.

shown in Fig.1. CellML1 represents a cardiac cell contraction model (Negroni Lascano model [11]) where the variable $L$ represents cell length, and $T$ represents cell status (named from the contraction related molecule Troponin). CellML2 represents a blood circulation model (Heldt model [12]) which calculates volume change ($dV/dt$) from blood pressure $P$. CellML3 represents Finite Element Method (FEM) calculation which gives heart volume $V$ from pressure $P$ and cell status $T$. Cell length $L$ is calculated from heart volume $V$. The remaining part represents a convergence step which is used to calculate heart pressure from heart volume. In this example, this calculation is necessary since FEM used in this example can only calculate shape from applied pressure and force.

The example TecML file contains the time evolution calculation and a convergence calculation schemes for the coupled CellML models (Fig.2). The recurrence relation index variables $n$ and $m$ are also declared in the file.

The example RelML file describes the correspondence between the variables in the CellML files and the TecML file (Fig.3). We also have to describe the initial values of certain variables and the termination conditions for the recurrence relations described in the TecML file.

From these input files, we can obtain the declarative description of recurrence relations as follows.

$$V[0] = V_{init} \quad (5)$$

$$T[0] = T_{init} \quad (6)$$

where $V_{init}$ and $T_{init}$ are the initial volume and cell status, respectively. Since the derivatives of both $V[n]$ and $T[n]$ are calculated from $L[n]$ and $P[n]$ which appear as results of CellML3 calculation, the convergence part is calculated in the next part.

$$\begin{aligned}
V_{target}[n][0] &= V[n] && (7)\\
P[n][0] &= P[n-1] && (8)\\
P_1[n][0] &= P[n-1] && (9)\\
P_2[n][0] &= P[n-1] && (10)\\
V[n][m] &= f_3(P[n][m], T[n]) && (11)\\
L[n][m] &= f_4(V[n][m]) && (12)\\
e[n][m] &= |V[n][m] - V_{target}[n][m]| && (13)\\
nP[n][m] &= f_{Pnext}(P[n][m], V[n][m], \\
&\quad V_{target}[n][m], P_1[n][m], P_2[n][m]) && (14)
\end{aligned}$$

$$\begin{aligned}
nP_1[n][m+1] &= P[n][m] && (15)\\
nP_2[n][m+1] &= P_1[n][m] && (16)\\
V_{target}[n][m+1] &= V_{target}[n][m] && (17)\\
P[n][m+1] &= nP[n][m] && (18)\\
P_1[n][m+1] &= nP_1[n][m] && (19)\\
P_2[n][m+1] &= nP_2[n][m] && (20)\\
P[n] &= P[n][m] \text{ if } (e[n][m] < \epsilon) && (21)\\
L[n] &= L[n][m] \text{ if } (e[n][m] < \epsilon) && (22)
\end{aligned}$$

The pressure value $P[n][m]$ and the cell length $L[n][m]$ are passed on once the condition for $e$ is reached. These values along with the current volume $V[n]$ and cell (troponin) status $T[n]$ is used for the recurrence relation calculations of CellML1 and CellML2 as shown by the following equations.

$$\begin{aligned}
k_T[n] &= f_1(L[n], T[n]) && (23)\\
k_V[n] &= f_2(P[n]) && (24)\\
T[n+1] &= T[n] + k_T[n] \times \delta && (25)\\
V[n+1] &= V[n] + k_T[n] \times \delta && (26)\\
T_{end} &= T[n] \text{ if } (n = 1000) && (27)\\
V_{end} &= V[n] \text{ if } (n = 1000) && (28)
\end{aligned}$$

In this example, the *while* for the first model terminates and the final output of the calculation is passed on as input for another CellML model. The next calculation, described by the separate but coupled model, is implemented using a separate *while* loop. The output of this second *while* loop is then fed back into the first model as input. Finally, from the above recurrence relations, we can generate program to calculate every variable at every time point as Fig.4.

```
// Executable Code
  V[0] = Vinit;
  T[0] = Tinit;
  n = 0;
  do {
    Vtarget[n][0] = V[n];
    P_3[n][0] = P[n-1];
    P1[n][0] = P[n-1];
    P2[n][0] = P[n-1];
    m = 0;
    do {
       V_3[n][m] = f3(P_3[n][m], T[n]);
       L_3[n][m] = f4(V_3[n][m]);
       e[n][m] = fabs(V_3[n][m] - Vtarget[n][m]);
       nP[n][m] = fPnext(P_3[n][m],V_3[n][m],Vtarget[n][m],
                         P1[n][m],P2[n][m]);
       Vtarget[n][m+1] = Vtarget[n][m];
       nP1[n][m+1] = P[n][m];
       nP2[n][m+1] = P1[n][m];
       m = m + 1;
    } while(!(e[n][m-1] < eps));
    P[n] = P_3[n][m-1];
    L[n] = L_3[n][m-1];
    kT[n] = f1(L[n], T[n]);
    kv[n] = f2(P[n]);
    T[n+1] = T[n] + kT[n] * delta;
    V[n+1] = V[n] + kV[n] * delta;
    n = n + 1;
  } while(!((n-1)==1000));
  Tend = T[n-1];
  Vend = V[n-1];
```

Fig. 4. The generated executable code for the coupling simulation.

## IV. Conclusions

We proposed a coupling calculation description language together with biological function simulation program code generator using model and coupling calculation scheme description file written by proposed language. By using this system, we can easily generate simulation program of complex combination of elementary models such as cell models, ventricular geometry models and circulation models.

The process of combining multiple models requires variable matching based on the biological knowledge. This process can be supported by the use of biological ontology database which will be one of our future works.

The part of the system is publicly available in our project website[8], and the full system will be publicly available in the future.

## References

[1] A.A. Cuellar, C.M. Lloyd, P.F. Nielsen, D.P. Bullivant, D.P. Nickerson and P.J. Hunter, "An overview of CellML 1.1, a biological model description language." *Simulation*, vol. 79, pp. 740-747, 2003.

[2] http://models.cellml.org/cellml

[3] Y. Asai, Y, Suzuki, Y. Kido, H. Oka, E. Heien, M. Nakanishi, T. Urai, K. Hagihara, Y. Kurachi and T. Nomura, "Specifications of insilicoML 1.0: a multilevel biophysical model description language." *J Physiol Sci*, vol. 58, pp. 447-458, 2008.

[4] M. Hucka et al., "The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models", Bioinformatics, 9(4):524–531, 2003.

[5] G.R. Christie, P.F. Nielsen, S.A. Blackett, C.P. Bradley and P.J. Hunter, "FieldML: concepts and implementation." *Physical and Engineering Sciences*, vol. 367, pp. 1869-1884, 2009.

[6] http://www.opencor.ws/

[7] http://physiodesigner.org/

[8] http://cellmlcompiler.sourceforge.net/

[9] http://www.cmiss.org/openCMISS

[10] A. Amano, N. Soejima, T. Shimayoshi, H. Kuwabara and Y. Kunieda, "A general CellML simulation code generator using ODE solving scheme description." in *33rd Annual International IEEE EMBS Conference*, 2011.

[11] J.A. Negroni and E.C. Lascano, "A cardiac muscle model relating sarcomere dynamics to calcium kinetics." *J Mol Cell Cardiol*, vol. 28, pp. 915-929, 1996.

[12] T. Heldt, E.B. Shim, R.D. Kamm, and R.G. Mark, "Computational modeling of cardiovascular response to orthostatic stress." *J Appl Physiol*, vol. 92, pp. 1239-1254, 2002.

[13] C Pantelides, The Consistent Initialization of Differential-Algebraic Systems, SIAM J. Sci. and Stat. Comput. Volume 9, Issue 2, pp. 213–231, 1988.

[14] Tarjan, R. E., "Depth-first search and linear graph algorithms", SIAM Journal on Computing 1 (2): 146–160, 1972.