Proceedings of the 28th IEEE
EMBS Annual International Conference
New York City, USA, Aug 30-Sept 3, 2006

Development of a support tool for multi-agent based biological modeling

Toshihiro Kawazu, Shinsuke Odai, Noriko Shibuya, and Taishin Nomura

Abstract— **In late years physiomic modeling of biological organisms are performed flourishingly. However enormous labor is required for performing such a large-scale and complicated biological model development. We were aimed at developing a software tool which could reduce a burden of the model developers who try to build such a model. At first we defined the data structure that could describe structure and functions of modules consisting of biological organisms. Then we designed the base classes which expressed the data structure. A whole model would be realized by connecting a number of modules that realize the functioning of the base classes. By applying our tool to concrete model developments for a cardiac cell and a single ionic channel, we examined utility and effectiveness of the tool for modeling on a multi-scale.**

I. Introduction

Structure of biological organisms can be characterized at several levels of spatial scale, ranging from molecules, cells, organs to individual organisms. Possible integrated descriptions of the functional behavior of the physiological state of an individual based on its hierarchical structure of the organism have been termed as "physiome" [1]. Thus, physiomic modeling should deal with descriptions of structure and state of integrands, and ways to integrate and simulate those. In late years physiomic modeling of biological organisms are performed extensively [2]. However enormous labor is required for performing such a large-scale modeling. We were aimed at developing a software tool which could reduce a burden of the model developers who try to build such a model.

II. Outline of a Tool

This study aimed at developing a software tool that could support variety of users, ranging from biologist, physiologist to bioengineers, to deal with physiomic modeling of biological organisms and to facilitate dynamic simulation of the models. Basic requirements for the specifications of tool were as follows:

- *Generality*: To support modeling of various biological objects, such as proteins, cells, organs as an aggregate of cells.
- *Structural Modularity and Sub-modularity*: Each model can be defined as a module object. Structure of each module can be defined, if necessary, as a set of sub-modules with smaller spatial scales. The set of sub-modules may exhibit hierarchy and/or network structure.
- *Autonomy*: Each module has its "state," and can update the state autonomously as an agent so that the user can simulate its dynamics with ease.
- *Functional Modularity*: For a target module, the user can select modules that influence a way to update the state of the target module, and then define the influence.
- *Editablity*: The user can easily perform model revision, such as connecting and/or eliminating existing modules.
- *Module Management*: To provide an effective way to manage a number of sub-modules.

In this study, we constructed a prototype of a tool which satisfied these requirements. We then applied the tool to several concrete modeling targets to examine utility of the tool.

III. Structure and Functional Relations

A modeling of a biological organ at multiple scale could force the user to deal with a enormous number of modules, and relationships among the modules could become complicated. To overcome such difficulties, it is required for the tool to provide a way to make the global structure understandable for the user and to make handling structure of the model easy. To this end, we defined a basic data structure that could describe characteristics of the modules and relations between modules of living organisms. We considered relations among modules. They include the structural and the functional relations. Using these two relations, we provided the software environment to support model development of models of living body organs as an aggregate of modules and sub-modules.

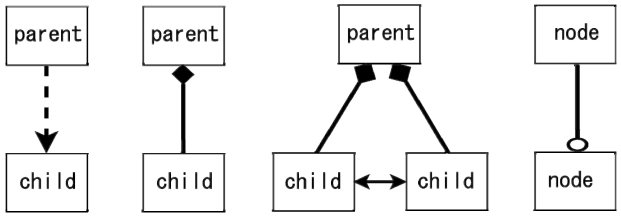

Fig. 1. Diagram representations of structure and functional relations. From the left, "include-like," "consist-of-like" hierarchical relations, and "attachment" and "functional" relations. The last two relations do not represent hierarchical structure. The functional relation has its direction, starting at the non-marked site and terminated at the open circle, along which state update notification signals are transmitted during dynamics simulations.

Manuscript received April 3, 2006. This work was supported in part by JSPS and Japanese Ministry of education and science under Grants #16300154 and #17650085.

T. Kawazu, S. Odai, N. Shibuya, and T. Nomura are with Graduate School of Engineering Science, at Osaka University, Toyonaka, Osaka, 560-8531 Japan. (phone: 06-6850-6532; fax: 06-6850-6557; e-mail: kawazu@bpe.es.osaka-u.ac.jp, odai@bpe.es.osaka-u.ac.jp, shibuya@bpe.es.osaka-u.ac.jp, taishin@bpe.es.osaka-u.ac.jp).

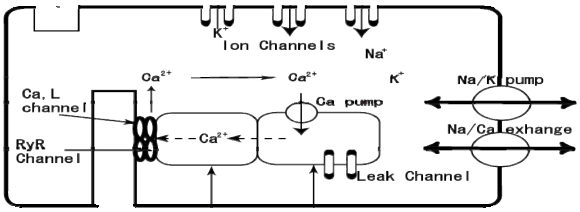

Fig. 2. Schematic picture of a cardiac cell.

A. Structural relation

We considered structure of living organisms using structural relations among modules consisting of the organ. The hierarchical structure is an important relation to represent structure of living organisms. However, the usual hierarchical structure that defines a unique parent of a module might not be enough to express complicated structure of living organisms. To deal with this situation, we defined the following two types of structural relations (Fig. 1). The first relation represents a hierarchical structure, referred to as the "include-like", to describe a situation in which a module is included in or physically inside another module. The second hierarchical relation was referred to as the "consist-of-like" hierarchical structure, in which a set of modules at a certain scale makes up a module at a larger scale. Separately from the hierarchical structure, we defined a structure, referred to as the "attachment," to represent a situation in which modules were interconnected with each other.

The three relations mentioned above can be used to represent a model of a cardiac cell as a concrete example as shown in Fig. 2. One can easily observe that the sarcoplasmic reticulum (SR) and ions are inside the cell membrane, which could be represented by the "include-like" hierarchical structure (Fig.3). The cell membrane consists of phosphatides, ion channels and ion exchange pumps, and each ion channel consists of several protein subunits. These can be expressed by the "consist-of-like" hierarchical structure. The network-SR and the junctional-SR are the modules with the same spatial scale, and they are connected with each other. This structure can be expressed by the "attachment" relationship. As such, the "attachment" relationship is convenient to express that several modules are physically connected with each other to define a whole module. Note that the attachment relation does not describe the hierarchical structure. Figure 3 summarizes a model structure representation of a cardiac cell shown in Fig. 2, using the structural relations defined in this study.

B. Functional relation

A module consisting of a biological organ may typically represent a physical object, and it possesses a “state.” In terms of object-oriented programming, a state of the module may be modeled as a property of the module. For a given module

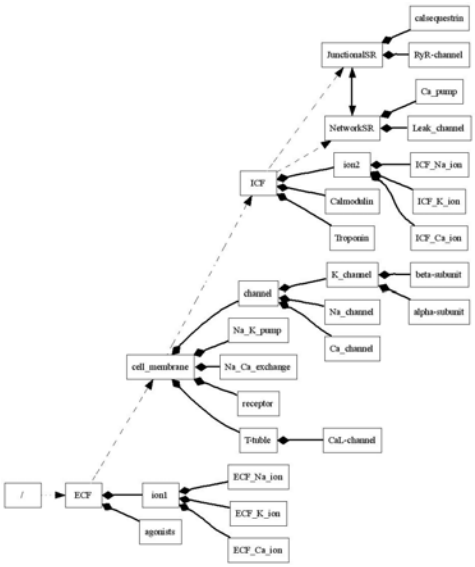

Fig. 3. Hierarchical representation of the cardiac cell model shown in Fig. 2.

object, a spatial position and its moving velocity could be considered as the state of the object. A state of the module changes in time under influences from states of other modules or that of itself. If a state of a module determines the state update of the target module, these two modules are said to be functionally-related (the right most diagram of Fig. 1). A functional relation has its direction, starting at one object and terminated at another object, along which a state update notification signal is transmitted during dynamic simulations. Because a way how a functional relation determines the state update of a certain module depends on the module under consideration, our support tool was designed just to provide a framework to notify update signal to a module that is functionally connected when a certain module needs to update its state. For specific modeling of an object, the concrete way of its state update will be defined by the user. The functional relation defined here is basically independent of underlying structure of the module, and hence the functional relations exhibit network structure.

IV. Model Construction and Edit

Based on the structural and functional relations defined in the previous chapter, we constructed a software system that could effectively manage structure and functions of living organisms.

A. Design of Base Classes

For our software implementation, we used C++ (Visual C++ .NET). We designed base classes that could represent the relations described above. The first base class is referred to as the “Node” class, representing a basic prototype of module. The second base class is referred to as the “Edge” class. The Edge class does represent the structural relations among Node class objects. That is, a Node object may be related to another Node object through an Edge object. A Node class object possesses two properties, referred to as

"*inEdge" and "*outEdge." The former is a list of pointers to Edge objects that are linked to parents, and the latter to children. Note that the parents are not necessarily unique. Each Edge object listed in a Node object possesses two properties "*head" and "*tail," which are, respectively, a pointer to the parent Node and a pointer to the child Node. Another property of the Edge class is "type" specifying a type of the relations among "include-like", "consist-of-like", and "attachment."

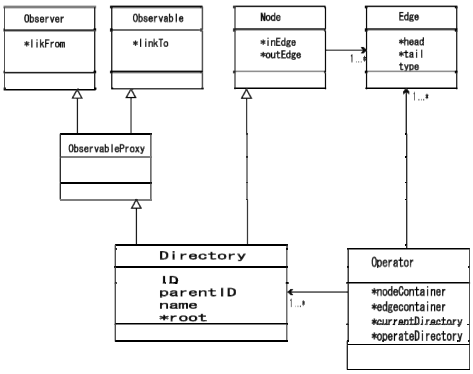

Fig. 4. Class diagram of base classes.

The "Operator" class was designed to manage two base classes "Node" and "Edge." The user may construct and edit his/her models through the Operator class object.

The functional relation among modules as Node objects was designed separately from the structural relations, using the observer pattern, one of the Java-Design Patterns. A state-update signal may be sent "automatically" to a Node object (module) through the functional relation. For our implementation, two classes, "Observable" class as a signal origin (signal sender) and "Observer" class as a signal destination (signal receiver), were defined. The property (*linkTo) of the Observable class is a list of pointers to Observer objects as the signal receivers, while the property (*linkFrom) of the Observable objects as the signal senders. The relation between Observer and Observable classes can represent the functional relation among modules. Note that, as detailed below, a module may inherit the characteristics of Observer and/or Observable classes, as well as those of Node class. Since a detailed way of state-update may depend on the modeling target, a method of this base class, which performs a state-update, was defined as a virtual function.

Using above four base classes, the "Directory" class was defined. The Directory class represents a module consisting a biological model that the user needs to construct, manage and simulate. It is the class that inherits the three classes. Hence, a Directory class object could be managed by the user, through the Operator object, for a model construction and edit in both the structural and functional purposes. Figure 4 is the class diagram that summarizes the relations among four base classes used to define a module as a Directory object, and the Operator class as the interface to the user.

B. Visualization of Model

When the user constructs a large scale model, it is important to visualize structure of the model. A utility to visualize the

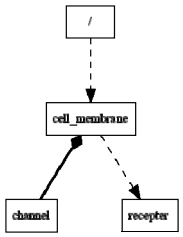

Fig. 5. The model that was built by the command..

structure (functional and structural relations among modules) of models was implemented to facilitate user's understanding. As an example, let us suppose that the user is dealing with modeling of a cardiac cell shown in Fig. 2. In our current version, the user will type commands, such as

- `mkdir cell_membrane`
- `cd cell_membrane`
- `mkdir channel`
- `mkdir recepter`
- `clink ./ channel -c`

By these commands, a small part of the model could be constructed, and the corresponding graph visualization was produced automatically (Fig. 5).

The structural relations among modules, such as ions and channels, when the user completed the model construction can be represented automatically as a tree-like graph as shown in Fig. 3.

In addition to the tree-like graph visualization, another way of automatic model visualization similar to Ben's diagram was also implemented, where the hierarchical structure was visualized as an inclusive relation of circular objects. See Fig. 6 in which the cardiac cell model shown by the tree-like graph in Fig. 3 is now transformed into another graph, which is closer to the model shown in Fig. 2. It is expected that such visualization is useful for the user, in particular when the target model of the user is complicated.

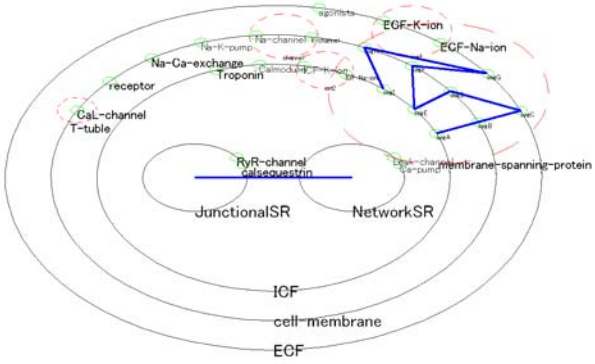

Fig. 6. Automatic visualization of model's structure. This figure visualizes conceptual structure of a cardiac cell model shown in Fig. 2.

V. MEMBRANE DYNAMICS SIMULATION

We examined the utility of our tool by constructing a model of single membrane channel dynamics, in which potassium ions flow through an ion channel of a cell membrane [3]. At first, with the relations that defined above, we expressed

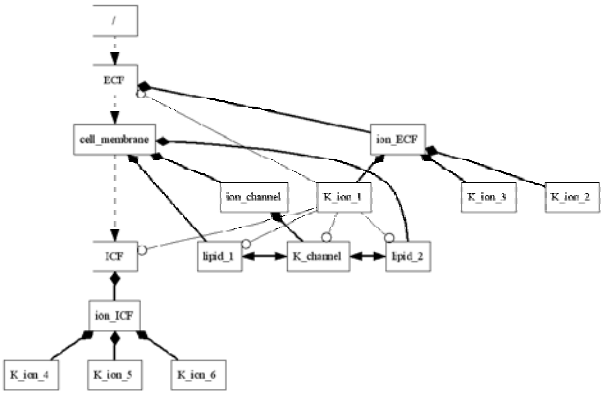

Fig. 7. Hierarchical description of single K channel model. Note that the functional relation is depicted only for K_ion_1 for simplicity, although there are a number of potassium ions

structure of a cell as hierarchical structure (Fig7). Each module inherits three base classes described in the previous section. We considered "position" of an ion as the "state" of the ion. A rule of state-update was as follows; each ion moves in accordance with the concentration gradient and electric potential difference between inside and outside of the cell. With this rule, functional relations between every single ion and other modules were introduced. For example, regarding the functional relation between an ion and the channel, each ion object send a signal to the channel object along the functional relation at every instant of its state-update.

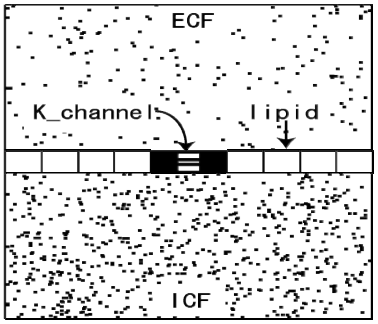

Fig. 8. A snapshot of simulation for the single channel dynamics. Small dots are potassium ions. The modeled space was separated by the membrane.

The channel object performs its own state-update associated with the gate opening and closing, and also performs the operation defined by the user upon every signal notification from ions. The user could define such an operation for each channel if necessary. For example, a potassium ion notifies the potassium channel along the functional relation. If the state of the gate of potassium channel is open and if that potassium ion is at the entrance of the channel, that potassium ion can get into the channel. If the source of signal notification was not potassium ion, that ion cannot pass through the channel regardless of the state of the gate. Each ion object itself does not have to judge whether or not it can pass through the channel, but the channel object recognizes

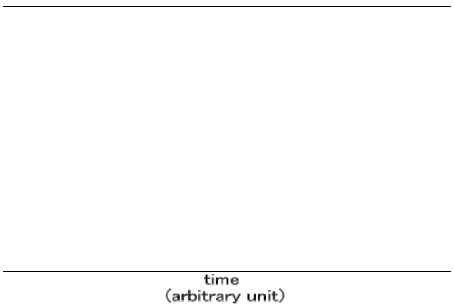

Fig. 9. Dynamics of a single-channel current.

the type of the ion object and is responsible for the judgment. Fig. 8 illustrates the Brownian-like movement of potassium ions and opening and closing of single potassium ion channel. Fig. 9 exemplifies a simulation result, in which the simulated single channel potassium current could reproduce well the experimental observation at least qualitatively.

When the user wishes to simulate with other ions and channels in addition to the potassium ions and the channel, the user may be able to add new objects and make functional relation according to the procedure illustrated above with relatively ease.

VI. Conclusion

We developed a tool which reduced a burden of the model developers who try to build such a model. To this end, we defined the data structure that could describe structure and function of modules consisting of biological organisms. We showed that, using several biological examples, a whole model could be realized by connecting a number of modules, and our tool might be useful for multi-scale biological modeling.

References

[1] Jim Bassingthwaight (2005, November, 28), *The Phsiome Project* [Online]. Available: http://www.physiome.org(URL)

[2] Edmund J. Crampin, Matthew Halsetead, Peter Hunter, Poul Nilsen, Denis Noble, Nicolas Smith, and Merryn Tawhai, "Computationl physiology and the physiome project," *Experimental Physiology,* 2004, Jan, 89(1), pp. 1-26.

[3] A. L. Hodgkin, A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol*, vol. 117, pp. 500-544, 1952.