Cell Modeling using Agent-based Formalisms

Kenneth Webb, Tony White
Sussex University, U.K, Carleton University, Canada
k.s.webb@sussex.ac.uk, arpwhite@scs.carleton.ca

Abstract

The systems biology community is building increasingly complex models and simulations of cells and other biological entities. This community is beginning to look at alternatives to traditional representations such as those provided by ordinary differential equations (ODE). Making use of the object-oriented (OO) paradigm, the Unified Modeling Language (UML) and Real-time Object-Oriented Modeling (ROOM) visual formalisms, we describe a simple model that includes membranes with lipid bilayers, multiple compartments including a variable number of mitochondria, substrate molecules, enzymes with reaction rules, and metabolic pathways. We demonstrate the validation of the model by comparison with Gepasi and comment on the reusability of model components.

1 Introduction

Researchers in bioinformatics and systems biology are increasingly using computer models and simulation to understand complex inter- and intra-cellular processes. The principles of object-oriented (OO) analysis, design, and implementation, as standardized in the Unified Modeling Language (UML), can be directly applied to top-down modeling and simulation of cells and other biological entities. This paper describes how an abstracted cell, consisting of membrane-bounded compartments with chemical reactions and internal organelles, can be modeled using tools such as Rational Rose RealTime (RRT), a UML-based software development tool. The resulting agent-based approach, embodied in CellAK (for Cell Assembly Kit), produces models that are similar in structure and functionality to those that can be specified using the Systems Biology Markup Language (SBML) [1], and CellML [2], and implemented using E-CELL [3], Gepasi [4], Jarnac [5], StochSim [6], Virtual Cell [7], and other tools currently available to the biology community. We claim that this approach offers greater potential modeling flexibility and power because of its use of OO, UML, ROOM, and RRT, which utilizes the actor concept in its implementation. The OO paradigm, UML methodology, and RRT tool, together represent an accumulation of best practices of the software development community, a community constantly expected to build more and more complex systems, a level of complexity that is starting to approach that of systems found in biology. CellAK represents, therefore, an agent-based cell modeling environment built on top of state of the art software modeling tools and practices.

All of these approaches mentioned above make a fundamental distinction between structure and behavior. This paper deals mainly with the top-down structure of membranes, compartments, small molecules, and the relationships between these, but also shows how bottom-up behavior of active objects such as enzymes, transport proteins, and lipid bilayers, is incorporated into this structure to produce an executable program.

We do not use differential equations to determine the time evolution of cellular behavior, as is the case with most of the cell modeling systems described in this paper. Differential equations find it difficult to model directed or local diffusion processes and subcellular compartmentalization [8]. CellAK more closely resembles Cellulat [9] in which a collection of autonomous agents (our active objects – enzymes, transport proteins, lipid bilayers) act in parallel on elements of a set of shared data structures called blackboards (our compartments with small molecule data structures). Differential equation-based models are also difficult to reuse when new aspects of cell structure need to be integrated. The motivation for this paper is the demonstration of a high degree of reuse in the agent-based models that have been developed; reuse of behavior and structure, both separately and in combination. Finally, we note that agent-based modeling of cells is becoming an area of increasing research interest [8, 9] thereby prompting the communication of this research.

This paper consists of 4 further sections. The next section introduces the Real-Time Object- Oriented Methodology (ROOM). A CellAK model is then described that uses the concepts of inheritance, containment, ports and connectors. Having introduced the model, a validation section is provided. The paper concludes with a review of key messages and references to future work.

2 The ROOM formalism

David Harel, originator of the hierarchical state diagram formalism used in the Universal Modeling Language (UML) [10], and an early proponent of visual formalisms in software analysis and design [11], has argued that biological cells and multi-cellular organisms can be modeled as reactive systems using real-time software development tools [12,13].

Reactive systems are those whose complexity stems not necessarily from complicated computation but from complicated reactivity over time. They are often highly concurrent and time-intensive, and exhibit hybrid behavior that is predominantly discrete in nature but has continuous aspects as well. The structure of a reactive system consists of many interacting components, in which control of the behavior of the system is highly distributed amongst the components. Very often the structure itself is dynamic, with its components being repeatedly created and destroyed during the system's life span, see [13 p.5].

Rational Rose RealTime (RRT) is a visual design and implementation tool for the production of telecommunication systems, embedded software, and other highly-concurrent real-time systems. It combines the features of UML with the real-time specific features and visual notation of the Real-time Object-Oriented Modeling (ROOM) [14]. A RRT application's main function is to react to events in the environment, and to internally-generated timeout events, in real-time.

Software developers design software with RRT by decomposing the system into an inheritance hierarchy of classes and a containment hierarchy of actors, using UML class diagrams. Each architectural actor, or capsule as they are called in RRT, contains a UML state diagram that is visually designed and programmed to react to externally generated incoming messages (generated within other capsules or sent from external systems), and to internally-generated timeouts. The agents in a CellAK system are the architectural actors in a RRT model. Messages are exchanged through ports defined for each capsule. Ports are instances of protocols, which are interfaces that define sets of related messages. All C++, C, or Java code in the system is executed within actors' state diagrams, along transitions from one state to another (which may be a self-transition to the same state). An executing RRT system is therefore an organized collection of communicating finite state machines. The RRT run-time scheduler guarantees correct concurrent behavior by making sure that each transition runs all of its code to completion before any other message is processed.

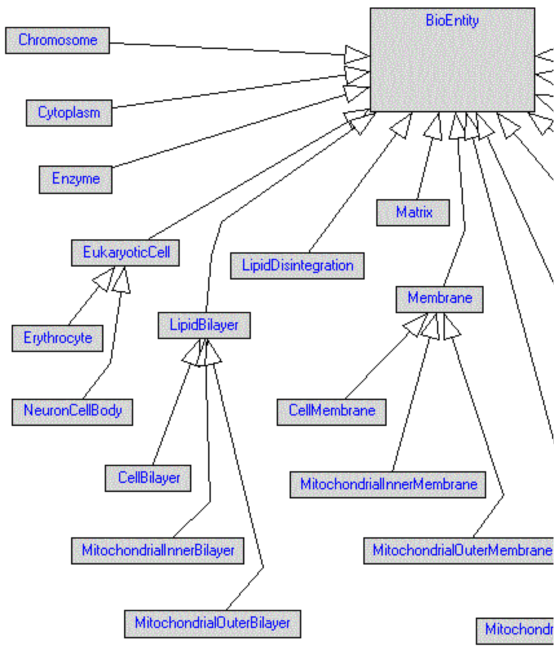

Figure 1: UML Diagram for BioEntities

The RRT design tool is visual. During design, to create the containment structure, capsules are dragged from a list of available classes into other classes. For example, the designer may drag an instance of Nucleus onto the visual representation of EukaryoticCell, thus establishing a containment relationship. Compatible ports on different capsules are graphically connected to allow the sending of messages. UML state diagrams are drawn to represent the behavior of each capsule. Other useful UML graphical tools include use case diagrams, and sequence diagrams. External C++, C, or Java classes can be readily integrated into the system. RRT allows the time evolution of a simulation to be watched, instrumented with breakpoints, and stepped through.

The developer generates the executing system using visual programming, dragging and dropping objects onto a graphical editor canvas. RRT generates all required code from the diagrams, and produces an executable actor-based program. The executable can then be run and observed using the design diagrams to dynamically monitor the run-time structure and behavior of the system.

The powerful combination of the actor/OO paradigm as embodied in the UML and ROOM visual formalisms with the added flexibility of the C, C++ or Java programming languages, bundled together in a development tool such as RRT, provide much that is appropriate for biological modeling.

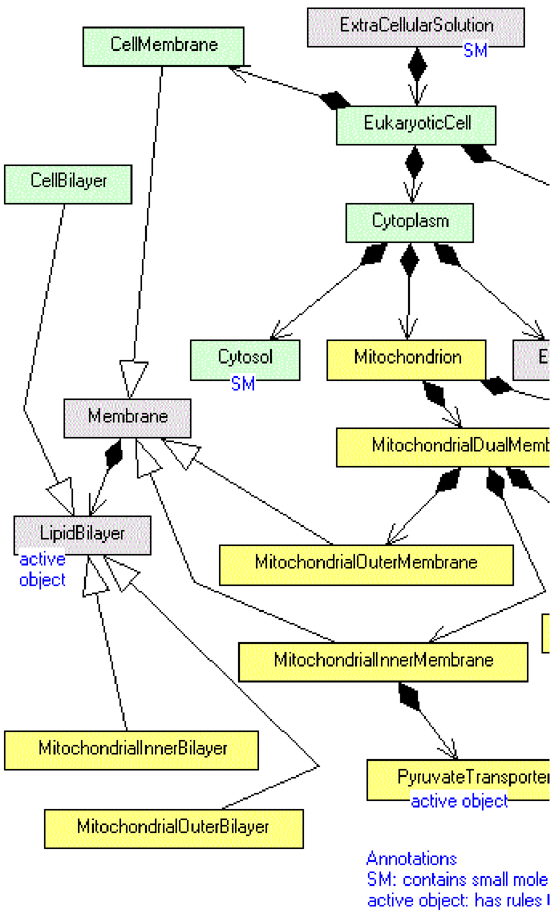

Figure 2: Containment Hierarchy

To summarize, benefits of the CellAK that are of use in cell and other biological modeling that have been identified so far in this paper include: support for concurrency and interaction between entities, scalability to large systems, use of inheritance and containment to structure a system, ability to implement any type of behavior that can be implemented in C, C++ or Java, actor instantiation from a class, ease of using multiple instances of the same class, and subclassing to capture what entities have in common and how they differ.

3 The Model

3.1 Classes, Capsules and Containment

The purpose of the small example system described here is to model and simulate metabolic pathways, especially the glycolytic pathway that takes place within the cytoplasm, and the TCA cycle that takes place within the mitochondrial matrix. It also includes a nucleus to allow for the modeling of genetic pathways in which changes in the extra cellular environment can effect changes in enzyme and other protein levels. The model is easily extensible, to allow for specialized types of cells.

Figure 1 shows a set of candidate entities organized into an inheritance hierarchy, drawn as a UML class diagram. The lines with a triangle at one end are the standard UML notations for inheritance. Erythrocyte and NeuronCellBody are particular specializations of the more generic EukaryoticCell type. CellBilayer, MitochondrialInnerBilayer, and MitochondrialOuterBilayer are three of potentially many different subclasses of LipidBilayer. These three share certain characteristics but typically differ in the specific lipids that constitute them. The figure also shows that there are four specific Solution entities, each of which contains a mix of small molecules dissolved in the Solvent water. All entity classes are subclasses of BioEntity.

Figure 2 shows a different hierarchy, that of containment. This UML class diagram shows that at the highest level, a EukaryoticCell is contained within an ExtraCellularSolution. The EukaryoticCell in turn contains a CellMembrane, Cytoplasm, and a Nucleus. This reductionist decomposition continues for several more levels. It includes the dual membrane structure of a Mitochondrion along with its inter-membrane space and solution and its internal matrix space and solution. Part of the inheritance hierarchy is also shown in these figures. Each Membrane contains a LipidBilayer, but the specific type of bilayer (CellBilayer, MitochondrialInnerBilayer, MitochondrialOuterBilayer) depends on which type of membrane (CellMembrane, MitochondrialInnerMembrane, MitochondrialOuterMembrane) it is contained within.

3.2 Specifying Adjacency

A model is constructed of capsules, which are instances of classes shown in Figure 1. Capsules are arranged in a containment hierarchy as shown in Figure 2. Capsules represent the agents in a CellAK simulation. Connectivity between capsules determines adjacency; i.e. how changes

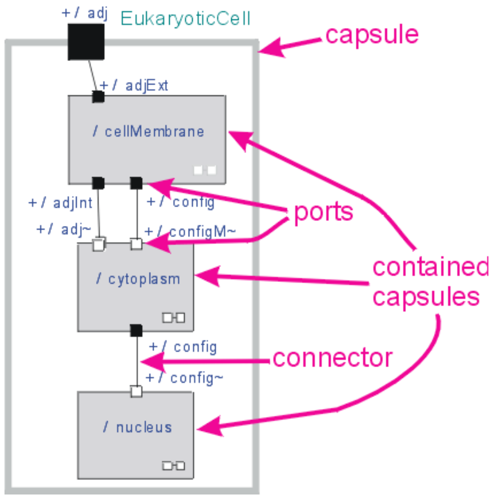

Figure 3: Eukaryotic Cell Capsule Structure

in the state of one capsule affect another. Changes occur through the exchange of messages.

In a EukaryoticCell, CellMembrane is adjacent to and interacts with Cytoplasm, but is not adjacent to and therefore cannot interact directly with Nucleus. Interactions between CellMembrane and Nucleus must occur through Cytoplasm. It is important to have a structural architecture that will place those things adjacent to each other that need to be adjacent, so they can be allowed to interact.

Adjacency is represented using protocols. A protocol is a specific set of messages that can be exchanged between capsules to allow interaction. The Configuration protocol has two signals - ConfigSig and MRnaSig. When the simulation starts, the Chromosome within the Nucleus sends a ConfigSig message to the Cytoplasm, which will recursively pass this message to all of its contained capsules. When an active object such as an Enzyme receives the ConfigSig message, it determines its type and takes on the characteristics defined in the genome for that type. When a Solution such as Cytosol receives the ConfigSig message, it extracts the quantity of the various molecules that it contains, for example how many glucose and how many pyruvate molecules. In addition to being passed as messages through ports, configuration information may be also be passed in to a capsule as a parameter when it is created. This is how the entire Mitochondrion containment hierarchy is configured. In this approach, Nucleus is used for a purpose in the simulation that is similar to its actual role in a biological cell. The MRnaSig (messenger RNA signal) message can be used to reconfigure the system by creating new Enzyme types and instances as the simulation evolves in time.

The Adjacency protocol allows configured capsules to exchange messages that will establish an adjacency relationship. Capsules representing active objects (Enzymes, PyruvateTransporter and other types of TransportProtein, LipidBilayer) that engage in chemical reactions by acting on small substrate molecules, will send SubstrateRequest messages. Capsules that contain small molecules (types of Solution such as Cytosol, ExtraCellularSolution, MitochondrialIntermembranesol, Matrixsol) will respond with SubstrateLevel messages. Figure 3 is a capsule structure diagram that shows EukaryoticCell and its three contained capsules with named ports and connector lines between these ports. The color of the port (black or white) indicates the relative direction (in or out) of message movement.

Figure 3 represents a *significantly* simplified model; the final model includes all of the capsules shown in Figure 2. The full model is shown in Figure 4. A full description of Figure 4 is, however, beyond the scope of this paper.

Defining the desired behavior of the system is achieved by specifying patterns of message exchange between capsules.

In the sample model, the glycolytic pathway is implemented through the multiple enzymes within Cytoplasm, all acting concurrently on the same set of small molecules within Cytosol. The TCA metabolic pathway is similarly implemented by the concurrent actions of the multiple enzymes within Matrix acting on the small molecules of the Matrixsol. Movement of small molecules across membranes is implemented by the various lipid bilayers. For example, lipidBilayer within MitochondrialOuterMembrane transports pyruvate from the Cytosol to the MitochondrialIntermembranesol, and pyruvateTransporter within MitochondrialInnerMembrane transports pyruvate across this second membrane into the Matrixsol.

3.3 Enzyme Behaviour

Figure 5 shows the UML state diagram representing the behavior of an Enzyme active object. When first created, it makes the initialize transition. As part of this transition it executes a line code that sends a message out its adj port. When it subsequently receives a SubstrateLevel response message through the same adj port, it stores the SmallMolecule reference that is part of that message, creates a timer so that it can be invoked at a regular interval, and makes the transition to the Active state.

The state diagrams for lipid bilayers and transport proteins are much the same, but include additional states because they need to connect to two small molecule containers, one inside and the other outside.

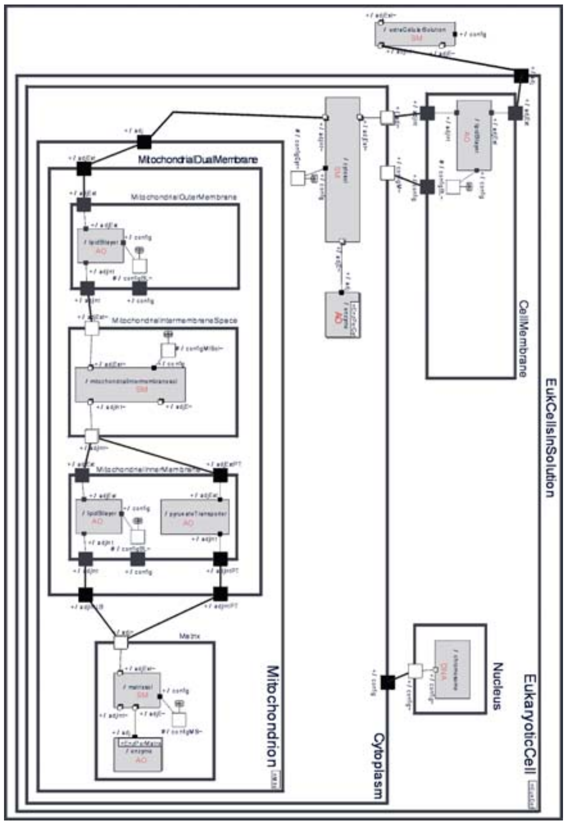
EukCellsInSolution
EukaryoticCell
CellMembrane
Cytoplasm
Nucleus
Mitochondrion
MitochondrialDualMembrane
MitochondrialOuterMembrane
MitochondrialIntermembraneSpace
MitochondrialInnerMembrane
Matrix

Figure 4: The complete structure of the sample model, with all capsules, ports, and connectors. This is an enhancement of Figure 3, with additional details added.

3.4 Kinetics and Enzyme Reactions

Enzyme reactions can take various forms. In this paper, we consider the simplest case, in which an enzyme irreversibly converts a single substrate molecule into a different product molecule. By irreversible is meant that the enzyme cannot also convert the product into the substrate. More complex reactions include combining two substrates into one resulting product, splitting a single substrate into two products, and making use of activators, inhibitors, and coenzymes.

In the C++ code below, which implements irreversible Michaelis-Menten kinetics [15 p.148+], [4] `sm->` is a reference to the SmallMolecule data structure that in this case is located in Cytosol, while `gene->` refers to a specific gene in the Chromosome. All processing by active objects makes use of these two types of data, data that they know about because of the two types of message exchange that occur during initial configuration.

```
1. Irreversible, 1 Substrate, 1 Product, 0 Activator, 0 Inhibitor, 0 Coenzyme
2. case Irr_Sb1_Pr1_Ac0_In0_Co0:
3.  s = sm->molecule[gene->substrateId[0]].get();
4.  nTimes = enzymeLevel * ((gene->substrateV * s) / (gene->substrateK + s));
5.  sm->molecule[gene->substrateId[0]].dec( nTimes );
6.  sm->molecule[gene->productId[0]].inc( nTimes );
7.  break;
```

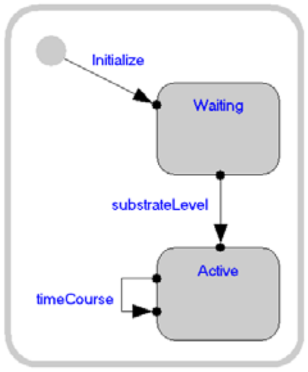

Figure 5: Enzyme state machine

The gene in CellAK is encoded as a set of features that includes protein kinetic constants. For example, in the code above, `gene->substrateV` refers to V the upper limit of the rate of reaction, and `gene->substrateK` is the Michaelis constant K_m that gives the concentration of the substrate molecule s at which the reaction will proceed at one-half of its maximum velocity.

The Gepasi software package [4] performs the same processing using ODEs. Gepasi implements irreversible Michaelis-Menton kinetics, which is implemented on line 4 of the code above in our model.

3.5 Validation

The main focus in CellAK has been on a qualitative model, but this approach also provides quantitative results which very closely approximate those computed using Gepasi, a tool that does claim to produce accurate quantitative results. In addition to the practical value of having CellAK generate accurate results, these also help to validate its design and implementation.

A simplified Glycolytic Pathway model was run in parallel using CellAK and Gepasi. The model includes the ten standard enzymes of glycolysis, and the eleven standard substrate and product metabolites ([15]. 308). All enzymes are implemented as irreversible, and there are no activators, inhibitors or coenzymes. Nine of the enzyme reactions convert one substrate into one product. The sole exception is the fourth enzyme reaction (Aldolase) that converts one substrate (Fructose-1,6-biphosphate) into two products (DihydroxyacetonePhosphate and Glyceraldehyde-3-

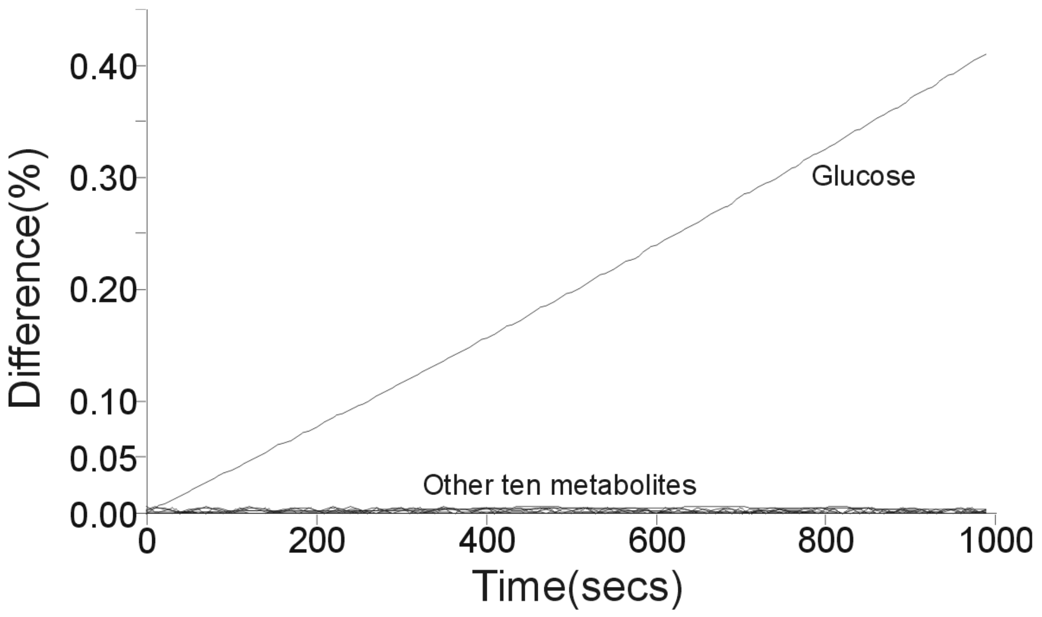

Figure 6: Percentage Difference between Gepasi and CellAK Results

phosphate). The units of time in both models are seconds, but more realistically should be thought of simply as discrete timesteps.

The results of this experiment are shown in Figure 6. Initially there are 1000000 units of each metabolite. Over the course of the simulation, during 1000 timesteps, for ten out of the eleven metabolites, the difference between the CellAK and Gepasi results is never greater than 0.005%.

There is continuously more Glucose in the CellAK model with the passage of time than in the Gepasi version. In CellAK the cell bilayer constantly replenishes the amount of Glucose in the cytosol by transporting it at a low rate from the extra cellular solution. This low rate, as currently implemented, is not sufficient to keep the Glucose quantitty constant in the cytosol. In both the Gepasi and CellAK results, the Glucose level decreases from 1000000 to around 900000 (900160 Gepasi, 903893 CellAK) after 1000 seconds.

4 Conclusions

This paper has described a modeling approach and tool, CellAK, developed using principles from agent-based modeling that is suitable for application to sophisticated cell modeling. We have demonstrated the validation of the model against Gepasi. The visual nature of the tool is considerably simpler to understand when compared to conventional differential equation based models and, being container based, can more effectively support system level models proposed by Tomita. We believe that this paper clearly confirms the value of agent-based modeling reported in [8]. Further, we have reused several of the classes and protocols in models of neurons with considerable success; research that is reported elsewhere.

Clearly other modeling work is possible. Other active objects in CellAK (polymers) are also composed of repeating units of monomers. Becker [15 p.30] states that there are three major types of polymers in a cell. This suggests a general principle. Active objects have an influence on other active objects in CellAK by having an effect on their constituent monomers. This enhancement should now be implemented for enzymes, transport proteins, and other proteins in CellAK. However, proteins are considerably more complex than lipid bilayers. The amino acids that constitute a protein are coded for in the DNA, the order of amino acids is of critical importance, and the string of amino acids folds into a three-dimensional shape. The behavior of a protein is therefore an extremely complex function of its fine-grained structure. A more tractable problem is found in the interactions of proteins with each other, such as when one protein regulates (activates or inactivates) another protein through the process of phosphorylation [15 p.158], which involves a relatively simple reversible structural modification (a change in the fine-grained structure of another protein). The approach described in this paper could be applied relatively easily to the modeling of networks of such interacting proteins.

References

1. Hucka, M., et al., 2003. The systems biology markup language (SBML): a medium for representation and

exchange of biochemical network models. Bioinformatics 19, 524-531.

2. Hedley, W., et al., 2001. A short introduction to CellML. Philosophical Transactions - Mathematical Physical and Engineering Sciences 359, 1073-1089.
3. Tomita, M., et al., 1999. E-Cell: software environment for whole-cell simulation. Bioinformatics 15, 72-84.
4. Mendes, P., 1997. Biochemistry by numbers: simulation of biochemical pathways with Gepasi 3. Trends. Biochem. Sci. 22, 361-363.
5. Sauro, H., 2000. JARNAC: a system for interactive metabolic analysis. Animating the Cellular Map 9th International BioThermoKinetics Meeting. University Press, ISBN 0-7972-0776-7.
6. Morton-Firth, C., Bray, D., 1998. Predicting Temporal Fluctuations in an Intracellular Signalling Pathway. Journal of Theoretical Biology 192, 117-128.
7. Loew, L., Schaff, J., 2001. The Virtual Cell: a software environment for computational cell biology. TRENDS in Biotechnology 19, 401-406.
8. Khan, S, et al., 2003. A Multi-Agent System for the Quantitative Simulation of Biological Networks. AAMAS'03, 385-392.
9. Gonzalez, P., et al., 2003. Cellulat: an agent-based intracellular signalling model. BioSystems 68, 171-185.
10. Harel, D., 1987. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming 8, 231-274.
11. Harel, D., 1988. On Visual Formalisms. Communications of the ACM 31, 514-530.
12. Harel, D., 2003. A Grand Challenge for Computing: Full Reactive Modeling of a Multi-Cellular Animal. LNCS 2623, 2-2.
13. Kam, N., Harel, D., et al., 2003. Formal Modeling of C. elegans Development: A Scenario-Based Approach. LNCS 2602, 4-20.
14. Selic, B., Gullekson, G., Ward, P., 1994. Real-time Object-Oriented Modeling. John Wiley & Sons, New York.
15. Becker, W., Reece, J., Poenie, M., 1996. The World of the Cell, 3rd ed. Benjamin/Cummings, Menlo Park, CA.
16. Harvey, I., Bossomaier, T., 1997. Time Out of Joint: Attractors in Asynchronous Random Boolean Networks. ECAL97.