UML as a Cell and Biochemistry Modeling Language[1]

Ken Webb

B.A. (Honours) Cognitive Science
Carleton University
Canada

Tony White[*]

School of Computer Science
Carleton University
Canada

Abstract

The systems biology community is building increasingly complex models and simulations of cells and other biological entities, and are beginning to look at alternatives to traditional representations such as those provided by ordinary differential equations (ODE). The lessons learned over the years by the software development community in designing and building increasingly complex telecommunication and other commercial real-time reactive systems, can be advantageously applied to the problems of modeling in the biology domain. Making use of the object-oriented (OO) paradigm, the Unified Modeling Language (UML) and Real-time Object-Oriented Modeling (ROOM) visual formalisms, and the Rational Rose RealTime (RRT) visual modeling tool, we describe a multi-step process we have used to construct top-down models of cells and cell aggregates. The simple example model described in this paper 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 relevance of abstraction, reuse, objects, classes, component and inheritance hierarchies, multiplicity, visual modeling, and other current software development best practices. We show how it is possible to start with a direct diagrammatic representation of a biological structure such as a cell, using terminology familiar to biologists, and by following a process of gradually adding more and more detail, arrive at a system with structure and behavior of arbitrary complexity that can run and be observed on a computer. We discuss our CellAK (Cell Assembly Kit) approach in terms of features found in SBML, CellML, E-CELL, Gepasi, Jarnac, StochSim, and Virtual Cell.

Keywords: agent-based modeling, UML, cell simulation

[1] Carleton University Cognitive Science Technical Report 2003-05
URL http://www.carleton.ca/iis/TechReports

[*] Corresponding author. Tel.: 613-520-2600 x2208 E-mail address: arpwhite@scs.carleton.ca

November 2003

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 the process of 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 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) (Hucka *et al.*, 2003a; Hucka *et al.*, 2003b), and CellML (Hedley *et al*., 2001), and implemented using E-CELL (Tomita *et al.*, 1999), Gepasi (Mendes, 1993; Mendes, 1997), Jarnac (Sauro, 2000), StochSim (Morton-Firth & Bray, 1998), Virtual Cell (Schaff *et al*., 2000; Loew & Schaff, 2001; Slepchenko *et al*., 2002), 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. 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.

All of these approaches listed in the previous paragraph 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 (Khan, *et al*., 2003) and they lack the ability to deal with non-equilibrium solutions. Further, differential equation-based models are difficult to reuse when new details are added to the model. CellAK more closely resembles Cellulat (Gonzalez *et al*., 2003) 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). The dynamics of a CellAK model result from messages passing between active objects. Agent-based modeling of cells is becoming an area of increasing research interest (Gonzalez *et al*., 2003; Khan, *et al*., 2003) owing in no small measure to the desire to understand cellular processes at an increasing level of detail.

This paper describes a process that starts with the identification of biological entities and their relationships with each other, progresses through the gradual addition of details, and ends with an executable program that simulates biochemical pathways. This relatively simple process can be used to model any chemical-like system that involves active objects transforming and moving passive small molecules, such as cells, a circulatory system, neural circuits, or organisms. We believe this process to be superior to other modeling approaches owing to its use of standard techniques from software engineering, the visual nature of the modeling process and the significant potential for reuse of the model components.

The remainder of the paper is organized as follows. Section 2 introduces object-oriented concepts by discussing a eukaryotic cell. Section 3 introduces UML, formalizing the example introduced in section 2. Section 4 describes the principal concepts behind the Real Time Object Oriented Methodology (ROOM). Having described ROOM, section 5 provides details of a process used in CellAK for cell modeling. Section 6 provides an extended discussion of the model created, contrasting it with prior art. In section 7, future work is described and the paper concludes with key messages in section 8.

2. Object-Oriented (OO) Paradigm

The process of software development has evolved considerably in the last 20 years. The current more generally accepted paradigm for commercial software development is called the object-oriented approach, which includes OO analysis and design methodologies, and OO programming languages such as Java and C++. OO replaced the earlier imperative paradigm in which a computer program was thought of as a system of procedures calling other procedures.

An OO object is a software entity that encapsulates or hides its own internal details or attributes. The scope of an internal attribute is such that it is only known within the object rather than at the global level as was typically the case with the older paradigm. For example, an Organelle object should not allow any other object in the system, such as EukaryoticCell or other instances of Organelle, to directly manipulate its private internal structure. Objects are a way of breaking the system into a manageable set of modules that can be developed and tested by individuals, and integrated into a larger system being developed by a team. Some objects are typically contained within other objects, resulting in a containment hierarchy.

The OO concept of class allows multiple instances of the same type of object to be created. For example, a cell model may need many instances of the Organelle class. A class is defined once and reused as many times as needed. This creates abstractions that can be reused as parts of larger abstractions, and also allows for multiplicity (multiple instances of a class).

Subclasses allow developers to explicitly capture what two objects have in common through inheritance from a superclass, as well as how they are different. Mitochondrion and Chloroplast objects, two subclasses of Organelle, both encapsulate a functionality that can be used by EukaryoticCell, but that differs in the details.

3. Unified Modeling Language (UML)

Starting in the late 1980s various individuals and software development communities developed their own graphics-based methods for object-oriented analysis and design. These methods became increasingly necessary as computer systems became more and more complex, and it was no longer possible to simply sit down at a computer workstation and start entering lines of code in some computer programming language. In the mid 1990s Grady Booch, Jim Rumbaugh, and Ivar Jacobson merged their slightly different approaches into a common Unified Modeling Language (UML). The UML standardization process is managed by the Object Management Group (OMG) (OMG, 2003). It is standard practice in the computer industry to present analysis, design, and implementation models of a system, using the UML common visual notation (Booch *et al*., 1998).

Figure 1 shows a UML class diagram that specifies the simple system used as an example in the previous section on the OO paradigm. The connecting line with unfilled triangle from the Mitochondrion and Chloroplast subclasses to Organelle is the symbol for inheritance in UML. Any number (multiplicity of 0..*) of Mitochondrion and Chloroplast objects can be created within EukaryoticCell. The connecting line with filled diamond from Organelle to EukaryoticCell is the symbol for containment in UML. Functions associated with classes are also defined; e.g. generateATP() in the Mitochondrion class.

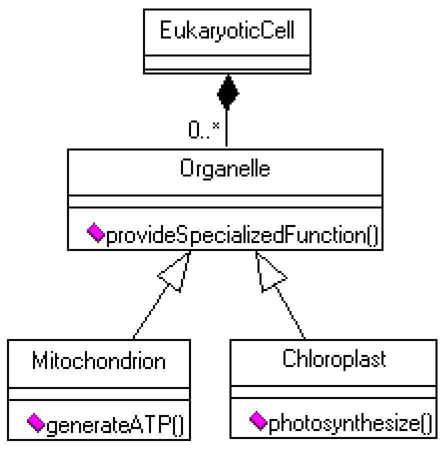


Figure 1 **- Simple example system.**

UML is starting to be used to a limited extent within the biology community. The Systems Biology Markup Language (SBML) specification documents use many UML diagrams to formalize the SBML data structures (Hucka *et al*., 2003).

There are three main advantages to using UML as a basis for defining SBML data structures. First, compared to using other notations or a programming

> language, the UML visual representations are generally easier to grasp by readers who are not computer scientists. Second, the visual notation is implementation-neutral: the defined structures can be encoded in any concrete implementation language—not just XML, but C or Java as well. Third, UML is a de facto industry standard that is documented in many sources. Readers are therefore more likely to be familiar with it than other notations. (Hucka *et al*, 2003, p.3)

However, although biotechnology tools are being written in OO programming languages such as Java and C++, and although some tools such as Virtual Cell (NRCAM, 2003) and E-CELL (Takahashi *et al*., 2002, p.68) are expressing their OO software design using UML, the end user interfaces of these systems do *not* make use of OO and UML concepts. Because cells and other biological entities naturally exhibit the principles embodied in OO and UML, at least when viewed from a top-down perspective, it makes sense to use these computer approaches when modeling cells.

As efforts go forward to develop whole-cell (Tomita, 2001) and other increasingly complex models containing multiple compartments, biologists will encounter many of the same issues such as scalability that led the software development community to develop and use graphics-based formalisms such as UML.

This paper will present examples of a number of UML diagram types and visual notations used in these diagrams. This paper does not present a comprehensive review of UML, providing only sufficient information to clarify modeling concepts and diagrams.

4. The ROOM formalism and the Rational Rose RealTime tool

David Harel, originator of the hierarchical state diagram (statecharts) formalism used today in UML (Harel, 1987), and an early proponent of visual formalisms in software analysis and design (Harel, 1988), has argued that biological cells and multi-cellular organisms can be modeled as reactive systems using real-time software development tools (Harel, 2002; Kam, Harel *et al*., 2003). Two such commercially-available tools are I-Logix Rhapsody (I-Logix, 2003), and

Rational Rose RealTime (Rational Software, 2003), the latter being used to implement the system described in this paper.

> Reactive systems are those whose complexity stems not necessarily from complicated computation but from complicated reactivity over time. They are most 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. (Kam, Harel *et al.*, 2003, 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) (Selic *et al.*, 1994). 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 objects, using UML class diagrams. Each architectural object, 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. 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 objects' 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.

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. This naturally mirrors the view understood by the biologist, making the model far more accessible when compared to a differential equation-based formulation. 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.

The developer generates the executing system by making a selection from a menu. RRT generates all required code from the diagrams, and produces an executable 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 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. Models are much more accessible to non-mathematicians using this formalism.

To summarize, benefits of the CellAK methodology 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, object 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. Examples of capsules, protocols, ports, and the various diagrams and concepts mentioned in this section, will be provided in subsequent sections of this paper.

5. Process

This paper will present a simple process that has been used to develop cellular models. The process has five iterated steps. Each step has a number of sub-steps. These steps and the principles behind them have been adapted loosely from a number of computer industry sources (Quatrani, 1998; Kruchten, 2000).

The design methodology presented here is top-down rather than bottom-up, but as we will see the run-time dynamics can be much more bottom-up. A cell is considered as an entity that consists of various compartments, each containing active objects that act chemically on various types and quantities of small molecules. An active object is defined here as a RRT capsule that acts in a biologically-plausible manner on some substrate molecule or set of substrates, possibly located in multiple compartments. Each compartment may contain other compartments to any arbitrary depth.

Abstraction is an important principle of software development. Start by identifying entities that exist in the application domain, in this case cellular biology. For example, think in terms of membranes, enzymes, organelles, and small molecules rather than computer-centric processes. These are the types of entities that would appear in a cell biology textbook (Becker, 1996). Consider how these entities relate to each other. At this point, pay minimal attention to considerations of how these will actually be implemented in lines of software code. Start with a high level of abstraction, and gradually add detail until the final concrete system is ready to be executed. UML allows an initial application-level model (such as at the level of biology) to be gradually evolved into a design, and then a programming language implementation.

The five-step CellAK process described here has been successfully used to develop models and executing simulations of cells and of cell aggregates. A simple cell model is used to motivate the discussion for each step. This process can be used to develop models and simulations using a variety of software development languages and tools.

Step 1: Identify entities, inheritance and containment hierarchies

The first step can be divided into three sub-steps:

1. Identify entities in the problem domain (biology),
2. Identify inheritance relations between these entities, and
3. Identify containment relations between them.

The purpose of the small example system described here will be 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 should also include 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 should also be extensible, to allow for specialized types of cells.

Figure 2 shows a set of candidate entities organized into an inheritance hierarchy, drawn as a UML class diagram using RRT. These are only candidate entities because further analysis may uncover other entities that should be included, or some of these may prove unnecessary. The lines with a triangle at one end are standard UML notation 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.

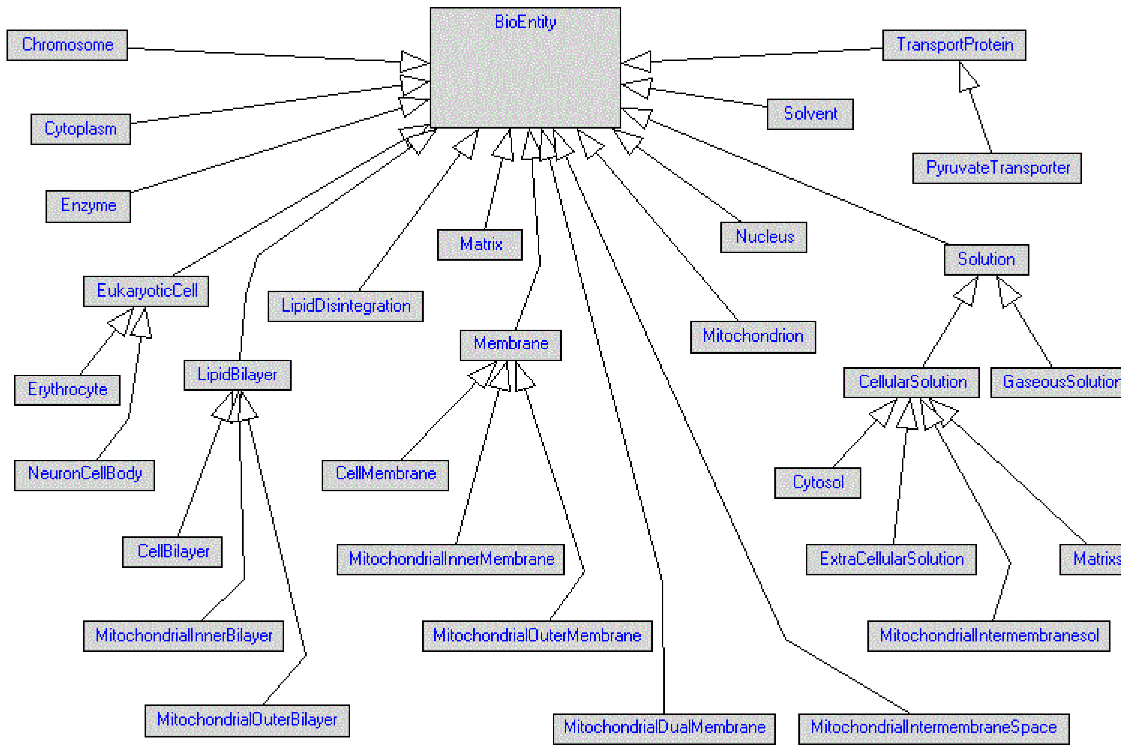


Figure 2 **- Set of entities organized into an inheritance UML class diagram.**

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. This will make it possible in a later design stage for instances of each class to share programming code such as the ability to display information about themselves, or the ability to be scheduled at some regular interval. For now these are just potentialities we are setting up by making everything a subclass of BioEntity.

Figure 3 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,

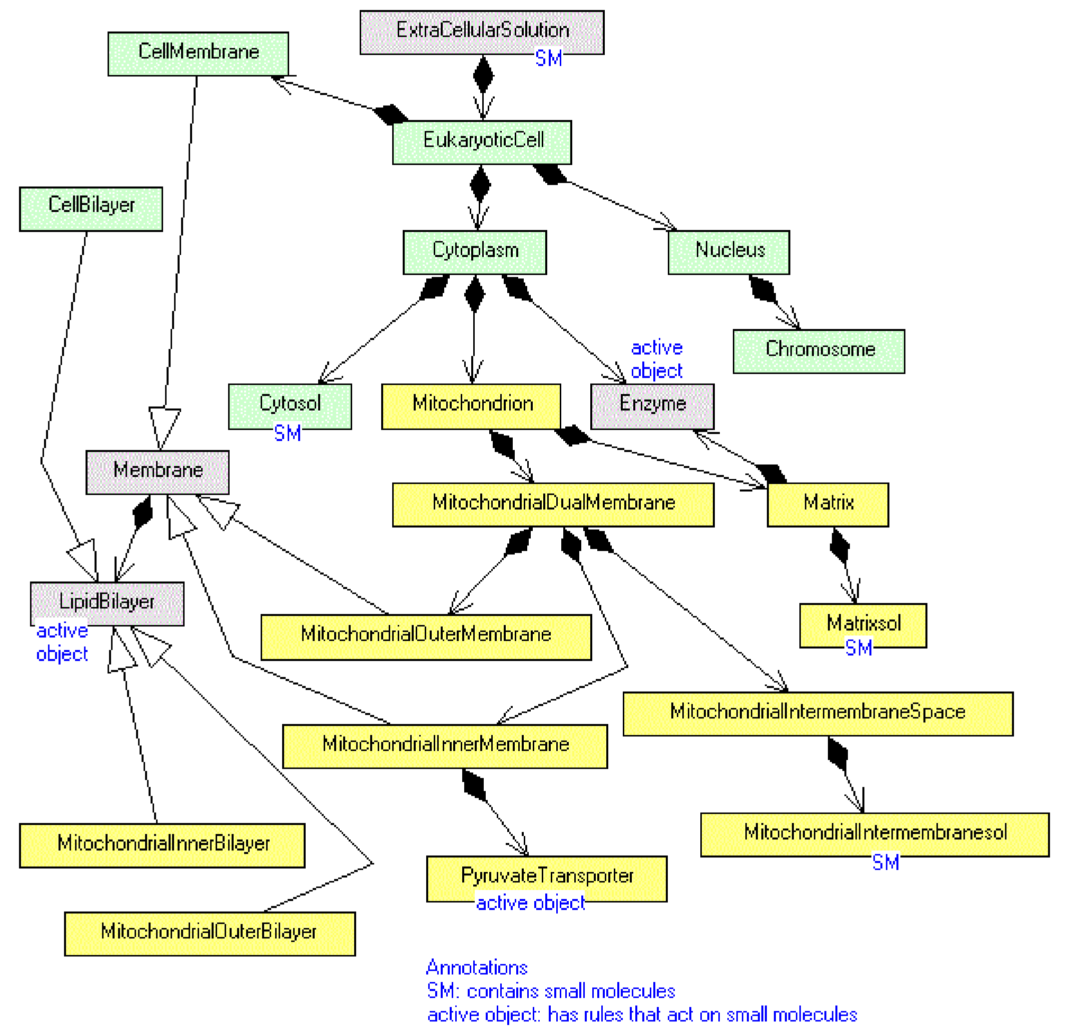

ExtraCellularSolution
SM
CellMembrane
EukaryoticCell
CellBilayer
Cytoplasm
Nucleus
Chromosome
active object
Cytosol
SM
Mitochondrion
Enzyme
Membrane
MitochondrialDualMembrane
Matrix
LipidBilayer
active object
MitochondrialOuterMembrane
Matrixsol
SM
MitochondrialIntermembraneSpace
MitochondrialInnerMembrane
MitochondrialInnerBilayer
MitochondrialIntermembranesol
SM
PyruvateTransporter
active object
MitochondrialOuterBilayer
Annotations
SM: contains small molecules
active object: has rules that act on small molecules

Figure 3 - **The containment hierarchy as a UML class diagram.**

MitochondrialOuterBilayer) depends on which type of membrane (CellMembrane, MitochondrialInnerMembrane, MitochondrialOuterMembrane) it is contained within.

The UML class diagram, Figure 3 has been annotated with text to show which entities (the four Solution subclasses identified in the inheritance hierarchy) will contain small molecules (SM) such as glucose, pyruvate, and the other substrates and products of the metabolic pathways that are part of this simulation. Small molecules are captured in the model as a pair of passive (non-capsule) classes (SmallMolecules containing multiple instances of SmallMolecule, one instance for each type such as glucose or pyruvate). The three entity types identified as active objects (Enzyme, PyruvateTransporter, LipidBilayer) will act on the small molecules to create a dynamic metabolism. That dynamics will be described in step 4.

Figure 4 shows a set of ROOM capsule structure diagrams that present the same information as in the UML diagram of Figure 3, but laid out as a series of nested rectangles. The software developer creates a new entity by dragging and dropping existing entities from a list in a browser to a space that represents the new container. For example, to create the Cytoplasm class requires dragging instances of Cytosol, Mitochondrion, and Enzyme into the rectangle that represents Cytoplasm.

There will typically be many enzyme types active at the same time, so a multiplicity factor nEnzPerCyt (number of enzymes per cytoplasm) is declared, the specific value of which can be delayed until later. The model includes several other multiplicity factors including the number of EukaryoticCell instances in ExtraCellularSolution (nEukCell), the number of Mitochondrion instances in Cytoplasm (nMito), and the number of Enzyme types in Matrix (nEnzPerMatrix). Multiplicities can range from 0 to hundreds or even thousands of instances.

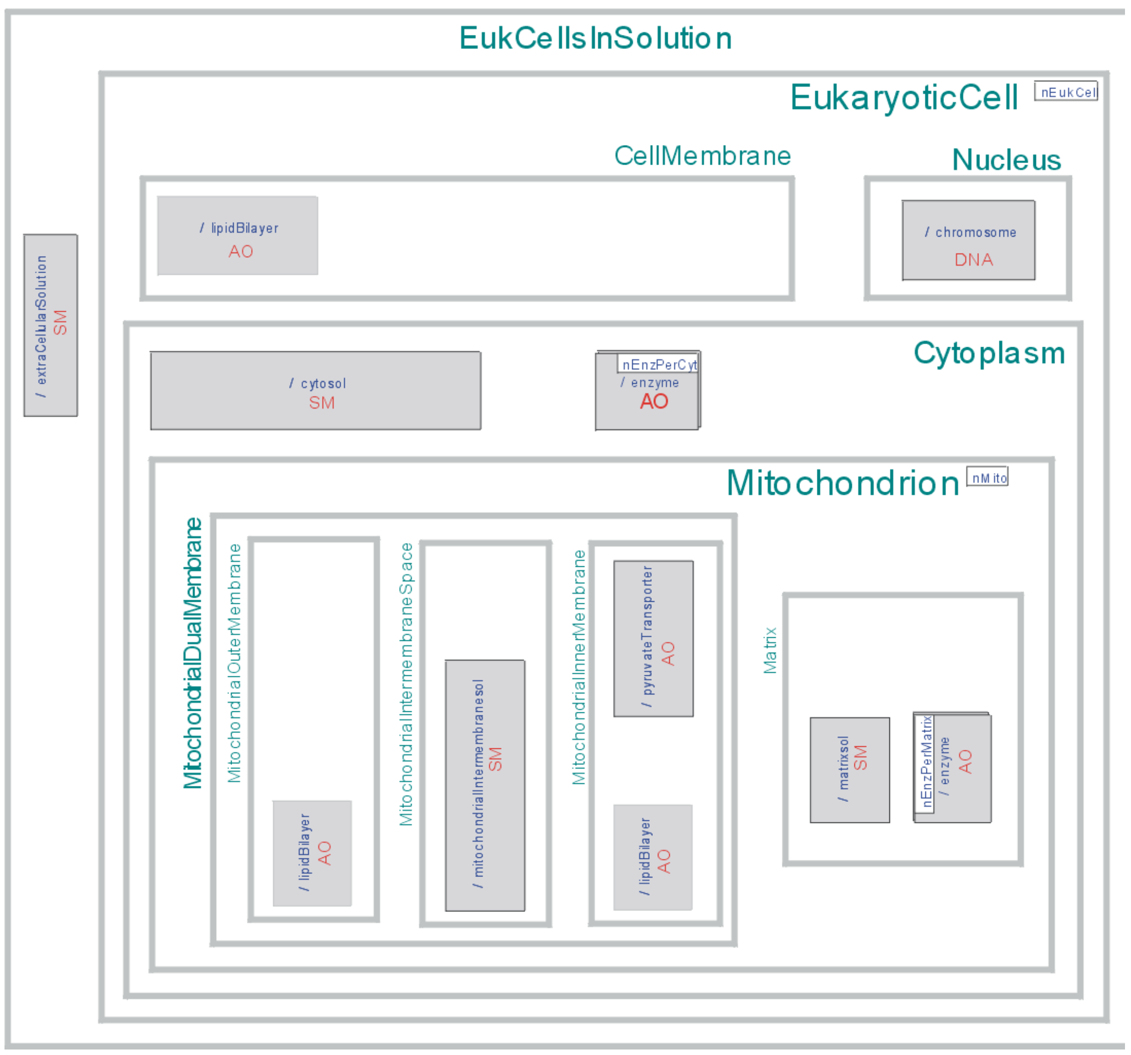


Figure 4 - **The containment hierarchy as a set of ROOM capsule structure diagrams**

The result of step 1 is often called a Domain Model, especially if it incorporates a large number of entities belonging to one domain, in this case biology, that can later be used to build many separate models.

Once the architectural structure is in place, the more fine-grained structure of the small molecules can be specified, again using UML. In CellAK, each type of small molecule is an instance of the Substrate class (a C++ class rather than a RRT capsule) which contains a count of the number of molecules of that molecule type (from 0 to 10^{15}), and also contains operations to increase (`inc(amount)`) and decrease (`dec(amount)`) the count by a designated amount and to get (

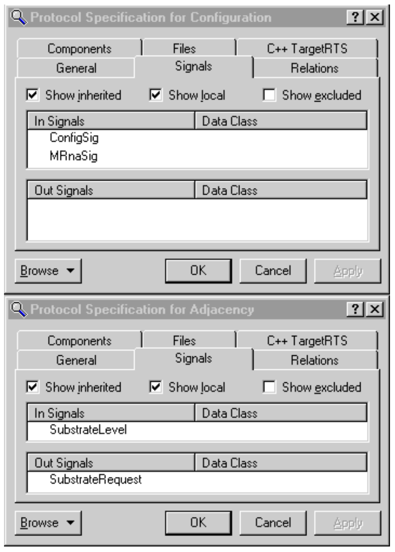
`(get()`) and set (`set(amount)`) the current value of the count. A separate SmallMolecules class includes an array or dictionary of all the possible types of small molecules that may be found in a CellAK model.

Step 2: Establish relationships between entities

The second step involves several sub-steps, which would typically be done in parallel:

1. Identify adjacency and other relationships between capsules (relationships not identified in step 1),
2. Identify and specify protocols (interaction types between entities),
3. Create ports (instances of protocols) on capsules, and
4. Connect ports using connectors.

This step establishes the adjacency structure of the biological and chemical entities in the system, and their potential for interaction. 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. In many cases the static layout defined in step 1 suggests which entities will interact, but not in all cases. For example, within MitochondrialInnerMembrane, both LipidBilayer and PyruvateTransporter are adjacent and could in theory interact with each other, but this will not be allowed in the simple simulation described here. 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.



Protocol Specification for Configuration

☑ Show inherited ☑ Show local ☐ Show excluded

In Signals	Data Class
ConfigSig	
MRnaSig	

Out Signals	Data Class

Protocol Specification for Adjacency

☑ Show inherited ☑ Show local ☐ Show excluded

In Signals	Data Class
SubstrateLevel	

Out Signals	Data Class
SubstrateRequest	

Figure 5 – **Configuration and Adjacency Protocols.**

A protocol is a specific set of messages that can be exchanged between capsules to allow interaction. Figure 5 is a RRT dialog that shows the two protocols used in the sample system. 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. The contents of this message is a reference to a structure, specific to this cell, that defines the genome, the quantities of the various small molecules, and other starting conditions. 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 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 over 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 (to be described in step 4) 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 6 is a RRT capsule structure diagram that shows EukaryoticCell and its three contained capsules with named ports and connector lines between these ports. The ports are added by dragging from the protocol symbol in a browser window to the capsule structure diagram. The ports whose names begin with *adj* are instances of the Adjacency protocol, while *config* and *configM* are instances of the Configuration protocol. The color of the port (black or white) indicates the relative direction (in or out) of message movement.

Figure 7 shows the final result once all protocols, ports and connectors are in place. This figure continues the step-by-step progression that has led from identifying biological entities, through organizing these entities into inheritance (Figure 2) and containment (Figure 3) hierarchies, creating capsule structure diagrams (Figure 4), identifying adjacency and genetic configuration relationships and specifying these as protocols (Figure 5), to creating instances of these protocols as ports and connecting the ports using connectors. The structural architecture of the system is now complete. By following the series of connector lines between capsules in Figure 7, you can confirm which entities are in an adjacency relationship with which other entities. For example, the lipidBilayer and pyruvateTransporter capsules within MitochondrialInnerMembrane are both adjacent to the mitochondrialIntermembranesol and matrixsol capsules.

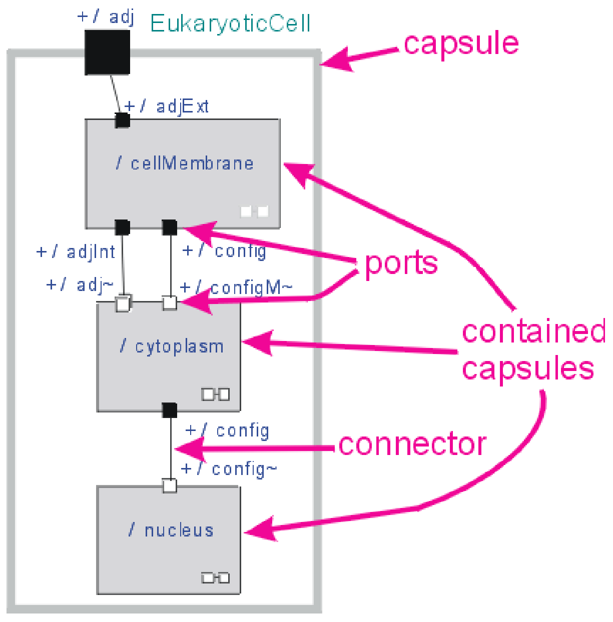


Figure 6 - **The three contained capsules within the EukaryoticCell capsule structure diagram. The capsules are connected through ports that are instances of the two protocols shown in Figure 5.**

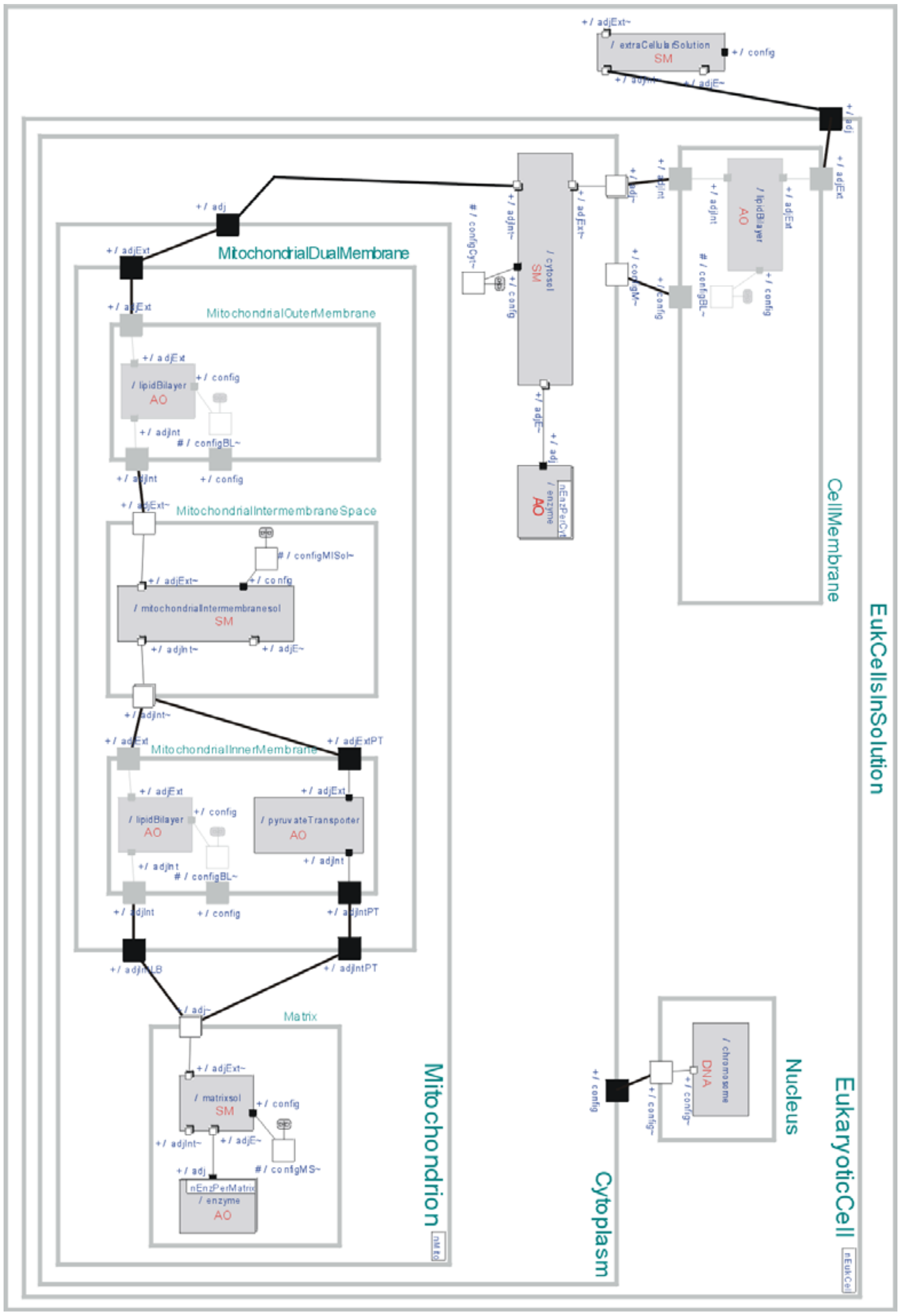


Figure 7 - **The complete structure of the sample model, with all capsules, ports, and connectors. This is an enhancement of Figure 4, with additional details added.**

Step 3: Define external and internal behavior patterns

The third step adds behavior to the existing structure.

1. Define the desired behavior of the system by specifying patterns of message exchange between capsules, and
2. Define the detailed behavior of each capsule using state diagrams, the combined effect of which will produce this desired overall pattern of message exchange.

Figure 8 is a UML sequence diagram that shows the adjacency configuration processing that would be expected to occur in the small system described in this paper. Capsule instances are shown at the top of the diagram, annotated with AO (active object) or SM (container for small molecules). When it starts up, each active object sends a SubstrateRequest message out each of its adj ports (instances of the Adjacency protocol). If a port is connected to a capsule such as a Solution that contains small molecules, that capsule will respond with a SubstrateLevel message containing a reference to its small molecule data structure. Each enzyme (enzyme_1 to enzyme_N), plus cellBilayer and mitochondrialOuterBilayer, sends a SubstrateRequest to Cytosol. Cytosol responds with SubstrateLevel messages containing the reference pSM. Time in a sequence diagram is represented by the thin line pointing downward from each capsule instance.

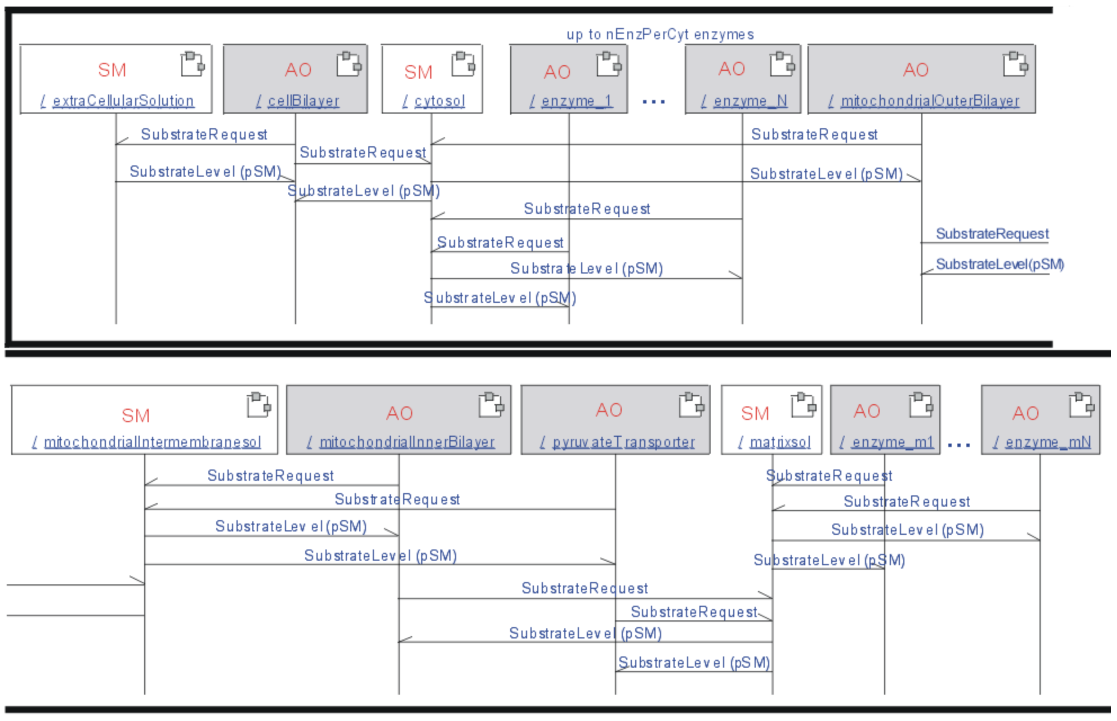


Figure 8 **- Adjacency configuration as a UML sequence diagram.The diagram is split into two parts so it will fit on the page.**

The resulting reference structure for the entire system is shown in Figure 9. This is exactly the same diagram as Figure 7, with two types of configured reference structures superimposed, one representing adjacency, and the other representing the influence of genes. LipidBilayer and pyruvateTransporter both point to the small molecule data structures within both mitochondrialIntermembranesol and matrixsol. Because there are multiple instances of EukaryoticCell, Mitochondrion, and Enzyme, some of the pointers have double arrow heads to graphically represent this multiplicity. The instances of LipidBilayer also point to an internal small molecule data structure that contains the lipids that they are composed of, allowing for lipid creation in the Cytoplasm, lipid transport, and disintegration to be modeled.

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.

Figure 9 also shows the influence of the genes. Each enzyme, transporter, and other protein, is configured to reference a detailed description of its functionality. The description is contained within a table of gene data residing within the Chromosome object inside the Nucleus.

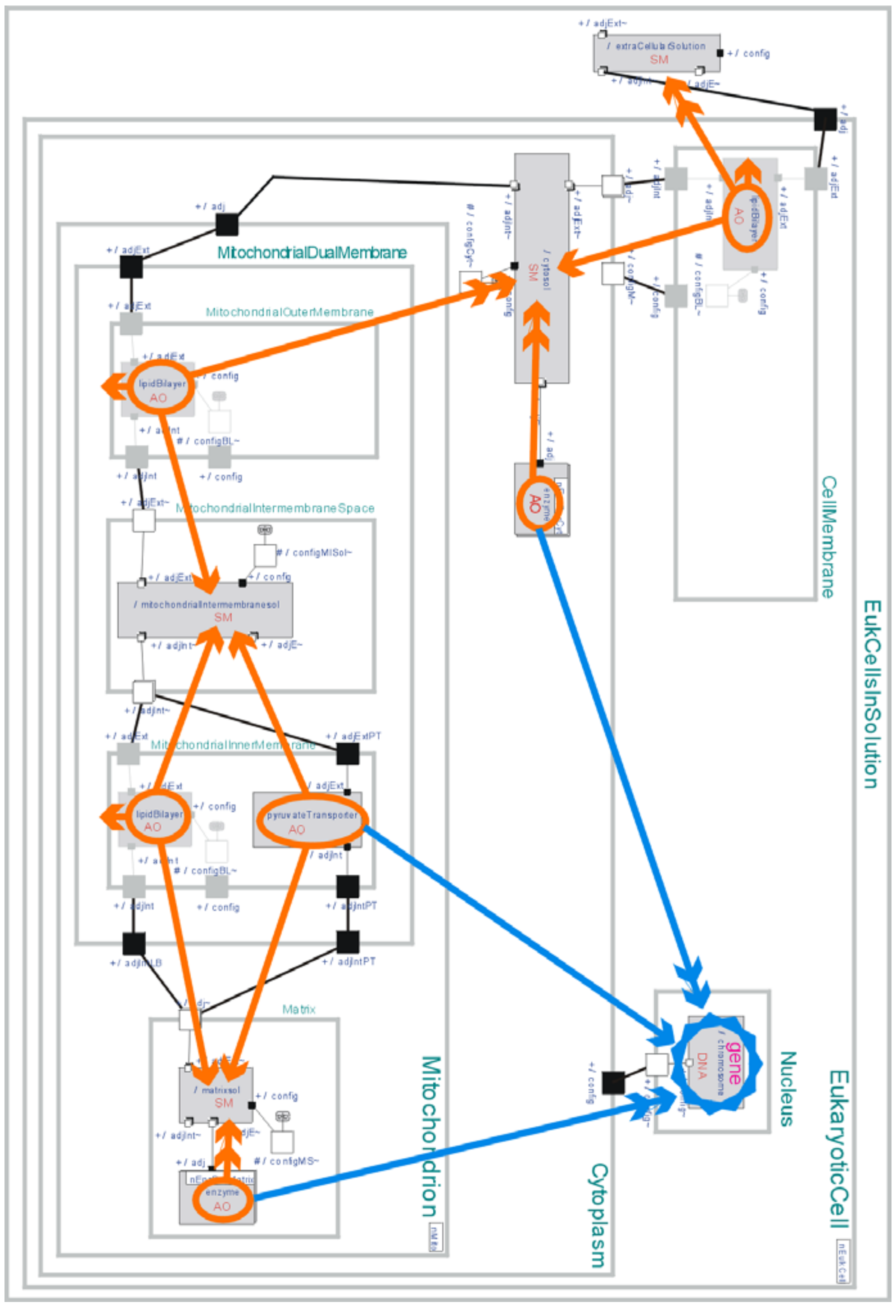


Figure 9 - **Structure of the entire system with references from active objects (AO) to compartments that contain small molecules (SM), and to the genes that reside at the Chromosome.**

Figure 10 shows the UML state diagram representing the behavior of an Enzyme active object. When first created, it makes the initialize transition, the line from the large grey circle in the upper left to the Waiting state. As part of this transition it executes a line of C++ code `adj.SubstrateRequest().send();` that sends a message out its adj port. When it subsequently receives a SubstrateLevel response message through the same adj port, it stores the pSM 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.

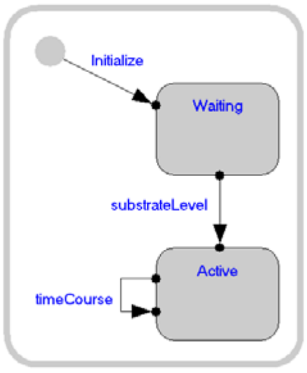


Figure 10 - **Enzyme behavior drawn as a UML state diagram.**

Step 4: Implement detailed behavior

The fourth step involves adding C++ programming language code to the state diagrams. The two types of configuration, adjacency and gene configuration, come together at runtime as each active object repeatedly times out at discrete intervals (the timeCourse transition shown on **Figure 10**) to perform its simple processing.

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. These more complex reaction types have been implemented in CellAK using the same approach as shown in **Figure 11**, but are not discussed further in this paper.

In the C++ code in Figure 11, which implements irreversible Michaelis-Menten kinetics (Becker, 1996, p.148+; Mendes, 2003) `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 11 **- One of many possible Enzyme reaction types, as implemented in C++.**

The gene in CellAK is encoded as a set of features that includes protein kinetic constants. For example, in **Figure 11**, `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 (Mendes, 1997) performs the same processing using ODEs. Gepasi implements irreversible Michaelis-Menton kinetics by operating on the following formula (Mendes, 2003):

$$v = \frac{V * S}{K_m + S}$$

where *v* is the amount of change in the quantity of the substrate and product, and *S* is the initial quantity of substrate. This formula is implemented on line 4 of Figure 11.

The reaction rules could be as simple or as complex as needed. Because RRT can incorporate existing C, C++ or Java code, the reaction rules could make use of existing code from other biochemistry modeling tools. The combined action of multiple reaction rules over time results in the two metabolic pathways that are part of the simple example model. These are the glycolytic pathway and the TCA cycle.

In larger models using CellAK, containing 1000+ cells and varying numbers of organelles, the timeCourse transition shown in **Figure 10** has been replaced with a simple scheduler implemented in C++. Each active object registers with the scheduler as part of its substrateLevel transition, and is subsequently directly invoked at a regular interval.

Step 5: Validate

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 (Becker, p. 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-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 12**. 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 reults is never greater than 0.005%.

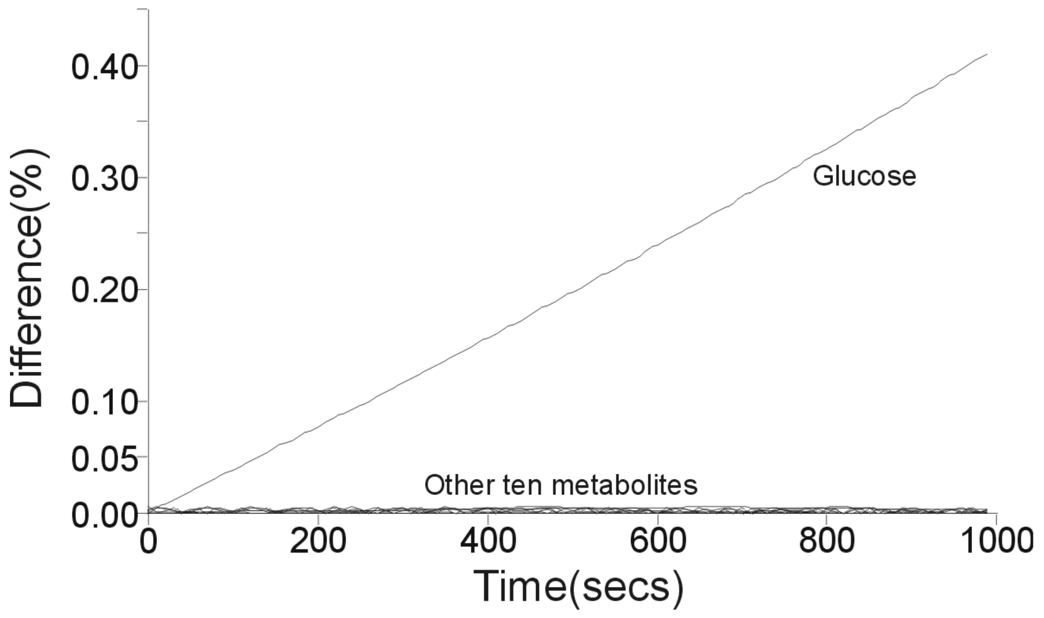


Figure 12 – **Percentage Difference between Gepasi and CellAK Results**

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.

6. Discussion

Any model or simulation of a cell must take into account two separate architectures. The first is the top-down containment structure - the membranes, the solutions such as Cytosol, the small molecules, and the active objects such as enzymes. The other architecture is the bottom-up behavior - the dynamic reactions between molecules, and the rules and parameters that define these reactions.

OO, UML, ROOM and RRT make a fundamental distinction between structural modeling and behavioral modeling of computer systems (Selic *et al*., 1994; Harel, 2001, p.55). Feitelson and Treinin (2002, p.34) suggest a biological parallel when they state that "the particulars of many cellular structures seem not to be encoded in DNA, and they are never created from scratch; rather, each cell inherits templates for these structures from its parent cell." Some cellular structure (a hierarchy of cell and organelle compartments with collections of small molecules) must exist before the DNA-encoded enzymes and other proteins can carry out their behavioral work. This structure represents a slow evolution over many millions of years, and it also represents the constraints that physics and chemistry enforce on the otherwise open-ended realm of possibilities. The behavior comes about through a process of Darwinian evolution and involves the genes. This is reflected in the need to separately specify both structure (capsules, compartments, components; and objects, molecular species, small molecules) and behavior (state machines, operations, rules, reactions) in the startup configuration data for all the computer systems mentioned in this paper. Table 1 shows the terminology employed by each of these systems. In each case, behavior operates on objects (fine-grained structural elements) within a structural architecture.

System	**Structure Architectural**	**Structure Fine-grained**	**Behavior**
UML, ROOM/RRT	capsule structure	attributes, data objects	state machine behavior, operations
CellAK	capsule hierarchies	small molecule objects	active object operations, gene-specified parameters
SBML	compartments	species	reactions, rules
E-CELL	cell components	substances	reaction rules
Gepasi	compartments	metabolites	reactions, kinetics
Jarnac	compartments	species nodes	reactions
Virtual Cell	regions	species	reactions, fluxes
CellML	components, groups	variables	reactions, math

Table 1 - The terminology used by the various computer systems described in this paper differs somewhat, but they all share the same underlying concepts, including a fundamental distinction

between structure and behavior, and a secondary distinction between structural architecture and fine-grained structure.

Each approach mentioned in this paper has mechanisms to deal with each of these architectures, and with how the two come together at run-time. CellAK, with its OO, UML and RRT roots, focuses primarily on the structural architecture, but does provide a simple mechanism to simulate the time evolution of any number of chemical reactions occurring concurrently within and between an arbitrary number of compartments. Tools such as Gepasi, with their roots in biochemistry and differential equation modeling, focus primarily on the moment-by-moment time evolution of the behavioral architecture, but also provide varying amounts of support for aspects of the complex structural architecture. In CellAK, reaction modeling is implemented using message passing and Michaelis-Menton kinetics are coded directly into the message processing behaviour. Interactions between biological elements are, therefore, directed, as messages are passed from one active object to another. This makes it possible to model very detailed, localized phenomena in the cell, which are extremely difficult to capture in a differential equation representation. Modeling using message passing also makes it straightforward to introduce new components to the containment hierarchy, as existing interactions are unaffected by the introduction. This is considerably harder with a differential equation representation as extreme care must be taken when adding, removing or modifying terms in the dynamical equations for the system. Stated another way, CellAK separates containment and interaction; differential equations do not.

The scope or namespace mechanism is standard in software development. An object (or attribute) called pyruvate can be simultaneously used within two or more parts of the system, and refer to a different entity in each case. If there are pyruvate molecules in Cytosol, MitochondrialIntermembranesol, and Matrixsol, all three can simply be called pyruvate and do not need to be distinguished by using separate names such as pyruvateCyt, pyruvateInt, and pyruvateMtrx. In an OO system each object is automatically scoped, has its own namespace. This is not true with many of the biochemistry modeling tools. E-CELL, Gepasi, Jarnac and StochSim do not appear to support a scope or namespace mechanism. All entities are globally scoped and belong to the same global namespace. All four of these tools do allow for multiple compartments, but the molecular species within each compartment must be given separate names. None of these tools can be said to support objects in the OO sense of entities that encapsulate or hide their contents.

The SBML and CellML specifications are neutral on the issue of scoping and namespaces. Models can be represented in either of these two markup language syntaxes, with the same molecular species name such as pyruvate appearing in more than one compartment. The semantics of the modeling tool that reads the SBML or CellML file may allow for separate namespaces, or it may concatenate the species and compartment names to derive globally unique names.

None of the biochemistry modeling tools support the OO concepts of class and inheritance. There is no mechanism to define a type or class of entity such as Mitochondrion, and then make use of it in more than one part of the model by simply naming its type. CellML does provide a simple class-like reuse mechanism through its ability for one file to import another file. None of the modeling tools provide a concept of subclass, or of a multiplicity of objects of the same type.

CellAK, as an OO system, does make use of classes, static and dynamic instantiation of objects from classes, subclasses, and multiplicity. Once a class such as Enzyme has been designed, any number of instances of this class (objects) can be created within Cytoplasm and Matrix. At run-time each instance of Enzyme can be configured to be a different type of enzyme, new enzymes can be dynamically created as conditions in the simulation change (through molecular signaling, MRNA transcription from DNA, and ribosomal translation to protein), enzymes can be dynamically destroyed as they age, and entities can be dynamically reconfigured (connected to each other, disconnected, and reconnected to other entities). If 100 instances of Mitochondrion are defined within Cytoplasm, each will automatically, because of its class definition, point to the common small molecule data structure in Cytoplasm and to its own small molecule data structure in MitochondrialIntermembranesol. Pyruvate will move to each of these internal compartments from the common Cytoplasm, according to the kinetic rate constant defined within the MitochondrialOuterBilayer subclass of LipidBilayer.

A system with 1000 cells each containing 10 instances of Mitochondrion is as easy to run in an OO system such as CellAK as in a system with 1 cell and 1 (or 0) Mitochondria. E-Cell, for example, with its named components, requires considerably more effort when changing configurations in the way described in the previous sentence.

The use of visual programming in CellAK makes the generated models far more accessible to biochemists and biologists. Physical structures are naturally represented in a containment hierarchy and interactions are explicitly represented using ports and protocols. The alternative differential equation representation hides interactions in terms in equations, which are significantly more obscure for the non-mathematician. With interactions “hidden”, it makes model reuse more difficult. Reuse is explicitly encouraged with CellAK.

One way to use CellAK is to develop a library of components with standard interfaces using the protocols described in this paper. It then becomes possible to rapidly develop new models and simulations by plugging a selection of components together.

A number of complex simulations have been constructed to test the concepts presented in this paper. We will briefly discuss the implementation of a circulatory system, of a circuit of neurons, and of an ecology. All of these simulations can be thought of as test harnesses, separate environments to test the original EukaryoticCell and the concepts discussed in this paper.

In the CirculatorySystem object, instances of a subclass of EukaryoticCell called Erythrocyte move within a structure of Vein, Artery, Capillary (three subclasses of BloodVessel), and various Heart objects, all implemented as RRT capsules. Each of the 100 or so erythrocytes (each within its own unit of BloodPlasma) is continuously reconfigured to refer successively to a cyclical series of external spaces including Lung (a compartment containing a high level of oxygen and low level of carbon dioxide small molecules), DigestiveSystem (a compartment containing high levels of glucose), and Brain (a compartment containing low levels of oxygen and glucose, and high levels of carbon dioxide). The reaction rules and kinetic rate constants defined for the CellBilayer class determine the relative concentrations of oxygen, carbon dioxide and glucose within the BloodPlasma, Lung, DigestiveSystem and Brain compartments during the time evolution of the simulation. The entire cell structure described in this paper, as illustrated in Figure 9, was reused without alteration and embedded within each of the 100 cells in the CirculatorySystem simulation object.

This ability to embed an object created for one simulation within a larger simulation is possible because of the mechanisms provided by OO, UML and RRT, and by the simple SubstrateRequest - SubstrateLevel protocol (interface) that all active objects and small molecule compartments have in common (see Figure 5 and Figure 8).

Another test involved EukaryoticCell specialized as a NeuronCellBody, as part of a Neuron. Neuron and Synapse objects were combined to form a neural circuit that could transmit neural (chemical and electrical) messages. The resulting Brain, a separately developed DigestiveSystem containing EukaryoticCell subclasses called MucosalCell, and CirculatorySystem were combined into a HumanBeing (subclass of Animal) simulation. HumanBeing was subsequently combined with Plant, Atmosphere, Lake and Sun to produce a simulation of a simple ecology.

7. Future Work

CellAK is currently implemented using a proprietary commercial toolset, Rational Rose RealTime (RRT). The architectural approach and process described in this paper should be ported to a non-proprietary environment, which would make it more accessible to a larger community.

Specific configurations, such as that shown in Figure 9, are currently hardcoded using RRT. It should be possible to read in and write out a configuration using an XML specification, preferably either SBML (Hucka *et a l.*, 2003b) or CellML (Hedley *et al.*, 2001). This would allow model exchange with tools such as Gepasi, thus helping to automate the validation process. It is our view that the creation of an interface to the System Biology Workbench would also be beneficial.

CellAK, as described in this paper, is a top-down approach, in keeping with the process normally followed when using the OO paradigm, the UML and ROOM visual formalisms, and the RRT toolset. CellAK configurations could instead be implemented in a more generic way, to allow them to be manipulated using bottom-up evolutionary mechanisms. This becomes possible if the approach described in this paper is reinterpreted as the following alternative five steps:

1. Manipulate the containment hierarchy by representing it explicitly as a tree, and then operating on it using evolutionary mechanisms (Koza, 2001). Although the RRT containment hierarchy is already tree-like, it cannot be dynamically manipulated at run-time. An inheritance hierarchy could optionally also be implemented to provide names for node types, and to help constrain the range of configuration and runtime behaviors available for each node type.

2. Create additional lateral connections between nodes in the tree. Replace the RRT mechanism of static ports, protocols and connections with a set of algorithms that nodes could follow to dynamically configure themselves by navigating the tree structure to locate adjacent nodes. These would be the nodes with which they would subsequently dynamically interact.

3 and 4. Provide primitives from which active objects could evolve simple behaviors such as the enzyme behavior shown in **Figure 11**. Allow higher-level patterns of interaction including metabolic and other networks to emerge from local interactions, rather than having these prespecified.

5. Validate.

8. Conclusion

Simulation tools for non-mathematicians using visual modeling and programming are needed in order to increase the accessibility of modeling in biology. CellAK is a step towards the realization of Harel's "full reactive modeling" and Tomita's "whole cell simulation". This paper has suggested that biochemical modeling and simulation tools will need to provide a richer set of mechanisms for top-down structural architecture if they truly wish to construct whole-cell simulations. It is our hypothesis that the approaches currently in use by the software development community, the OO paradigm, the UML methodological formalism, and the specifics of tools such as RRT, can be advantageously applied to the problem of whole-cell simulation. A hybrid approach would combine the most successful top-down architectural mechanisms used within the commercial software industry, with proven bottom-up algorithmic mechanisms provided by the biochemistry modeling community. This paper has proposed a tentative five-step process for the development of such hybrid systems.

References

Becker, W., Reece, J., Poenie, M., 1996. The World of the Cell, 3rd ed.. Benjamin/Cummings, Menlo Park, CA.

Booch, G., Rumbaugh, J., Jacobson, I., 1998. The Unified Modeling Language User Guide. Addison-Wesley, Reading, MA.

Feitelson, D., Treinin, M., 2002. The Blueprint for Life?. Computer, July 2002, 34-40.

Gonzalez, P., et al., 2003. Cellulat: an agent-based intracellular signalling model. BioSystems 68, 171-185.

Harel, D., 1987. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming 8, 231-274.

Harel, D., 1988. On Visual Formalisms. Communications of the ACM 31, 514-530.

Harel, D., 2001. From Play-In Scenarios to Code: An Achievable Dream. IEEE Computer, Jan 2001, 53-60.

Harel, D., 2002. A Grand Challenge for Computing: Full Reactive Modeling of a Multi-Cellular Animal. . In Workshop on Grand Challenges for Computing Research, Edinburgh, Scotland, November 2002. http://www.wisdom.weizmann.ac.il /~dharel/papers/GrandChallenge.doc

Hedley, W., et al., 2001. A short introduction to CellML. Philosophical Transactions - Mathematical Physical and Engineering Sciences 359, 1073-1089.

Hucka, M., et al., 2003. Systems Biology Markup Language (SBML) Level 1: structures and facilities for basic model definitions. http://www.cds.caltech.edu/erato

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.

I-Logix, 2003. I-Logix Rhapsody and Statemate. http://www.ilogix.com.

Kam, N., Harel, D., et al., 2003. Formal Modeling of C. elegans Development: A Scenario-Based Approach. In Proceedings of Computational Methods in Systems Biology: First International Workshop, CMSB 2003, Rovereto, Italy, February 24-26, 2003, LNCS 2602, 4-20.

Khan, S, et al., 2003. A Multi-Agent System for the Quantitative Simulation of Biological Networks. AAMAS'03, 385-392.

Koza, J., et al., 2001. Reverse Engineering of Metabolic Pathways from Observed Data using Genetic Programming. PSB 2001, 434-445.

Kruchten, P., 2000. The Rational Unified Process: An Introduction (2nd Edition). Addison-Wesley, Reading, MA.

Loew, L., Schaff, J., 2001. The Virtual Cell: a software environment for computational cell biology. TRENDS in Biotechnology 19, 401-406.

Mendes, 2003. Gepasi 3.30. Available via the World Wide Web at http://www.gepasi.org.

Mendes, P., 1993. GEPASI: a software package for modelling the dynamics, steady states and control of biochemical and other systems. Comput. Appl. Biosci. 9, 563-571.

Mendes, P., 1997. Biochemistry by numbers: simulation of biochemical pathways with Gepasi 3. Trends. Biochem. Sci. 22, 361-363.

Morton-Firth, C., Bray, D., 1998. Predicting Temporal Fluctuations in an Intracellular Signalling Pathway. Journal of Theoretical Biology 192, 117-128.

NRCAM, 2003. The Virtual Cell. http://www.nrcam.uchc.edu.

OMG, 2003. Unified Modeling Language (UML). http://www.omg.org/uml.

Quatrani, T., 1998. Visual Modeling with Rational Rose and UML. Addison-Wesley, Reading, MA.

Rational Software, 2003. Rational Rose RealTime. http://www.rational.com/products/rosert.

Sauro, H., 2000. JARNAC: a system for interactive metabolic analysis. http://www.sys-bio.org

Schaff, J., et al., 2000. Physiological Modeling with Virtual Cell Framework. Methods in Enzymology 321, 1-22.

Selic, B., Gullekson, G., Ward, P., 1994. Real-time Object-Oriented Modeling. John Wiley & Sons, New York.

Slepchenko, B., et al., 2002. COMPUTATIONAL CELL BIOLOGY: Spatiotemporal Simulation of Cellular Events. Annual Review of Biophysics and Biomolecular Structure 31, 423-442.

System Biology Workbench, http://www.sbw-sbml.org.

Takahashi, K., et al., 2002. Computational Challenges in Cell Simulation: A Software Engineering Approach. IEEE Intelligent Systems, Sept/Oct 2002, 64-71.

Tomita, M., 2001. Whole-cell simulation: a grand challenge of the 21st century. Trends in Biotechnology 19, 205-210.

Tomita, M., et al., 1999. E-Cell : software environment for whole-cell simulation. Bioinformatics 15, 72-84.