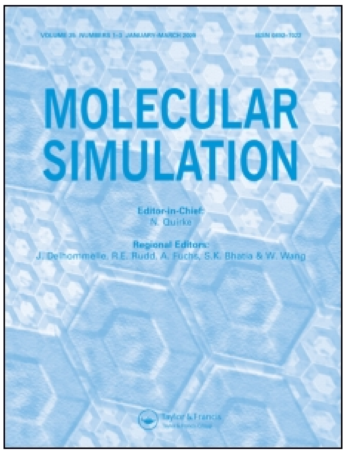


Molecular Simulation
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713644482

DEVS modelling and simulation of the cellular metabolism by mitochondria

Gabriel Wainer[a]; Roxana Djafarzadeh[b]
[a] Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada [b] School of Information Technology and Engineering, University of Ottawa, Ottawa, ON, Canada

Online publication date: 03 November 2010

To cite this Article Wainer, Gabriel and Djafarzadeh, Roxana(2010) 'DEVS modelling and simulation of the cellular metabolism by mitochondria', Molecular Simulation, 36: 12, 907 — 928
To link to this Article: DOI: 10.1080/08927022.2010.481793
URL: http://dx.doi.org/10.1080/08927022.2010.481793

Molecular Simulation
Vol. 36, No. 12, October 2010, 907–928

DEVS modelling and simulation of the cellular metabolism by mitochondria

Gabriel Wainer[a]* and Roxana Djafarzadeh[b1]

[a]*Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, ON, Canada K1S 5B6;* [b]*School of Information Technology and Engineering, University of Ottawa, 800 King Edward Avenue, Ottawa, ON, Canada K1N 6N5*

(*Received 10 August 2009; final version received 28 March 2010*)

We present a method for modelling and simulating metabolic pathways in the cells (namely, the glycolysis and Krebs' cycle), using the discrete event system specification (DEVS) formalism. The hierarchical nature of DEVS makes it ideal for describing naturally hierarchical systems as the Cell, while its discrete-event approach improves performance due to the asynchronous nature of the events involved. DEVS time-based nature can adequately represent the timing of the chemical reactions. We show how this methodology enables creating a precise and easy way to model and simulate biological systems, including advanced visualisation of the experiments. The results presented, which focus on the simulation of the cellular metabolism pathways in mitochondria, show the potential of our approach.

Keywords: DEVS; CD++; mitochondria; glycolysis; Krebs' cycle

1. Introduction

Simulation is becoming increasingly important in the analysis and design of complex systems such as those involving biological processes. The application of simulation to biological systems aims at developing methods to represent and analyse complex biological phenomena. In general, research in this field focuses on problems involving varied phenomena at many different levels, whose complexity makes impossible to use traditional mathematical representations. In general, biological systems can be seen as composed of many subsystems and components, each having its own unique characteristics and behaviour that contributes to the overall form and function of the whole. Moreover, these systems are highly complex, exhibiting many simultaneous interactions and strong non-linearity. Computer simulation provides adequate means to study these problems, as one can focus on the analysis of particular experimental conditions.

The reasons for building such models and running simulations include the need to replicate the function of living organisms, in order to test the current knowledge about them, and allowing the study of conditions that are difficult (or even impossible) to create experimentally. Researchers in systems biology stray from the traditional reductionist objective of identifying simple causes and effects, and focus instead on the detailed behaviour of biological systems.

Traditionally, the simulation of biological systems posed many technical challenges, including the selection of accurate model parameters, the validation of simulation results and the optimisation of simulation code for computational efficiency. On the one hand, modelling and simulation methods can extend the solution space well beyond what is possible with theoretical and purely analytical solutions. On the other hand, researchers and practitioners have to constantly seek the best trade-off between simulation accuracy and computational load.

Nevertheless, it is also important to design the simulation software properly. Modellers should be able to easily understand their code, modify it and have confidence that the desired computations are described. This is difficult for two reasons. First, the systems are often complex, and, as a consequence, models of those systems tend to become complex. Second, realistic simulations may require the integration of multiple complex algorithms. Several algorithms would be necessary if one wished to simulate, for example, the deformation of a cell membrane, surrounded by reacting and diffusing chemicals, in a changing electric field [1].

It has been shown that this complexity can be better dealt with if one builds models of biological systems that can be designed as hierarchies [2,3]. In this case, the components of the hierarchy would typically reflect real-world entities in the organism. Based on these ideas, numerous methods have been proposed.

Some of the research in this field has focused on the creation of advanced tools with application to particular problems [for instance, the E-Cell project [4], the Silicon Cell [5], the CyberCell Database (CCDB) [6], etc.]. Other research in this field has focused, instead, on the utilisation of advanced modelling and simulation methodologies applied to this kind of problems. For instance, in [7], the authors propose a new method to simulate coupled

*Corresponding author. Email: gwainer@sce.carleton.ca

ISSN 0892-7022 print/ISSN 1029-0435 online
© 2010 Taylor & Francis
DOI: 10.1080/08927022.2010.481793
http://www.informaworld.com

chemical reactions using both stochastic and non-Markov processes; in [8], Gillespie proposes to simulate a single trajectory of a chemical system using a stochastic algorithm; in [9], numerous methods in this area are presented and discussed. In [3], the authors show the results of simulation experiments of mammary duct formation using Cellular Automata [10], and in [11], a swarming environment is used to show how blood flows through liver lobules. Various teams have used Cellular Automata [12–14].

In this work, we show how one can deal with the above two categories: we present an advanced modelling methodology for the field of systems biology (which permits that a modeller can work with advanced models, maintaining their software application and making easy changes and running with high performance). On the other hand, we show how the methodology can be applied to build domain-specific basic libraries that experts could use for experimentation. The library presented here is focused on models applied to the study of mitochondria. In this way, researchers in any field could modify and improve the models in the library, and use them for different applications. To do so, we show various models to understand and control dynamics of mitochondrial metabolism through computer simulation (the idea is to be able to do this at the organelle scale).

Our solution is based on the use of the discrete event system specification (DEVS) formalism [15]. DEVS provides a framework for the construction of hierarchical models in a modular fashion, allowing model reuse, and reducing development and testing time. The hierarchical nature of DEVS makes it ideal for describing naturally hierarchical systems as the Cell. Likewise, its discrete-event nature improves the execution performance due to the asynchronous nature of the events occurring in the cell. DEVS also uses explicit timing information; hence, we can adequately represent timing of the reactions occurring at different levels of abstraction. The advantages of DEVS have been thoroughly discussed in the literature. DEVS is compelling because it separates models from simulators, and because it provides a framework with which models can be defined as hierarchies of interacting submodels [16–18] (a thorough discussion of the advantages of DEVS is presented in Section 2).

As we will show, the complexity, multiple layers of behaviour and local confinement of the behavioural elements make DEVS extremely effective. The definition of these models using the DEVS formalism brought two major benefits: *effectiveness* (models that can be understood, while improving quality) and *performance* (the discrete-event nature of DEVS makes time to advance in a discontinuous and irregular way depending on the next relevant event, which is adequate for models in biology). The approach presented here also allows users to construct advanced model visualisations, which is an important aspect for systems biology that is usually dismissed or postponed. Nevertheless, visual information can provide the biology practitioners with better ways of validating the models, moreover considering that in this field the data are usually scarce, and new theories can be derived from observation. As shown in [19], one can easily integrate independent DEVS simulations with advanced 3D visualisation engines. We explain how these models can be applied to the study of mitochondria, in such a way that the researchers in this field could modify and improve them, and use them for different applications. We chose the mitochondrion, as it fulfils different important roles in cellular metabolism [20], and a number of genetic diseases (including diabetes, deafness, heart, Alzheimer's, Parkinson's etc.) are associated with mutations in mitochondrial DNA. Studying the mitochondrion also affects other fields of interest, including the study of apoptosis and forensic science [21]. For this purpose, we built a mitochondrial model that included two biological pathways. The model puts emphasis on cellular metabolism and energy production aspects.

2. Background

As discussed in Section 1, in recent years, there has been an increasing trend in using modelling and simulation of applications in systems biology. As seen in the literature of this area, some of the works have focused on the creation of advanced tools with application to a particular problem. Some examples of these advanced tools include:

- The E-Cell project [4] aims to simulate the whole cell, including not just metabolic pathways but also protein synthesis and signal transduction. The E-Cell system uses a set of reaction rules and initial values, and users can run simulations and observe dynamic changes in quantities and concentrations of intra- and extracellular metabolites and substances. The activities of biochemical reactions can be monitored and the amounts of substances can be altered by users during the simulation.
- The Silicon Cell project [5] models cellular metabolism and regulation, trying to capture the physical and chemical constraints in cells. Their *in silico* approach incorporates experimental data about individual components of organisms to derive accurate models of cellular metabolism.
- The CCDB [6] is a comprehensive, Web-accessible database designed to support and coordinate international efforts in modelling a cell of *Escherichia coli* (*E. coli*; a bacteria). The CCDB brings together observed and derived quantitative data from numerous sources (covering the genomic, proteomic and metabolic character of *E. coli*, strain K12). The database is self-updating but also

supports annotations by the community, and it provides an extensive array of viewing, querying and search options (including a powerful, easy-to-use relational data extraction system).

Instead, other groups have focused their research effort on how to improve and use advanced modelling and simulation methodologies applied to this kind of problems, for instance:

- In [7], the authors state two fundamental ways to view coupled systems of chemical equations: continuous (differential equations on the concentrations) or discrete (stochastic processes on the number of molecules), approximated by numerical simulation methods. One of such methods (the next reaction method) is proposed to simulate coupled chemical reactions using both stochastic and non-Markov processes.
- Gillespie [8] defined one of the most widely used techniques to date. His idea was to simulate a single trajectory of a chemical system using a stochastic algorithm, whose outcomes are equivalent to the ones obtained by the chemical master equation. The algorithm uses a number of molecules (and their reactions), and then generates the next reaction at random (using a probability proportional to the number of molecules). The algorithm makes the assumption that the system is equilibrium (all molecules are distributed in a uniform manner). The algorithm cannot be applied to numerous phenomena, and the performance is an issue; therefore, there have been numerous efforts addressing these problems.
- In [9], the authors present a complete survey and discussion on many different techniques applied in the field, ranging from the next sub-volume method (one of the techniques that extends Gillespie's approach for isolation of different conditions by membranes), up to a variety of formal methods: Markov processes, stochastic Petri nets, stochastic π-calculus, etc. In particular, they analyse the use of methods such as DEVS or Statecharts [22], which are usually employed for modelling the micro-level view on the molecules in the system (i.e. defining, as in this paper, a detailed description of the molecules' interactions). They argue that these techniques are not yet well adapted for representing stochastic simulations (whose effects and interactions cannot be expressed easily, moreover when thousands of particles are available) and further research in this field is needed.
- In [3], the authors show the results of simulation experiments of mammary duct formation using Cellular Automata [10] for approximation of a reaction–diffusion process. In [11], the authors combine a swarming environment to show how blood flows through liver lobules.
- Numerous researchers have used Cellular Automata as the basic modelling method. For instance, Cellular Automata were used for modelling tumour-immune systems [12], by creating a 2D cell space through which immune cells wander in search of a tumour. The authors show how to model a core of necrotic cells, surrounded by a ring of dormant cells, surrounded in turn by a ring of proliferative cells.

As mentioned in Section 1, we have explored the use of the DEVS formalism [15] to deal with these two kinds of issues at the same time: we want to explore the use of new methods in the field of biological studies, while constructing libraries that are ready-to-use by researchers in the field. As we will show in the following sections, DEVS allowed us to achieve these goals, while providing the means of creating advanced 3D visualisations to improve validation of the simulation results.

DEVS provides a framework for the construction of hierarchical models in a modular fashion, allowing model reuse, and reducing development and testing time. The hierarchical nature of DEVS makes it ideal for describing naturally hierarchical systems as the cell. Likewise, its discrete-event nature improves the execution performance due to the asynchronous nature of the events occurring in the cell. DEVS also uses explicit timing information; hence, we can adequately represent timing of the reactions occurring at different levels of abstraction. The advantages of DEVS have been thoroughly discussed in the literature. DEVS is compelling because it separates models from simulators, and because it provides a framework with which models can be defined as hierarchies of interacting submodels.

Our models have been created using DEVS, a system's theoretical approach that allows the definition of hierarchical modular models. Why DEVS? Because, as discussed earlier, this methodology has proved to be very successful, generic and it is gaining popularity. DEVS supports hierarchical and modular construction of models, reducing the development and testing effort. DEVS decouples the model, experiments and the simulation; thus, various experiments on the same model can be executed on different simulation engines (allowing for portability and interoperability at a high level of abstraction). A well-defined and formally proven separation of concerns permits models and simulators to be independently verified and reused in later combinations with minimal re-verification. Each DEVS model can be built as a behavioural (atomic) or a structural (coupled) model. A real system modelled using DEVS can be described as a set of atomic or coupled submodels. The atomic model is the lowest level and defines dynamics, while the coupled are structural models composed of one or more atomic and/or coupled models. An atomic DEVS model is defined as

$$M = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_{\text{a}} \rangle,$$

where M is an atomic DEVS model defined by the following: X is the external input event set; S is the sequential state set; Y is the external output event set; δ_{int}: $S \rightarrow S$ is the internal transition function; δ_{ext}: $Q{*}X \rightarrow S$ is the external transition function, where $Q = \{(s,e) | s \in S$, and $e \in [0,t_{\text{a}}(s)]\}$ is the total state set and e is the elapsed time since last state transition; λ: $S \rightarrow Y$ is the output function; and t_{a}: $S \rightarrow R_0^+ \cup \infty$ is the time advance function.

Each atomic model can be seen as having an interface consisting of *input* (X) and *output* (Y) *ports* to communicate with other models. Every *state* (S) in the model is associated with a *time advance* (t_{a}) function, which determines the duration of the state. Once the time assigned to the state is consumed, an internal transition is triggered. At that moment, the model execution results are spread through the model's output ports by activating an *output function* (λ). Then, an *internal transition function* (δ_{int}) is fired, producing a local state change. Input external events are collected in the input ports. An external transition function (δ_{ext}) specifies how to react to those inputs.

A coupled DEVS model is formed by configuring several atomic models or coupled models, and it is defined as

$$\text{CM} = \langle X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, \text{ select} \rangle,$$

where CM is a coupled DEVS model defined by the following: X is the external input event set; Y is the external output event set; $D \in N$, $D < \infty$ is an index for the components of the coupled model; M_i: $\forall i \in D$, M_i is a basic DEVS model (an atomic or a coupled model), defined by

$$M_i = \langle X_i, S_i, Y_i, \delta_{\text{int}i}, \delta_{\text{ext}i}, \lambda_i, t_{\text{a}i} \rangle,$$

I_i is the set of influences of i and $\forall j \in I_i$ (the models that can be influenced by outputs of model i); Z_{ij}: $Y_i \rightarrow X_j$, $\forall j \in D$, Z_{ij} is the i to j output translation function; and select is a function, the tie-breaking selector.

Coupled models are defined as a set of basic components (atomic or coupled), which are interconnected through the models' interfaces. The models' coupling defines how to convert the outputs of a model into inputs for the others, and how to handle inputs/outputs from/to external models.

As discussed in [16–18], DEVS provides the following major advantages compared to other methodologies:

(a) Independency between modelling and simulation mechanisms, which allows DEVS models to be executed interchangeably in single-processor, parallel or real-time engines without changes. Likewise, the use of DEVS as the basic formal specification mechanism enables one to define interactions with models in other formalisms (i.e. PDE, Cellular Automata, Petri nets, finite elements, finite differences, etc.), and integration between these formalisms could be easily used for defining complex models with diverse methods [23]. This approach provides *evolvability* of the models through a technique that is easy to understand, and able to be combined with other techniques.

(b) DEVS enables high-performance execution due to its discrete-event nature. In [17,18], we showed varied experiments in which the application of DEVS can improve the execution performance (of both discrete and continuous models) in several orders of magnitude. Also, as the simulation engines are independent from the models themselves, we introduced a parallel simulation engine that can improve performance further without modifications to the model. It has been shown that DEVS combined with parallel simulation techniques can produce speedups of up to 1000 times [17,18]. This is particularly important for biological systems, in which timescales vary at the different levels of the modelled hierarchy (from thousands of reactions per second at the intracellular level up to seconds or minutes at the organelle level). In order to provide precise timing behaviour; traditional discrete-time simulators (such as those used for simulating PDEs) need to find a common timeslot for all the levels in the system in order to be able to activate every component in the model at the smallest possible rate.

(c) DEVS provides the advantages of being a formal approach: a formal conceptual model can be validated, improving the error detection process and reducing testing time (thus improving the quality and development costs of a simulation). DEVS provides facilities to translate the formal specifications into executable models, facilitating the verification of the simulator and the validation against the real system.

(d) The existence of an internal transition function is a unique feature that eases the definition of certain properties. Internal state changes can be captured, describing complex internal interactions in a simple and elegant way (in biological systems, it is good to have a mechanism to represent state changes without the influence of external factors). DEVS is also a time-based formalism, in which representing timing information in the models is straightforward. Although the state associated with any submodel may change only at discrete times, DEVS exploits an 'elapsed time' parameter to facilitate the representation of continuously changing system properties. This is valuable to represent the time information for the cell components. Modelling of these phenomena is difficult under other techniques.

(e) DEVS theory has been extended to be able to express hybrid systems (i.e. those with both discrete and continuous components). The theory can be applied to predictive quantisation of arbitrary ordinary differential equation models [24], represented as combined discrete-event/differential equation formalism approximated by DEVS.

Arguably the most compelling reason to use DEVS is that the formalism facilitates hierarchical model design. The separation of model and simulator is also a significant advantage, but the separation of different aspects of the model is the key to addressing the complexity of a biological system and the methods used to simulate it.

Despite the popularity of the DEVS formalism and the widespread use of simulation in the study of biological systems, the application of DEVS to biological models is still relatively rare. The results presented here show one of the first attempts. Two other groups (one led by Adelinde M. Uhrmacher at the University of Rostock, and a second one by Prof. A. Hunt at UCSF) have also applied DEVS to various models of biological systems. The following examples show some recent efforts that have used DEVS for modelling biological systems. In most of these examples, the authors used the DEVS software for defining a concrete application (instead of building a ready-to-use library).

For instance, in [11,25], the authors presented a DEVS model of liver lobules, which make up the main functional and structural component of the liver (which is comprised of thousands of them). When blood flows through a lobule, it undergoes several chemical reactions in multiple stages. In [11], the lobule was modelled like a hexagonal cylinder including multiple zones (as seen in Figure 1). There are three stages (zones), comprising several interconnected nodes, whose number is proportional to the approximated lobule volume of that particular zone. Each node is responsible for receiving a substance, and transforming it, and each node works interdependently of each other.

In [25], a DEVS model of the lobule was built based on these assumptions (which was easily defined as a DEVS coupled model following the structure presented in Figure 1). The model's behaviour was defined based on the equations introduced in [11], where each node has its own set of parameters to determine the output when given a certain input. Each node is given a delay to represent the time it takes for a substance reaction to reach completion.

In [25,26], a model of interaction of synapsin and vesicles was defined using DEVS as the underlying methodology. Synapsin is a neuron-specific phosphopro-

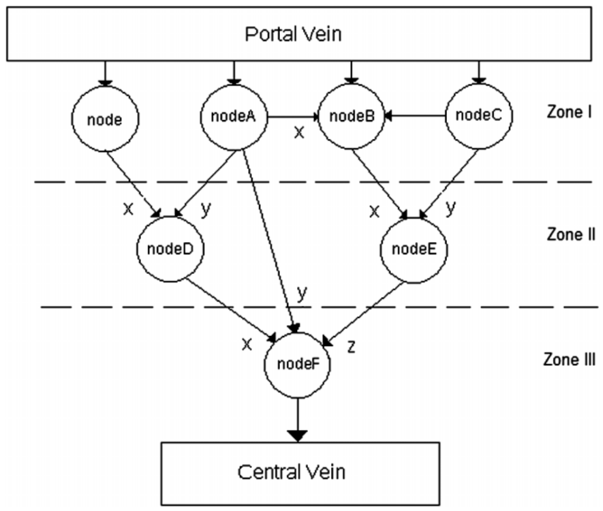


Figure 1. Zones and nodes.

tein that binds to small synaptic vesicles and actin filaments in a phosphorylation-dependent pattern. The idea was to model the reserve pool of synaptic vesicles, predicting the number of synaptic vesicles released from the reserve pool as a function of time under the influence of action potentials (APs) at differing frequencies. The model defines the molecular interactions of *synapsin* (S) with *vesicles* (V) which occur inside a nerve cell, and the behaviour of synapsin movements until reaching a vesicle and binding to it. Once binding has occurred, depending on *offrate*, V and S can break their bindings and separate. The *onrate* and *offrate* describe how often bindings occur or break then after. Different scenarios were modelled: (1) V is stationary (with a fixed position on cell space) and S is mobile; (2) V is mobile and S is stationary; and (3) V and S are both mobile (leads to a maximum number of total movements and therefore bindings). Binding patterns are in such a way that each S can bind to more than one V, and V can bind to more than one S. Examples of such binding are presented in Figure 2, which shows a stable image of synapsin–vesicles bindings where single/double/multiple bindings had occurred within the neuron.

In [27], an advanced DEVS model was used to improve the performance of the execution of AP functions. The model presented an approximation of Hodgkin–Huxley equations programmed as a DEVS atomic model (whose results are presented in Figure 3(a)). It then presented a polynomial approximation to the original PDE defining the cell's behaviour (Figure 3(b)), which transformed the coefficients in the polynomials into discrete-event signals using DEVS. Each cell used polynomial coefficients to compute the current state, and to inform the cell's state to the neighbouring cells in the heart tissue. The specification of the local computing function included in each of the cells will now receive the

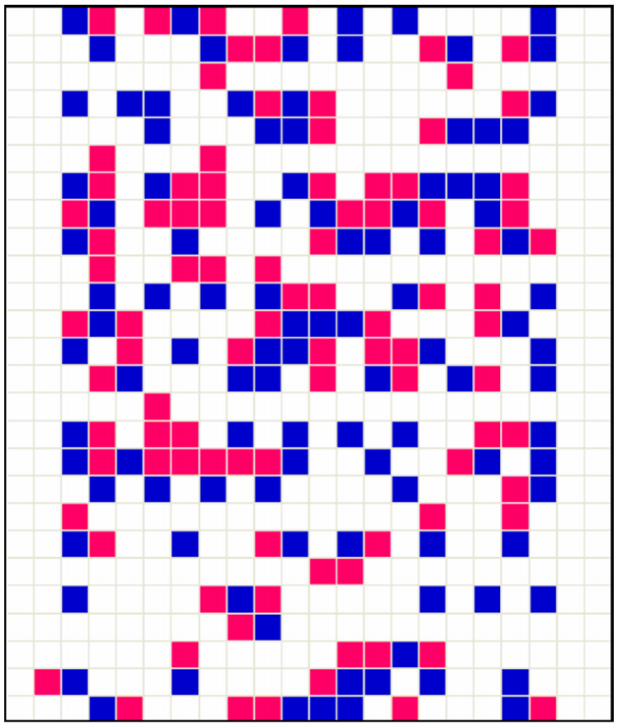


Figure 2. Vesicle–synapsin binding and clustering.

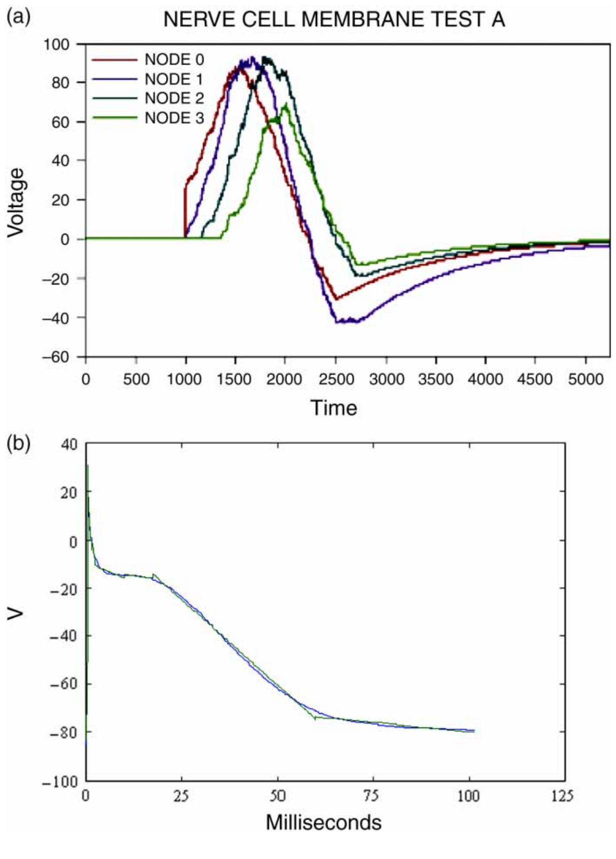


Figure 3. (a) AP approximation based on Hodgkin–Huxley equations and (b) DEVS approximation of heart tissue AP.

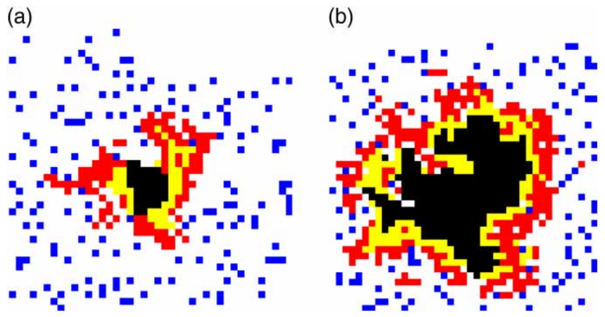


Figure 4. Tumour-immune system simulation: (a) immune cells attacking the tumour and (b) tumour proliferating.

coefficient events from the neighbouring cells. The cell's outputs will now be the current cell states specified as polynomial coefficients. Timing of activation for each polynomial can be easily defined using the model delay functions.

DEVS was also used to model tumour-immune systems that involve growing tumours interacting with immune cells [28], based on the work done in [12]. Tumours can be regarded as a core of necrotic cells, surrounded by dormant cells, surrounded in turn by proliferative cells. The growth of a tumour is effected by the division of its proliferative cells, but inhibited by nearby immune cells. The use of DEVS for defining this model was advantageous, as it facilitated the formal specification and reuse of cellular models. Simulation results indicated that, in a qualitative sense, the desired behaviour of tumours and immune cells was captured. This can be seen in Figure 4, which shows the simulation results for this system, in a case where the tumour overwhelms the immune system. In Figure 4(a), the immune cells have cleared away a large section of proliferative cells on the upper side of the tumour. Nevertheless, later, in Figure 4(b), the tumour has regained this region. It thereafter proceeded to expand towards the boundaries of the cell space.

Although these works show the feasibility of using DEVS for modelling and simulation of biological systems, they are all focused on particular models. The rest of the paper focuses on how to define a library using the open-source DEVS software that can be easily modified and reused. We focus on the design aspects, the integration with visualisation engines, and on the creation of domain-specific models. We show the multiple advantages of the DEVS formalism, permitting the reader to understand how these models could be created (using the CD++ toolkit, which implements DEVS theories [29,30]). We explain how these models can be applied to the study of mitochondria, in such a way that the researchers in this field could modify and improve them, and use them for different applications.

Mitochondria are small double-membrane organelles found in the cytoplasm of eukaryotic cells. Mitochondria are responsible for converting nutrients into the energy yielding molecule, adenosine tri-phosphate (ATP), to fuel

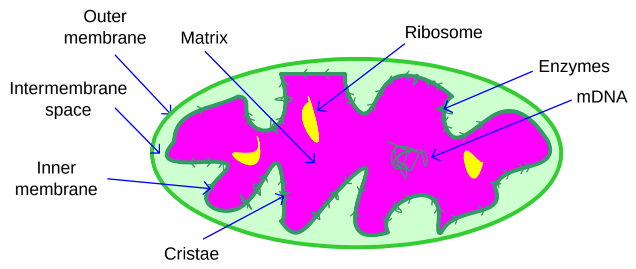


Figure 5. Scheme of the mitochondria.

the cell's activities [31]. Mitochondria can be divided into four components: outer membrane; inter-membrane space; inner membrane; and the matrix (see Figure 5).

The smooth outer membrane holds numerous transport proteins, which shuttle materials in and out of the mitochondrion. The components between the outer and inner membranes have important roles in electron transport and oxidative phosphorylation. The inner membrane has many folds, called *cristae*, which are the sites of ATP synthesis. The cristae enclose a liquid-filled region known as the matrix, which contains mitochondrial genome. These gene products include various enzymes involved in the process of aerobic respiration, proteins necessary for the import of proteins, and proteins and nucleic acids required for the mitochondrial genome [32].

The chief function of the mitochondria is to create energy for cellular activity by the process of aerobic respiration. In this process, glucose is broken down in the cell's cytoplasm via the glycolysis process, to form pyruvic acid. In a series of reactions, part of which is called Krebs' cycle, the pyruvic acid reacts with water to produce carbon dioxide and hydrogen. Energy is released as the electrons flow from the coenzyme down the electron transport chain to the oxygen atoms. The enzyme ATPase, which is embedded in the inner membrane, adds a phosphate group to adenosine diphosphate (ADP) in the matrix to form ATP. Aerobic respiration is an ongoing process and mitochondria can produce hundreds of thousands of ATP molecules/minute. ATP is transported to the cytoplasm, where it is used for virtually all energy-requiring reactions. As ATP is used, it is converted into ADP, which is returned by the cell to the mitochondrion and is used to build more ATP [33].

Various efforts have focused on the creation of models of mitochondria. For instance, [34] created a model of oxidative phosphorylation in different tissues, with the goal to include these virtual mitochondria in a virtual cell. We want to carry out simulation-based studies of this metabolism cycle. In the following sections, we show how to carry out simulation-based studies of this metabolism cycle using DEVS.

3. A DEVS model of glycolysis

Glycolysis, also called Embden–Meyerhof pathway, is a sequence of reactions used by virtually all cells to metabolise glucose [32]. The role of glycolysis is to produce energy. Glycolysis generates about 15% of the energy produced by aerobic respiration, and it is the basis for the metabolism in virtually all the living creatures. It consists of a sequence of 10 reactions that converts a glucose molecule into two pyruvate molecules with the production of NADH and ATP. Specific enzymes control each of the different reactions, as shown in Figure 6.

The glycolysis pathway was defined as a DEVS coupled model, and it was implemented using CD++. We present, as an example, the definition of Step 1, in which glucose is phosphorylated by ATP to form glucose-6-phosphate and ADP [35].

Translating the step into a DEVS atomic model is straightforward. The idea is to define each of the reactions in Figure 7 as the corresponding activation of the DEVS functions. Internal state changes are driven by the internal transition function; external events are handled by the external transitions. Timing of the reactions can be precisely defined by the time advance function. In this case, the atomic model is defined by

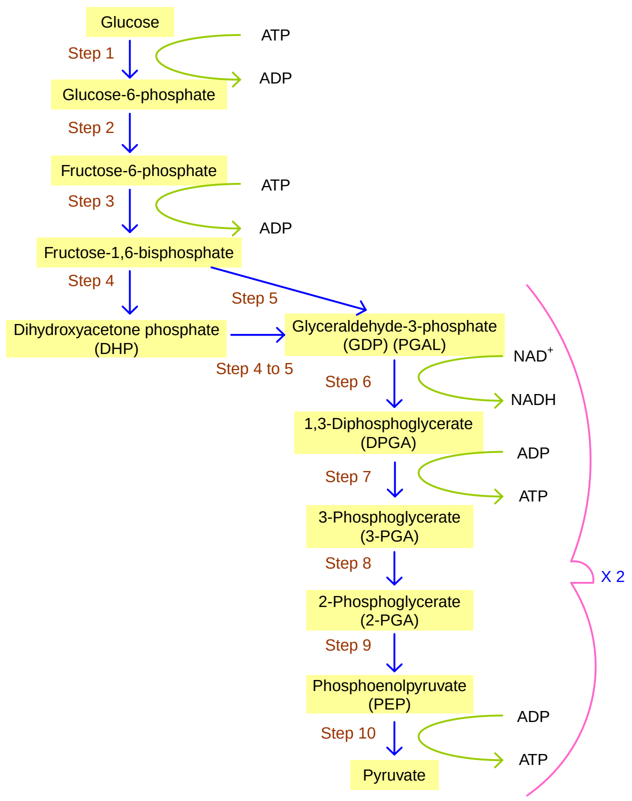

Glucose
Step 1
ATP
ADP
Glucose-6-phosphate
Step 2
Fructose-6-phosphate
Step 3
ATP
ADP
Fructose-1,6-bisphosphate
Step 4
Step 5
Dihydroxyacetone phosphate (DHP)
Glyceraldehyde-3-phosphate (GDP) (PGAL)
Step 4 to 5
Step 6
NAD+
NADH
1,3-Diphosphoglycerate (DPGA)
Step 7
ADP
ATP
3-Phosphoglycerate (3-PGA)
Step 8
X 2
2-Phosphoglycerate (2-PGA)
Step 9
Phosphoenolpyruvate (PEP)
Step 10
ADP
ATP
Pyruvate

Figure 6. Glycolysis pathway.

$$\text{Step 1} = \langle S, X, Y, \delta_{\text{int}}, \delta_{\text{ext}}, t_{\text{a}}, \lambda \rangle, \text{t}$$

$S = \{$atpc, glucosec, ifhex, counter, phase, sigma$\}$;
$X = \{$glucose, ATPi, hexokinase$\}$;
$Y = \{$glucose_6_phosphate, ADP, H$\}$;

δ_{ext} (phase = passive, e, ATPi) = $\{$ATPi = TRUE$\}$;
δ_{ext} (phase = passive, e, glucose) = $\{$glucose = TRUE$\}$;
δ_{ext} (phase = passive, e, hexokinase) = $\{$hexokinase = TRUE$\}$;
λ (phase = passive, ATPi = glucose = hexokinase = TRUE) = glucose_6_phosphate, ADP, H;

δ_{int} (phase = passive, ATPi = glucose = hexokinase = TRUE) = $\{$phase = active$\}$;
δ_{int} (phase = active) = $\{$phase = passive$\}$;

t_{a} (passive, any(ATPi, glucose, hexokinase) = TRUE) = 0.2 ms;
t_{a} (phase = active) = reaction_time(Step1).

As we can see, the external transition function is invoked every time *glucose*, *ATPi* or *hexokinase* are received, and the reactions previously discussed of the step to the enzyme (*hexokinase*) or each of the inputs (*glucose and ATPi*) as previously described in Figure 6. The internal transition function schedules an internal event after a preparation time describing the timing for the transfer. If there is glucose, ATPi and hexokinase in the system, then the reaction will happen.

When the time interval expires, the *output function* is invoked and the first value in Step 1 is sent through the corresponding output port. In this case, the output function is activated when all the conditions of the external function have been satisfied, i.e. all three input events are in, and the reaction can happen. As a result, *ADP*, *glycose_6_phosphate* and *H* will be sent out through the corresponding output ports.

After calling the output function, the internal transition function is invoked. The internal transition function will produce an internal state change according to the substances available in the mitochondria. This function updates the number of substances available according to the reaction, and it then *passivates*, waiting for the next input.

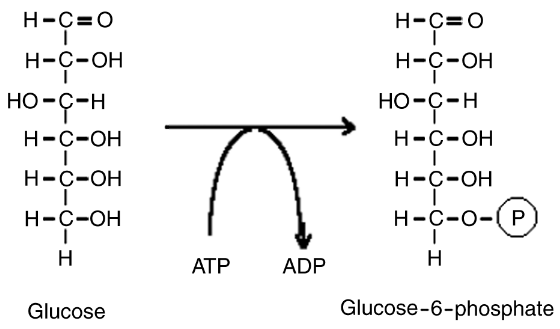


Figure 7. Step1 of glycolysis [35].

In this case, δ_{int} *passivates* the model and resets the counter (i.e. an internal event with infinite delay is scheduled, waiting for the next input).

The remaining steps of Figure 6 (steps 2–10) were developed using a similar approach: the behaviour of each component was carefully specified with an analysis of inputs and outputs for each step; then, each step was defined as a DEVS model following the specification. Finally, each model was implemented in CD++, and tested separately. Numerous simulations were carried out individually for each of the steps, validating the results with those found in the literature [20,21]. Performing individual tests of all possible cases for each step is very simple, as DEVS provides well-defined interfaces, and the hierarchical decomposition mechanism allows one to do exhaustive analysis of each subcomponent, finding errors very quickly and being able to fix them without much effort. Once every step was thoroughly tested, the main model was built as a coupled model connecting all the submodels (see Figure 8 and the corresponding textual notation on Table 9 in the appendix).

This model was defined in CD++ [29,30], which is a modelling and simulation tool defined using the specifications of the DEVS formalism. CD++ is built as a class hierarchy of models related to simulation processing entities. DEVS atomic models can be programmed and incorporated onto the Model basic class hierarchy using C++. Once an atomic model is defined, it can be combined with others into a multi-component model using a specification language specially defined with this purpose. CD++ also provides client/server services, which provide remote access to a high-performance simulation server. Using these facilities, the users can develop and test their models in local workstations, and

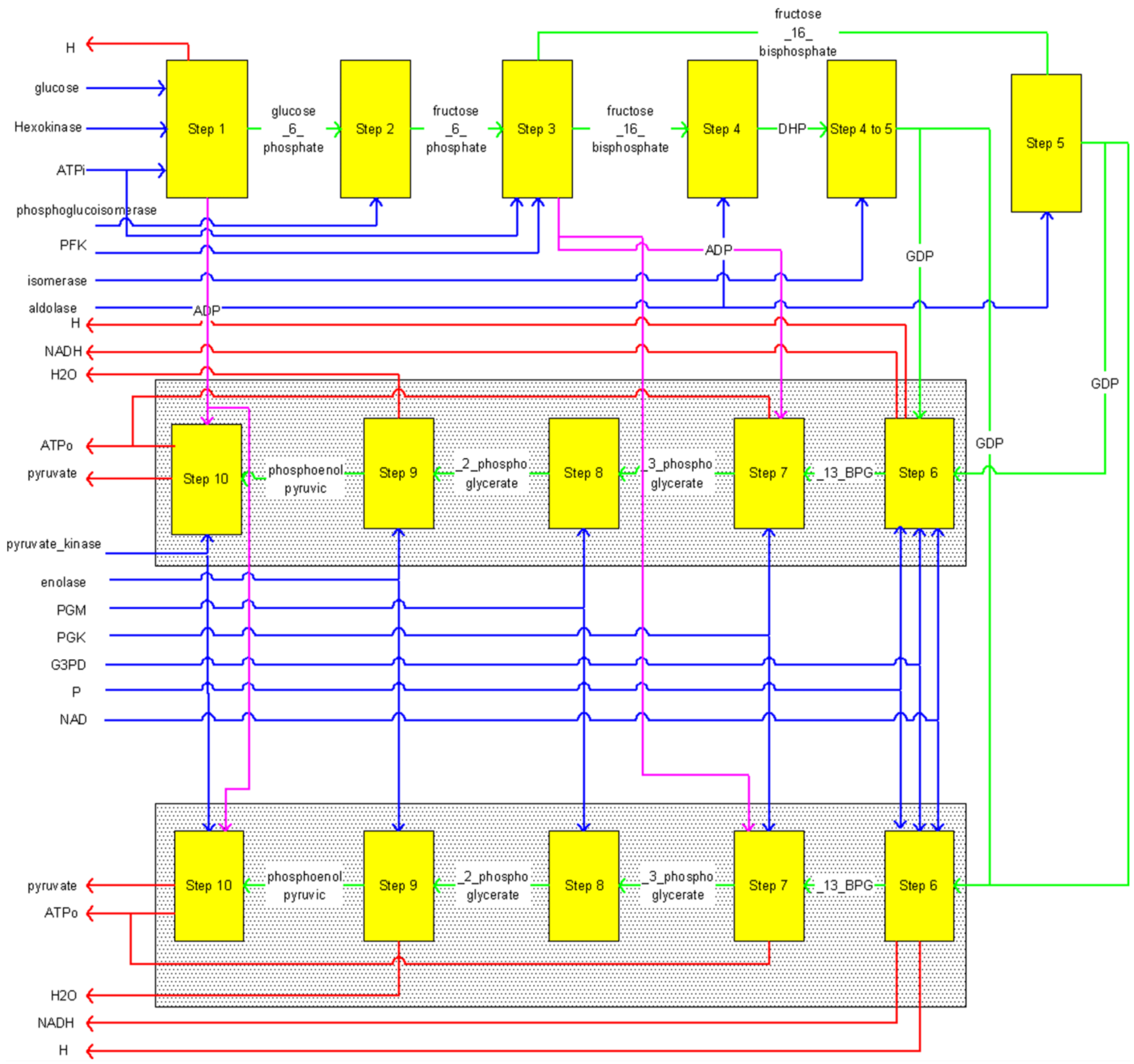


Figure 8. Coupled model of glycolysis.

submit them in a remote CD++ high-performance server. Then, they can receive, visualise and analyse the results on a local computer, improving model definition and execution. First, the components of the coupled model are defined. Then, the input and output ports are included. Finally, the links show the translation functions.

When we execute this model in CD++, we can study the model's behaviour by analysing the model's outputs. One simulation scenario we created validating the glycolysis model is presented in Table 1.

These simulation results accurately describe the reactions occurred during glycolysis [33], following the results of Figure 2. Tables 1 and 2 show the input/output trajectories for the glycolysis model and *Step 1*, respectively. As we can see in Table 2, by time 30:00, we have

Table 1. Inputs/outputs for glycolysis model.

Inputs	Outputs
10:00 glucose 2	50:000 h 2
18:00 ATPi 3	72:000 nadh 2
50:00 hexokinase 1	72:000 h 2
51:00 phospohGlucoisomerase 1	72:000 atpo 2
52:00 PFK 2	72:000 h2o 2
53:00 isomerase 1	72:000 atpo 2
55:00 aldolase 1	72:000 pyruvate 2
62:00 G3PD 1	
63:00 PGK 1	
64:00 PGM 1	
65:00 enolase 1	
67:00 pyruvKinase 1	
70:00 NAD 3	
72:00 P 2	

Table 2. Inputs/outputs for step 1.

Inputs	Outputs
15:00 hexokinase 1	30:000 adp 2
30:00 glucose 2	30:000 glucose_6_phosphate 2
30:00 ATPi 6	30:000 h 2
40:00 glucose 4	40:000 adp 4
40:00 ATPi 1	40:000 glucose_6_phosphate 4
55:00 glucose 1	40:000 h 4
55:00 ATPi 1	55:000 adp 1
65:00 glucose 1	55:000 glucose_6_phosphate 1
	55:000 h 1
	65:000 adp 1
	65:000 glucose_6_phosphate 1
	65:000 h 1

all the three inputs required to produce a reaction. At time 30:00, two *glucose*s and six *ATPi* enter the system generating two *ADP*, two *glucose_6_phosphate* and two *H* molecules.

Figure 9 shows the execution results for *Step 1* using the CD++ modeller. CD++ allows the execution of the same model using a local application or a remote server. Once the simulation is done, the user can analyse the simulation results using different visualisation tools. In this way, specialists located remotely are able to run different experiments, and to analyse them locally using the visual tools presented here. In Figure 9, we see that at time 40:00, four *glucose* molecules enter the system generating four more outputs of each of the *ADP*, *glucose_6_phosphate* and *H* molecules.

In [19], we presented CD++/Maya, an application to generate 3D visual representation of DEVS models. Autodesk Maya [36] is a software application for 3D digital animation and visual effects. It provides a suite of tools for 3D world creation, including animation, rendering, dynamics, etc. In CD++/Maya, the log files generated from CD++ can be translated into 3D representations. We used CD++/Maya to create 3D graphics and animations for the glycolysis model. Figure 10 shows a snapshot of the animation. In this case, we show that, while the enzymes are entering the reactions, their names are being added as text to the top of the screen for clarity.

4. Krebs' cycle modelling

Krebs' cycle, also called the tri-carboxylic acid cycle and the citric acid cycle (CAC), oxidises pyruvate formed during the glycolysis pathway into CO_2 and H_2O. This cycle is a series of chemical reactions of central importance in all living cells that utilise oxygen. The CAC takes place within the mitochondria in eukaryotes, and within the cytoplasm in prokaryotes. For each turn of the cycle, 12 ATP molecules are produced, one directly from the cycle and 11 from the reoxidation of the three NADH molecules and one FADH2 molecule produced by the cycle by oxidative phosphorylation [37]. Glucose is converted by glycolysis into pyruvate. Pyruvate enters the mitochondria, linking glycolysis to Krebs' cycle. This step (Step A), as seen in Figure 11, is also called the bridging step. Pyruvate dehydrogenase – a complex of three enzymes and five coenzymes – oxidises pyruvate using NAD^+ to form acetyl CoA, NADH and CO_2.

We defined a model of Krebs' cycle (depicted in Figure 11), using identical principles than the ones used in the previous section. In Step A, as seen in Figure 12,

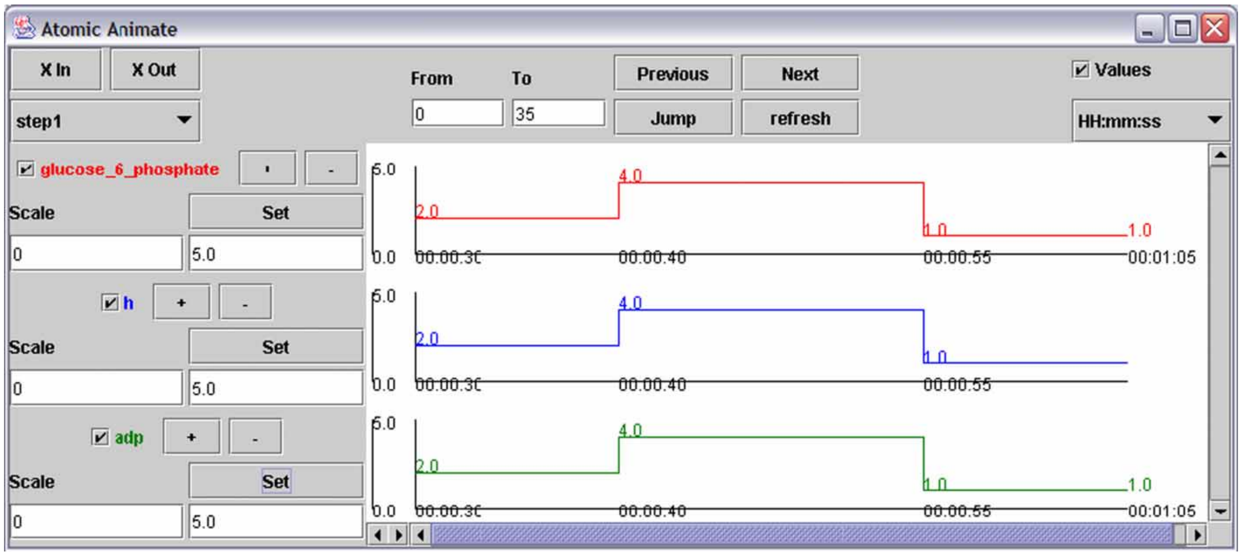


Figure 9. Atomic animation of Step 1 of glycolysis.

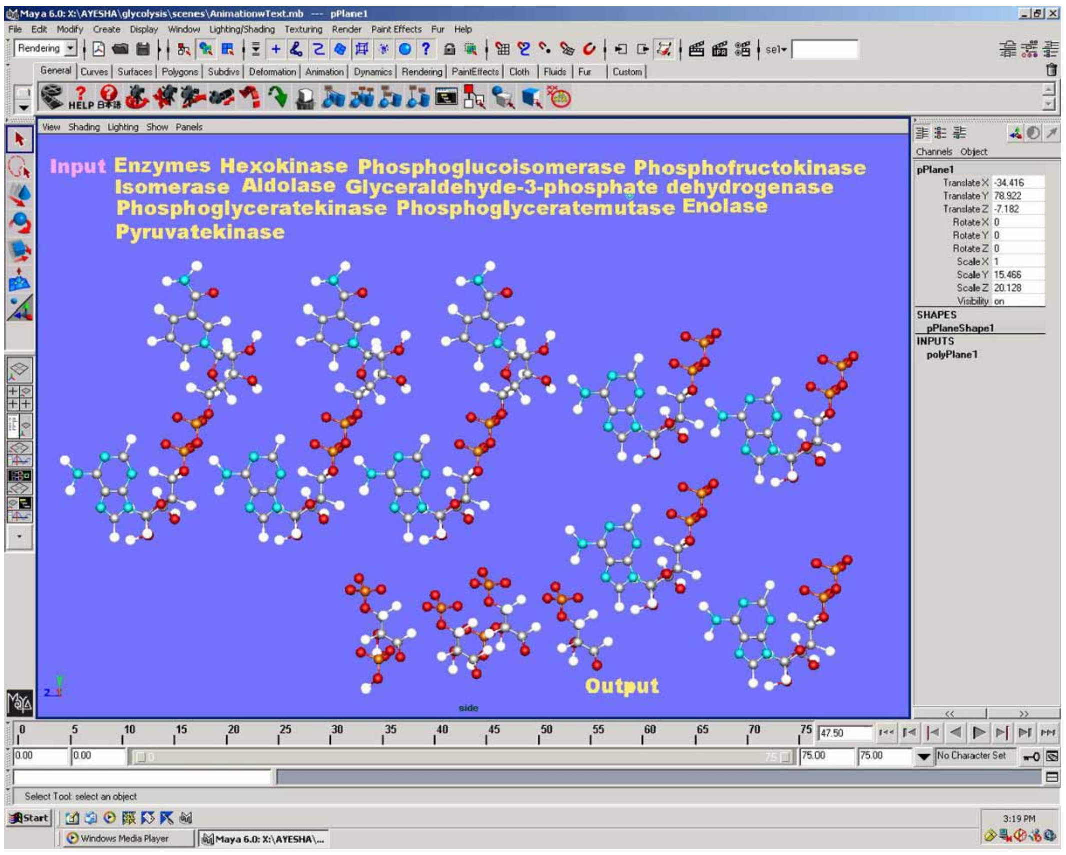

Maya 6.0: X:\AYESHA\glycolysis\scenes\AnimationwText.mb --- pPlane1
Input Enzymes Hexokinase Phosphoglucoisomerase Phosphofructokinase
Isomerase Aldolase Glyceraldehyde-3-phosphate dehydrogenase
Phosphoglyceratekinase Phosphoglyceratemutase Enolase
Pyruvatekinase
Output
pPlane1
Translate X -34.416
Translate Y 78.922
Translate Z -7.182
Rotate X 0
Rotate Y 0
Rotate Z 0
Scale X 1
Scale Y 15.466
Scale Z 20.128
Visibility on
SHAPES
pPlaneShape1
INPUTS
polyPlane1
Select Tool: select an object

Figure 10. Visualisation of the glycolysis execution in CD++/Maya: Step 6.

pyruvate is degraded and combined with coenzyme A to form acetyl coenzyme A. NADH and CO_2 are released during this process (Step A is the link between glycolysis and Krebs' cycle).

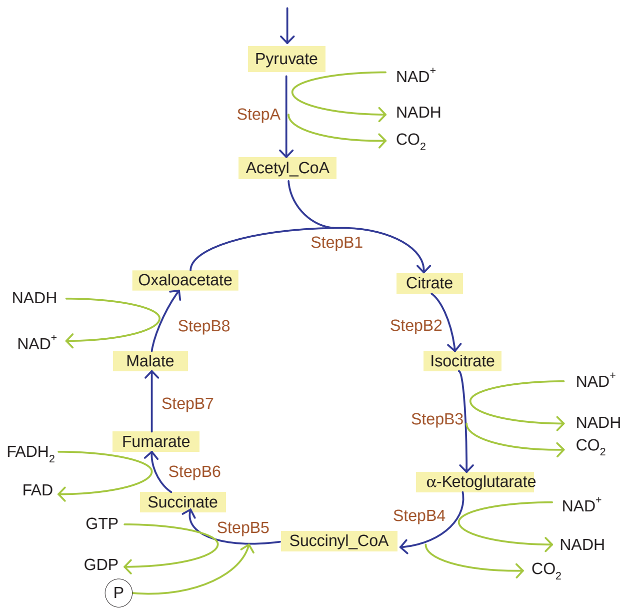


Figure 11. Krebs' cycle reactions.

The atomic model for Step A of Krebs' cycle is shown in Figure 13.

The DEVS atomic model for *Step A* is defined as

$$\text{Step A} = \langle S, X, Y, \delta_{\text{int}}, \delta_{\text{ext}}, t_a, \lambda \rangle,$$

$S = \{$pyruvatec, pyruvateDehydrogenase, hscoaic, nadc, counter, phase, sigma$\}$; $X = \{$pyruvate, pyruvateDehydrogenase, HSCoAi, NAD$\}$; $Y = \{$acetyl_CoA, NADH, H,

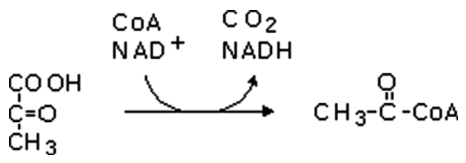


Figure 12. Step A of Krebs' cycle [35].

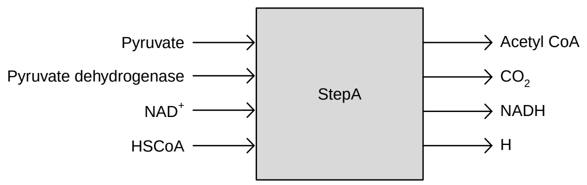


Figure 13. Atomic model for Step A of Krebs' cycle.

Table 3. External transition function (δ_{ext}).

```
δext (s, e, x) {
    if (x = pyruvate) {
        Pyruvatec = pyruvatec + x;
        if (hscoaic > 0 and nadc > 0 and ifpyruvateDehydrogenase = true)
            ta(active, prep_time);
        }
        else if (x = HSCoAi) {
          hscoaic = hscoaic + x;
          if (pyruvatec > 0 and nadc > 0 and ifpyruvateDehydrogenase = true)
              ta(active, prep_time);
        }
        else if (x = NAD) {
          nadc = nadc + msg.value();
          if (pyruvatec > 0 and hscoaic > 0 and ifpyruvateDehydrogenase = true)
             ta(active, prep_time);
        }
        else if (x = pyruvateDehydrogenase) {
          ifpyruvateDehydrogenase = true;
          if (pyruvatec >  0 and hscoaic > 0 and nadc > 0)
             ta(active, prep_time);
        }
}
```

CO2}; δ_{int} = internal function; δ_{ext} = external function; t_a = time advance (it controls the timing of internal transitions: when the sigma state variable is present, this function just returns the value of sigma; sigma holds the time remaining to the next internal event); and λ = output function.

Table 3 shows the external transition function. The external transition function will be invoked every time pyruvate, pyruvate dehydrogenase, HSCoAi or NAD arrives. When an event arrives at any of the input ports, its value is added to a counter kept for that kind of input (pyruvatec, ifpyruvate dehydrogenase, hscoaic and nadc). Once the value of the counter of the input event is incremented, the condition for the other requirements is checked. For example, if the input event is pyruvate, the pyruvate counter value is incremented by the value of the pyruvates entered, and then it is checked against other input counters: hscoaic; nadc; and ifpyruvate dehydrogenase. If there are NAD, HSCoAi and pyruvate dehydrogenase present in the system, then the reaction will happen. In the case of pyruvate dehydrogenase, since it is an enzyme and its presence is only needed for the reaction, its value is set to true.

Table 4 shows the internal function for Step A of Krebs' cycle. Before calling this method, the sigma value is zero because the interval to the internal transition has expired. When the preparation time interval expires and after calling the output function, this method is invoked.

Table 5 shows the output function for Step A of Krebs' cycle. When the preparation time interval expires, this method is invoked and the Acetyl_CoA, NADH, CO_2 and H have to be sent out through the output ports. The internal transition function updates the counter of molecules available in the mitochondria, and passivates the model waiting for the next arrival of a substance.

The definition of Krebs' coupled model using CD++ is presented in Figure 14, which was transformed into a CD++ specification, seen in the Appendix. The figure presents a sketch of the coupled model for Krebs' cycle.

All the remaining steps were developed using a similar approach. To do this, the behaviour of each component was carefully specified with an analysis of inputs and outputs. Each step was defined as a DEVS model following the specification. Afterwards, each model was implemented in CD++, connecting all the submodels defined, as in Figure 15. CD++ permits defining coupled models as this one using a graphical user interface. Figure 15 presents the visual input creation of this coupled model, which is exported to CD++ notation to be executed.

Table 6 lists the input event files (stepA.ev) which include a list of inputs with their timestamp and value, and the output files (stepA.out) which include the outputs resulted from simulation, for three different scenarios.

In scenario 1, the first input event pyruvate enters at time 00:00:15:00 and has a value 5. The second input is HSCOi with a value 1 and it enters at time 00:00:15:00. The third input is NAD with timestamp 00:00:35:00 and a value 2. The last input is the enzyme pyruvate dehydrogenase which enters at time 00:00:42:00 with a value 1. All the outputs are generated at time 00:00:42:000, at the same time that the last input enters. This means that all the four inputs are needed for any output to be generated.

Table 4. Internal transition function (δ_{int}).

```
δint (s, e) {
  counter = 0;
  if (pyruvatec > = 1 and hscoaic > = 1 and nadc > = 1 and ifpyruvateDehydrogenase = true) {
            if (pyruvatec > = hscoaic and hscoaic > = nadc) {
               pyruvatec = pyruvatec − nadc;
               hscoaic = hscoaic − nadc;
               counter = nadc;
               nadc = 0;
            }
            else if ((pyruvatec > = nadc) && (nadc > = hscoaic)) {
                pyruvatec = pyruvatec − hscoaic;
                nadc = nadc − hscoaic;
                counter = nadc;
                hscoaic = 0;
            }
            else if ((hscoaic > = nadc) && (nadc > = pyruvatec)) {
                nadc = nadc − pyruvatec;
                hscoaic = hscoaic − pyruvatec;
                counter = pyruvatec;
                pyruvatec = 0;
            }
            else if ((hscoaic > = pyruvatec) && (pyruvatec > = nadc)){
                pyruvatec = pyruvatec − nadc;
                hscoaic = hscoaic − nadc;
                counter = nadc;
                nadc = 0;
            }
            else if ((nadc > = pyruvatec) && (pyruvatec > = hscoaic)){
                pyruvatec = pyruvatec − hscoaic;
                nadc = nadc − hscoaic;
                counter = hscoaic;
                hscoaic = 0;
            }
            else if ((nadc > = hscoaic) && (hscoaic > = pyruvatec)) {
                hscoaic = hscoaic − pyruvatec;
                nadc = nadc − pyruvatec;
                counter = pyruvatec;
                pyruvatec = 0;
            }
            else if ((pyruvatec = = hscoaic) && (hscoaic = = nadc)) {
                counter = pyruvatec;
                pyruvatec = nadc = hscoaic = 0;
            }
       }
   }
```

Table 5. Output function (λ).

```
λ (s) {
  if (counter is not zero)
  send outputs through the ports; //Acetyl_CoA, NADH, CO2, H
}
```

In Scenario 2, the input event file produces an empty output file. This is because the enzyme pyruvate dehydrogenase is missing from the input list. In scenario 3, we have two sets of outputs generated. The first set is generated at time 00:01:25:000, and the second set are generated at time 00:03:01:000. As we can see, pyruvate dehydrogenase enters at time 00:00:42:00 with a value 1. Since only the presence of this enzyme is needed, one unit of this input is enough. The number of total outputs (2 unit of each) matches the lowest value of any other input event which in this case is equal to the total number of HSCoAi in the input list (1 unit at time 00:01:25:00, and another unit at time 00:03:01:00).

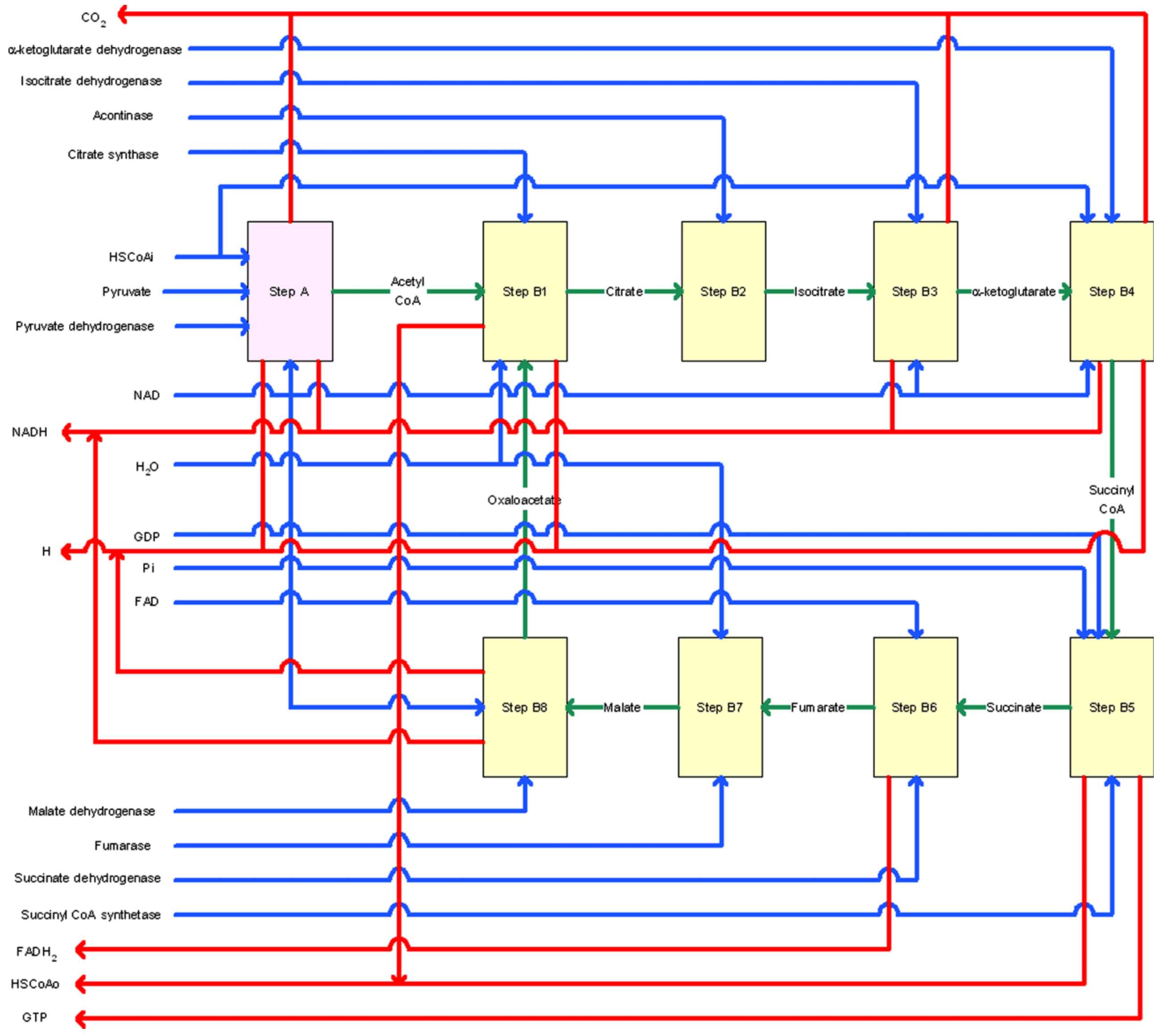


Figure 14. Coupled model of Krebs' cycle.

After testing all the atomic models one by one, and adding them on top of each other and testing them again, we can make sure that all the submodels are free of error. Once all the basic atomic models are tested, the Krebs' coupled model is formed by connecting all the previously tested atomic models.

Table 7 shows the input and output files for three different scenarios for the Krebs' coupled model.

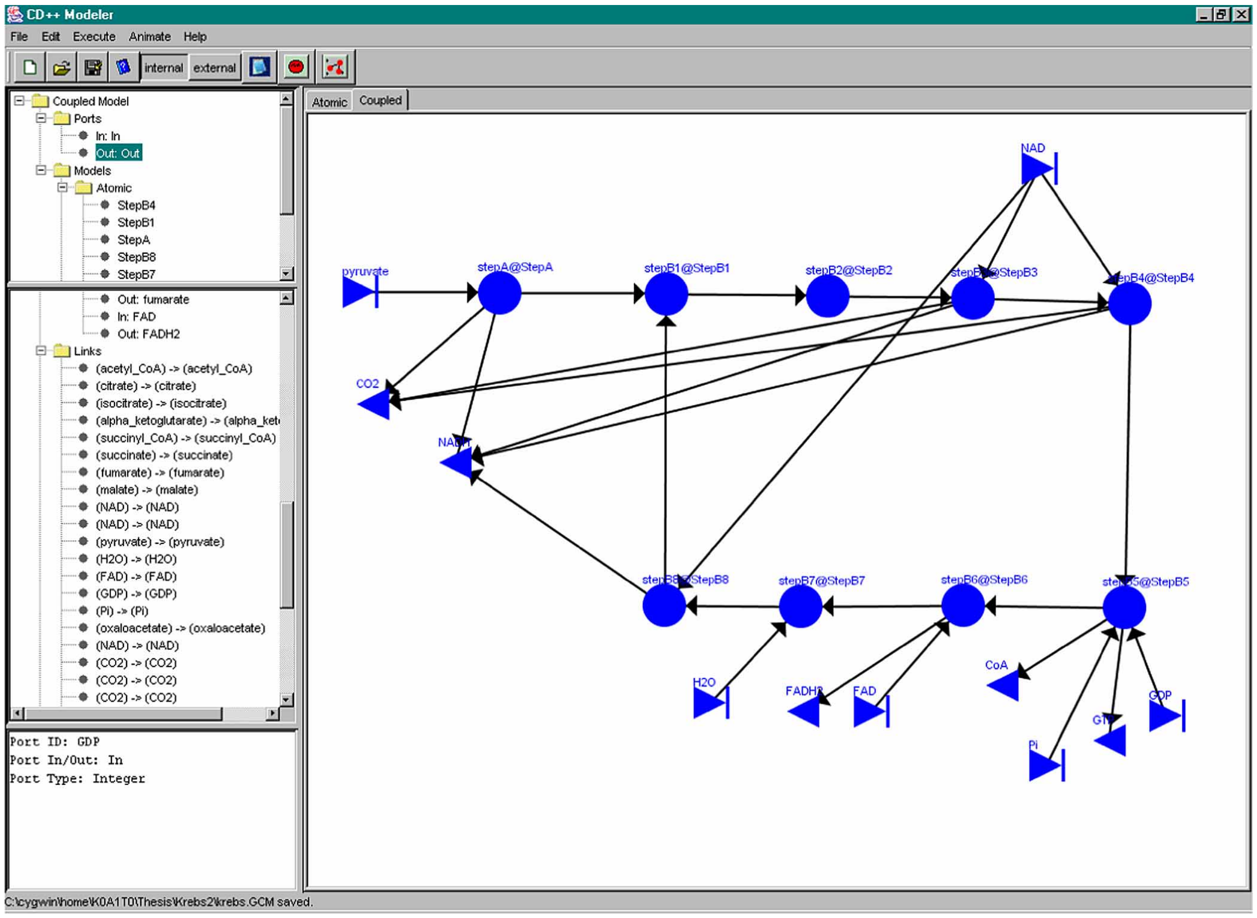
These simulation results accurately describe the reactions occurred in the mitochondria, as described in Figure 16.

Table 8 shows a list of the input and output events for Krebs' model. Figure 16 demonstrates the atomic animation for this set of inputs and outputs. This time, for each event chosen in the input event, a timeline starting exactly based on the time specified in Table 8 is drawn, with the value of the event marked on the line.

Figure 17 shows snapshots of some of the reactions in the Krebs' cycle animation done in CD++/Maya. Figure 17(a) shows the beginning of the reaction, in which one pyruvate and four NAD^+ appear. Figure 17(b) shows the formation of acetyl CoA, and the production of carbon dioxide and NADH as by-products.

5. Discussion and conclusions

During the last several decades, computer simulation has become an integral part in both the basic and applied fields



```
Port ID: GDP
Port In/Out: In
Port Type: Integer
```

C:\cygwin\home\K0A1T0\Thesis\Krebs2\krebs.GCM saved.

Figure 15. Defining the Krebs' coupled model using the CD++ modeller.

of biological research. The hierarchical and discrete-event capabilities of DEVS make it an ideal method for modelling biological events. As illustrated herein, CD++ can be used to model and simulate biological pathways using a systematic method with models that consist of sets of lower-level interactions. We show how suitable hierarchies can help biologists and medical researchers simulate complex biological systems with code that can be

Table 6. Input event files and corresponding output files resulted from simulating Step A of Krebs' cycle.

Scenario	StepA.ev	StepA.out

1	`00:00:15:00 pyruvate 5`
	`00:00:25:00 HSCoAi 1`
	`00:00:35:00 NAD 2`
	`00:00:42:00 pyruvateDehydrogenase 1`
		`00:00:42:000 acetyl_coa 1`
		`00:00:42:000 nadh 1`
		`00:00:42:000 co2 1`
		`00:00:42:000 h 1`

2	`00:00:15:00 pyruvate 5`
	`00:00:25:00 HSCoAi 1`
	`00:00:35:00 NAD 2`	

3	`00:00:15:00 pyruvate 1`
	`00:00:35:00 NAD 2`
	`00:00:42:00 pyruvateDehydrogenase 1`
	`00:01:25:00 HSCoAi 1`
	`00:02:15:00 pyruvate 5`
	`00:02:35:00 NAD 4`
	`00:03:01:00 HSCoAi 1`
		`00:01:25:000 acetyl_coa 1`
		`00:01:25:000 nadh 1`
		`00:01:25:000 co2 1`
		`00:01:25:000 h 1`
		`00:03:01:000 acetyl_coa 1`
		`00:03:01:000 nadh 1`
		`00:03:01:000 co2 1`
		`00:03:01:000 h 1`

Table 7. Event and output files of Krebs' coupled model.

Scenario	Krebs.ev	Krebs.out
1	00:00:10:00 pyruvate 1	00:01:11:000 nadh 1
	00:00:25:00 NAD 2	00:01:11:000 co2 1
	00:00:40:00 H2O 2	00:01:11:000 h 1
	00:00:57:00 HSCoAi 2	00:02:01:000 hscoao 1
	00:01:11:00 pyruvateDehydrogenase 1	00:02:01:000 h 1
	00:02:01:00 oxaloacetate 1	00:02:30:000 co2 1
	00:02:14:00 acontinase 1	00:02:30:000 nadh 1
	00:02:30:00 isocitrateDehydrogenase 1	00:02:30:000 co2 1
	00:02:30:00 alpha_ketoglutarateDehydrogenase 1	00:02:30:000 h 1
	00:02:50:00 succinylCoA_Synthetase 1	00:03:11:000 hscoao 1
	00:03:10:00 GDP 1	00:03:11:000 gtp 1
	00:03:11:00 Pi 1	00:03:28:000 fadh2 1
	00:03:15:00 FAD 1	00:03:45:000 nadh 1
	00:03:28:00 succinateDehydrogenase 1	00:03:45:000 h 1
	00:03:40:00 fumarase 1	
	00:03:45:00 malateDehydrogenase 1	
2	00:00:10:00 pyruvate 1	00:03:12:000 nadh 1
	00:00:25:00 NAD 2	00:03:12:000 co2 1
	00:00:40:00 H2O 1	00:03:12:000 h 1
	00:03:45:00 malateDehydrogenase 1	00:03:45:000 hscoao 1
	00:00:57:00 HSCoAi 2	00:03:45:000 h 1
	00:01:14:00 citrateSynthase 1	00:03:45:000 nadh 1
	00:02:14:00 acontinase 1	00:03:45:000 co2 1
	00:02:30:00 isocitrateDehydrogenase 1	00:03:45:000 nadh 1
	00:02:30:00 alpha_ketoglutarateDehydrogenase 1	00:03:45:000 co2 1
	00:02:50:00 succinylCoA_Synthetase 1	00:03:45:000 h 1
	00:03:10:00 GDP 1	00:03:45:000 hscoao 1
	00:03:11:00 Pi 1	00:03:45:000 gtp 1
	00:03:12:00 pyruvateDehydrogenase 1	00:03:45:000 fadh2 1
	00:03:15:00 FAD 1	00:03:45:000 nadh 1
	00:03:28:00	
	succinateDehydrogenase 1	00:03:45:000 h 1
	00:03:40:00 fumarase 1	
	00:03:45:00 oxaloacetate 1	
3	00:00:10:00 pyruvate 1	00:03:12:000 nadh 1
	00:00:25:00 NAD 2	00:03:12:000 co2 1
	00:00:40:00 H2O 1	00:03:12:000 h 1
	00:03:45:00 malateDehydrogenase 1	
	00:00:57:00 HSCoAi 2	
	00:02:14:00 acontinase 1	
	00:02:30:00 isocitrateDehydrogenase 1	
	00:02:50:00 succinylCoA_Synthetase 1	
	00:03:10:00 GDP 1	
	00:03:11:00 Pi 1	
	00:03:12:00 pyruvateDehydrogenase 1	
	00:03:15:00 FAD 1	
	00:03:28:00 succinateDehydrogenase 1	
	00:03:40:00 fumarase 1	
	00:03:45:00 oxaloacetate 1	

easily understood and modified. The different examples presented here show how one can organise the models in a manageable way.

Using this method, complex simulations can be built and validated incrementally. The use of DEVS, combined with domain-specific libraries, provides a system that allows even non-computer science specialists the ability to develop complex simulation models for specific research problems. This approach also enables reuse of simulation components and allows seamless integration of these components into more complex simulation models. This capability would allow for sharing and interoperability of model components, thus aiding the development of a wide variety of model implementations for the broader medical research field.

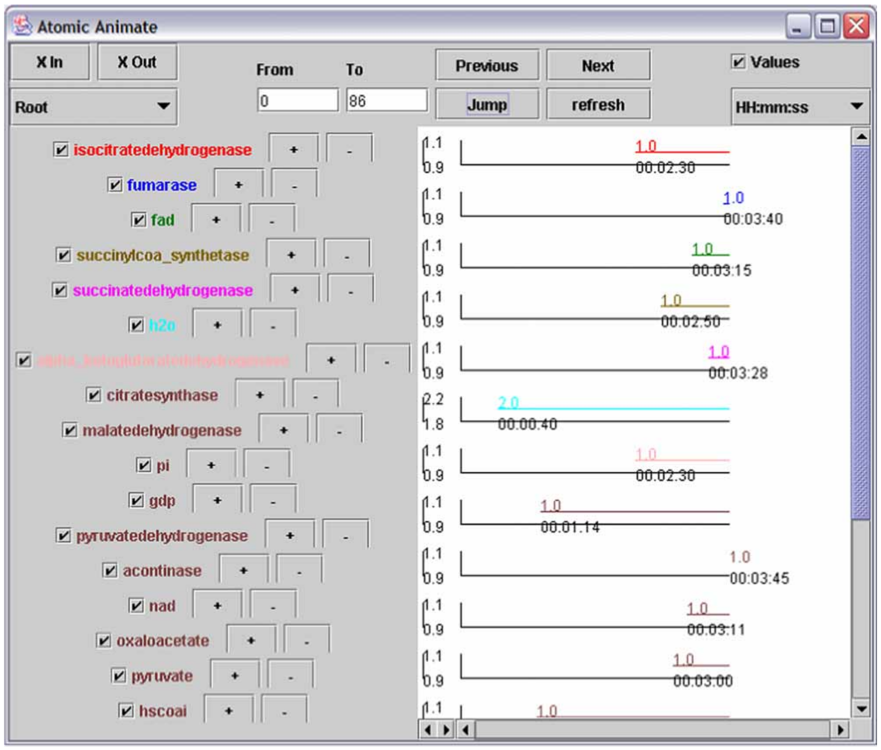


Figure 16. Atomic animation of Krebs' cycle.

As illustrated in this paper, the use of DEVS (which was used to model and simulate biological pathways – glycolysis and Krebs' cycle) provided a systematic method in which a model consists of a set of lower-level interactions. The use of DEVS enables proving the correctness of the simulation engines and permits to model the problem even by a non-computer science specialist. The high-level language of DEVS reduces the algorithmic complexity for the modeller while allowing complex cellular timing behaviours. Sharing and interoperability of model implementations, focusing in different model examples in the area of medicine, is a means of developing independent models that can be integrated at the level of DEVS interactions, and how models can be composed into simulations that can execute in distributed environments.

We used an example based on the development of models for mitochondria, which are complex organelles that can be modelled as systems with hierarchical

Table 8. Inputs and outputs for Krebs' model.

Inputs (krebs.ev)	Outputs (krebs.out)
`00:00:10:00 pyruvate 1`	`00:01:11:000 nadh 1`
`00:00:25:00 NAD 2`	`00:01:11:000 co2 1`
`00:00:40:00 H2O 2`	`00:01:11:000 h 1`
`00:00:57:00 HSCoAi 2`	`00:02:01:000 hscoao 1`
`00:01:11:00 pyruvateDehydrogenase 1`	`00:02:01:000 h 1`
`00:01:14:00 citrateSynthase 1`	`00:02:30:000 nadh 1`
`00:02:01:00 oxaloacetate 1`	`00:02:30:000 co2 1`
`00:02:14:00 acontinase 1`	`00:02:30:000 nadh 1`
`00:02:30:00 isocitrateDehydrogenase 1`	`00:02:30:000 co2 1`
`00:02:30:00 alpha_ketoglutarateDehydrogenase 1`	`00:02:30:000 h 1`
`00:02:50:00 succinylCoA_Synthetase 1`	`00:03:11:000 hscoao 1`
`00:03:10:00 GDP 1`	`00:03:11:000 gtp 1`
`00:03:11:00 Pi 1`	`00:03:28:000 fadh2 1`
`00:03:15:00 FAD 1`	`00:03:45:000 nadh 1`
`00:03:28:00 succinateDehydrogenase 1`	`00:03:45:000 h 1`
`00:03:40:00 fumarase 1`	
`00:03:45:00 malateDehydrogenase 1`	

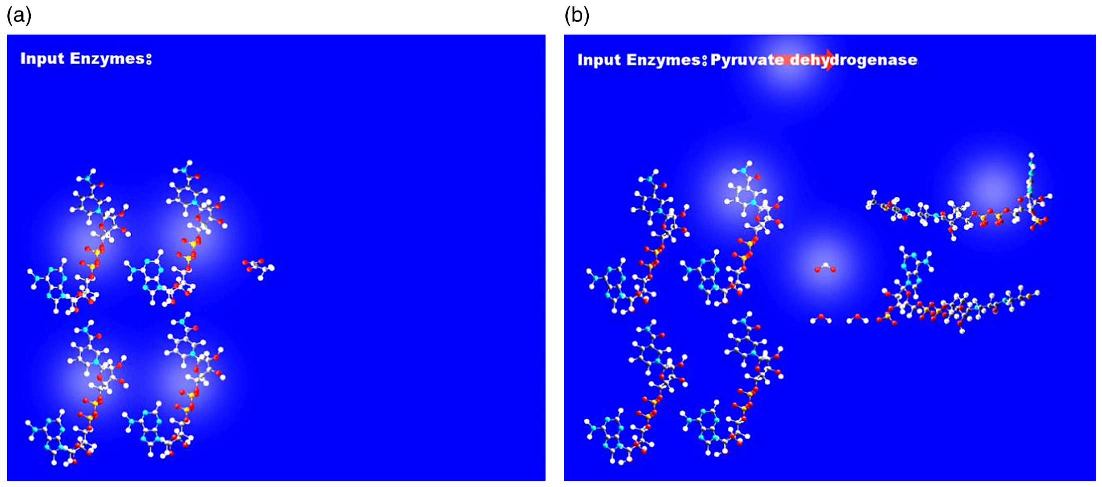


Figure 17. (a) Krebs' cycle begins and (b) acetyl CoA is formed.

structure. Many complex reactions can occur in mitochondria, typically arranged to form multiple self-compensating feedback loops. Moreover, the chemical reactions always occur due to the presence of suitable enzymes, and in specifically dedicated sites in the cell or within its organelles. This combination of high overall complexity, multiple layers of behaviour (cell, organelles and specific reaction sites) and local confinement of the behavioural elements (i.e. the chemical reactions) makes a block decomposition approach such as the one advocated by the DEVS formalism extremely effective. In addition to that, the CD++ toolkit fully supports hierarchical and compositional system specification in the modelling, testing, deployment and execution phases. Deterministic models based on differential equations used for other applications are regarded as inadequate for these processes of considerable complexity.

The definition of these models using the DEVS formalism brings two major benefits to scientists interested in studying the behaviour of cells or similarly challenging biological systems. The first benefit is modelling effectiveness, to be understood as a combination of different factors that ease the work of the creators and maintainers of models, as well as improving its quality.

A first important factor is the increased understandability yielded by hierarchical model representations. Here, the structured DEVS language and the modelling tools work together to make the models of complex biological systems manageable within the user's grasp. These features also bring about another significant factor, composability and information hiding. Details are effectively encapsulated and hidden, so that more and more models can be assembled together, leveraging multiple aggregation layers without jeopardising the overall model comprehension.

Beyond the two core factors outlined above, there are other ways in which our approach based on DEVS increases modelling effectiveness. First, the modular and hierarchical decomposition of the models allows distributing the modelling tasks themselves across a team of scientists, due to the clear interfaces defined between the component parts of a coupled model. Second, and somewhat analogously, the model structure eases also the testing work, by allowing naturally planning and executing multi-level and hierarchical test strategies. Creating advanced 3D visualisation environments is also straightforward. Visualisation is an important aspect of modelling and simulation, and it is very important in the domain of biological systems (as it provides a mechanism for validating the execution of the simulations, and the relationships with *in vitro* experiments). Nevertheless, this aspect is usually dismissed or postponed for later versions of the model. Instead, in our case, simulation results can be easily used to construct useful 2D and 3D images, including animations via Autodesk Maya.

Another major benefit brought about by DEVS is execution performance improvement. In this discrete-event simulation execution model, time advances in a discontinuous and irregular way depending on the next relevant event. This approach effectively replaces the quantised continuum of the temporal and spatial physical dimensions with a logical state space, evolving according to systems theoretical principles.

Other approaches that do not abstract away from the continuous physical space and time, such as [38,39], are able to process very precise information (e.g. the exact space disposition of molecules or the probability of a molecule passing through a small gap). Nevertheless, these models have typically to pay the price of increased computational resources and lower scalability with respect to the model size and complexity. Moreover, these approaches typically require some tuning of the simulation time steps, which can also cause a higher number of simulation runs and thus an increased computation load. When we compare the size of the teams, the effort involved and the duration of the efforts for other applications, the advantage of DEVS becomes apparent (for instance, projects such as the E-Cell are ongoing efforts that started in 1996 and have required dozens of full-time researchers; in our case, a small team was able to build the basic building blocks of energy pathways at a reduced cost, thanks to the advantages provided by the DEVS formal specifications).

Beyond the performance benefits stemming from DEVS discrete-event simulation approach, there are also direct contributions in terms of deployment and execution. The communication among the different processors follows a message-passing paradigm, and various kinds of topologies are possible. In particular, distributed execution on different machines is possible; the model-driven organisation of such processors does not assume any specific multicomputer architecture, so that it is extremely suited to modern infrastructures for scientific calculations such as Grid Computing.

Acknowledgements

This work has been partially funded by the NSERC, and by Prof. Tofy Mussivand at the Ottawa Heart Institute. Rhys Goldstein (Autodesk Research) has provided useful feedback in the later version of the manuscript.

Note

1. Email: roxanadj@site.uottawa.ca

References

[1] R. Goldstein and G.A. Wainer, *DEVS-based design of spatial simulations of biological systems*, Proceedings of Winter Simulation Conference, Austin, TX, 2009.

[2] C. Maus, M. John, M. Röhl, and A. Uhrmacher, *Hierarchical modeling for computational biology*, in *Formal Methods for Computational Systems Biology*, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 81–124.

[3] M.R. Grant, C.A. Hunt, X. Lan, J.E. Fata, and M.J. Bissell, *Modeling mammary gland morphogenesis as a reaction–diffusion process*, Proceedings of 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, IEMBS '04, San Francisco, CA, 2004, pp. 679–682.

[4] K. Takahashi, N. Ishikawa, Y. Sadamoto, H. Sasamoto, S. Ohta, A. Shiozawa, F. Miyoshi, Y. Naito, Y. Nakayama, and M. Tomita, *E-cell 2: Multi-platform E-cell simulation system*, Bioinformatics 19 (2003), pp. 1727–1729.

[5] G. Olivier Brett and L. Snoep Jacky, *Web-based kinetic modelling using JWS online*, Bioinformatics 20 (2004), pp. 2143–2144.

[6] S. Sundararaj, A. Guo, B. Habibi-Nazhad, M. Rouani, P. Stothard, M. Ellison, and D.S. Wishart, *The cybercell database (CCDB): A comprehensive, self-updating, relational database to coordinate and facilitate* in silico *modeling of* Escherichia coli, Nucl Acids Res. 32 (2004), pp. 293–295.

[7] M.A. Gibson and J. Bruck, *Efficient exact stochastic simulation of chemical systems with many species and many channels*, J. Phys. Chem. A 104 (2000), pp. 1876–1889.

[8] D.T. Gillespie, *A general method for numerically simulating the stochastic time evolution of coupled chemical reactions*, J. Comput. Phys. 22 (1976), p. 403.

[9] R. Ewald, C. Maus, A. Rolfs, and A.M. Uhrmacher, *Discrete event modeling and simulation in systems biology*, J. Simul. 1 (2007), pp. 81–96.

[10] J. von Neumann, *Theory of Self-Reproducing Cellular Automata*, University of Illinois Press, Urbana, IL, 1966.

[11] C.A. Hunt, G. Ropella, M. Roberts, and L. Yan, *Biomimetic* in silico *devices*, Proceedings of Computational Methods in Systems Biology 2004; Lecture Notes in Bioinformatics 3082 (2005), pp. 35–43.

[12] R. Huricha and X. Ruanxiaogang, *A simple cellular automaton model for tumor-immunity system*, Proceedings. 2003 IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, Changsha, Hunan, China.

[13] O. Inghe, *Genet and ramet survivorship under different mortality regimes – A cellular automata model*, J. Theor. Biol. 138 (1989), pp. 257–270.

[14] K. Takahashi, S.N.V. Arjunan, and M. Tomita, *Space in systems biology of signaling pathways – Towards intracellular molecular crowding* in silico, FEBS Lett. 579 (2005), pp. 1783–1788.

[15] B.P. Zeigler, H. Praehofer, and T.G. Kim, *Theory of Modeling and Simulation*, 2nd ed., Academic Press, London, 2000.

[16] G. Wainer, *Applying Cell-DEVS methodology for modeling the environment*, Simul.: Trans. Soc. Model. Simul. Int. 82 (2006), pp. 635–660.

[17] G. Wainer, *ATLAS: A language to specify traffic models using Cell-DEVS*, Simul. Model. Pract. Theory 14 (2006), pp. 313–337.

[18] G. Wainer, S.S. Daicz, L. De Simoni, and D. Wasserman, *Using the ALFA-1 simulated processor for educational purposes*, ACM J. Edu. Resour. Comput. 1 (2001), pp. 111–151.

[19] G. Wainer and Q. Liu, *Tools for graphical specification and visualization of DEVS models*, Simulation 85 (2009), pp. 131–158.

[20] S. Krauss, *Mitochondria: Structure and role in respiration*, in *Nature Encyclopedia of Life Sciences*, Nature Publishing Group, New York, 2001.

[21] J. Poulton and L. Bindoff, *Mitochondrial respiratory chain disorders*. *Nature Encyclopedia of Life Sciences*, Nature Publishing Group, New York, 2000.

[22] D. Harel and M. Politi, *Modeling Reactive Systems with Statecharts*, McGraw-Hill, New York, 1998.

[23] H. Saadawi and G. Wainer, *Defining models of complex 2D physical systems using Cell-DEVS*, Simul. Model. Pract. Theory 15 (2007), pp. 1268–1291.

[24] F.E. Cellier and E. Kofman, *Continuous System Simulation*, Springer Science + Business Media, New York, 2006.

[25] G. Wainer, B. Al-aubidy, A. Dias, R. Bain, S. Jafer, M. Dumontier, and J. Cheetham, *Advanced DEVS models with applications to biomedicine*, Proceedings of AIS'2007 Artificial Intelligence, Simulation and Planning, Buenos Aires, Argentina, 2007.

[26] R. Bain, S. Jafer, M. Dumontier, G. Wainer, and J. Cheetham, *Vesicle, synapsin and actin concentration time series modelling at the presynaptic nerve terminal (poster)*, Proceedings of Symposium on Progress in Systems Biology 2006, Ottawa, ON, Canada, 2006.

[27] N. Giambiasi and G. Wainer, *Using G-DEVS and Cell-DEVS to model complex continuous systems*, Simul.: Trans. Soc. Model. Simul. Int. 81 (2005), pp. 137–151.

[28] R. Goldstein and G.A. Wainer, *Modelling tumor-immune systems with Cell-DEVS*, June, 2008.

[29] G. Wainer, *Discrete-Event Modeling and Simulation: A Practitioner's Approach*, CRC Press, Boca Raton, FL, 2009.

[30] G. Wainer, *CD++: A toolkit to develop DEVS models*, Softw. Pract. Exp. 32 (2002), p. 1261.

[31] B. Alberts, D. Bray, L. Lewis, M. Raff, K. Roberts, and D. Watson, *Molecular Biology of the Cell*, 1st ed., Garland Publishing, Inc., New York and London, 1983.

[32] P. Thorsness and T. Hanekamp, *Mitochondria: Origin*. in *Nature Encyclopedia of Life Sciences*, Nature Publishing Group, New York, 2000. Available at http://www.Els.Net/[Doi:10.1038/npg.Els.0001381] 2000.

[33] H. Curtis and N. Barnes, *Biology*, 5th ed., W.H. Freeman, New York, 1989.

[34] M. Aimar-Beurton, B. Korzeniewski, T. Letellier, S. Ludinard, J. Mazat, and C. Nazaret, *Virtual mitochondria: Metabolic modelling and control*, Mol. Biol. Rep. 29 (2002), pp. 227–232.

[35] J. Maber, *Step by step glycolysis*, Dept Biochemistry and Molecular Biology, The University of Leeds, 2004. Available at http://www.Jonmaber.Demon.Co.uk/glysteps

[36] ALIAS, Corp *Maya 6 features in detail*, Available at http://www.Alias.Com/eng/products-services/maya/file/maya6_features_in_detail.Pdf 2004.

[37] B.D. Hames and N.M. Hooper, *Instant Notes in Biochemistry*, 2nd Ed., Springer, New York, 2000.

[38] S.S. Andrews and D. Bray, *Stochastic simulation of chemical reactions with spatial resolution and single molecule detail*, Phys. Biol. 1 (2004), pp. 137–151.

[39] J. Stiles and T. Bartol, *Monte Carlo methods for simulating realistic synaptic microphysiology using MCell*, in *Computational Neuroscience: Realistic Modeling for Experimentalists*, E. De Schutter, ed., Boca Raton, FL, CRC Press, 2001, pp. 87–127.

Appendix. Model implementation in CD++

Besides the functions defined in the formal DEVS specification, CD++ requires defining the model basic architecture in C++. This includes the creation of the state variables and a constructor.

Step1.h: Model Definition

```
class Step1: public Atomic {

protected:
     Model &initFunction();
     Model &externalFunction(ExternalMessage&);
     Model &internalFunction(InternalMessage&);
     Model &outputFunction(InternalMessage&);

private:
     const Port & glucose;    // inputs
     const Port &ATPi;
     const Port &hexokinase;

     Port &glucose_6_phosphate;    // outputs
     Port &ADP;
     Port &H;

     Time preparationTime;
     double atpc;
     double glucosec;
     bool ifhex;
     double counter;
       };  // class Step1
```

Step1.cpp: Model Implementation

```
Step1 :: Step1(const string &name): Atomic(name)
glucose(addInputPort(‘glucose”))
ATPi(addInputPort(“ATPi”))
hexokinase(addInputPort(“hexokinase”))
glucose_6_phosphate(addOutputPort(“glucose_6_phosphate”))
ADP(addOutputPort(“ADP”))
H(addOutputPort(“H”))
preparationTime(0, 0, 10, 0) {
       counter = atpc = glucosec = 0;
       ifhex = false;
}

Model &Step1 :: externalFunction (const ExternalMessage &msg) {
   if(msg.port() =  = glucose) {
      glucosec = glucosec + msg.value();
      if ((atpc > 0)  = true))
         holdIn(active, Prep_Gluc);
   }
   else if(msg.port() =  = ATPi) {
       atpc = atpc + msg.value();
       if ((glucosec > 0)  = true))
       holdIn(active, Prep_ATPi);
   }
   else if (msg.port() =  = hexokinase) {
      ifhex = true;
      if ((glucosec > 0) 0))
      holdIn(active, Prep_Hexo);
   }
}
Model &;Step1 :: outputFunction(InternalMessage &msg) {
   if (counter ! = 0) {
       sendOutput(msg.time(), ADP, counter);
       sendOutput(msg.time(), glucose_6_phosphate, counter);
       sendOutput(msg.time(), H, counter);
   }
   return *this;
   }
   Model &Step1 :: internalFunction(const InternalMessage
        counter = 0;
         if (state() = = idle) {
              passivate();
   }
   else {
       if ((atpc > = 1) = 1) = true)) {
          if (atpc > glucosec) {
              atpc = atpc - glucosec;
              counter = glucosec;
              glucosec = 0;
          }
          else if (atpc < glucosec) {
              glucosec = glucosec-atpc;
              counter = atpc;
              atpc = 0;
          }
          else if (atpc = = glucosec) {
              counter = atpc;
              atpc = glucosec = 0;
          }
          holdIn(passive, Time :: Zero);
       }
       else {
          passivate();
       }
   '}
   return *this;
}
```

Table 9. Glycolysis coupled model.

```
[top]
components: step1@Step1 step2@Step2 step3@Step3 step4@Step4
step4to5@Step4to5 step5@Step5 step6@Step6 step7@Step7 step8@Step8
step9@Step9 step10@Step10

out: H ADP NADH H2O pyruvate ATPo
in: glucose ATPi hexokinase phosphoglucoisomerase PFK isomerase aldolase G3PD NAD P PGK PGM enolase
pyruvate_kinase

Link: glucose glucose@step1
Link: ATPi ATPi@step1
Link: hexokinase hexokinase@step1
...
Link: aldolase aldolase@step4
Link: isomerase isomerase@step4to5
...
Link: ATPo@step7 ATPo
Link: H2O@step9 H2O
Link: pyruvate@step10 pyruvate
Link: ATPo@step10 ATPo
...
```