Modeling and Simulation of Cellular Metabolism and Energy production by Mitochondria

by Roxana Djafarzadeh

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements of the degree of Master of Applied Sciences in Electrical and Computer Engineering

Ottawa Carleton Institute for Electrical and Computer Engineering

School of Information Technology and Engineering, Faculty of Engineering, University of Ottawa

Approved by _____

Dr. Tofy Mussivand, Thesis Supervisor

Approved by _____

Dr. Gabriel A. Wainer, Thesis Co-Supervisor

Approved by _____

Chair

MODELING AND SIMULATION OF CELLULAR METABOLISM AND ENERGY PRODUCTION BY MITOCHONDRIA

BY ROXANA DJAFARZADEH

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE DEGREE OF MASTER OF APPLIED SCIENCES IN ELECTRICAL AND COMPUTER ENGINEERING

OTTAWA CARLETON INSTITUTE FOR ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

FACULTY OF ENGINEERING

UNIVERSITY OF OTTAWA

SEPTEMBER 2004

ABSTRACT

It is the intent of this thesis to investigate and develop a simulation model for metabolic pathways; namely, Glycolysis and Krebs cycle, then to further improve it to a complete virtual mitochondria. CD++ tool is chosen to be used as the simulation tool for this purpose. CD++ is a modelling tool for simulation of complex physical systems, which can be used to simulate a variety of models. The simulation client provides users with the ability to create simulation models, send the simulation model to a remote CD++ server for execution and visualize the results with sophisticated and easy-to-use Graphical User Interfaces (GUI) locally, support multiple view visualization, run several models simultaneously, and as a result improving the analysis of simulation models. The simulation server can be used by various users around the world to perform multi-observer simulation using remote execution of the models.

Simulation of the known and proven biological models will offer the confidence to attack more complex biological problems and prove the practicality of this tool.

ACKNOWLEDGMENTS

I wish to thank Dr. Tofy Mussivand, FRSC (Cardiovascular Devices Division, University of Ottawa Heart Institute), for introducing me to Mitochondria, reviewing my work, and also for his financial support.

I wish to thank Dr. G. Wainer (Carleton University), for providing me with the right tools and advice, and without whom this achievement would not be possible.

Special thanks to Dr. G. Schatz, President (The Swiss Science and Technology Council), the father of mitochondrial research, whose papers and emails encouraged me to continue my work regardless of what is going on around me.

I also wish to thank Technology Partnerships Canada for always being willing to extend my contracts. Special thanks to Jeff Parker, Ken Laycock, Alison Tait, and Terrence Leblanc for paving the road and making this achievement a lot easier for me. God bless you.

And last, but not least, I thank my cats Jasmine and Poupée for their unconditional love, Sr. Dolores and Rica for always believing in me, Dr. Sitwell for taking care of my migraines and providing me with medication when my medical plan expired, Tunhi for always being there for a cup of coffee and a spiritual talk, and friends and family.

Finally, I would like to dedicate this work to my father who always encouraged me to further my education.

LIST OF SYMBOLS

ADP	Adenosine Diphosphate
API	Application Program Interface
ATP	Adenosine Triphosphate
CD++	A Modeling and Simulation Toolkit
DEVS	Discrete Events Systems specification
ETC	Electron Transport Chain
FAD	Flavin Adenine Dinucleotide
$FADH_2$	$FAD + 2H^+ + 2e^-$
GUI	Graphical User Interface
NAD^+	Nicotinamide Adenine Dinucleotide
NADH	$NAD^+ + H^+ + 2e^-$

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 - INTRODUCTION

Simulation is becoming increasingly important in the analysis and design of complex systems such as biological processes. The simulation process begins with with any problem to be solved or understood, such as the spread of a virus through a group of cells. A model is an abstract representation of a real system. In most cases, these models can be defined as mathematical representations of the real models and can be analyzed using mathematical techniques. However, the complexity of biological systems sometimes makes it impossible to use analytical methods.

1.1 MOTIVATION

Many important problems in cell biology require the dense nonlinear interactions between functional modules to be considered. Computer simulation tools are widely used to study complex systems. For these complex systems, a formal specification of the simulation model is necessary before its implementation. The conceptual simulation model must be validated before its implementation in a simulation environment. Several modeling paradigms have been developed to specify dynamic systems formally; each one is better suited to model different kind of system [31].

The importance of computer simulation in understanding cellular processes is now widely accepted, and a variety of simulation algorithms useful for studying certain

subsystems have been designed. Many of these are already widely used, and a large number of models constructed on these existing formalisms are available. A significant computational challenge is how we can integrate such sub-cellular models running on different types of algorithms to construct higher order models.

This thesis addresses one solution for simulating complex biological models: Applying CD++ – a toolkit for M&S based on the DEVS formalism – in models of complex biological models [3].

1.2 OBJECTIVES

It is the objective of this thesis to propose technological research, and development methods of modeling and simulation, and visualization methods for methabolic pathways. The application of this paradigm is shown through modeling of complex biological systems developed in CD++. These complex applications can be implemented in a simple fashion, and they can be executed effectively. The simulation models of glycolysis and Krebs cycle are highlighted in Chapters 4 and 5 respectively.

The goal is to understand and control dynamics of mitochondrial metabolism through computer simulation in a whole organelle scale. For this purpose, a mitochondrial model was constructed which included the biological pathways – the glycolysis and the Krebs cycle and the electron transport chain at the inner-membrane. Currently, the model is examined with emphasis on cellular metabolism and energy production aspect of mitochondria, based on the two biological pathways: Glycolysis and Krebs cyle.

The DEVS formalism was proposed to model discrete events systems. Basic DEVS models, called atomic models, are specified as black boxes and several DEVS models can be integrated together forming a structural model, called a coupled model. The CD++ tool implements the DEVS models, which provides a general framework to define and simulate complex generic models [3].

Given this analysis of the subject, the contributions of this thesis are the proposed modeling and simulation tool for complex biological systems, and it is believed that existing applications can benefit from these results.

1.3 THESIS PLAN

To reach the objectives of this thesis, two chapters are dedicated to general discussion on mitochondria, and different simulation tools. The next chapters explain modeling of the two biological pathways: glycolysis and Krebs cycle. Chapter 6 will shortly touch the electron transport chain as a research in progress, and chapter 7 will explain the graphical user interfaces. The detailed organization of the thesis is as follow:

- Chapter 2 is a background for non-biologists and it is divided in following sections: Mitochondria, Glycolysis, Krebs cycle, and Electron Transport Chain.

- Chapter 3 is intended for non-engineers and discusses the basics of Modeling and Simulation in general first, and then CD++ tool which is used as the simulation tool.

- Chapter 4 presents the modeling and simulation of Glycolysis using CD++ tool.

- Chapter 5 presents the Krebs cycle modeling and simulation using CD++ tool.

- Chapter 6 gives an introduction to Electron transport chain and oxidative phosphorylation, and the future work in progress to complete the simulation of mitochondria.

- Chapter 7 describes all the graphical user interfaces available for this tool.

- Chapter 8 presents conclusions, and suggests the direction for future work.

- Appendix A includes the code for Glycolysis model.

- Appendix B includes the code for Krebs cycle model.

CHAPTER 2 – BACKGROUND

Mitochondria are small double-membrane organelles found in the cytoplasm of eukaryotic[1] cells. Mitochondria are responsible for converting nutrients into the energy yielding molecule, Adenosine TriPhosphate (ATP), to fuel the cell's activities. This function is the reason why mitochondria are frequently referred to as the powerhouse of the cell [5].

2.1 MITOCHONDRIA

Mitochondria are unusual organelles in that they contain their own DNA (deoxyribonucleic acid), and ribosomes (protein-producing organelles). Within the mitochondria, the DNA directs the ribosomes to produce proteins, many of which function as enzymes (biological catalysts) in ATP production. The number of mitochondria in a cell depends on the cell's function. Cells with particularly heavy energy demands, such as muscle cells, have more mitochondria than other cells [13][15][25].

2.1.1 MITOCHONDRIAL STUCTURE

A mitochondrion is typically bean shaped ranging in size from 0.5-1 micrometer in length. Mitochondria can be divided into four components: outer membrane, inter-membrane space, inner membrane, and the matrix (see figure 2.1). The smooth outer

[1] Eukaryote: Cells with a nucleus

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. Cristae are the sites of ATP synthesis, and their folded structure greatly increases the surface area where ATP synthesis occurs. Transport proteins, molecules in the electron transport chain, and enzymes that synthesize ATP are among the molecules embedded in the cristae. The cristae enclose a liquid-filled region known as the inner compartment, also called matrix, which contains mitochondrial genome and the majority of gene products contained within the mitochondria. These gene products include various enzymes involved in the process of aerobic respiration, proteins necessary for the import of proteins into mitochondria, and proteins and nucleic acids required for the mitochondrial genome [14][25].

It is important to note that cristae have a very high importance in not only containing and organizing the electron transport chain and the ATP pumps, but also separating the matrix from the space that contains the hydrogen ions (protons), allowing the gradient needed to drive the pump [5].

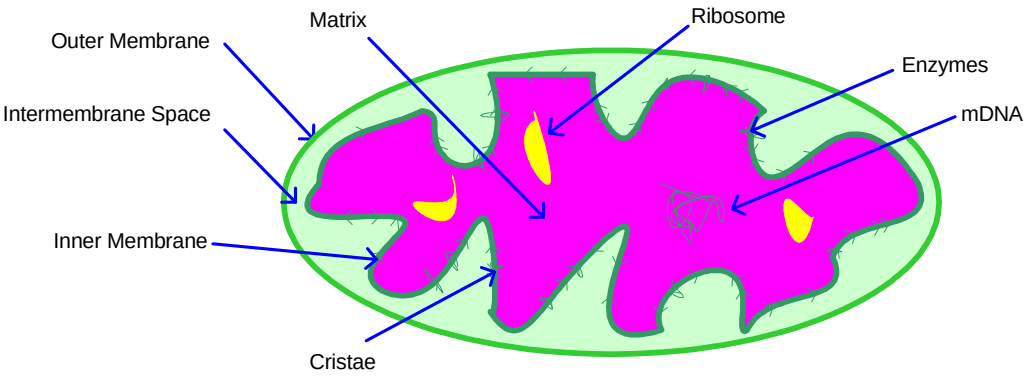

Figure 2.1 - 2-D Model of Mitochondria

2.1.2 MITOCHONDRIAL FUNCTION

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 a process called, glycolysis, to form pyruvic acid, which is then transported into the mitochondrion. In a series of reactions, part of which is called Krebs cycle[2], the pyruvic acid reacts with water to produce carbon dioxide and ten hydrogen atoms. The electron transport chain separates the electron and proton in each of the ten hydrogen atoms. The ten electrons are sent through the electron transport chain and some eventually combine with oxygen to form water.

[2] The Krebs cycle was named after Sir Hans Adolf Krebs (1900-1981), who proposed the key elements of this pathway in 1937 and was awarded the Nobel Prize in Medicine, for its discovery in 1953. See chapter 4 for more details.

Energy is released as the electrons flow from the coenzymes down the electron transport chain to the oxygen atoms, and this energy is trapped by the components of the electron transport chain. As the electrons flow from one component to another, the components pump random protons from the matrix to the outer compartment. The protons cannot return to the matrix except by the enzyme ATPase which is embedded in the inner membrane. As the protons flow back into the matrix, ATPase 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 each minute in a typical cell. The ATP is transported to the cytoplasm of the cell where it is used for virtually all energy-requiring reactions it performs. 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 [15][25].

2.1.3 ORIGIN OF MITOCHONDRIA

Mitochondria have significant features that resemble those of prokaryotes, primitive cells that lack a nucleus. Mitochondrial DNA is circular, like the DNA of prokaryotes, and its ribosomes are also similar to prokaryotic ribosomes. Mitochondria divide independently of the cell through binary fission, the method of cell division typical of prokaryotes.

The prokaryote-like features of mitochondria lead many scientists to support the endosymbiosis hypothesis. This hypothesis states that millions of years ago, free-living prokaryotes capable of aerobic respiration were engulfed by other, larger prokaryotes but

not digested, possibly because they were able to resist digestive enzymes. The two cells developed a symbiotic, or cooperative, relationship in which the host cell provided nutrients and the engulfed cell used these nutrients to carry out aerobic respiration, which provided the host cell with an abundant supply of ATP. The engulfed cells evolved into mitochondria, which retain the DNA and ribosomes characteristic of their prokaryotic ancestors.

A recent comparison of samples of human mitochondrial DNA suggests that humans have descended from a woman who lived in Africa 140,000 to 290,000 years ago. Genetic samples taken from African, Asian, Australian, European, and New Guinean ethnic groups revealed a specific number of mitochondrial DNA types. The African mitochondrial DNA occupies the longest and oldest of the branches, giving rise to the other ethnic groups [13][23].

2.1.4 IMPORTANCE OF MITOCHONDRIA

The DNA in mitochondria is used to track certain genetic diseases, and to trace the ancestry of organisms that contain eukaryotic cells. In many animal species, mitochondria tend to follow a pattern of maternal inheritance. When a cell divides, the mitochondria replicate independently of the nucleus. The two daughter cells formed after cell division each receive half of the mitochondria as the cytoplasm divides. When an egg is fertilized by a sperm, the sperm's mitochondria are left outside the egg. The fertilized zygote inherits only the mother's mitochondria. This maternal inheritance

creates a family tree that is not affected by the typical shuffling of genes that occurs between a mother and father [17].

While the DNA within mitochondria directs the synthesis of enzymes for aerobic respiration, it also codes for proteins important in the nervous system, circulatory system, and other body functions. A number of genetic diseases, including diabetes mellitus, deafness, heart disease, Alzheimer's disease, Parkinson disease, Leber's Hereditary Optic Neuropathy (a condition of complete or partial blindness) are associated with mutations in mitochondrial DNA. A relatively new medical specialty, mitochondrial medicine, seeks to understand the role of mitochondrial DNA mutations in genetic diseases [17].

Recently, the mitochondrion has also been identified as an important component in the pathway of programmed cell death, apoptosis. In response to certain signals, mitochondria swell and release cytochrome c, which in turn is another apoptotic pathway coactivator. Those mitochondria also serve as a source for reactive oxygen species that contribute to cell death during apoptosis [17].

Another use of mitochondrial DNA analysis is in forensic science. For example, the identities of the skeletons alleged to be those of Tsar Nicholas II, the last Russian tsar, and his family were recently established using mitochondrial DNA. The mitochondrial DNA of a living maternal relative of the tsar's family was found to be an exact match to the suspected remains of the tsar's wife, Alexandra, and three children [13][25].

2.2 GLYCOLYSIS

Glycolysis, also called Embden-Meyerhof pathway, is a sequence of reactions used by virtually all cells to metabolize glucose. It involves ten steps during which glucose is broken down to two molecules of pyruvate. In this process, a net of two molecules of ATP are formed. [12]

The role of glycolysis (Glyco=sweet, sugar; lysis=to split) is to produce energy. Glycolysis takes place outside mitochondria, in the cytosol. This produces about 15% of the energy of aerobic respiration. Glycolysis is the basis for energy metabolism in virtually all the living creatures in a sequence of ten reactions which converts a glucose molecule into two pyruvate molecules with the production of NADH and ATP. Specific enzymes control each of the different reactions. This process happens in two phases, where in the first phase glucose is converted into two Glyceraldehyde-3-Phosphate molecules (GDP), and the second phase two pyruvate molecules. There is a net gain of 2 ATP at the end of glycolysis. Glycolysis itself does not require oxygen. [12]

2.2.1 THE PATHWAY OF GLYCOLYSIS

The first step in glycolysis is phosphorylation[3] of glucose by hexokinase. This reaction consumes one ATP molecule. Glucose 6-phosphate is then rearranged to form fructose 6-phosphate by phosphoglucoisomerase. Phosphofructokinase (PFK) then uses another ATP molecule to form fructose 1,6-bisphosphate (also called fructose 1,6-

[3] Phosphorylation: The addition of a phosphate group to a compound is called phosphorylation [P.172 Chapter7/Harvesting the Energy in Nutrients].

diphosphate). At this point the molecule splits into 2 molecules by aldolase: Dihydroxyacetone phosphate (DHP), and Glyceraldehyde 3-phosphate (GDP or PGAL). Isomerase converts dihydroxyacetone phosphate immediately into glyceraldehyde 3-phosphate. After this step, everything takes place twice, once for each GDP derived from original glucose (see table 1) [12].

Table 1 - First Phase: Preparatory Steps

Step 1	Glucose enters the cell and is phosphorylated on the number six carbon.	This ATP coupled reaction: • Is catalyzed by hexokinase. (Kinase is an enzyme involved in phosphate transfer.) • Requires an initial investment of ATP. • Makes glucose more chemically reactive. • Produces glucose-6-phosphate. Since the plasma membrane is relatively impermeable to ions, the addition of an electrically charged phosphate group traps the sugar in the cell.
Step 2	An isomerase catalyzes the rearrangement of glucose-6-phosphate to its isomer, fructose-6-phosphate.	
Step 3	Carbon one of fructose-6-phosphate is phosphorylated.	This reaction: • Required an investment of still another ATP. • Is catalyzed by phosphofructokinase, an allosteric enzyme that controls the rate of glycolysis.
Step 4	Aldolase cleaves the six-carbon sugar into two isomeric three-carbon sugars.	• This is the reaction for which glycolysis is named. • For each glucose molecule that begins glycolysis, there are two product molecules for this and each succeeding step.
Step 5	An isomerase catalyzes the reversible conversion between the two three-carbon sugars.	This reaction: • Never reaches equilibrium because only one isomer, glyceraldehyde phosphate, is used in the next step of glycolysis. • Is thus pulled towards the direction of glyceraldehyde phosphate, which is removed as fast as it forms.

		• Results in the net effect that, for each glucose molecule, two molecules of glyceraldehyde phosphate progress through glycolysis.

Each of glyeraldehyde 3-phosphate can then be oxidized by a molecule of NAD^+, in the presence of glyceraldehyde 3-phosphate dehydrogenase, to form 1,3-bisphosphoglycerate. Next, phosphoglycerate kinase generates a molecule of ATP while forming 3- phosphoglycerate. This is where for each 2 molecules of ATP used, 2 molecules of ATP have been synthesiszed. It is important to note that the phosphate needed to generate 1,3-bisphosphoglycerate, comes from inorganic phosphate (P_i) dissolved in the cell's cytoplasm, and not from ATP. As substrate level phosphorylation requires ADP, when ADP is missing, and there is plenty of ATP, this reaction does not occur, making this step an important regulatory point of the pathway. Phosphoglyceromutase then forms 2-phosphoglycerate. Phosphoenolpyruvate is then formed in the presence of enolase. Here another substrate-level phosphorylation produces a molecule of ATP, and pyruvate in the presence of pyruvate kinase. This serves as an additional regulatory step (see table 2) [12].

Table 2 - Second Phase: Oxidative Steps

Step 6	An enzyme catalyzes two sequential reactions:	• Glyceraldehyde phosphate is oxidized and NAD^+ is reduced to $NADH + H^+$. o This reaction is very exergonic (Delta G = -10.3 kcal/mol) and is coupled to the endergonic phosphorylation phase. o For every glucose molecule, 2 NADH are produced. • Glyceraldehyde phosphate is phosphorylated on carbon number one. o The phosphate source is inorganic phosphate, which

		is always present in the cytosol. o The new phosphate bond is a high energy bond at least as energetic as the phosphate bonds of ATP.
Step 7	ATP is produced by substrate level phosphorylation.	• In a very exergonic reaction, the phosphate group with the high energy bond is transferred from 1, 3-diphosphoglyceric acid to ADP. • For each glucose molecule, two ATP molecules are produced. The ATP ledger now stands at zero as the initial debt of two ATP from steps one and three is repaid.
Step 8		• In preparation for the next reaction, a phosphate group on carbon three is enzymatically transferred to carbon two.
Step 9	Enzymatic removal of a water molecule:	• Creates a double bond between carbons one and two of the substrate. • Rearranges the substrate's electrons, which transforms the remaining phosphate bond into an unstable high energy bond.
Step 10	In this last step of glycolysis, ATP is produced by substrate level phosphorylation.	• In a highly exergonic reaction, a phosphate group is transferred from PEP to ADP. • For each glucose molecule, this step produces two ATP.

After the formation of fructose 1,6 bisphosphate, many of the reactions are energetically unfavorable. The only reactions that are favorable are the 2 substrate-level phosphorylation steps that result in the formation of ATP. These two reactions pull the glycolytic pathway to completion. [12]

Table 3 - Inputs, Outputs, and Enzymes involved in Glycolysis

Steps	Inputs	Outputs	Enzyme
1	Glucose, ATP	Glucose 6-phosphate, ADP	Hexokinase
2	Glucose 6-phosphate	Fructose 6-phosphate	Phosphogluco-isomerase
3	Fructose 6-phosphate, ATP	Fructose 1,6-diphosphate, ADP	Phospho-fructokinase
4	Fructose 1,6-diphosphate	Dihydroxyacetone phosphate (DHP)	Aldolase
5	Fructose 1,6-diphosphate	Glyceraldehyde 3-phosphate (GDP) (PGAL)	Triose phosphate isomerase
6 (X2)	Glyceraldehyde 3-phosphate, Pi, NAD^+	1,3-diphosphoglycerate, NADH, H^+	Glyceraldehyde-3P-dehydrogenase

7 (X2)	1,3-biphosphoglycerate, ADP	3-phosphoglycerate, ATP	Phosphoglycerate kinase (PGK)
8 (X2)	3-phosphoglyceric acid	2-phosphoglyceric acid	Phophoglycerate mutase
9 (X2)	2-phosphoglyceric acid	Phosphoenolpyruvic acid, H_2O	Enolase
10 (X2)	Phosphoenolpyruvic acid, ADP	Pyruvic acid, ATP	pyruvate kinase

Under standard condition, reactions 1, 3, 7, 10 are exergonic. Under cellular conditions, reactions 1, 3, and 10 are exergonic. Exergonic reactions are the site of regulation of glycolysis.

2.3 KREBS CYCLE

During glycolysis, glucose is broken down into pyruvate. Each reaction is designed to produce some hydrogen ions (electrons) that can be used to make ATP. However, only 4 ATP molecules can be made by one molecule of glucose through this pathway. That is why mitochondria and oxygen are so important.

The breakdown process continues through the Kreb's cycle inside the mitochondria in order to get enough ATP. Pyruvate is carried into the mitochondria and there, it is converted into Acetyl Co-A which enters the Krebs' cycle. This first reaction produces carbon dioxide because it involves the removal of one carbon from the pyruvate [11].

Table 4 - Summary of Kreb's Cycle

Molecule	Enzyme	Reaction Type	Reactants/ Coenzymes	Products/ Coenzymes
1. Citrate	Aconitase	Dehydration		H_2O
2. CIS-Aconitate	Aconitase	Hydration	H_2O	
3. Isocitrate	Isocitrate Dehydrogenase	Oxidation	NAD^+	$NADH+H^+$
4. Oxalosuccinate	Isocitrate Dehydrogenase	Decarboxylation		
5. α-Ketoglutarate	α-Ketoglutarate Dehydrogenase	Oxidative Decarboxylation	NAD^+ CoA-SH	$NADH+H^+$ CO_2

6. Succinyl-CoA	Succinyl-CoA Synthetase	Hydrolysis	GDP P_i	GTP CoA-SH
7. Succinate	Succinate Dehydrogenase	Oxidation	FAD	$FADH_2$
8. Fumarate	Fumarase	Addition (H_2O)	H_2O	
9. L-Malate	Malate Dehydrogenase	Oxidation	NAD^+	$NADH+H^+$
10. Oxaloacetate	Citrate Synthase	Condensation		
11. Acetyl-CoA				

2.3.2 REGULATION OF KREBS CYCLE

The Krebs cycle is regulated at the steps catalyzed by citrate synthase, isocitrate dehydrogenase and α-ketoglutarate dehydrogenase via feedback inhibition by ATP, citrate, NADH and succinyl COA, and stimulation of isocitrate dehydrogenase by ADP. Pyruvate dehydrogenase, which converts pyruvate to acetyl CoA to enter the cycle, is inhibited by acetyl CoA and NADH. This enzyme itself is inactivated by phosphorylation, catalyzed by pyruvate dehydrogenase kinase. A high ratio of NADH/NAD+, acetyl CoA.CiA or ATP/ADP stimulates phosphorylation of pyruvate dehydrogenase inactivating the enzyme. Pyruvate inhibits the kinase. Dephosphorylation by a phosphate reactivates pyruvate dehydrogenase.

Overall, the cycle speeds up when there is a low concentration of ATP and NADH, and high concentration of ADP, and slows down as ATP, and then NADH, succinyl CoA and citrate accumulates [11].

2.3.3 ENERGY YIELD

From each turn of Krebs cycle, a total of 12 ATP molecules are produced: One directly from the cycle, 9 from three NADH, and 2 from one $FADH_2$ molecules produced by the cycle by the oxidative phosphorylation.

2.4 ELECTRON TRANSPORT CHAIN

The final metabolic pathway of cellular respiration, after glycolysis and the Krebs cycle, is Electron Transport Chain. Most of ATP molecules generated from a single glucose molecule during cellular respiration come from oxidative phosphorylation. In eukaryotes, this process takes place in the inner membrane of mitochondria.

The synthesis of ATP is a simple matter of stoichiometry[4]. Most of the ATP is generated by the proton gradient that develops across the inner mitochondrial membrane. Three ATPs are generated from each NADH, while only two ATPs are generated by each $FADH_2$. There is never 38 ATP produced in real conditions. It probably never increases 30 ATP [15][18].

2.4.1 COMPLEXES I, II, III, IV, AND V

The respiratory chain consists of five complexes of integral membrane proteins and two freely diffused molecules that shuttle electrons from one complex to the next:

[4] Stoichiometry: The fixed ratios of reactants to products in a chemical reaction.

Ubiquinone (Coenzyme Q), and Cytochrome c. There five complexes associated with the electron transfer chain are shown in the following table:

Table 5 - 5 Complexes of Inner Membrane

Enzyme	Name
Complex I	NADH dehydrogenase (or) NADH-CoQ oxidoreductase
Complex II	Succinate dehydrogenase (or) Succinate-CoQ oxidoreductase
Complex III	Cytochrome b-c1 complex (or) CoQ-Cytochrome c oxidoreductase
Complex IV	Cytochrome c oxidase
Complex V	ATP synthase (or) F_0F_1 Particle

Complexes I, III, and IV transport protons across mitochondria membrane. Coenzyme Q and Cytochrome c are only mobile electron carriers transferring electrons between complexes. All four complexes operate independently from each other.

Complex I, (NADH dehydrogenase complex, or NADH-CoQ oxidoreductase), catalyzes the transfer of electons from NADH to Coenzyme Q (CoQ):

$$NADH + CoQ + H^+ \rightarrow NAD^+ + CoQH_2$$

In the process of reducing CoQ, it also translocates protons, helping to provide the electrochemical potential used to produce ATP.

Complex III, (CoQ-Cytochrome c reductase complex, cytochrome bc_1 complex), catalyzes the reduction of cytochrome c by accepting reducing equivalents from Coenzyme Q (CoQ):

$$CoQH_2 + 2Fe^{+3}\text{-cytochrome c} \rightarrow CoQ + 2\ Fe^{+2}\text{-cytochrome c}$$

Complex IV, (cytochrome c oxidase), is the terminal electron acceptor in the chain, taking four reducing equivalents from cytochrome c and converting molecular oxygen to water. In this process, it translocates protons, helping to establish a Chemiosmotic Potential that ATP Synthase then uses to synthesize ATP.

Complexes I, III, and IV are proton pumps. A proton pump is an integral membrane protein that is capable of moving protons across the membrane of a cell, mitochondrion, or other sub-cellular compartment, thereby creating a difference or gradient in both pH and electrical charge (ignoring differences in buffer capacity) and tending to establish an electrochemical potential.

Complex II is part of the Krebs cycle and does not pump protons, and Complex V uses the electrochemical potential generated to create ATP. Complex IV is the terminus of the electron transfer chain, consuming oxygen and making water. Cytochrome c^5 is also an essential part of the electron transfer chain. It is a soluble protein loosely associated with the inner mitochondrial membrane and transfers electrons between Complexes III and IV [1].

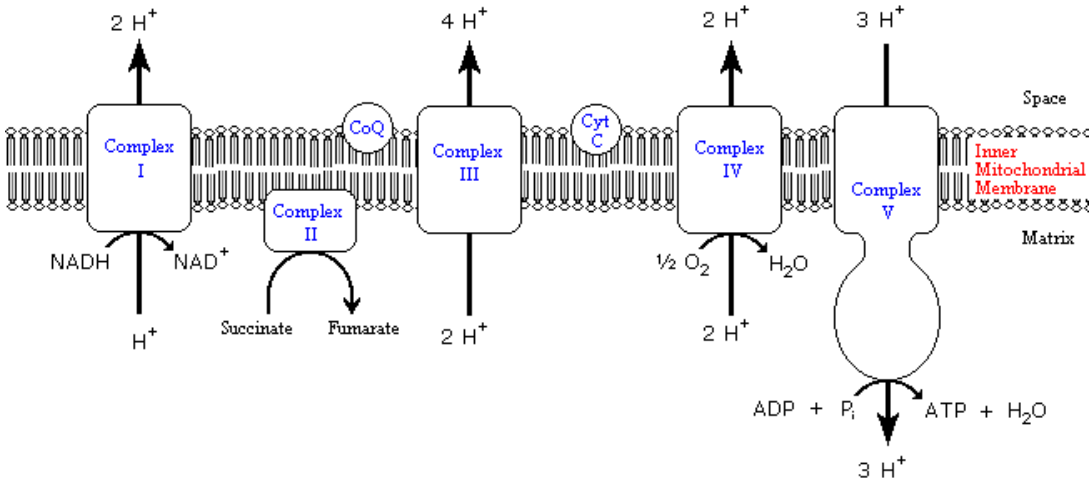

Figure 2.2 - Oxidative Phosphorylation using 5 Complexes
[Source: http://www.gwu.edy/~mpb/oxidativephos.htm]

2.4.2 OXIDATIVE PHOSPHORYLATION

Oxidative phosphorylation simply means the process that couples the removal of hydrogen ions from one molecule and giving phosphate molecules to another molecule[6]. As the Kreb's cycle runs, hydrogen ions (or electrons) are donated to the two carrier

---

5 Cytochrome C is a small heme protein found loosely associated with the inner membrane of the mitochondrion. It is a soluble protein, unlike other cytochromes and is an essential component of the electron transfer chain. It is capable of undergoing oxidation and reduction, does not bind to oxygen. It transfers electrons between complexes III and IV [8].

6 When you take hydrogen ions or electrons away from a molecule, you "oxidize" that molecule. When you give hydrogen ions or electrons to a molecule, you "reduce" that molecule. When you give phosphate molecules to a molecule, you "phosphorylate" that molecule.

molecules in 4 of the steps. They are picked up by either NAD or FAD and these carrier molecules become NADH and FADH as they now carry a hydrogen ion.

These electrons are carried chemically to the electron transport chain in the mitochondrial cristae. The NADH and FADH essentially serve as a ferry in the lateral plane of the membrane diffusing from one complex to the next. At each site is a hydrogen (or proton) pump which transfers hydrogen from one side of the membrane to the other. This creates a gradient across the inner membrane with a higher concentration of Hydrogen ions in the inter-membrane space. The electrons are carried from complex to complex by ubiquinone and cytochrome C.

The third pump in the series catalyzes the transfer of the electrons to oxygen to make water. This chemiosmotic pumping creates an electrochemical proton gradient across the membrane which is used to drive the energy producing machine: The ATP synthase. This molecule is found in small elementary particles that project from the cristae. This process requires oxygen which is why it is called aerobic metabolism. The ATP synthase uses the energy of the hydrogen ion gradient (also called proton gradient) to form ATP from ADP and Phosphate. It also produces water from the hydrogen and the oxygen. Thus, each compartment in the mitochondrion is specialized for one phase of these reactions [1][18].

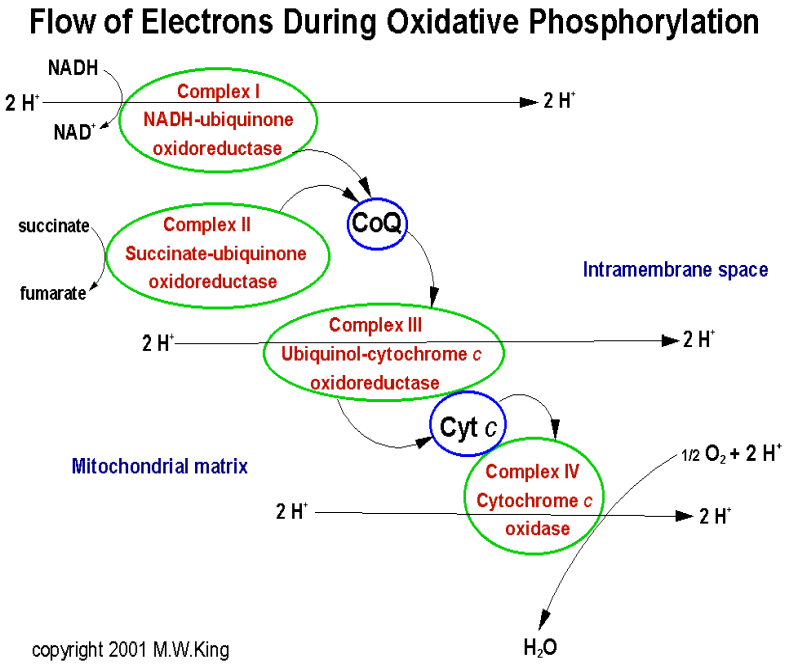

Figure 2.3 - Flow of Electrons During Oxidative Phosphorylation [Source: Source: 2001 M.W.King]

2.5 SUMMARY

Cellular respiration is the process of oxidizing food molecules, like glucose, to carbon dioxide and water. The energy released is trapped in the form of ATP for use by all the energy-consuming activities of the cell. Mitochondria are membrane-enclosed organelles distributed through the cytosol with the main function of converting the potential energy of food molecules into ATP. Mitochondria have: An outer membrane that encloses the entire structure, an inner membrane that encloses a fluid-filled matrix, and the area

between the two is called inter-membrane space. The inner membrane is elaborately folded named cristae projecting into the matrix.

The whole idea behind having mitochondria is to get as much ATP out of glucose or other food products, as possible. If we have no oxygen, we get only 4 molecules of ATP energy packets for each glucose molecule in glycolysis process. However, if we have oxygen, then through the Krebs cycle we can produce many more ATPs. From the Krebs cycle we get 24-28 ATP molecules out of one molecule of glucose converted to pyruvate (plus the 4 molecules out of glycolysis). So, one can see how much more energy we can obtain from a molecule of glucose in presence of mitochondria and oxygen [1][15].

Table 6 - A summary of Steps, Location, Inputs, and Outputs

Phase	Location	Summary	Starting Materials	End Products
Glycolysis	Cytoplasm	Glucose is degraded to pyruvate, producing 2 ATP molecules and hydrogens; can proceed anaerobically	Glucose, ATP, NAD^+	Pyruvate, ATP, NADH
Acetyl coenzyme A	Mitochondria	Pyruvate is degraded and combined with coenzyme A to form acetyl CoA; hydrogens and carbon dioxide are released	Pyruvate, coenzyme A	Acetyl CoA, carbon dioxide, NADH
Citric Acid Cycle	Mitochondria	Series of reactions in which the acetyl portion of acetyl CoA is degraded to carbon dioxide; hydrogens are released	Acetyl CoA, Water	Carbon Dioxide, NADH, $FADH_2$, ATP
Electron Transport Chain	Mitochondria	Chain of several electron transport molecules; H's are passed along the chain; energy released is used to form a proton gradient; ATP is synthesized as protons move across the gradient; oxygen is the final H-acceptor	NADH, $FADH_2$, Oxygen	ATP, Water

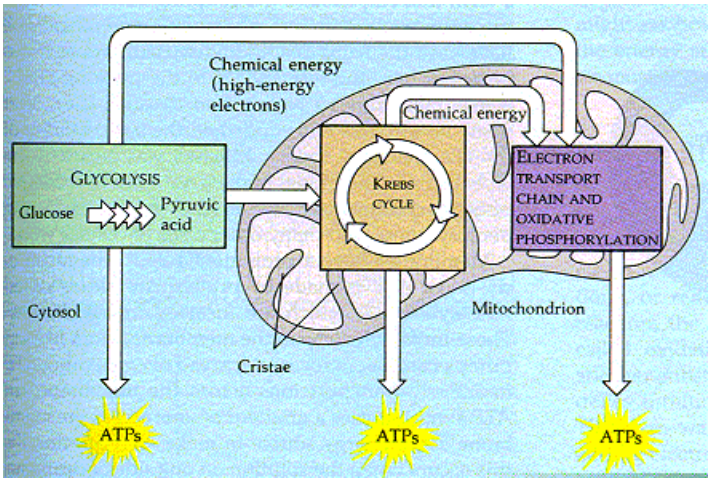

Figure 2.4 - Metabolic Pathways from Glycolysis to Electron Transport Chain [Source: Source: Natural Toxins Research Center at Texas A&M University – Kingsville]

Table 7 - Total ATPs Generated in the process of the Oxidation of One Molecule of Glucose [12]

In the Cytoplasm		
Glycolysis:	2 ATP	2 ATP
In the Mitochondrion		
Glycolysis:	2 NADH → 6 ATP	6 ATP[7]
Respiration:	*x2*	
Pyruvic acid → Acetyl CoA:	1 NADH → 3 ATP	6 ATP
Krebs cycle:	1 ATP 3 NADH → 9 ATP 1 FADH2 → 2 ATP	24 ATP
Total		38 ATP

[7] In some cells, the energy cost of transporting the electrons from the NADH molecules formed in glycolysis across the inner mitochondrial membrane lowers the net yield from these 2 NADH to 4 ATP, thus the total maximum yield in these cells is 36 ATP.

CHAPTER 3 – MODELING AND SIMULATION TOOLS

This chapter begins with a general discussion of simulation and biosimulations. Then it jumps to the DEVS formalism, atomic models, coupled models, and cell-DEVS models. Later, CD++ toolkit is introduced with the reasons why it is chosen as our development toolkit for this biomedical application.

3.1 SIMULATION

Simulation includes physical, mathematical, and computer models. Although simulation predates computers, the practice of simulation has highly increased in importance with the arrival of digital computers. Nowadays, Computer simulations are being used in business, economics, engineering, space technology and medicine. Computer simulation is a powerful technique with a broad range of applications. It is a tool that can provide solutions to many complex systems. Simulation involves models that represent the subject under investigation.

Simulation is a powerful tool for analyzing and understanding a wide variety of complex systems. The simulation process begins with a problem to be solved, such as a network performance, the spread of a virus through a group of cells, or a biological process. By observing the real system, different entities are identified. A model is an

abstract representation of such system that is constructed accordingly. The execution of the model is carried by a simulator. The simulator consists of a computer system that executes the instructions of that model to generate its behaviour. Finally, the obtained results are compared to those of the real system for validation [37].

3.2 SYSTEMS AND MODELS

A system is a collection of independent objects called components, with a defined purpose. A model is a representation of a system or a component of that system. There are many types of models. To be useful, the model must represent the entity being investigated with regard to characteristic under investigation. Sometimes the behaviour of the model is of interest. At other times the model is analyzed for specific answers [48].

A system (reality) is usually defined as any ordered set of interrelated physical (or abstract) objects. Our interest in a specific system may be in one or more of these activities: analysis, design, control, or improved understanding or performance. By modeling we mean the study of the mechanisms inside a system, and using basic physical (biologic, economic, and so on) laws and relationships, a model is inferred. Models are therefore not reality, and a model, no matter how complex, is only a representation of reality and should never be confused with it. The computerized model is an operational computer program that implements a system's model. A record of predicted behaviour of the system is obtained from computer run(s). Model verification is defined as the

"substantiation that a computerized model represents the system's model within specified limits of accuracy". The level of agreement between the observed and predicted behaviour of the physical system is the essence of model validation. Validating a model requires comparing its behaviour (simulation results) with that of the real system (measured or observed data).

3.3 BIOSIMULATION

Computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output. Bio-Simulation aims at developing information technology with which we can easily represent and simulate complex biological systems and apply the technology to medicine and biology. It is the use of mathematical techniques for simulation of biological phenomena, including programming techniques.

Medical simulation is the quantitative description of biophysical behavior in terms of mathematical equations. The reasons for performing simulations include the desire to replicate the function of living organisms, to test of our understanding, and to investigate conditions that are difficult or even impossible to create experimentally.

The level of complexity in biological systems exceeds that of many other engineering applications. For instance, analysis of experimental results from the ionic currents from cardiac cell membranes suggest that there are perhaps tens of different types of channels all carrying potassium in and out of the cell, each with different kinetic behavior.

Software to deal with problems of this complexity will always lag behind a complete description of reality, but coming as close as possible requires specialized algorithms and code and high performance computers.

Modeling is the geometric counterpart to simulation in that the goal is not to describe function, but to quantitatively capture anatomy and physical locations of objects in space. From the locations of points in space, modeling seeks to define connections between these points in order to define areas, surfaces or volumes. Models in biomedical applications define anatomy of tissues and organs in the body by means of discrete points joined to form polygonal elements such as rectangles, triangles, hexahedra, and tetrahedra. There is a natural synergy between modeling and simulations in that many simulations require a geometric description of the tissue whose function is to be simulated.

Another category of computer applications is scientific visualization. Visualization is an essential component of virtually any problem and provides a means for viewing geometric models, experimental results, simulation results, and clinical observations. For example, visualizing a three-dimensional head model along with the MRI scans from the patient and the results from a source localization simulation requires the integration of many different types of visualization techniques - visualization of the geometrical mesh, visualization of the MRI data using volume rendering, visualization of the potentials and currents from the simulation using surface shading – all integrated into a single frame.

Modeling, simulation, and visualization have been oriented towards independent, sequential processing. In contrast, recent developments in software development are aimed at providing more integration and interactivity within the software system, allowing communication between elements of the system and a high degree of user control over the function of the program.

3.3.1 OTHER RELATED WORK

This unit is an introduction to the concepts and methods used in modeling and simulating biological systems. Biological systems are composed of many subsystems and components, each having its own unique characteristics and behaviour while contributing to the overall form and function of an entire system. These systems are highly complex; many components interact simultaneously and exhibit non-linear behaviour. These interactions and non-linearities must be taken into account when attempts are made to understand or predict their behaviour. Because of these complexities, classical mathematical methods used to study non-living physical or chemical systems have been inadequate for living systems. Simulation, based on quantitative models of biological processes and their interactions, can provide considerable insight into the behaviour of living systems and into ways of managing them to achieve specific goals.

It is becoming increasingly difficult to apply traditional theoretical methods to the formulation of coherent pictures of cell and organ function because it is no longer possible for a human theorist to integrate all of the available information. Instead, computer technologies must now be used to perform this integration.

Limitations in computer power have been a major obstacle for biomedical modeling, but there are strong arguments that High Performance Computing will bring about a dramatic change in making modeling more accurate and more useful.

Simulation is a tool that enables a better understanding of complex physical and natural systems. In the past few years, several simulation models of real biological systems have been developed such as: E-cell from Japan, virtual mitochondria in Europe, Cyber-cell, and virtual-cell.

3.3.1.1 E-CELL PROJECT

E-cell project was started 1996 to model and simulate various cellular processes with the aim of simulating the whole cell. E-cell not only models metabolic pathways but also protein synthesis and signal transduction. It is a virtual cell made of 127 genes sufficient for self-support. The metabolisms include transcription, translation, membrane transport, glycolysis pathway, and the phospholipid biosynthesis pathway for membrane structure. They claim that the major bottleneck in cell modeling is lack of quantitave data, and their system is a useful tool to conduct quantitative simulation based on those data obtained by mass-production of those quantitave metabolic data [27].

This model combines the Gillespie-Gibson stochastic algorithm and deterministic differential equations. Dramatic improvements in performance were obtained without significant accuracy drawbacks [27]. A modular, object-oriented simulation meta-algorithm based on a discrete-event scheduler and Hermite polynomial interpolation has

been developed and implemented. It is shown that this new method can efficiently handle many components driven by different algorithms and different timescales [28].

3.3.1.2 *QUANTITATIVE SIMULATION OF MITOCHONDRIAL ENERGY METABOLISM USING E-CELL SIMULATION ENVIRONMENT*

A kinetic model of mitochondrial energy metabolism using E-cell system[8] has been designed. This mitochondrial model includes the Electron Transport Chain and the Krebs cycle. Kinetic parameters for all the enzymes in the model are estimated. They continue to refine their model till the overall behaviour of mitochondrial energy metabolism becomes consistent with experimental data. Theirs final goal is to apply this system for analyses of human mitochondrial diseases [36].

3.3.1.3 *VIRTUAL MITOCHONDRIA: METABOLIC MODELLING AND CONTROL*

In this Model, Oxidative Phosphorylation was modelled in a quantitave way. Several kinetic and thermodynamic models of this process have been developed, and tested for a broad range of experimentally-measured prarameter values and system properties. This model was used to predict new properties of the system and the existence of new phenomena. They concluded that the basic knowledge of the kinetic parameters of enzymes or enzymatic complexes will enable us to predict the metabolic fluxes, their regulation and their control. Their goal was to apply to mitochondria, the method

[8] A simulation environment developed for whole-cell simulation

developed for whole cells in the post-genomic area, i.e. to construct and to analyse the metabolic maps from the genes [21].

3.3.1.4 THE SILICON CELL: COMPUTING THE LIVING CELL

Applying a computationally-based methodology to modeling cellular metabolism and regulation that captures the physical and chemical constraints inherently placed on cells, which limit cellular behavior. In silico approach incorporates experimental data about individual components of organisms to derive highly accurate models of cellular systems centered around metabolism – the chemical engine that drives the living process and forms the basis for much of human disease. For a target organism, genomic data is combined with biochemical and physiological data, as determined through today's high-throughput technologies, and then assembled into a virtual model in silico. Within these in silico models, the function of genetic circuits can be characterized: the coordinated activity of multiple genes, proteins and metabolites. The in silico model incorporates the known pattern of these reactions into a related network of pathways.

In silico modeling technology can be used to accurately predict the cellular functions, behavior, fitness and performance characteristics of cells under changing environmental, physiological and genetic conditions [2].

3.4 THE DEVS FORMALISM

DEVS (Discrete EVent System Specification) is a theoretical approach, which allows the definition of hierarchical models that can be easily reused [49]. One of the main advantages of DEVS framework is that it separates modeling from simulation. A real system modeled using DEVS can be described as a set of communicating atomic or coupled submodels. The atomic model is the lowest level and contains structural dynamics, while the coupled model is composed of one or more atomic and/or coupled models.

3.4.1 ATOMIC DEVS MODEL

An atomic DEVS model is defined by input ports, output ports, state variable, state transition functions, output function, and time advance function:

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, t_a >$$

Where:

X: external input events set;
S: sequential state set;
Y: external output events set;
δ_{int}: internal transition function; $S \rightarrow S$
δ_{out}: external transition function; $X * Q \rightarrow X$
λ: output function; $S \rightarrow Y$
t_a: time advance function; $t_a: S \rightarrow R_o \cup \infty$
Q: total state set; $Q = \{(s, e) \mid s \in S, e \in [0, t_a(s)]\}$, e: elapsed time

Each atomic model is provided with input and output ports that allow the model to communicate with other models. The input events set is made up of all possible inputs that may occur on the input ports, similarly the output set consists of all possible outputs the atomic model may have. The external transition function is invoked when an event occurs on an input port; this function determines what state change if any is required as a result of the event and the current state. The model remains in its current state for an amount of time determined by the time advance function, when this time has expired the output function is invoked, which sends output events from the output set on the output ports based on the current state. Following the invocation of the output function, the internal function is immediately invoked, which determines state change if any is required as a result of the current state.

3.4.2 COUPLED DEVS MODEL

A coupled DEVS model is composed of a set of atomic or/and coupled submodels (internal couplings, external input couplings, and external output couplings):

$$CM = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, select >$$

Where:

X: external input event set;

Y: external output event set;

D: 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 coupled model), defined by: $M_i = < X_i, S_i, Y_i, \delta_{inti}, \delta_{exti}, \lambda_i, t_{ai} >$

I_i: the set of influencees of model I (the models that can be influenced by outputs of model i),

Z_{ij}: $\forall$ j $\in$ D, Z_{ij} is the i to j translation function;

Select: tie-breaking selector.

Coupled models are defined by a set of basic components which are interconnected through their model interfaces. The influences set determines which components should receive the outputs of each component. The translation function converts the outputs of one component to the inputs of other components [21].

3.5 THE CD++ TOOLKIT

The CD++ toolkit was developed in order to implement the theoretical concepts already specified by the DEVS formalism. (Wainer 2002) is a tool??????. The toolkit has been built as a set of independent software pieces, each of them independent of the operating environment chosen. The defined models are built as a class hierarchy, and each of them is related with a simulation entity that is activated whenever the model needs to be executed. New models can be incorporated into this class hierarchy by writing DEVS models in C++, overloading the basic methods representing DEVS specifications: external transitions, internal transitions and output functions. CD++ employs a virtual time simulation approach, which allows skipping periods of inactivity.

The abstract simulation technique enables defining and using different simulation engines without affecting existing models [26] [30].

3.5.1 CD++ ATOMIC AND COUPLED MODEL DEFINITION

Atomic models are created within the CD++ toolkit by creating C++ classes that are derivatives of the class Atomic. Atomic is an abstract class that declares a model's API (Application Program Interface) and defines some service functions the user can use to write the model. The atomic class then provides a set of services and requires a set of functions to be redefined. The following primitives are used to define the Atomic model's behaviour:

- holdIn(state,time): Instructs the model to remain in state for the specified time, following which the output and internal transition methods will be invoked. It corresponds to the ta(s) function of DEVS.

- passivate(): It is equivalent to holdIn(passive, infinity). It sets the next internal transition time to infinity. The model will only be activated again if an external event is received.

- getCurrentState(): It returns the current model's phase.

- sentOutput(time, port, value): Sends a message out on port:*port* at time:*time* with value:*value*.

- state(): returns the current state of the model.

The new class must overload the initFunction, externalFunction, internalFunction and outputFunction methods within the Atomic class. Each Atomic model instantiates ports which are unidirectional either input or output. These ports are used to exchange event messages between different atomic models.

Initialization method is invoked when the simulation starts; it performs the method body as well as setting the model state to passive and setting the time for the next schedule event to infinity.

- initFunction(): method invoked by the simulator at the beginning the simulation.
- externalFunction(ExternalMessage &): method invoked when an external event arrives to a port. It corresponds to the dext function of the DEVS formalism.
- internalFunction(InternalMessage &): method defining the dint function of the DEVS formalism.
- outputFunction(const CollectMessage &): in charge of transmitting the output events of the model. It corresponds to the l function of the DEVS formalism.

After creating a new Atomic model class, the class must be registered with the simulator by invoking the SingleModelAdm::Instance().registerAtomic method from within the MainSimulator::registerNewAtomics() method. Following registration the new atomic model should be added to the simulator makefile, and this makefile should be executed. This will compile the simulator and all new atomic models. Following compilation, the new atomic model may be instantiated within a model (MA) file, which defines a coupled DEVS model. Once an atomic model is defined, it can be combined

with others into a multicomponent model using a specification language specially defined for this purpose.

A model (MA) file consists of components, atomic model instances and links. A sample model file is listed below:

Table 8 - A CD++ Sample Model File [Source: Step6 of Glycolysis]

```
[[top]
components : step6@Step6
out : _13_BPG NADH H
in : GDP NAD P G3PD
Link : GDP GDP@step6
Link : NAD NAD@step6
Link : P P@step6
Link : G3PD G3PD@step6
Link : _13_BPG@step6 _13_BPG
Link : NADH@step6 NADH
Link : H@step6 H

[step6]
preparation : 00:00:05:000
```

The model file is made up of components that contain instances of Atomic models. There must always be at least one component in the MA file; the [top] model always defines the top level component. In formal specifications, four properties must be configured: components, output ports, input ports and links between models. The following syntax is used:

Components: A component is specified by inserting a line with the component name surrounded by square brackets. It describes the models integrating a coupled model and lists the components of the coupled model (atomic or coupled). For atomic models, an instance and a class name must be specified, allowing a coupled model to use more than one instance of a given atomic class. For coupled models, only the model name must be

given, and it must be defined as another group in the same file. The example above has the two components [top] and [step6]. The syntax is *modelName@className*, allowing more than one instance of the same model with different names. Atomic model instances have syntax: *Instance_name@atomic_model_name*, While component instances have syntax: *Instance_name@component_name*.

Out: The keyword out is followed by a list of output ports for the component.

In: The keyword in is followed by a list of input ports for the component.

Link: It describes the internal and external coupling scheme. It specifies links between ports on any two of the following: ports on atomic model instances, ports on component instances, ports on the component to which the link belongs. The syntax of a link is: *source_port@model*, *dest_port@model*. The name of the model is optional and, if it is not indicated, the coupled model being defined is used [23] [24].

3.6 SUMMARY

DEVS is an increasingly accepted framework for understanding and supporting the activities of modeling and simulation. DEVS is a sound formal framework based on generic dynamic systems, including well defined coupling of components, hierarchical, modular construction, support for discrete event approximation of continuous systems and support for repository reuse. DEVS theory provides a rigorous methodology for representing models, and it does present an abstract way of thinking about the world with independence of the simulation mechanisms, underlying hardware and midware [25].

DEVS allows modular description of models that can be integrated using a hierarchical approach. DEVS has been successfully used in previous efforts in model interoperability providing ease for reuse of simulation models.

Another advantage of using DEVS is that different exisiting techniques such as cellular automata, Petri Nets, State Charts, and Queueing models have been mapped to DEVS. This permits sharing information at the level of the model, and different submodels can be specified using different techniques, while keeping independence at the level of the simulation engine. Existing DEVS tools have shown their ability to execute such wide variety of models with high performance in standalone or distributed environments [25].

CHAPTER 4 - A SIMULATION MODEL OF GLYCOLYSIS

Glycolysis, also called Embden-Meyerhof pathway, is a sequence of reactions used by virtually all cells to metabolize glucose. It involves ten steps during which glucose is broken down to two molecules of pyruvate. In this process, a net of two molecules of ATP are formed [1].

This pathway has been discussed in detail in chapter two, but an over overview of the pathway will be given before getting into a simulation model of glycolysis.

4.1 INTRODUCTION

The role of glycolysis (Glyco=sweet, sugar; lysis=to split) is to produce energy. Glycolysis takes place outside mitochondria, in the cytosol. This produces about 15% of the energy produced by aerobic respiration. Glycolysis is the basis for energy metabolism in virtually all the living creatures in a sequence of ten reactions which converts a glucose molecule into two pyruvate molecules with the production of NADH and ATP. Specific enzymes control each of the different reactions. This process happens in two phases, where in the first phase glucose is converted into two Glyceraldehyde-3-Phosphate molecules (GDP), and the second phase two pyruvate molecules. There is a net gain of 2 ATP at the end of glycolysis. Glycolysis itself does not require oxygen [1].

4.2 THE PATHWAY

As mentioned in chapter two, the first step in glycolysis is phosphorylation[9] of glucose by hexokinase. This reaction consumes 1 ATP molecule. Glucose 6-phosphate is then rearranged to form fructose 6-phosphate by phosphoglucoisomerase. Phosphofructokinase (PFK) then uses another ATP molecule to form fructose 1,6-bisphosphate (also called fructose 1,6-diphosphate). At this point the molecule splits into 2 molecules by aldolase: Dihydroxyacetone phosphate (DHP), and Glyceraldehyde 3-phosphate (GDP or PGAL). Isomerase converts dihydroxyacetone phosphate immediately into glyceraldehyde 3-phosphate. After this step, everything takes place twice, once for each GDP derived from original glucose [1].

Each of glyeraldehyde 3-phosphate can then be oxidized by a molecule of NAD^+, in the presence of glyceraldehyde 3-phosphate dehydrogenase, to form 1,3-bisphosphoglycerate. Next, phosphoglycerate kinase generates a molecule of ATP while forming 3- phosphoglycerate. This is where for each 2 molecules of ATP used, 2 molecules of ATP have been synthesiszed. It is important to note that the phosphate needed to generate 1,3-bisphosphoglycerate, comes from inorganic phosphate (P_i) dissolved in the cell's cytoplasm, and not from ATP. As substrate level phosphorylation requires ADP, when ADP is missing, and there is plenty of ATP, this reaction does not occur, making this step an important regulatory point of the pathway.

[9] Phosphorylation: The addition of a phosphate group to a compound is called phosphorylation [P.172 Chapter7/Harvesting the Energy in Nutrients].

Phosphoglyceromutase then forms 2-phosphoglycerate. Phosphoenolpyruvate is then formed in the presence of enolase. Here another substrate-level phosphorylation produces a molecule of ATP, and pyruvate in the presence of pyruvate kinase. This serves as an additional regulatory step.[1]

After the formation of fructose 1,6 bisphosphate, many of the reactions are energetically unfavorable. The only reactions that are favorable are the 2 substrate-level phosphorylation steps that result in the formation of ATP. These two reactions pull the glycolytic pathway to completion. [1]

Table 9 - Inputs, Outputs, and Enzymes involved in Glycolysis

Steps	Inputs	Outputs	Enzyme
1	Glucose, ATP	Glucose 6-phosphate, ADP	Hexokinase
2	Glucose 6-phosphate	Fructose 6-phosphate	Phosphogluco-isomerase
3	Fructose 6-phosphate, ATP	Fructose 1,6-diphosphate, ADP	Phospho-fructokinase
4	Fructose 1,6-diphosphate	Dihydroxyacetone phosphate (DHP)	Aldolase
5	Fructose 1,6-diphosphate	Glyceraldehyde 3-phosphate (GDP) (PGAL)	Triose phosphate isomerase
6 (X2)	Glyceraldehyde 3-phosphate, Pi, NAD^+	1,3-diphosphoglycerate, NADH, H^+	Glyceraldehyde-3P-dehydrogenase
7 (X2)	1,3-biphosphoglycerate, ADP	3-phosphoglycerate, ATP	Phosphoglycerate kinase (PGK)
8 (X2)	3-phosphoglyceric acid	2-phosphoglyceric acid	Phophoglycerate mutase
9 (X2)	2-phosphoglyceric acid	Phosphoenolpyruvic acid, H_2O	Enolase
10 (X2)	Phosphoenolpyruvic acid, ADP	Pyruvic acid, ATP	pyruvate kinase

Under standard condition, reactions 1, 3, 7, 10 are exergonic. Under cellular conditions, reactions 1, 3, and 10 are exergonic. Exergonic reactions are the site of regulation of glycolysis.

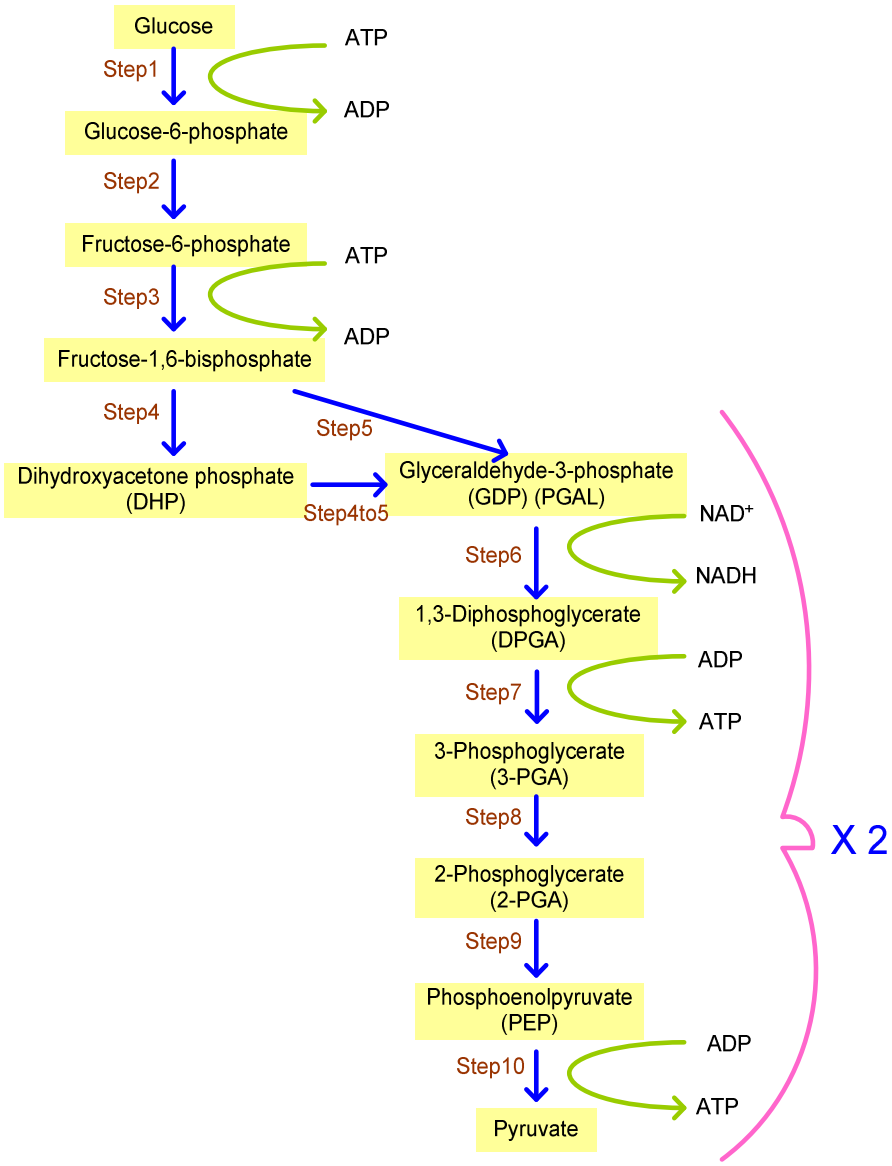

Figure 4.1 – Glycolysis Reactions

4.3 ATOMIC MODELS

The glycolysis pathway was completely implemented using CD++. The source code for the step one of glycolysis is included in section 4.3.1 along with pseucode, while the the source code for the steps two to ten are documented in appendix A.

4.3.1 STEP 1

Glucose is phosphorylated by ATP to form glucose 6-phosphate and ADP. This reaction is catalyzed by the enzyme hexokinase.

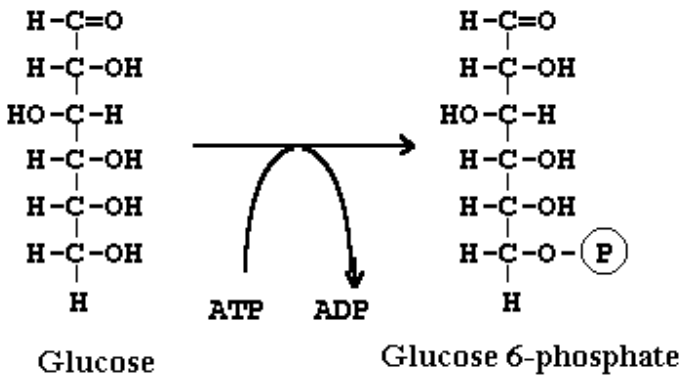

Figure 4.2 – Step1 of Glycolysis [Source: Dept Biochemistry & Molecular Biology, The University of Leeds, U.K., http://www.jonmaber.demon.co.uk/glysteps/step01b.htm]

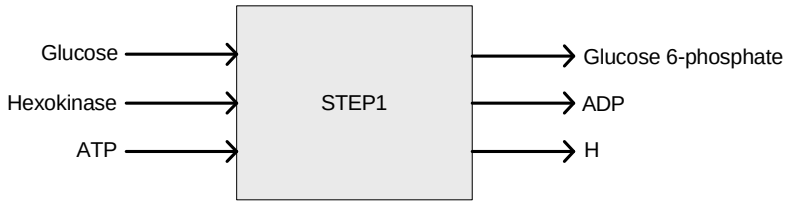

Figure 4.3 - Atomic Model for Step1 of Glycolysis

Glycolysis modeled using DEVS can be described as a composition of atomic and coupled components. The atomic model is defined by:

$$Step1 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$$

Where

S = {atpc, glucosec, ifhex, counter, phase, sigma}

X = {glucose, ATPi, hexokinase}

Y = {glucose_6_phosphate, ADP, H}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = Output function

Table 10 - External Function for Step1 of Glycolysis ($\boldsymbol{\delta_{ext}}$)

```
δext ( s, e, x ) {
        if (x=glucose)
        {
          glucosec=glucosec+x;

          if (atpc>0 and ifhex=true)
            holdin(active);
          else
            holdin(passive);
        }

        elseif (x=ATPi)
        {
```

```
            atpc=atpc+x;

            if (glucosec>0 and ifhex=true)
               holdin(active);
            else
               holdin(passive);
          }

          elseif (x=hexokinase)
          {
           ifhex=true;

           if (glucosec>0 and atpc>0)
               holdin(active);
           else
               holdin(passive);
          }
}
```

Table 11 - Internal Function for Step1 of Glycolysis (δ_{int})

```
δint ( s, e ) {
          counter=0;
          if (s=passive)
          {
             passivate;
          }
          else
          {
              if (atpc>1 and glucosec>1 and ifhex=1)
              {
                    if (atpc>glucosec)
                              {
                               atpc=atpc-glucosec;
                               counter=glucosec;
                               glucosec=0;
                              }
                    elseif (atpc<glucosec)
                              {
                               glucosec=glucosec-atpc;
                               counter=atpc;
                               atpc=0;
                              }
                    elseif (atpc==glucosec)
                              {
                               counter=atpc;
                               atpc=0;
                               glucosec=0;
                              }
                    holdin(passive);
              }
```

```
            else
            {
                passivate;
            }
        }
}
```

Table 12 - Output Function for Step1 of Glycolysis (λ)

```
λ ( s ) {
        if (counter is not zero)
        send outputs through the ports: glucose_6_phosphate,ADP,H;
}
```

Table 13 - Source Code for Step1 of Glycolysis: Step1.h

```
/*******************************************************************
*  DESCRIPTION:     Atomic Model Step1 of Glycolysis
*  AUTHOR:          Roxana Djafarzadeh
*  EMAIL:           mailto://rdjafar@site.uottawa.ca
*  DATE of Creation: 21/10/2003
*  Modified:        12/11/2003
*******************************************************************/

#ifndef __STEP1_H
#define __STEP1_H

#include <list>
#include "atomic.h"     // class Atomic

class Step1 : public Atomic
{
public:
        Step1( const string &name = "Step1" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &glucose;
        const Port &ATPi;
        const Port &hexokinase;
   // outputs
        Port &glucose_6_phosphate;
        Port &ADP;
        Port &H;

        Time preparationTime;
        double atpc;
        double glucosec;
        bool ifhex;
   double counter;
```

```
};        // class Step1

// ** inline ** //
inline
string Step1::className() const
{
          return "Step1" ;
}

#endif   //__STEP1_H
```

Table 14 - Source Code for Step1 of Glycolysis: Step1.cpp

```
/*****************************************************************
*  DESCRIPTION:     Atomic Model Step1 of Glycolysis
*  AUTHOR:          Roxana Djafarzadeh
*  EMAIL:           mailto://rdjafar@site.uottawa.ca
*  DATE of Creation: 21/10/2003
*  Modified:         24/11/2003
*  Modified:         08/12/2003
*****************************************************************/

/** include files **/
#include "step1.h"      // class step1
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*****************************************************************
* Function Name: Step1
* Description: Step1 of Glycolysis
*****************************************************************/
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 )
{
          string time( MainSimulator::Instance().getParameter( description(),
"preparation" ) ) ;

          if( time != "" )
                    preparationTime = time ;
}

/*****************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*****************************************************************/
Model &Step1::initFunction()
{
          atpc = 0;
          glucosec = 0;
          ifhex = false;
          counter = 0;

          return *this ;
}
```

```
/*****************************************************************
* Function Name: externalFunction
* Description:
*****************************************************************/
Model &Step1::externalFunction( const ExternalMessage &msg )
{

        if( msg.port() == glucose )
        {
                glucosec = glucosec + msg.value() ;

                if ( (atpc > 0 ) && (ifhex == true ) )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }


        else if( msg.port() == ATPi )
        {
                atpc = atpc + msg.value() ;

                if ( (glucosec > 0 ) && (ifhex == true ) )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }


        else if ( msg.port() == hexokinase )
        {
                ifhex = true ;

                if ( (glucosec > 0 ) && (atpc > 0) )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }

        return *this;
}

/*****************************************************************
* Function Name: internalFunction
* Description:
*****************************************************************/
Model &Step1::internalFunction( const InternalMessage & )
{
   counter = 0;
        if  ( state() == passive )
        {
                passivate();
        }
        else
        {

                if ( (atpc >= 1) && (glucosec >= 1 ) && (ifhex == 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 = 0;
                                glucosec = 0;
                        }

                        holdIn(passive, Time::Zero );
                }

                else
                {
                        passivate();
                }
        }

        return *this ;

}

/*****************************************************************
* Function Name: outputFunction
* Description:
*****************************************************************/
Model &Step1::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
                sendOutput( msg.time(), ADP, counter );
      sendOutput( msg.time(), glucose_6_phosphate, counter );
                sendOutput( msg.time(), H, counter );
        }

        return *this ;
}
```

4.3.2 STEP 2

Glucose 6-phosphate is converted to fructose 6-phosphate by phosphoglucoisomerase.

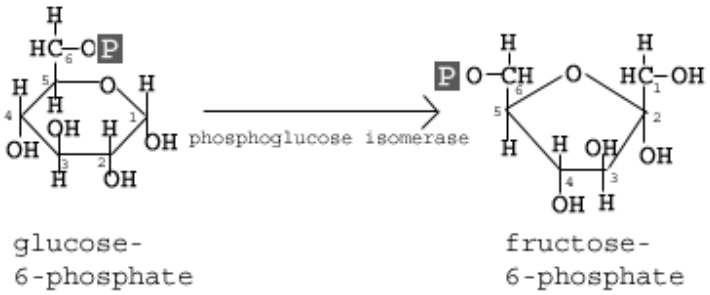

Figure 4.4 - Step2 of Glycolysis [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

Figure 4.5 - Atomic Model for Step2 of Glycolysis

$Step2 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {g6pc, ifpgisomerase, counter, phase, sigma}

X = {glucose_6_phosphate, phosphoglucoisomerase }

Y = {fructose_6_phosphate}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = Output function

*Table 15 - External Function for Step2 of Glycolysis (**δ_{ext}**)*

```
δext ( s, e, x ) {
        if( x = glucose_6_phosphate )
        {
                g6pc = g6pc + x ;
                if ( ifpgisomerase == true )
                        holdIn( active );
                else
                        holdIn(passive );
        }

        else if ( x = phosphoglucoisomerase )
        {
                ifpgisomerase = true ;
                if (g6pc > 0 )
                        holdIn( active );
                else
                        holdIn( passive );
        }
}
```

*Table 16 - Internal Function for Step2 of Glycolysis (**δ_{int}**)*

```
δint ( s, e ) {
        counter=0;
        if ( s = passive )
        {
                passivate;
        }
        else
        {
                if  ( g6pc>=1 and ifpgisomerase=true )
                {
                        counter = g6pc;
                        g6pc = 0;
                        holdIn( passive );
                }
                else
                {
                        passivate;
                }
        }
}
```

Table 17 - Output Function for Step2 of Glycolysis (λ)

```
λ ( s ) {
```

```
        if (counter is not zero)
        send outputs through the ports: fructose_6_phosphate;
}
```

4.3.3 STEP 3

Fructose 6-phosphate is phosphorylated by ATP to fructose 1,6-bisphosphate and ADP, using enzyme phosphofructokinase (PFK).

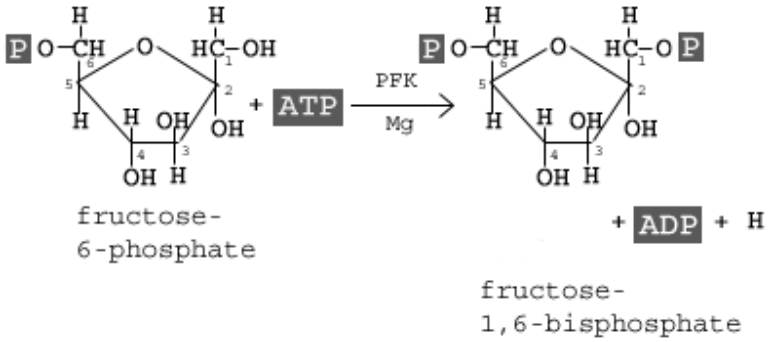

Figure 4.6 – Step3 of Glycolysis [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

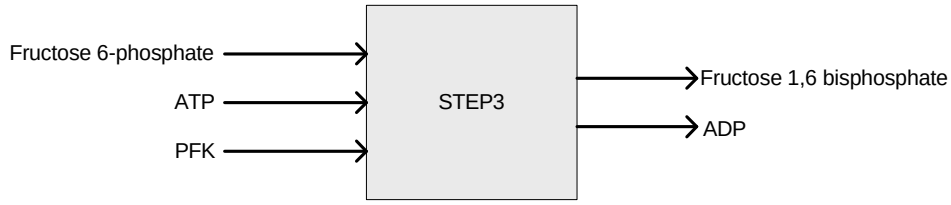

Figure 4.7 - Atomic Model for Step3 of Glycolysis

$Step3 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {f6pc, atpc, ifpfk, counter, phase, sigma}

X = {fructose_6_phosphate, ATP, PFK}

Y = {fructose_16_bisphosphate, ADP}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = Output function

Table 18 - External Function for Step3 of Glycolysis ($\boldsymbol{\delta_{ext}}$)

```
δext ( s, e, x ) {
        if ( x = fructose_6_phosphate )
        {
                f6pc = f6pc + x ;
                if ( atpc > 0 and ifpfk = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if( x = ATPi )
        {
                atpc = atpc + x ;
                if ( f6pc > 0 and ifpfk = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = PFK )
        {
                ifpfk = true ;
                if ( f6pc > 0 and atpc > 0 )
                        holdIn( active );
                else
                        holdIn( passive );
        }
}
```

Table 19 - Internal Function for Step3 of Glycolysis ($\boldsymbol{\delta_{int}}$)

```
δint ( s, e ) {
        counter=0;
        if ( s = passive )
        {
```

```
                passivate;
        }
        else
        {
                if ( atpc >= 1 and f6pc >= 1 and ifpfk = true )
                {
                        if (atpc > f6pc)
                        {
                                atpc = atpc - f6pc;
                                counter = f6pc;
                                f6pc = 0;
                        }

                        else if (atpc < f6pc)
                        {
                                f6pc = f6pc - atpc;
                                counter = atpc;
                                atpc = 0;
                        }

                        else if (atpc = f6pc)
                        {
                                counter = atpc;
                                atpc = 0;
                                f6pc = 0;
                        }

                        holdIn( passive );
                }
                else
                {
                        passivate;
                }
        }
}
```

Table 20 - Output Function for Step3 of Glycolysis (λ)

```
λ ( s ) {
        if (counter is not zero)
        send outputs through the ports: fructose_16_bisphosphate, ADP;
}
```

4.3.4 STEP 4

Aldolase splits fructose 1,6-bisphosphate into dihydroxyacetone phosphate (DHP) and glyceraldehydes 3-phosphate (GDP). This has been shown in two steps (steps 4 and 5).

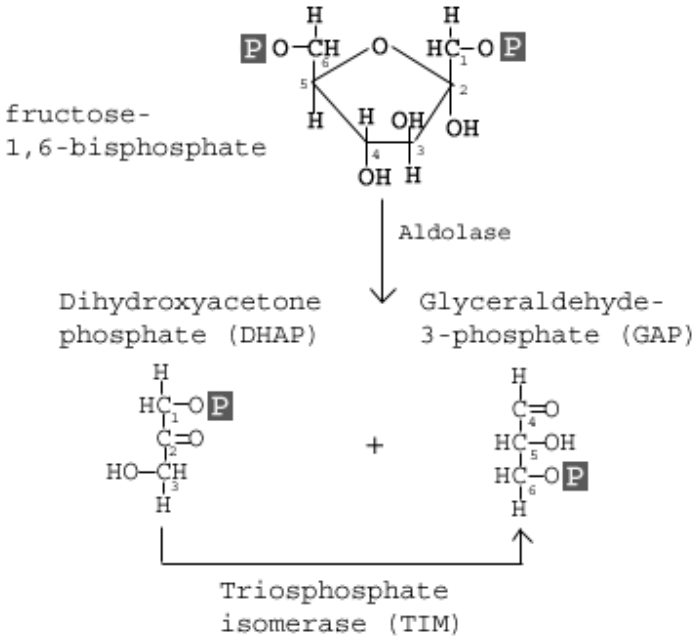

Figure 4.8 – Step 4 and 5 of Glycolysis [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

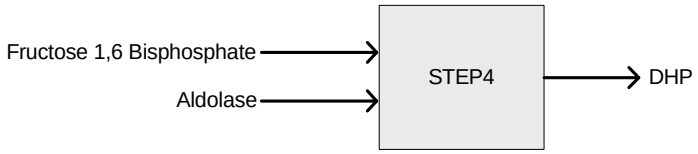

Figure 4.9 - Atomic Model for Step4 of Glycolysis

$Step4 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {f16pc, ifaldolase, counter, phase, sigma}

X = {fructose_16_bisphosphate, aldolase}

Y = {DHP}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = Output function

Table 21 - External Function for Step4 of Glycolysis ($\boldsymbol{\delta_{ext}}$)

```
δext ( s, e, x ) {
        if( x = fructose_16_bisphosphate )
        {
                f16pc = f16pc + x ;
                if ( ifaldolase == true )
                        holdIn(active);
                else
                        holdIn(passive);
        }

        else if ( x = aldolase )
        {
                ifaldolase = true ;
                if ( f16pc > 0 )
                        holdIn(active);
                else
                        holdIn(passive);
        }
}
```

Table 22 - Internal Function for Step4 of Glycolysis ($\boldsymbol{\delta_{int}}$)

```
δint ( s, e ) {
        counter=0;
        if ( s = passive )
        {
                passivate;
        }
        else
        {
                if ( f16pc >= 1 and ifaldolase=true )
                {
                        counter = f16pc;
                        f16pc = 0;
                        holdIn( passive );
                }
                else
                {
                        passivate;
                }
        }
```

```
}
```

Table 23 - Output Function for Step4 of Glycolysis (λ)

```
λ ( s ) {
        if (counter is not zero)
        send output through the port: DHP;
}
```

4.3.5 STEP 5

Glyceraldehydes 3-phosphate (GDP) is the other product of fructose 1,6-bisphosphate.

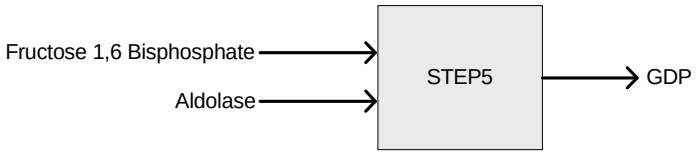

Figure 4.10 - Atomic Model for Step5 of Glycolysis

Step5 = < *S, X, Y,* δ_{int}, δ_{ext}, *ta,* λ >

Where

S = {f16pc, ifaldolase, counter, phase, sigma}

X = {fructose_6_phosphate, aldolase}

Y = {GDP}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = Output function

*Table 24 - External Function for Step5 of Glycolysis (**δ_{ext}**)*

```
δext ( s, e, x ) {
        if ( x = fructose_16_bisphosphate )
```

```
        {
                f16pc = f16pc + x ;
                if ( ifaldolase == true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = aldolase )
        {
                ifaldolase = true ;
                if (f16pc > 0 )
                        holdIn( active );
                else
                        holdIn( passive );
        }
}
```

*Table 25 - Internal Function for Step5 of Glycolysis (**δ_{int}**)*

```
δint ( s, e ) {
        counter=0;
        if ( s = passive )
        {
                passivate;
        }
        else
        {
                if ( f16pc>=1 and ifaldolase=true )
                {
                        counter = f16pc;
                        f16pc = 0;
                        holdIn( passive );
                }

                else
                {
                        passivate;
                }
        }
}
```

Table 26 - Output Function for Step5 of Glycolysis (λ)

```
λ ( s ) {
        if (counter is not zero)
        send output through the port: GDP;
}
```

4.3.6 STEP 4 TO 5

Dihydroxyacetone phosphate (DHP) is converted to glyceraldehydes 3-phosphate (GDP) by triose phosphate isomerase (isomerase).

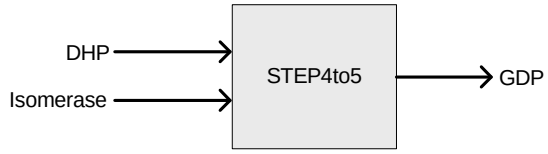

Figure 4.11 - Atomic Model for Step4to5 of Glycolysis

Step3 = < *S, X, Y,* δ_{int}, δ_{ext}, *ta,* λ >

Where

S = {dhpc, ifisomerase, counter, phase, sigma}

X = {DHP, isomerase}

Y = {GDP}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = Output function

*Table 27 - External Function for Step4to5 of Glycolysis (**δ_{ext}**)*

```
δext ( s, e, x ) {
        if( x = DHP )
        {
                dhpc = dhpc + x ;
                if ( ifisomerase = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = isomerase )
        {
```

```
                ifisomerase = true ;
                if ( dhpc > 0 )
                        holdIn(active);
                else
                        holdIn(passive);
        }
}
```

Table 28 - Internal Function for Step4to5 of Glycolysis (δ_{int})

```
δint ( s, e ) {
        counter=0;
        if ( s = passive )
        {
                passivate;
        }
        else
        {
                if ( dhpc>=1 and ifisomerase=true )
                {
                        counter = dhpc;
                        dhpc = 0;
                        holdIn( passive );
                }
                else
                {
                        passivate;
                }
        }
}
```

Table 29 - Output Function for Step4to5 of Glycolysis (λ)

```
λ ( s ) {
        if (counter is not zero)
        send the output through the output port: GDP;
}
```

4.3.7 STEP 6

Glyceraldehydes 3-phosphate (GDP) is converted to 1,3-diphosphoglycerate. This reaction is catalyzed by glyceraldehydes 3-phosphate dehydrogenase (G3PD), and uses P_i, and NAD^+.

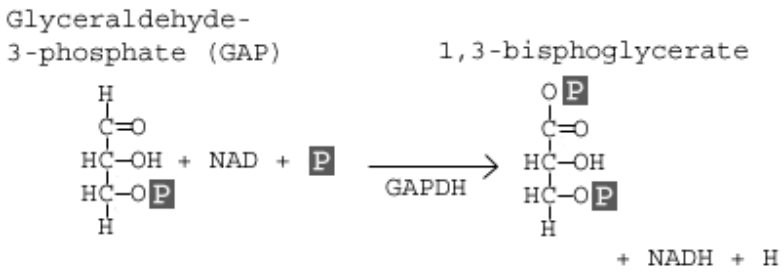

Figure 4.12 – Step6 of Glycolysis [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

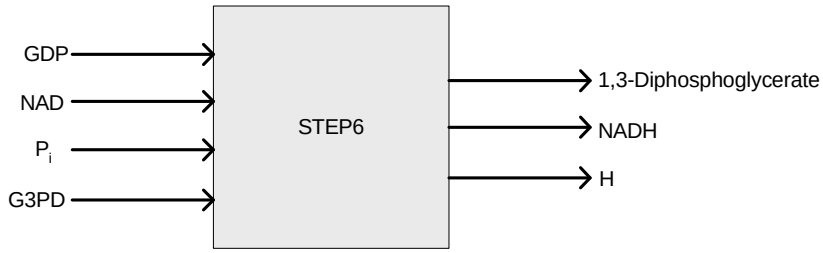

Figure 4.13 - Atomic Model for Step6 of Glycolysis

$Step6 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {gdpc, nadc, pic, ifg3pd, counter, phase, sigma}

X = {GDP, NAD, Pi, G3PD}

Y = {_13_BPG, NADH, H}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = Output function

Table 30 - External Function for Step6 of Glycolysis (δ_{ext})

```
δext ( s, e, x ) {
        if( x = GDP )
        {
                gdpc = gdpc + x ;
                if ( nadc>0 and pc>0 and ifg3pd=true )
                        holdIn( active );
                else
                        holdIn(passive);

        }

        else if( x = NAD )
        {
                nadc = nadc + x ;
                if ( gdpc>0 and pc>0 and ifg3pd=true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if( x = P )
        {
                pc = pc + x ;
                if ( (gdpc > 0 and nadc > 0 and ifg3pd = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = G3PD )
        {
                ifg3pd = true ;
                if ( gdpc > 0 and nadc > 0 and pc > 0 )
                        holdIn( active );
                else
                        holdIn( passive );
        }
}
```

Table 31 - Internal Function for Step6 of Glycolysis (δ_{int})

```
δint ( s, e ) {
        counter=0;
        if ( s = passive )
        {
                passivate;
        }
        else
        {
                if (gdpc>=1 and nadc>=1 and pc>=1 and ifg3pd= true)
                {
```

```
if ( gdpc>=nadc and nadc>=pc )
{
        gdpc = gdpc - pc;
        nadc = nadc - pc;
        counter = pc;
        pc = 0;
}

else if ( gdpc>=pc and pc>=nadc )
{
        gdpc = gdpc - nadc;
        pc = pc - nadc;
        counter = nadc;
        nadc = 0;
}

else if ( nadc>=gdpc and gdpc>=pc )
{
        gdpc = gdpc - pc;
        nadc = nadc - pc;
        counter = pc;
        pc = 0;
}

else if ( nadc>=pc and pc>=gdpc )
{
        nadc = nadc - gdpc;
        pc = pc - gdpc;
        counter = gdpc;
        gdpc = 0;
}

else if ( pc>=gdpc and gdpc>=nadc )
{
        gdpc = gdpc - nadc;
        pc = pc - nadc;
        counter = nadc;
        nadc = 0;
}

else if ( pc>=nadc and nadc>=gdpc )
{
        nadc = nadc - gdpc;
        pc = pc - gdpc;
        counter = gdpc;
        gdpc = 0;
}

else if ( gdpc=nadc and gdpc=pc )
{
        counter = gdpc;
        gdpc = 0;
```

```
                                        nadc = 0;
                                        pc = 0;
                                }
                                holdIn( passive );
                        }
                        else
                        {
                                passivate;
                        }
                }
}
```

Table 32 - Output Function for Step6 of Glycolysis (λ)

```
λ ( s ) {
        if (counter is not zero)
        send outputs through the output ports: _13_BPG, NADH, H;
}
```

4.3.8 STEP 7

Phosphoryl group from 1,3-diphosphoglycerate is transferred to ADP, generating ATP and 3-phosphoglycerate. This reaction is catalyzed by phosphoglycerate kinase (PGK).

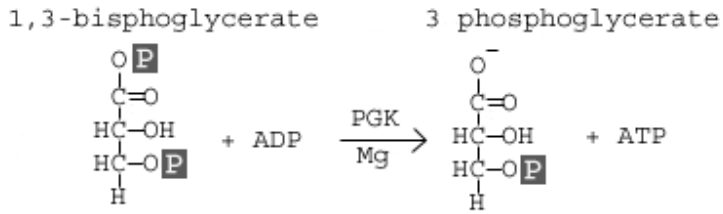

Figure 4.14 – Step7 of Glycolysis [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

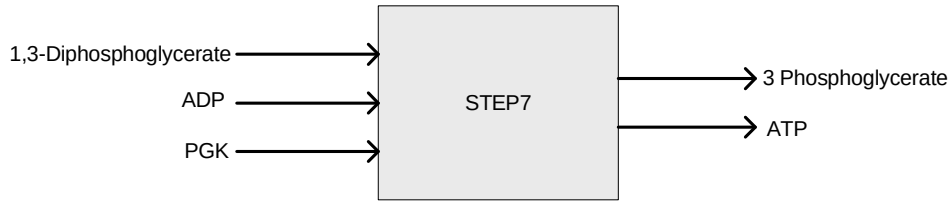

Figure 4.15 - Atomic Model for Step7 of Glycolysis

Step7 = < *S, X, Y,* δ_{int}, δ_{ext}, *ta,* λ >

Where

S = {_13bpgc, adpc, ifpgk, counter, phase, sigma}

X = {_13_BPG, ADP, PGK}

Y = {_3_phosphoglycerate, ATPo}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = Output function

*Table 33 - External Function for Step7 of Glycolysis (**δ_{ext}**)*

```
δext ( s, e, x ) {
        if( x = _13_BPG )
        {
                _13bpgc = _13bpgc + x ;
                if ( adpc > 0 and ifpgk = true  )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if( x = ADP )
        {
                adpc = adpc + x ;
                if ( _13bpgc > 0 and ifpgk = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = PGK )
```

```
        {
                ifpgk = true ;
                if ( _13bpgc > 0 and adpc > 0 )
                        holdIn( active );
                else
                        holdIn( passive );
        }
}
```

*Table 34 - Internal Function for Step7 of Glycolysis (**δ_{int}**)*

```
δint ( s, e ) {
        counter=0;
        if ( s = passive )
        {
                passivate;
        }
        else
        {
                if (_13bpgc>=1 and adpc>= 1 and ifpgk=true )
                {
                        if (_13bpgc > adpc)
                        {
                                _13bpgc = _13bpgc - adpc;
                                counter = adpc;
                                adpc = 0;
                        }

                        else if (_13bpgc < adpc)
                        {
                                adpc = adpc - _13bpgc;
                                counter = _13bpgc;
                                _13bpgc = 0;
                        }

                        else if (_13bpgc = adpc)
                        {
                                counter = adpc;
                                adpc = 0;
                                _13bpgc = 0;
                        }

                        holdIn( passive );
                }
                else
                {
                        passivate;
                }
        }
}
```

Table 35 - Output Function for Step7 of Glycolysis (λ)

```
λ ( s ) {
        if (counter is not zero)
        send outputs through the output ports: _3_phosphoglycerate, ATPo;
}
```

4.3.9 STEP 8

3-phosphoglycerate is converted to 2-phosphoglycerate by phosphoglycerate mutase (PGM).

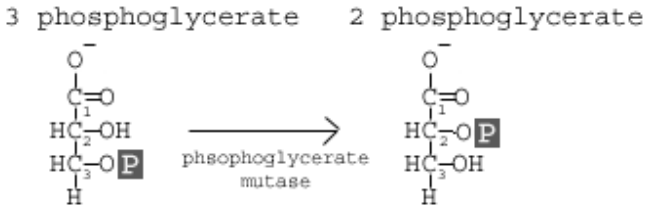

Figure 4.16 – Step8 of Glycolysis [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

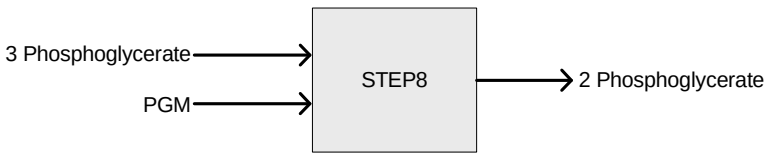

Figure 4.17 - Atomic Model for Step8 of Glycolysis

$Step8 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {_3pgc, ifpgm, counter, phase, sigma}

X = {_3_phosphoglycerate, PGM}

Y = {_2_phosphoglycerate}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = Output function

Table 36 - External Function for Step8 of Glycolysis ($\boldsymbol{\delta_{ext}}$)

```
δext ( s, e, x ) {
        if( x = _3_phosphoglycerate )
        {
                _3pgc = _3pgc + x ;
                if ( ifpgm == true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = PGM )
        {
                ifpgm = true ;
                if (_3pgc > 0 )
                        holdIn( active );
                else
                        holdIn( passive );
        }
}
```

Table 37 - Internal Function for Step8 of Glycolysis (δ_{int})

```
δint ( s, e ) {
        counter=0;
        if ( s = passive )
        {
                passivate;
        }
        else
        {
                if ( _3pgc >= 1 and ifpgm=true )
                {
                        counter = _3pgc;
                        _3pgc = 0;
                        holdIn( passive );
                }
                else
                {
                        passivate;
                }
        }
```

| }

Table 38 - Output Function for Step8 of Glycolysis (λ)

λ (s) {
if (counter is not zero)
send outputs through the output ports: _2_phosphoglycerate;
}

4.3.10 STEP 9

Enolase catalyzes the dehydration of 2-phosphoglycerate to form phosphoenolpyruvic (PEP). Water is released in this process.

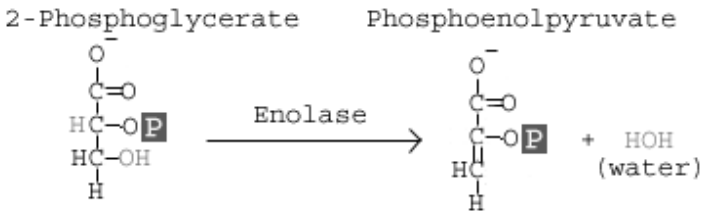

Figure 4.18 – Step9 of Glycolysis [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

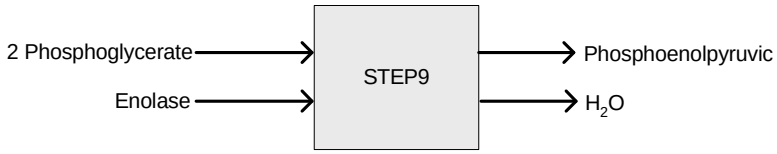

Figure 4.19 - Atomic Model for Step9 of Glycolysis

$Step9 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {_2pgc, ifenolase, counter, phase, sigma}

X = {_2_phosphoglycerate, enolase}

Y = {phosphoenolpyruvic, H2O}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = Output function

Table 39 - External Function for Step9 of Glycolysis (δ_{ext})

```
δext ( s, e, x ) {
        if( x = _2_phosphoglycerate )
        {
                _2pgc = _2pgc + x ;
                if ( ifenolase = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = enolase )
        {
                ifenolase = true ;
                if ( _2pgc > 0 )
                        holdIn( active );
                else
                        holdIn( passive );
        }
}
```

Table 40 - Internal Function for Step9 of Glycolysis (δ_{int})

```
δint ( s, e ) {
        counter=0;
        if ( s = passive )
        {
                passivate;
        }
        else
        {
                if ( _2pgc >= 1 and ifenolase = true  )
                {
                        counter = _2pgc;
                        _2pgc = 0;
                        holdIn( passive );
                }
```

```
                    else
                    {
                            passivate;
                    }
            }
}
```

Table 41 - Output Function for Step9 of Glycolysis (λ)

```
λ ( s ) {
        if (counter is not zero)
        send outputs through the output ports: phosphoenolpyruvic, H2O;
}
```

4.3.11 STEP 10

In this step, pyruvate kinase catalyzes the transfer of the phosphoryl group from phosphoenolpyruvic (PEP) to ADP to form ATP and pyruvate.

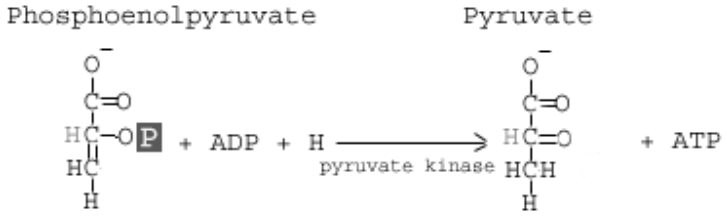

Figure 4.20 - Step10 of Glycolysis [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

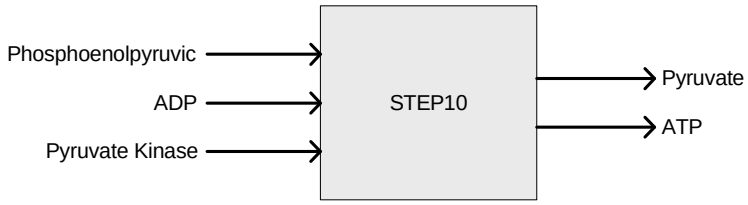

Figure 4.21 - Atomic Model for Step10 of Glycolysis

Step10 = < *S, X, Y,* δ_{int}, δ_{ext}, *ta,* λ >

Where

S = {pepc, adpc, ifpk, counter, phase, sigma}

X = {phosphoenolpyruvic, ADP, pyruvate_kinase}

Y = {pyruvate, ATP}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = Output function

*Table 42 - External Function for Step10 of Glycolysis (**δ_{ext}**)*

```
δext ( s, e, x ) {
        if( x = phosphoenolpyruvic )
        {
                pepc = pepc + x ;
                if ( adpc > 0 and ifpk = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if( x = ADP )
        {
                adpc = adpc + x ;
                if ( pepc > 0 and ifpk = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x= pyruvate_kinase )
```

```
        {
                ifpk = true ;
                if ( pepc > 0 and adpc > 0 )
                        holdIn( active );
                else
                        holdIn( passive );
        }
}
```

Table 43 - Internal Function for Step10 of Glycolysis (δ_{int})

```
δint ( s, e ) {
        counter=0;
        if ( s = passive )
        {
                passivate;
        }
        else
        {
                if ( pepc >= 1 and adpc >= 1 and ifpk = true )
                {
                        if (pepc > adpc)
                        {
                                pepc = pepc - adpc;
                                counter = adpc;
                                adpc = 0;
                        }

                        else if (pepc < adpc)
                        {
                                adpc = adpc - pepc;
                                counter = pepc;
                                pepc = 0;
                        }

                        else if (pepc = adpc)
                        {
                                counter = pepc;
                                pepc = 0;
                                adpc = 0;
                        }

                        holdIn( passive );
                }
                else
                {
                        passivate;
                }
        }
}
```

Table 44 - Output Function for Step10 of Glycolysis (λ)

```
λ ( s ) {
        if (counter is not zero)
        send outputs through the output ports: pyruvate, ATPo;
}
```

4.4 COUPLED MODEL

All the steps explained in the previous section were implemented in CD++, as showed in section 4.3.1 and the appendix. To do this, first the behaviour of each component was carefully specified with an analysis of inputs and outputs for each step as showed in sections 4.3.1 to 4.3.10. Each step was defined as a DEVS model following the specification. Afterwards, each model was implemented in CD++, and tested separately. Once every model was thoroughly tested, the main model was built as a coupled model connecting all the submodels previously defined. This model follows the design presented in Figure 4.23, and its detailed definition can be found in table 45.

Table 45 - Glycolysis.MA

```
[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 : phosphoglucoisomerase phosphoglucoisomerase@step2
Link : ATPi ATPi@step3
Link : PFK PFK@step3
Link : aldolase aldolase@step4
Link : isomerase isomerase@step4to5
Link : aldolase aldolase@step5
Link : NAD NAD@step6
Link : P P@step6
Link : G3PD G3PD@step6
```

```
Link : ADP ADP@step7
Link : PGK PGK@step7
Link : PGM PGM@step8
Link : enolase enolase@step9
Link : pyruvate_kinase pyruvate_kinase@step10
Link : glucose_6_phosphate@step1 glucose_6_phosphate@step2
Link : fructose_6_phosphate@step2 fructose_6_phosphate@step3
Link : fructose_16_bisphosphate@step3 fructose_16_bisphosphate@step4
Link : fructose_16_bisphosphate@step3 fructose_16_bisphosphate@step5
Link : DHP@step4 DHP@step4to5
Link : GDP@step4to5 GDP@step6
Link : GDP@step5 GDP@step6
Link : _13_BPG@step6 _13_BPG@step7
Link : _3_phosphoglycerate@step7 _3_phosphoglycerate@step8
Link : _2_phosphoglycerate@step8 _2_phosphoglycerate@step9
Link : phosphoenolpyruvic@step9 phosphoenolpyruvic@step10
Link : ADP@step1 ADP@step10
Link : H@step1 H
Link : ADP@step3 ADP@step7
Link : NADH@step6 NADH
Link : H@step6 H
Link : ATPo@step7 ATPo
Link : H2O@step9 H2O
Link : pyruvate@step10 pyruvate
Link : ATPo@step10 ATPo

[step1]
preparation : 00:00:00:000
[step2]
preparation : 00:00:05:000
[step3]
preparation : 00:00:15:000
[step4]
preparation : 00:00:05:000
[step4to5]
preparation : 00:00:05:000
[step5]
preparation : 00:00:05:000
[step6]
preparation : 00:00:05:000
[step7]
preparation : 00:00:05:000
[step8]
preparation : 00:00:05:000
[step9]
preparation : 00:00:05:000
[step10]
preparation : 00:00:05:000
```

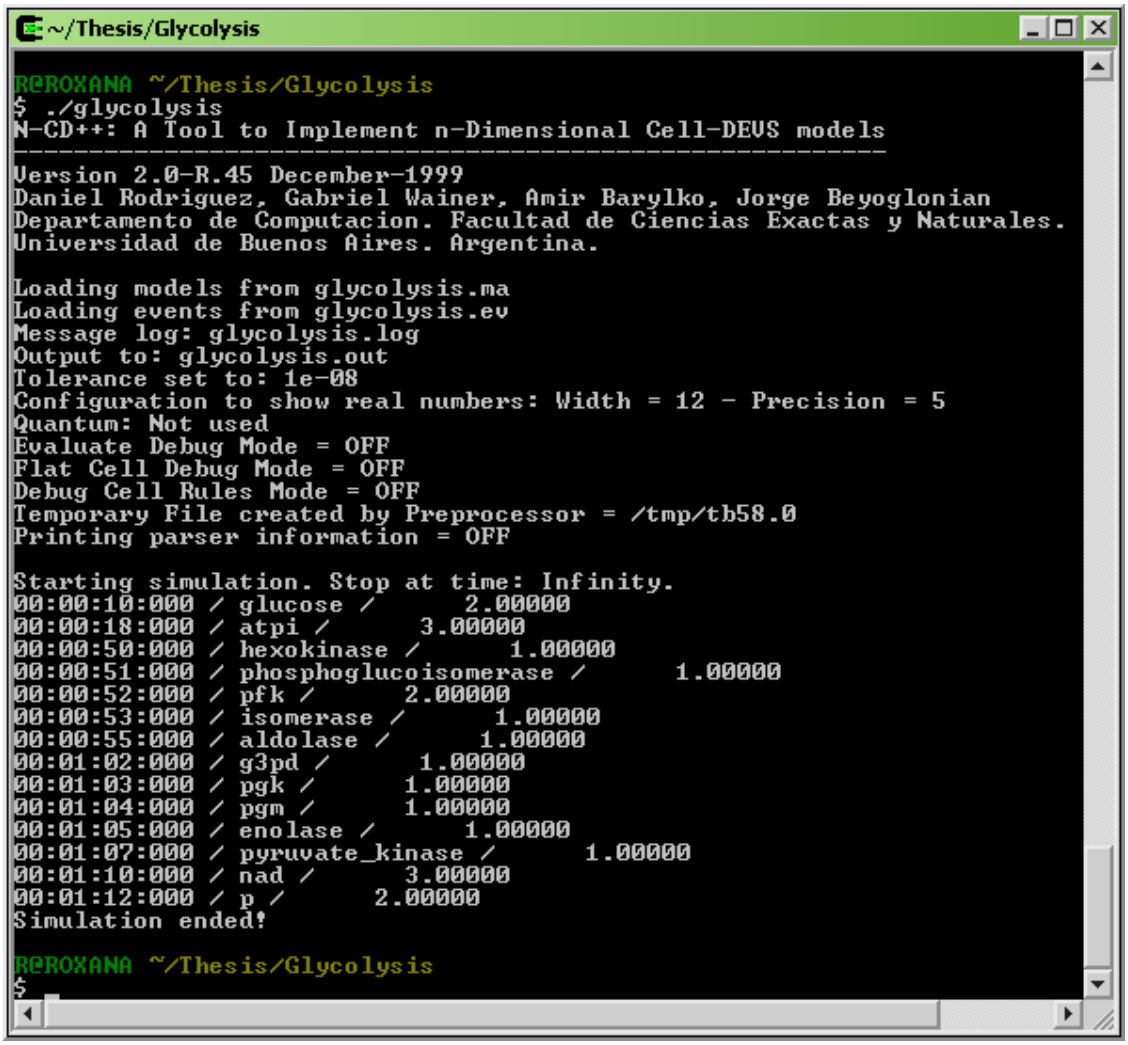

Figure 4.22 – Snapshot of Simulation Run for Glycolysis

Figure 4.22 shows a snapshot of cygwin[10] shell where glycolysis simulation is being run from the command line. Running the simulation for Glycolysis model will generate a log file that will look like the following table, where one can follow the details of

[10] Cygwin is an open source collection of tools that allows Unix or Linux applications to be compiled and run on a Windows operating system from within a Linux-like interface. The name *Cygwin* was created from a combination of *Cyg*nus and *Win*dows. [Source: searchEnterpriseLinux.com]

simulation where the inputs and outputs happen. Only a fragment of glycolysis.log has been shown in table 44 (refer to Appendix A for a full description of Glycolysis.log).

Table 46 - A Fragment of Glycolysis.log file

```
.
.
.
Mensaje * / 00:00:50:000 / top(01) para step2(03)
Mensaje D / 00:00:50:000 / step2(03) / ... para top(01)
Mensaje D / 00:00:50:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:50:000 / Root(00) para top(01)
Mensaje * / 00:00:50:000 / top(01) para step10(12)
Mensaje D / 00:00:50:000 / step10(12) / ... para top(01)
Mensaje D / 00:00:50:000 / top(01) / ... para Root(00)
Mensaje X / 00:00:51:000 / Root(00) / phosphoglucoisomerase /      1.00000 para top(01)
Mensaje X / 00:00:51:000 / top(01) / phosphoglucoisomerase /      1.00000 para step2(03)
Mensaje D / 00:00:51:000 / step2(03) / 00:00:00:000 para top(01)
Mensaje D / 00:00:51:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:51:000 / Root(00) para top(01)
Mensaje * / 00:00:51:000 / top(01) para step2(03)
Mensaje D / 00:00:51:000 / step2(03) / 00:00:00:000 para top(01)
Mensaje D / 00:00:51:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:51:000 / Root(00) para top(01)
Mensaje * / 00:00:51:000 / top(01) para step2(03)
Mensaje Y / 00:00:51:000 / step2(03) / fructose_6_phosphate /      2.00000 para top(01)
Mensaje D / 00:00:51:000 / step2(03) / ... para top(01)
Mensaje X / 00:00:51:000 / top(01) / fructose_6_phosphate /      2.00000 para step3(04)
Mensaje D / 00:00:51:000 / step3(04) / 00:00:00:000 para top(01)
Mensaje D / 00:00:51:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:51:000 / Root(00) para top(01)
.
.
.
```

The output file from Glycolysis model can be checked to verify the validity of the model. In this case, since glycolysis is a well proven biological model, one can easily compare the results of the simulation with the experimental results to verify the correct amount of outputs at the correct times.

Table 47 - Glycolysis.out: The Output File

```
00:00:50:000 h 2
00:01:12:000 nadh 2
00:01:12:000 h 2
00:01:12:000 atpo 2
```

```
00:01:12:000 h2o 2
00:01:12:000 atpo 2
00:01:12:000 pyruvate 2
```

The coupled model consists of the following atomic models: step1, step2, step3, step4, step4to5, step5, step6, step7, step8, step9, and step10. Since the steps 6 to 10 happen two times, a separate coupled model can be defined for atomics models: step6, step7, step8, step9, and step10 to further simplify the model. In this case our model will have two coupled models consisting of steps 6 to 10, and atomic models for steps1, 2, 3, 4, 4to5, and 5. In this paper, only the simulation for the first model has been shown. See figure 3.3 for a drawing of the glycolysis model.

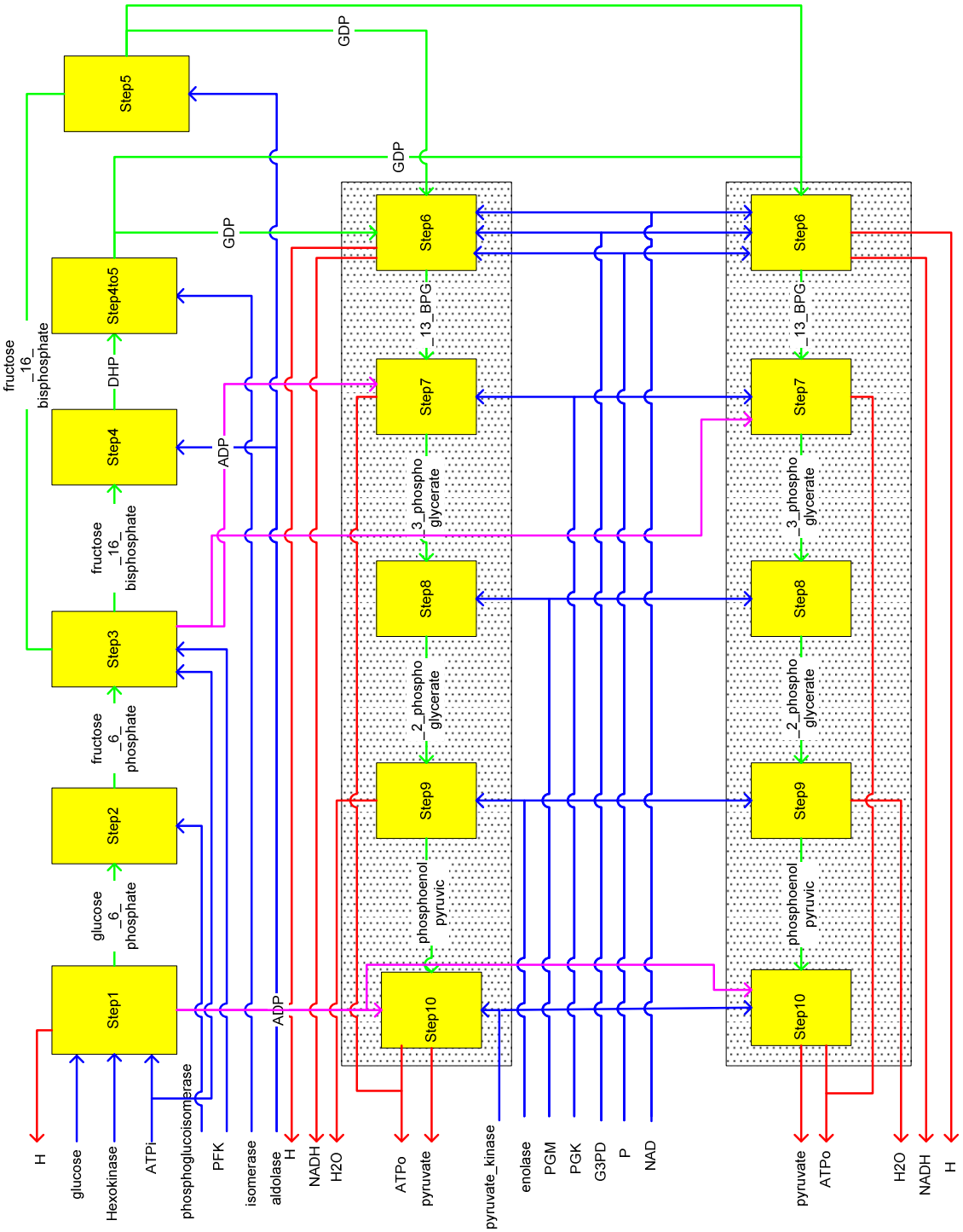

Figure 4.23 – Coupled Model of Glycolysis

The CD++ Modeler graphical user interface was used to represent the DEVS model for glycolysis that shows the execution results of the model (see Figure 4.23).

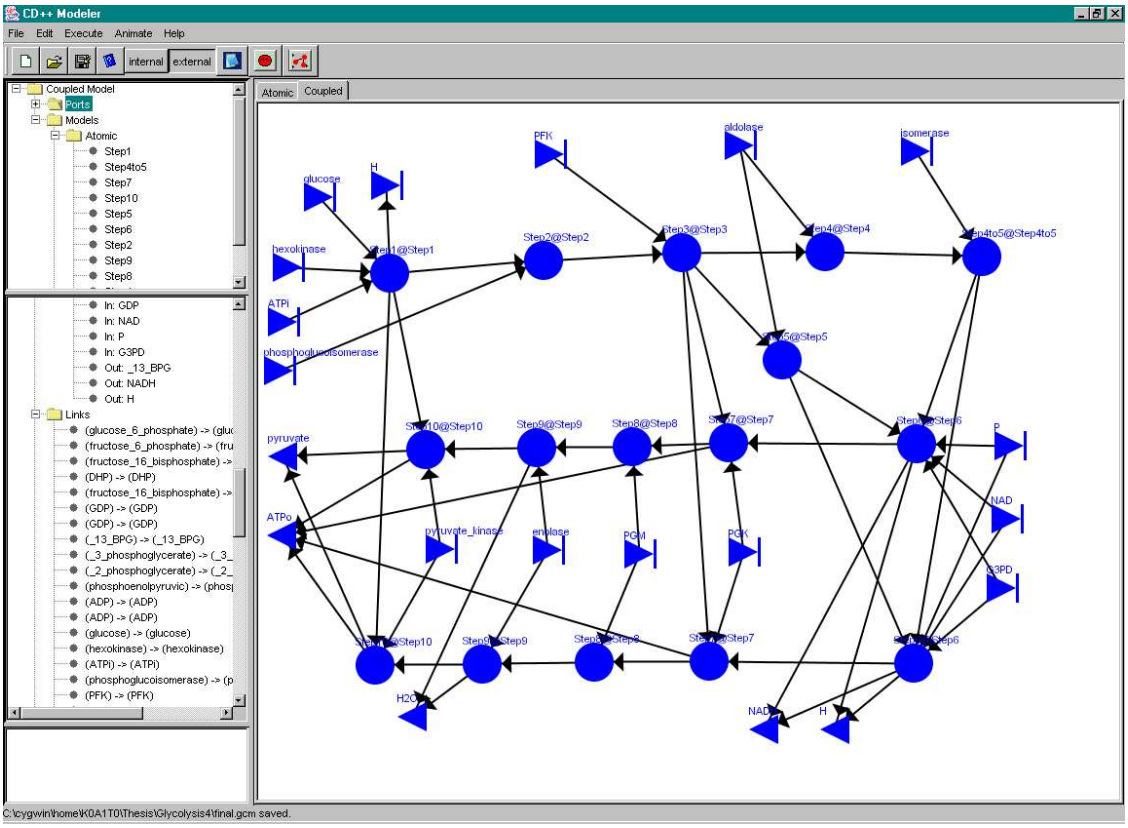

Figure 4.24 – Glycolysis.GCM

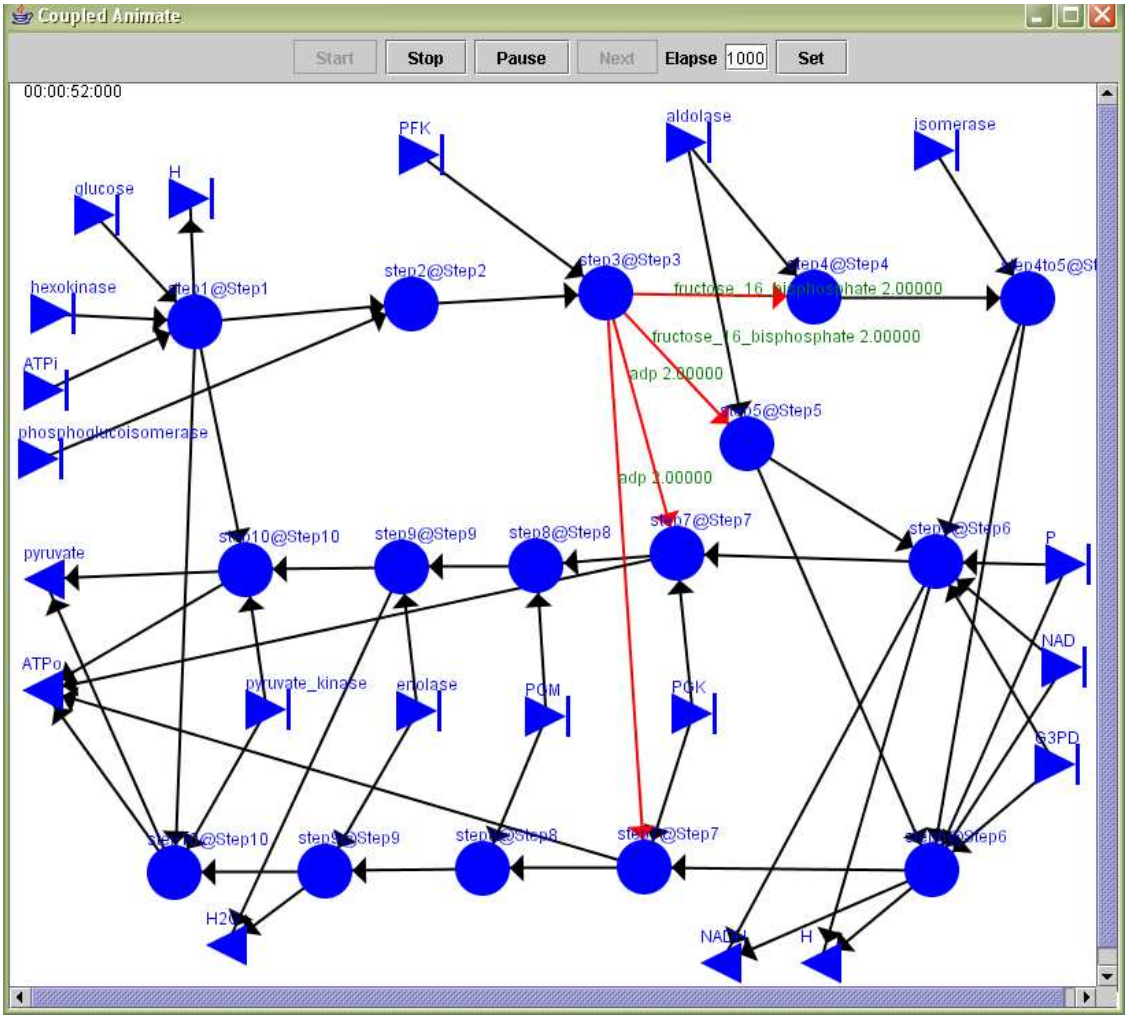

Figure 4.25 – Coupled Animation of Glycolysis

4.5 SUMMARY

The CD++ tool, based on the formalism allows the definition of complex biological models using a high-level specification language. In this way, the construction of simulations can be improved greatly.

The sum of all reactions in the glycolysis can be summarized as follow:

$$\text{Glucose} + 2ATP + 4ADP + 2P_i + 2NAD^+ \rightarrow \\ 2\text{Pyruvic Acid} + 2ADP + 4ATP + 2NADH + 2H^+ + 2H_2O \quad (4.1)$$

Therefore, one molecule of glucose, results in two molecules of ATP and two molecules of NADH as energy.

Table 48 - Products of Glycolysis

Products of Glycolysis
2 Pyruvate (Pyruvic Acid)
2 H_2O
2 ATP
2 NADH

As a well known pathway, this model can be used as a basis to prove the practicality of this approach, and eventually, more coupled models can be added to this model to build a more complex biological system.

CHAPTER 5-A SIMULATION MODEL OF KREBS' CYCLE

The Krebs cycle, also called the Tri-Carboxylic Acid (TCA) cycle and the Citric Acid Cycle (CAC), oxidises pyruvate formed during the glycolytic pathway into CO_2 and H_2O. This cycle is as series of chemical reactions of central importance in all living cells that utilize oxygen. The citric acid cycle takes place within the mitochondria in eukaryotes, and within the cytoplasm in prokaryotes. Acetyl-CoA produced during glycolysis, is the main input to Krebs cycle. Citrate is both the first and the last product of the cylce, which is regenerated by both oxaloacetate and acetyl-CoA.

For each trun of the cycle, 12 ATP molecules are produced, one directly from the cycle and 11 from the reoxidation of the three NADH and one $FADH_2$ molecules produced by the cycle by oxidative phosphorylation which will be described in the next chapter.

5.1 THE CYCLE

The Krebs cycle (see figure 5.1), as described in detail in chapter 2, has a preparatory stage plus a total of eight stages: [11]

1. The production of citrate from oxaloacetate and acetyl CoA catalyzed by citrate synthase

2. Isomerization of citrate to isocitrate catalyzed by acontinase

3. Oxidation of isocitrate to α-ketoglutarate catalyzed by isocitrate dehydrogenase (the reaction requires NAD^+)

4. Oxidation of α-ketoglutarate to succinyl CoA catalyzed by the α-ketoglutarate dehydrogenase complex (the reaction requires NAD^+)

5. Conversion of succinyl CoA to succinate catalyzed by succinyl CoA synthetase (the reaction requires P_i and GDP)

6. Oxidation of succinate to fumarate catalyzed by succinate dehydrogenase (the reaction requires FAD)

7. Hydration of fumarate to malate catalyzed by fumarase

8. Oxidation of malate to oxaloacetate catalazied by malate dehydrogenase (the reaction requires NAD^+)

Table 49 - Summary of Kreb's Cycle

Molecule	Enzyme	Reaction Type	Reactants/ Coenzymes	Products/ Coenzymes
1. Citrate	Aconitase	Dehydration		H_2O
2. CIS-Aconitate	Aconitase	Hydration	H_2O	
3. Isocitrate	Isocitrate Dehydrogenase	Oxidation	NAD^+	$NADH+H^+$
4. Oxalosuccinate	Isocitrate Dehydrogenase	Decarboxylation		
5. α-Ketoglutarate	α-Ketoglutarate Dehydrogenase	Oxidative Decarboxylation	NAD^+ CoA-SH	$NADH+H^+$ CO_2
6. Succinyl-CoA	Succinyl-CoA Synthetase	Hydrolysis	GDP P_i	GTP CoA-SH
7. Succinate	Succinate Dehydrogenase	Oxidation	FAD	$FADH_2$

8. Fumarate	Fumarase	Addition (H_2O)	H_2O	
9. L-Malate	Malate Dehydrogenase	Oxidation	NAD^+	$NADH+H^+$
10. Oxaloacetate	Citrate Synthase	Condensation		
11. Acetyl-CoA				

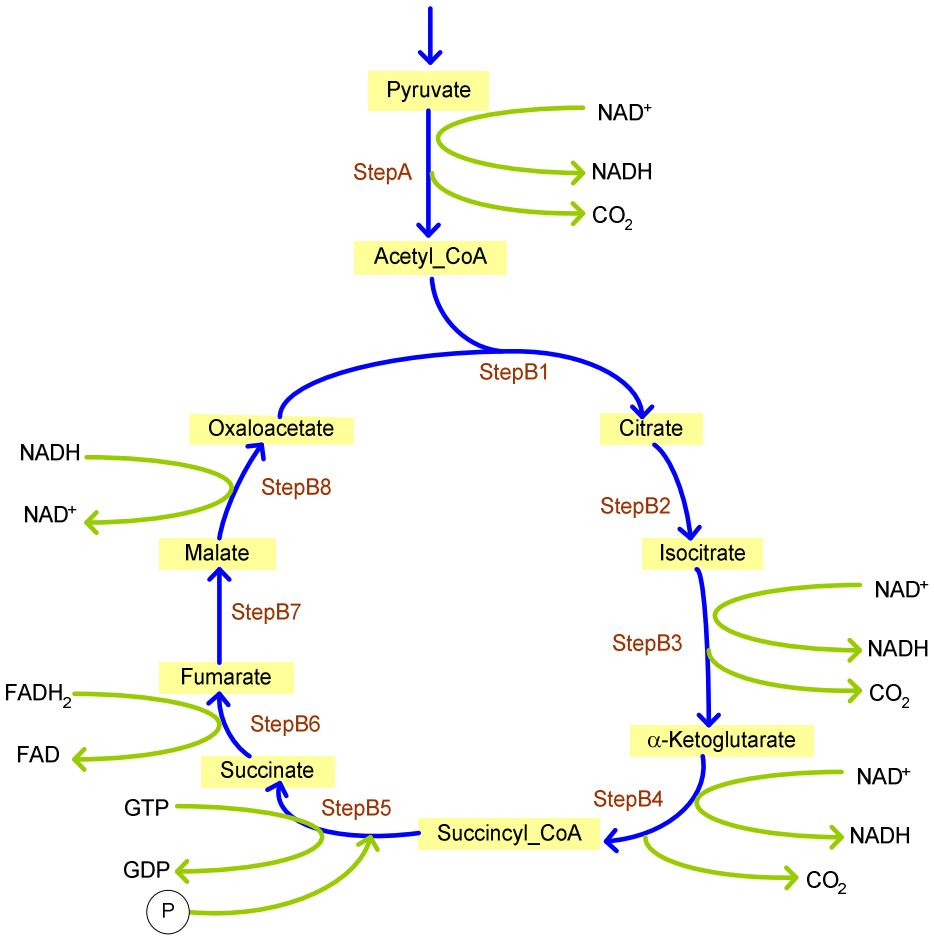

Figure 5.1 - Kreb's Cycle Reactions

5.2 ATOMIC MODELS

The Kreb's cycle was completely implemented using CD++. The source code for the step A of Krebs cycle is included in section 5.2.1 along with pseucode, while the source code for the steps B1 to B8 are documented in appendix B.

5.2.1 STEP A: THE BRIDGING STEP

A major source of energy is glucose which is converted by glycolysis into pyruvate. Pyruvate enters mitochondria, linking glycolysis to Kreb's cycle. Pyruvate dehydrogenase – a complex composed of three enzymes and five coenzymes – then oxidizes the pyruvate using NAD+ to forma acetyl CoA and CO_2. The formation of Acetyl-CoA from Pyruvic Acid is as follow:

$$2\text{Pyruvate} + 2\text{NAD}^+ + 2\text{CoA} \rightarrow \text{Acetyl-CoA} + 2\text{NADH} + 2\text{CO}_2 \quad (5.1)$$

Pyruvate is degraded and combined with coenzyme A to form acetyl coenzyme A. As we can see, NADH and CO_2 are released in the process. Since this reaction involves both and oxidation and a loss of CO_2, it is also called 'oxidative decarboxylation'.

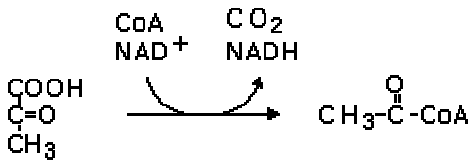

Figure 5.2 - StepA of Kreb's Cycle [Source: Dept Biochemistry & Molecular Biology, The University of Leeds, U.K., http://www.jonmaber.demon.co.uk/glysteps/step01b.htm]

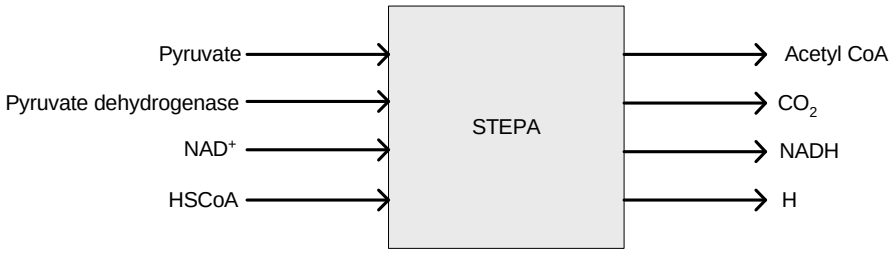

Figure 5.3 - Atomic Model for StepA of Kreb's Cycle

Krebs cycle modeled using DEVS can be described as a composition of atomic and coupled components. The atomic model is defined by:

$$StepA = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$$

Where

S = {pyruvatec, ifpyruvateDehydrogenase, hscoaic, nadc, counter, phase, sigma}

X = {pyruvate, pyruvateDehydrogenase, HSCoAi, NAD}

Y = {acetyl_CoA, NADH, H, CO2}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = output function

Table 50 - External Function for StepA of Kreb's Cycle (δ_{ext})

```
δext ( s, e, x ) {

      if ( x = pyruvate )
      {
          Pyruvatec = pyruvatec + x ;
          if (hscoaic>0 and nadc>0 and ifpyruvateDehydrogenase = true)
             holdin(active);
          else
             holdin(passive);
          }

          dlse if ( x = HSCoAi )
          {
              hscoaic = hscoaic + x;
              if (pyruvatec>0 and nadc>0  and ifpyruvateDehydrogenase = true)
                 holdIn(active);
              else
                 holdIn(passive);
          }

          else if ( x = NAD )
          {
              nadc = nadc + msg.value();
              if (pyruvatec>0 and hscoaic>0 and ifpyruvateDehydrogenase= true)
                 holdIn(active);
              else
                 holdIn(passive);
          }

          else if ( x = pyruvateDehydrogenase )
          {
              ifpyruvateDehydrogenase = true;
              if (pyruvatec> 0 and hscoaic>0 and nadc>0 )
                 holdIn(active);
              else
                 holdIn(passive);
          }
}
```

Table 51 - Internal Function for StepA of Kreb's Cycle (δ_{int})

```
δint ( s, e ) {
          counter=0;

          if (s=passive)
          {
             passivate;
          }

          else
```

```
{
    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=0;
                    nadc=0;
                    hscoaic=0;
                }

                holdin(passive);
            }
            else
            {
                passivate;
            }
        }
}
```

*Table 52 - Output Function for StepA of Kreb's Cycle (**λ**)*

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

Table 53 - Source Code for StepA of Krebs Cycle: StepA.h

```
/*****************************************************************
*  DESCRIPTION:      Atomic Model StepA of Krebs Cycle
*  AUTHOR:           Roxana Djafarzadeh
*  EMAIL:            mailto://rdjafar@site.uottawa.ca
*  DATE of Creation:  30/12/2003
*  Modified:         28/07/2004
*****************************************************************/

#ifndef __STEPA_H
#define __STEPA_H

#include <list>
#include "atomic.h"     // class Atomic

class StepA : public Atomic
{
public:
        StepA( const string &name = "StepA" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &pyruvate;
        const Port &pyruvateDehydrogenase;
        const Port &HSCoAi;
        const Port &NAD;
   // outputs
```

```
        Port &acetyl_CoA;
        Port &NADH;
        Port &CO2;
        Port &H;

        Time preparationTime;

        double pyruvatec;
        double ifpyruvateDehydrogenase;
        double hscoaic;
        double nadc;
   double counter;

};      // class StepA

// ** inline ** //
inline
string StepA::className() const
{
        return "StepA" ;
}

#endif  //__STEPA_H
```

Table 54 - Source Code for StepA of Krebs Cycle: StepA.cpp

```
/*******************************************************************
*  DESCRIPTION:     Atomic Model StepA of Krebs Cycle
*  AUTHOR:          Roxana Djafarzadeh
*  EMAIL:           mailto://rdjafar@site.uottawa.ca
*  DATE of Creation: 01/01/2004
*  Modified:        17/01/2004
*  Modified:        05/04/2004
*******************************************************************/

/** include files **/
#include "stepA.h"      // class stepA
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: StepA
* Description:
*******************************************************************/
StepA::StepA( const string &name )
: Atomic( name )
, pyruvate( addInputPort( "pyruvate" ) )
, pyruvateDehydrogenase( addInputPort( "pyruvateDehydrogenase" ) )
, HSCoAi( addInputPort( "HSCoAi" ) )
, NAD( addInputPort( "NAD" ) )
, acetyl_CoA( addOutputPort( "acetyl_CoA" ) )
, NADH( addOutputPort( "NADH" ) )
, CO2( addOutputPort( "CO2" ) )
, H( addOutputPort( "H" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(),
"preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}
```

```
/*********************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*********************************************************************/
Model &StepA::initFunction()
{
        pyruvatec = 0;
        ifpyruvateDehydrogenase = false;
        hscoaic = 0;
        nadc = 0;
        counter = 0;

        return *this ;
}

/*********************************************************************
* Function Name: externalFunction
* Description:
*********************************************************************/
Model &StepA::externalFunction( const ExternalMessage &msg )
{

        if( msg.port() == pyruvate )
        {
                pyruvatec = pyruvatec + msg.value();
                if ( (hscoaic > 0) && (nadc > 0) && (ifpyruvateDehydrogenase == true)
)
                        holdIn(active, Time::Zero);
           else
                        holdIn(passive, Time::Zero);
        }

   else if( msg.port() == HSCoAi )
        {
                hscoaic = hscoaic + msg.value();
                if ( (pyruvatec > 0) && (nadc > 0) && (ifpyruvateDehydrogenase ==
true) )
                        holdIn(active, Time::Zero);
           else
                        holdIn(passive, Time::Zero);
        }


   else if( msg.port() == NAD )
        {
                nadc = nadc + msg.value();
                if ( (pyruvatec > 0) && (hscoaic > 0) && (ifpyruvateDehydrogenase ==
true) )
                        holdIn(active, Time::Zero);
           else
                        holdIn(passive, Time::Zero);
        }


   else if( msg.port() == pyruvateDehydrogenase )
        {
           ifpyruvateDehydrogenase = true;

                if ( (pyruvatec > 0) && (hscoaic > 0) && (nadc > 0) )
                        holdIn(active, Time::Zero);
           else
                        holdIn(passive, Time::Zero);
        }
```

```
        return *this;

}

/*****************************************************************
* Function Name: internalFunction
* Description:
*****************************************************************/
Model &StepA::internalFunction( const InternalMessage & )
{

   counter = 0;

        if  ( state() == passive )

        {
                passivate();
        }

        else

        {
                if ( (pyruvatec >= 1) && (hscoaic >= 1) && (nadc >= 1) &&
(ifpyruvateDehydrogenase == true) )
                {

                        if ( (pyruvatec>=hscoaic) && (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=0;
                                nadc=0;
                                hscoaic=0;
                        }

                        holdIn(passive, Time::Zero);
                }

                else
                {
                        passivate();
                }

        }

        return *this ;

}

/*****************************************************************
* Function Name: outputFunction
* Description:
*****************************************************************/
Model &StepA::outputFunction( const InternalMessage &msg )
{

   if ( counter != 0 )
        {
                sendOutput( msg.time(), acetyl_CoA, counter );
                sendOutput( msg.time(), NADH, counter );
                sendOutput( msg.time(), CO2, counter );
                sendOutput( msg.time(), H, counter );
        }


        return *this ;

}
```

5.2.2 STEP B1

The unstable bond of acetyl CoA breaks and the two-carbon acetyl group bonds to the four-carbon oxaloacetic acid to form six-carbon citric acid (citrate).

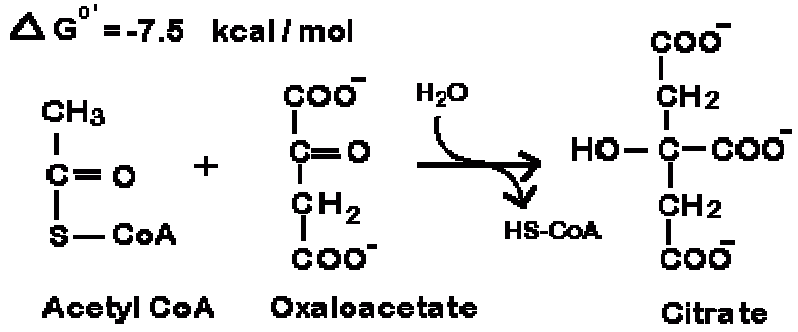

Figure 5.4 - StepB1 of Kreb's Cycle [Source: Dept Biochemistry & Molecular Biology, The University of Leeds, U.K., http://www.jonmaber.demon.co.uk/glysteps/step01b.htm]

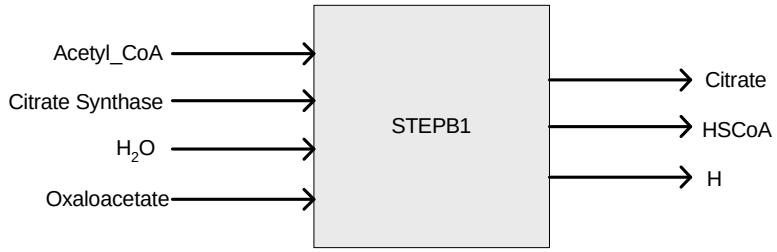

Figure 5.5 - Atomic Model for StepB1 of Kreb's Cycle

$StepB1 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {acetyl_CoAc, oxaloacetatec, ifcitrateSynthase, h2oc, counter, phase, sigma}

X = {acetyl_CoA, oxaloacetate, H2O, citrateSynthase}

Y = {citrate, HSCoAo, H}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = output function

Table 55 - External Function for StepB1 of Krebs Cycle (δ_{ext})

```
δext ( s, e, x ) {

      if (x=acetyl_CoA)
     {
          acetyl_CoAc=acetyl_CoAc+x;
          if (oxaloacetatec>0 and h2oc>0 and ifcitrateSynthase=true)
                holdin(active);
          else
                holdin(passive);
      }

      elseif (x=oxaloacetate)
      {
          Oxaloacetatec=oxaloacetatec+x;
          if (acetyl_CoAc>0 and h2oc>0 and ifcitrateSynthase = true)
                holdin(active);
          else
                holdin(passive);
     }

     elseif (x = H2O )
    {
          h2oc = h2oc + x;
          if (acetyl_CoAc>0 and oxaloacetatec>0 and ifcitrateSynthase = true)
                holdIn(active);
          else
                holdIn(passive);
          }

      elseif ( x = citrateSynthase )
          {
          ifcitrateSynthase = true ;
          if ( acetyl_CoAc>0 and oxaloacetatec>0 and h2oc>0  )
                holdIn(active);
          else
                holdIn(passive);
          }

}
```

Table 56 - Internal Function for StepB1 of Krebs Cycle (δ_{int})

```
δint ( s, e ) {
      counter=0;
      if (s=passive)
      {
          passivate;
       }
       else
       {
           if (acetyl_CoAc>=1 and oxaloacetatec>=1 and h2oc>=1 and ifcitrateSynthase =true)
           {
               if (acetyl_CoAc>=oxaloacetatec and oxaloacetatec>= h2oc)
               {
                        acetyl_CoAc = acetyl_CoAc - h2oc;
                        oxaloacetatec = oxaloacetatec - h2oc;
                        counter = h2oc;
                        h2oc = 0;
            }

            else if ( (acetyl_CoAc >= h2oc) && (h2oc >= oxaloacetatec) )
            {
                        acetyl_CoAc = acetyl_CoAc - oxaloacetatec;
                        h2oc = h2oc - oxaloacetatec;
                        counter = oxaloacetatec;
                        oxaloacetatec = 0;
            }

            else if ( (oxaloacetatec >= acetyl_CoAc) && (acetyl_CoAc >= h2oc) )
            {
                        oxaloacetatec = oxaloacetatec - h2oc;
                        acetyl_CoAc = acetyl_CoAc - h2oc;
                        counter = h2oc;
                        h2oc = 0;
            }

            else if ( (oxaloacetatec >= h2oc) && (h2oc >= acetyl_CoAc) )
            {
                        oxaloacetatec = oxaloacetatec - acetyl_CoAc;
                        h2oc = h2oc - acetyl_CoAc;
                        counter = acetyl_CoAc;
                        acetyl_CoAc = 0;
            }

            else if ( (h2oc >= acetyl_CoAc) && (acetyl_CoAc >= oxaloacetatec) )
            {
                        h2oc = h2oc - oxaloacetatec;
                        acetyl_CoAc = acetyl_CoAc - oxaloacetatec;
                        counter = oxaloacetatec;
                        oxaloacetatec = 0;
            }

            else if ( (h2oc >= oxaloacetatec) && (oxaloacetatec >= acetyl_CoAc) )
```

```
        {
                h2oc = h2oc - acetyl_CoAc;
                oxaloacetatec = oxaloacetatec - acetyl_CoAc;
                counter = acetyl_CoAc;
                acetyl_CoAc = 0;
        }

        else if ( (acetyl_CoAc == oxaloacetatec) && (acetyl_CoAc == h2oc) )
        {
                counter = acetyl_CoAc;
                acetyl_CoAc = 0;
                oxaloacetatec = 0;
                h2oc = 0;
        }

        holdin(passive);
        }
        else
        {
                passivate;
        }
    }
}
```

*Table 57 - Output Function for StepB1 of Krebs Cycle (**λ**)*

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

5.2.3 STEP B2

Citrate is converted to isocitrate by isomerizationcatalyzed by aconitase. This is actually a two-step reaction with cis-aconitase as an intermediate.

Figure 5.6 – StepB2 of Kreb's Cycle [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

Figure 5.7 - Atomic Model for StepB2 of Kreb's Cycle

$StepB2 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {citratec, ifacontinase, counter, phase, sigma}

X = {citrate, acontinase}

Y = {isocitrate}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = output function

Table 58 - External Function for StepB2 of Krebs Cycle (δ_{ext})

```
δext ( s, e, x ) {

      if ( x = citrate )
      {
           Citratec = citratec + x;
           if ( ifacontinase = true )
                 holdin(active);
           else
                 holdin(passive);
      }

      else if ( x = acontinase )
      {
           ifacontinase = true ;
           if (citratec > 0 )
                holdIn( active );
           else
                holdIn(passive );
       }

}
```

Table 59 - Internal Function for StepB2 of Krebs Cycle (δ_{int})

```
δint ( s, e ) {
      counter=0;
      if (s=passive)
      {
          passivate;
       }
       else
       {
           if ( citratec >= 1 and ifacontinase = true )
           {
               counter = citratec;
               citratec = 0;

               holdIn( passive );
            }

            else
            {
                    passivate;
             }
      }
}
```

*Table 60 - Output Function for StepB2 of Krebs Cycle (**λ**)*

```
λ ( s ) {
        if (counter is not zero)
        send outputs through the ports; // Isocitrate
}
```

5.2.4 STEP B3

Isocitrate is oxidized to α-ketuglutarate and CO_2 by isocitrate dehydrogenase. This enzyme requires NAD^+ which is reduced to NADH.

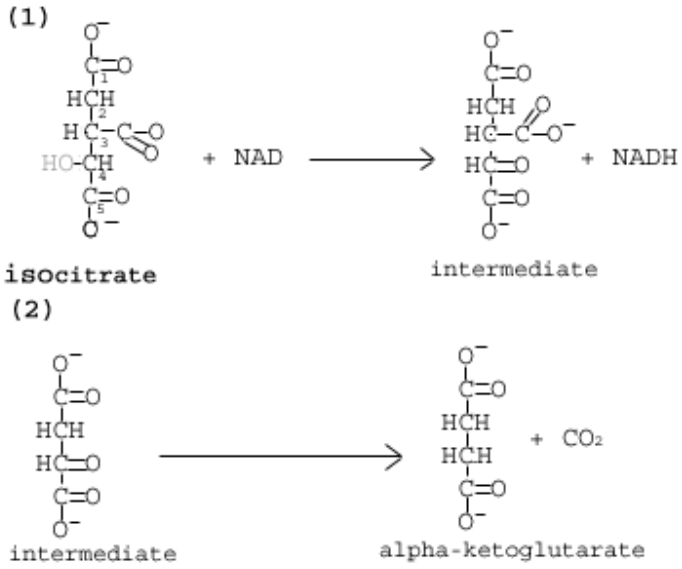

Figure 5.8 – StepB3 of Kreb's Cycle [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

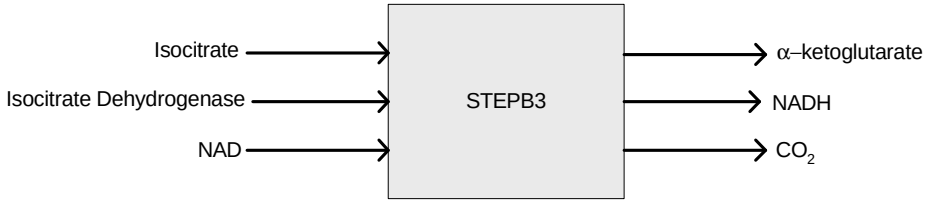

Figure 5.9 - Atomic Model for StepB3 of Kreb's Cycle

StepB3 = < *S, X, Y,* δ_{int}, δ_{ext}, *ta,* λ >

Where

S = {isocitratec, ifisocitrateDehydrogenase, nadc, counter, phase, sigma}

X = {isocitrate, isocitrateDehydrogenase, NAD}

Y = {alpha_ketuglutarate, NADH, CO2}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = output function

*Table 61 - External Function for StepB3 of Krebs Cycle (**δ_{ext}**)*

```
δext ( s, e, x ) {

        if ( x = isocitrate )
        {
                isocitratec = isocitratec + x ;
                if ( nadc>0 and ifisocitrateDehydrogenase=true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = NAD )
        {
                nadc = nadc + x ;
                if ( isocitratec>0 and ifisocitrateDehydrogenase=true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = isocitrateDehydrogenase )
        {
                ifisocitrateDehydrogenase = true ;
                if ( isocitratec > 0 and nadc > 0 )
                        holdIn( active );
                else
                        holdIn( passive );
        }

}
```

Table 62 - Internal Function for StepB3 of Krebs Cycle (δ_{int})

```
δint ( s, e ) {
      counter = 0;
      if ( s = passive )
      {
          passivate;
       }
       else
       {
            if(isocitratec>=1 and nadc>=1 and ifisocitrateDehydrogenase= true)
            {
                        if (isocitratec > nadc)
                        {
                                    isocitratec = isocitratec - nadc;
                                    counter = nadc;
                                    nadc = 0;
                        }

                        else if (isocitratec < nadc)
                        {
                                    nadc = nadc - isocitratec;
                                    counter = isocitratec;
                                    isocitratec = 0;
                        }

                        else if (isocitratec == nadc)
                        {
                                    counter = isocitratec;
                                    isocitratec = 0;
                                    nadc = 0;
                        }

                        holdIn( passive );
            }
            else
            {
                        passivate;
            }
      }
}
```

Table 63 - Output Function for StepB3 of Krebs Cycle (λ)

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

5.2.5 STEP B4

α-ketuglutarate is oxidized to succinyl CoA and CO_2 by the α-ketuglutarate dehydrogenase complex. This enzyme is a complex of three enzymes and uses NAD^+ as a cofactor.

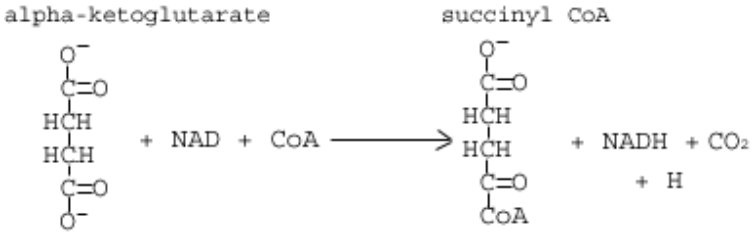

Figure 5.10 – StepB4 of Kreb's Cycle [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

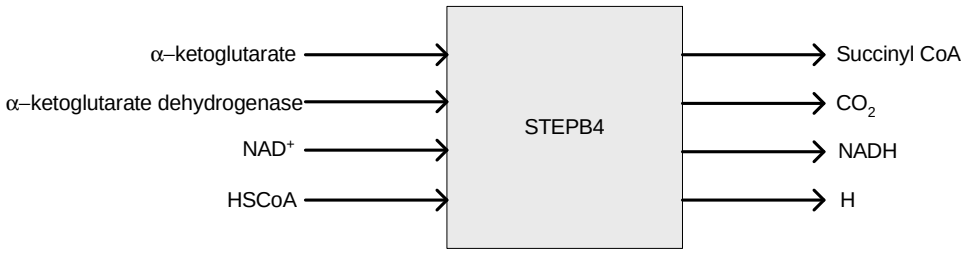

Figure 5.11 - Atomic Model for StepB4 of Kreb's Cycle

$StepB4 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {alpha_ketoglutaratec, ifalpha_ketoglutarateDehydrogenase, nadc, hscoaic, counter, phase, sigma}

X = {alpha_ketoglutarate, alpha_ketoglutarateDehydrogenase, NAD, HSCoAi}

Y = {succinyl_CoA, NADH, CO2, H}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = output function

Table 64 - External Function for StepB4 of Krebs Cycle ($\boldsymbol{\delta_{ext}}$)

```
δext ( s, e, x ) {
        if( x = alpha_ketoglutarate )
        {
                alpha_ketoglutaratec = alpha_ketoglutaratec + x ;
                if (nadc>0 and hscoaic>0 and ifalpha_ketoglutarateDehydrogenase=true)
                        holdIn( active );
                else
                        holdIn(passive );
        }

        else if( x = NAD )
        {
                nadc = nadc + x ;
                if (alpha_ketoglutaratec>0 and hscoaic>0 and ifalpha_ketoglutarateDehydrogenase=true)
                        holdIn( active );
                else
                        holdIn(passive );
        }

        else if( x = HSCoAi )
        {
                hscoaic = hscoaic + x ;
                if (alpha_ketoglutaratec>0 and nadc>0 and ifalpha_ketoglutarateDehydrogenase=true)
                        holdIn( active );
                else
                        holdIn(passive );
        }

        else if ( x = alpha_ketoglutarateDehydrogenase )
        {
                ifalpha_ketoglutarateDehydrogenase = true ;
                if  (alpha_ketoglutaratec>0 and nadc>0 and hscoaic>0)
                        holdIn( active );
                else
                        holdIn( passive );
        }

}
```

Table 65 - Internal Function for StepB4 of Krebs Cycle (δ_{int})

```
δint ( s, e ) {
      counter=0;
      if (s=passive)
      {
         passivate;
       }
       else
       {
         if (alpha_ketoglutaratec>=1 and nadc>=1 and hscoaic>=1and  ifalpha_ketoglutarateDehydrogenase=true)
           {
                    if (alpha_ketoglutaratec>=nadc and nadc>=hscoaic)
                    {
                              alpha_ketoglutaratec = alpha_ketoglutaratec - hscoaic;
                              nadc = nadc - hscoaic;
                              counter = hscoaic;
                              hscoaic = 0;
                    }

                    else if (alpha_ketoglutaratec>=hscoaic and hscoaic>=nadc)
                    {
                              alpha_ketoglutaratec = alpha_ketoglutaratec - nadc;
                              hscoaic = hscoaic - nadc;
                              counter = nadc;
                              nadc = 0;
                    }

                    else if (nadc>=alpha_ketoglutaratec and alpha_ketoglutaratec>= hscoaic)
                    {
                              nadc = nadc - hscoaic;
                              alpha_ketoglutaratec = alpha_ketoglutaratec - hscoaic;
                              counter = hscoaic;
                              hscoaic = 0;
                    }

                    else if (nadc>=hscoaic and hscoaic>=alpha_ketoglutaratec)
                    {
                              nadc = nadc - alpha_ketoglutaratec;
                              hscoaic = hscoaic - alpha_ketoglutaratec;
                              counter = alpha_ketoglutaratec;
                              alpha_ketoglutaratec = 0;
                    }

                    else if (hscoaic>=alpha_ketoglutaratec and alpha_ketoglutaratec>= nadc)
                    {
                              hscoaic = hscoaic - nadc;
                              alpha_ketoglutaratec = alpha_ketoglutaratec - nadc;
                              counter = nadc;
                              nadc = 0;
                    }

                    else if (hscoaic>=nadc and nadc>=alpha_ketoglutaratec)
```

```
                {
                        hscoaic = hscoaic - alpha_ketoglutaratec;
                        nadc = nadc - alpha_ketoglutaratec;
                        counter = alpha_ketoglutaratec;
                        alpha_ketoglutaratec = 0;
                }

                else if (alpha_ketoglutaratec=nadc and nadc=hscoaic)
                {
                        counter = alpha_ketoglutaratec;
                        alpha_ketoglutaratec = 0;
                        nadc = 0;
                        hscoaic = 0;
                }

                holdIn(passive);
        }

        else
        {
                passivate;
        }
    }
}
```

*Table 66 - Output Function for StepB4 of Krebs Cycle (**λ**)*

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

5.2.6 STEP B5

Succinyl CoA is converted to succinate by succinyl CoA synthetase. This reaction uses the energy released by the cleavage of the succinyl-CoA bond to synthesize GTP from P_i and GDP.

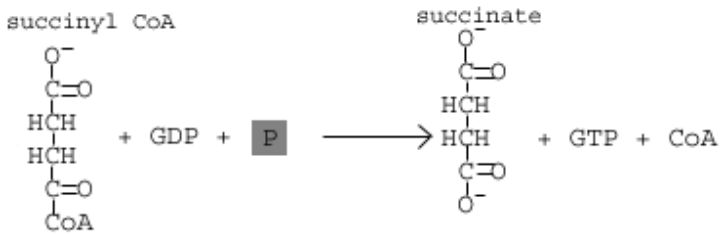

Figure 5.12 – StepB5 of Kreb's Cycle [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

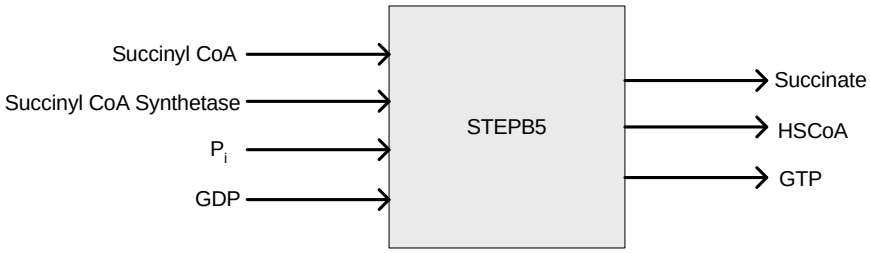

Figure 5.13 - Atomic Model for StepB5 of Kreb's Cycle

$StepB5 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {succinyl_CoAc, ifsuccinylCoA_Synthetase, pic, gdpc, counter, phase, sigma}
X = {succinyl_CoA, succinylCoA_Synthetase, Pi, GDP}

Y = {succinate, HSCoAo, GTP}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = output function

Table 67 - External Function for StepB5 of Krebs Cycle (δ_{ext})

```
δext ( s, e, x ) {

        if( x = succinyl_CoA )
        {
                succinyl_CoAc = succinyl_CoAc + x ;
                if (gdpc>0 and pic>0 and ifsuccinylCoA_Synthetase= true)
                        holdIn( active );
                else
                        holdIn(passive );
        }

        else if( x = GDP )
        {
                gdpc = gdpc + x ;
                if (succinyl_CoAc>0 and pic>0 and ifsuccinylCoA_Synthetase=true)
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if( x = Pi )
        {
                pic = pic + x ;
                if (succinyl_CoAc>0 and gdpc>0 and ifsuccinylCoA_Synthetase=true)
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = succinylCoA_Synthetase )
        {
                ifsuccinylCoA_Synthetase = true;
                if ( succinyl_CoAc > 0 and gdpc > 0 and pic > 0  )
                        holdIn( active );
                else
                        holdIn( passive );
        }

}
```

Table 68 - Internal Function for StepB5 of Krebs Cycle (δ_{int})

```
δint ( s, e ) {
      counter = 0;
      if  ( s = passive )
      {
          passivate;
       }
       else
       {
```

```
if ( succinyl_CoAc>=1 and gdpc>= 1 and pic>=1 )
{
        if ( succinyl_CoAc>=gdpc and gdpc>=pic )
        {
                succinyl_CoAc = succinyl_CoAc - pic;
                gdpc = gdpc - pic;
                counter = pic;
                pic = 0;
        }

        else if (succinyl_CoAc >= pic and pic>=gdpc )
        {
                succinyl_CoAc = succinyl_CoAc - gdpc;
                pic = pic - gdpc;
                counter = gdpc;
                gdpc = 0;
        }

        else if ( gdpc>=succinyl_CoAc and succinyl_CoAc>=pic )
        {
                gdpc = gdpc - pic;
                succinyl_CoAc = succinyl_CoAc - pic;
                counter = pic;
                pic = 0;
        }

        else if ( gdpc>=pic and pic>=succinyl_CoAc )
        {
                gdpc = gdpc - succinyl_CoAc;
                pic = pic - succinyl_CoAc;
                counter = succinyl_CoAc;
                succinyl_CoAc = 0;
        }

        else if ( pic>=succinyl_CoAc and succinyl_CoAc>=gdpc )
        {
                pic = pic - gdpc;
                succinyl_CoAc = succinyl_CoAc - gdpc;
                counter = gdpc;
                gdpc = 0;
        }

        else if ( pic>=gdpc and gdpc>=succinyl_CoAc )
        {
                pic = pic - succinyl_CoAc;
                gdpc = gdpc - succinyl_CoAc;
                counter = succinyl_CoAc;
                succinyl_CoAc= 0;
        }

        else if ( succinyl_CoAc=gdpc and gdpc = pic )
        {
                counter = succinyl_CoAc;
```

```
                succinyl_CoAc = 0;
                gdpc = 0;
                pic = 0;
        }

        holdIn( passive );
    }
    else
    {
        passivate;
    }
  }
}
```

*Table 69 - Output Function for StepB5 of Krebs Cycle (**λ**)*

```
λ ( s ) {
        if (counter is not zero)
        send outputs through the ports; // succinate, HSCoAo, GTP
}
```

5.2.7 STEP B6

Succinate is oxidized to fumarate by succinate dehydrogenase. FAD is tightly bound to this enzyme and is reduced to $FADH_2$.

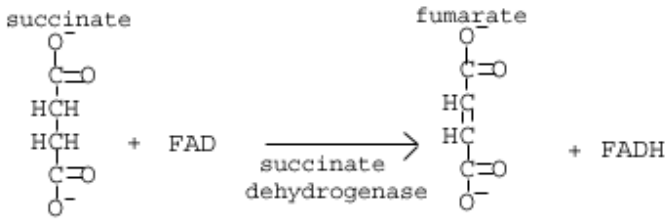

Figure 5.14 – StepB6 of Kreb's Cycle [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

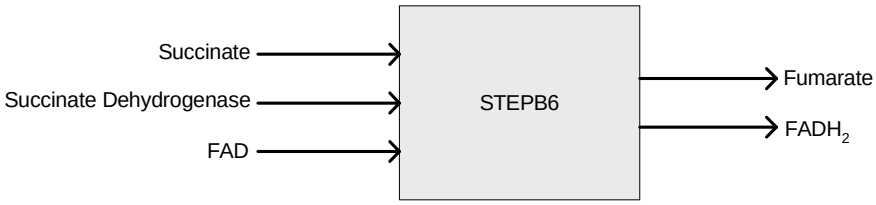

Figure 5.15 - Atomic Model for StepB6 of Kreb's Cycle

$StepB6 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {succinatec, ifsuccinateDehydrogenase, fadc, counter, phase, sigma}

X = {succinate, succinateDehydrogenase, FAD}

Y = {fumarate, FADH2}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = output function

Table 70 - External Function for StepB6 of Krebs Cycle ($\boldsymbol{\delta_{ext}}$)

```
δext ( s, e, x ) {

        if( x = succinate )
        {
                succinatec = succinatec + x ;
                if ( (fadc > 0 and ifsuccinateDehydrogenase = true )
                        holdIn( active );
                else
                        holdIn(passive );
        }

        else if ( x = FAD )
        {
                fadc = fadc + x ;
                if (succinatec > 0 and ifsuccinateDehydrogenase = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }
```

```
        else if ( x = succinateDehydrogenase )
        {
                ifsuccinateDehydrogenase = true;
                if ( succinatec > 0 and fadc = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

}
```

Table 71 - Internal Function for StepB6 of Krebs Cycle (δ_{int})

```
δint ( s, e ) {
    counter=0;
    if (s=passive)
    {
        passivate;
    }
    else
    {
        if (succinatec>=1 and fadc>= 1 and ifsuccinateDehydrogenase=true)
        {
                if ( succinatec > fadc )
                {
                        succinatec = succinatec - fadc;
                        counter = fadc;
                        fadc = 0;
                }

                else if ( succinatec < fadc )
                {
                        fadc = fadc - succinatec;
                        counter = succinatec;
                        succinatec = 0;
                }

                else if ( succinatec == fadc )
                {
                        counter = succinatec;
                        succinatec = 0;
                        fadc = 0;
                }

                holdIn( passive );
        }
        else
        {
                passivate;
        }
    }
}
```

*Table 72 - Output Function for StepB6 of Krebs Cycle (**λ**)*

```
λ ( s ) {
        if (counter is not zero)
        send outputs through the ports; // fumarate, FADH2
}
```

5.2.8 STEP B7

Fumarate is converted to malate by fumarase. This is a hydration reaction that requires the addition of a water molecule.

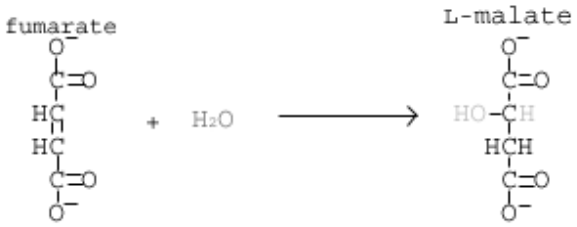

Figure 5.16 – StepB7 of Kreb's Cycle [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

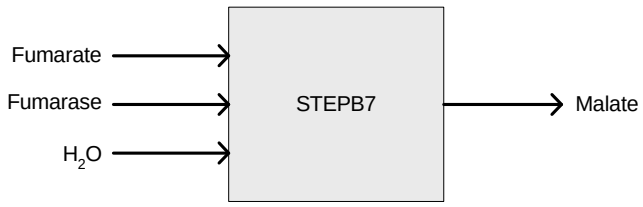

Figure 5.17 - Atomic Model for StepB7 of Kreb's Cycle

$StepB7 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {fumaratec, iffumarase, h2oc, counter, phase, sigma}

X = {fumarate, fumarase, H2O}

Y = {malate}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = output function

Table 73 - External Function for StepB7 of Krebs Cycle (δ_{ext})

```
δext ( s, e, x ) {
        if ( x = fumarate )
        {
                fumaratec = fumaratec + x ;
                if ( h2oc > 0  and iffumarase = true )
                        holdIn( active );
                else
                        holdIn(passive );
        }

        else if ( x = H2O )
        {
                h2oc = h2oc + x ;
                if ( fumaratec > 0 and iffumarase = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = fumarase )
        {
                iffumarase = true ;
                if ( fumaratec>0 and h2oc>0 )
                        holdIn( active );
                else
                        holdIn( passive );
        }

}
```

Table 74 - Internal Function for StepB7 of Krebs Cycle (δ_{int})

```
δint ( s, e ) {
    counter = 0;
    if ( s = passive )
```

```
    {
       passivate;
     }
     else
     {
          if ( fumaratec>=1 and h2oc>=1 and iffumarase=true )
          {
                    if ( fumaratec > h2oc )
                    {
                              fumaratec = fumaratec - h2oc;
                              counter = h2oc;
                              h2oc = 0;
                    }

                    else if ( fumaratec < h2oc )
                    {
                              h2oc = h2oc - fumaratec;
                              counter = fumaratec;
                              fumaratec = 0;
                    }

                    else if ( fumaratec == h2oc )
                    {
                              counter = fumaratec;
                              fumaratec = 0;
                              h2oc = 0;
                    }

                   holdIn( passive );
          }

           else
           {
                  passivate;
            }
    }
}
```

*Table 75 - Output Function for StepB7 of Krebs Cycle (**λ**)*

```
λ ( s ) {
          if (counter is not zero)
          send output ‘malate’ through the output port;
}
```

5.2.9 STEP B8

Malate is oxidized to oxaloacetate by the enzyme malate dehydrogenase. NAD^+ is required by this enzyme to accept the free pair of electrons and produce NADH.

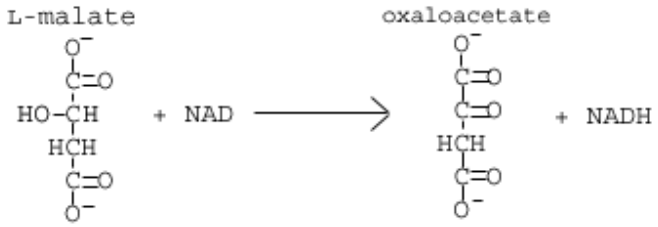

Figure 5.18 – StepB8 of Kreb's Cycle [Source: SparkNotes LLC, Part of the Barnes & Noble Learning Network, http://www.sparknotes.com/biology/]

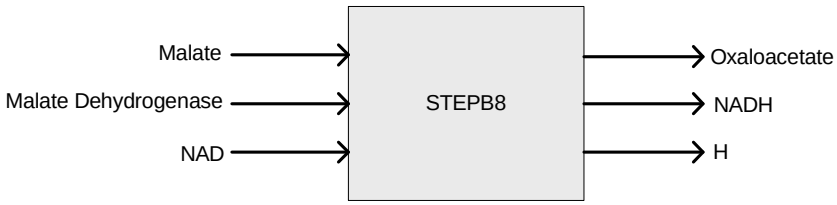

Figure 5.19 - Atomic Model for StepB8 of Kreb's Cycle

$StepB8 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

S = {malatec, ifmalateDehydrogenase, nadc, counter, phase, sigma}

X = {malate, malateDehydrogenase, NAD}

Y = {oxaloacetate, NADH, H}

δ_{int} = Internal function

δ_{ext} = External function

ta = Sigma

λ = output function

Table 76 - External Function for StepB8 of Krebs Cycle ($\boldsymbol{\delta_{ext}}$)

```
δext ( s, e, x ) {

        if( x = malate )
        {
                malatec = malatec + x ;
                if ( nadc > 0 and ifmalateDehydrogenase = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        elseif ( x = NAD )
        {
                nadc = nadc + x ;
                if ( malatec > 0 and ifmalateDehydrogenase = true )
                        holdIn( active );
                else
                        holdIn( passive );
        }

        else if ( x = malateDehydrogenase )
        {
                ifmalateDehydrogenase = true ;
                if ( malatec > 0 and nadc > 0 )
                        holdIn( active );
                else
                        holdIn( passive );
        }
}
```

Table 77 - Internal Function for StepB8 of Krebs Cycle ($\boldsymbol{\delta_{int}}$)

```
δint ( s, e ) {
    counter=0;
    if (s=passive)
    {
        passivate;
     }
     else
     {
          if (malatec>=1 and nadc>=1 and ifmalateDehydrogenase = true)
```

```
        {
                if ( malatec > nadc )
                {
                        malatec = malatec - nadc;
                        counter = nadc;
                        nadc = 0;
                }

                else if ( malatec < nadc )
                {
                        nadc = nadc - malatec;
                        counter = malatec;
                        malatec = 0;
                }

                else if ( malatec == nadc )
                {
                        counter = malatec;
                        malatec = 0;
                        nadc = 0;
                }

           holdIn( passive );
        }
         else
         {
                passivate;
         }
    }
}
```

*Table 78 - Output Function for StepB8 of Krebs Cycle (**λ**)*

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

5.3 COUPLED MODEL

All the steps explained in the previous section were all implemented in CD++. To do this, first the behaviour of each component was carefully specified with an analysis of inputs and outputs for each step. Each step was defined as a DEVS model following the specification. Afterwards, each model was implemented in CD++, and tested separately.

Finally the main model was built as a coupled model connecting all the submodels previously defined. This model follows the design presented in Figure 5.20, and its detailed definition can be found in table 75.

Table 79 - Krebs.MA

```
[top]
components : stepA@StepA stepB1@StepB1 stepB2@StepB2 stepB3@StepB3
stepB4@StepB4 stepB5@StepB5 stepB6@StepB6 stepB7@StepB7 stepB8@StepB8

out : FADH2 GTP HSCoAo H NADH CO2
in : pyruvate pyruvateDehydrogenase HSCoAi NAD oxaloacetate H2O citrateSynthase
acontinase isocitrateDehydrogenase alpha_ketoglutarateDehydrogenase GDP Pi
succinylCoA_Synthetase FAD succinateDehydrogenase fumarase malateDehydrogenase

Link : pyruvate pyruvate@stepA
Link : pyruvateDehydrogenase pyruvateDehydrogenase@stepA
Link : HSCoAi HSCoAi@stepA
Link : NAD NAD@stepA
Link : citrateSynthase citrateSynthase@stepB1
Link : H2O H2O@stepB1
Link : oxaloacetate oxaloacetate@stepB1
Link : acontinase acontinase@stepB2
Link : NAD NAD@stepB3
Link : isocitrateDehydrogenase isocitrateDehydrogenase@stepB3
Link : HSCoAi HSCoAi@stepB4
Link : NAD NAD@stepB4
Link : alpha_ketoglutarateDehydrogenase alpha_ketoglutarateDehydrogenase@stepB4
Link : GDP GDP@stepB5
Link : Pi Pi@stepB5
Link : succinylCoA_Synthetase succinylCoA_Synthetase@stepB5
Link : FAD FAD@stepB6
Link : succinateDehydrogenase succinateDehydrogenase@stepB6
Link : H2O H2O@stepB7
Link : fumarase fumarase@stepB7
Link : NAD NAD@stepB8
Link : malateDehydrogenase malateDehydrogenase@stepB8

Link : NADH@stepA NADH
Link : CO2@stepA CO2
Link : H@stepA H
Link : HSCoAo@stepB1 HSCoAo
Link : H@stepB1 H
Link : NADH@stepB3 NADH
Link : CO2@stepB3 CO2
Link : NADH@stepB4 NADH
Link : H@stepB4 H
Link : CO2@stepB4 CO2
Link : HSCoAo@stepB5 HSCoAo
```

```
Link : GTP@stepB5 GTP
Link : FADH2@stepB6 FADH2
Link : NADH@stepB8 NADH
Link : H@stepB8 H

Link : acetyl_CoA@stepA acetyl_CoA@stepB1
Link : citrate@stepB1 citrate@stepB2
Link : isocitrate@stepB2 isocitrate@stepB3
Link : alpha_ketoglutarate@stepB3 alpha_ketoglutarate@stepB4
Link : succinyl_CoA@stepB4 succinyl_CoA@stepB5
Link : succinate@stepB5 succinate@stepB6
Link : fumarate@stepB6 fumarate@stepB7
Link : malate@stepB7 malate@stepB8
Link : oxaloacetate@stepB8 oxaloacetate@stepB1

[stepA]
preparation : 00:00:05:000

[stepB1]
preparation : 00:00:10:000

[stepB2]
preparation : 00:00:25:000

[stepB3]
preparation : 00:00:40:000

[stepB4]
preparation : 00:00:45:000

[stepB5]
preparation : 00:00:55:000

[stepB6]
preparation : 00:01:05:000

[stepB7]
preparation : 00:01:20:000

[stepB8]
preparation : 00:01:35:000
```

Running the simulation for Krebs model will generate a log file that will look like the following table. Only a fragment of the krebs.log is shown in table 78 (for a full file, refer to Appendix B).

Table 80 - A section of Krebs.log file

```
Mensaje I / 00:00:00:000 / Root(00) para top(01)
Mensaje I / 00:00:00:000 / top(01) para stepa(02)
Mensaje I / 00:00:00:000 / top(01) para stepb1(03)
Mensaje I / 00:00:00:000 / top(01) para stepb2(04)
Mensaje I / 00:00:00:000 / top(01) para stepb3(05)
Mensaje I / 00:00:00:000 / top(01) para stepb4(06)
Mensaje I / 00:00:00:000 / top(01) para stepb5(07)
Mensaje I / 00:00:00:000 / top(01) para stepb6(08)
Mensaje I / 00:00:00:000 / top(01) para stepb7(09)
Mensaje I / 00:00:00:000 / top(01) para stepb8(10)
Mensaje D / 00:00:00:000 / stepa(02) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb1(03) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb2(04) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb3(05) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb4(06) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb5(07) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb6(08) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb7(09) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb8(10) / ... para top(01)
Mensaje D / 00:00:00:000 / top(01) / ... para Root(00)
Mensaje X / 00:00:10:000 / Root(00) / pyruvate /      5.00000 para top(01)
Mensaje X / 00:00:10:000 / top(01) / pyruvate /      5.00000 para stepa(02)
Mensaje D / 00:00:10:000 / stepa(02) / 00:00:00:000 para top(01)
Mensaje D / 00:00:10:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:10:000 / Root(00) para top(01)
Mensaje * / 00:00:10:000 / top(01) para stepa(02)
Mensaje D / 00:00:10:000 / stepa(02) / ... para top(01)
Mensaje D / 00:00:10:000 / top(01) / ... para Root(00)
Mensaje X / 00:01:11:000 / Root(00) / pyruvatedehydrogenase /      1.00000 para top(01)
Mensaje X / 00:01:11:000 / top(01) / pyruvatedehydrogenase /      1.00000 para stepa(02)
Mensaje D / 00:01:11:000 / stepa(02) / 00:00:00:000 para top(01)
Mensaje D / 00:01:11:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:01:11:000 / Root(00) para top(01)
Mensaje * / 00:01:11:000 / top(01) para stepa(02)
Mensaje D / 00:01:11:000 / stepa(02) / ... para top(01)
Mensaje D / 00:01:11:000 / top(01) / ... para Root(00)
Mensaje X / 00:01:14:000 / Root(00) / citratesynthase /      1.00000 para top(01)
…
```

The output file from Krebs model (see table 77) can be checked to verify the validity of the model.

Table 81 - Krebs.out: An Example of Output File

```
00:01:11:000 nadh 1
00:01:11:000 co2 1
```

```
00:01:11:000 h 1
00:02:01:000 hscoao 1
00:02:01:000 h 1
00:02:30:000 nadh 1
00:02:30:000 co2 1
00:02:30:000 nadh 1
00:02:30:000 co2 1
00:02:30:000 h 1
00:03:11:000 hscoao 1
00:03:11:000 gtp 1
00:03:28:000 fadh2 1
00:03:45:000 nadh 1
00:03:45:000 h 1
```

The coupled model for Krebs cycle consists of the following atomic models: stepA, stepB1, stepB2, stepB3, stepB4, stepB5, stepB6, stepB7, and stepB8. See figure 4.6 for a drawing of the Krebs model.

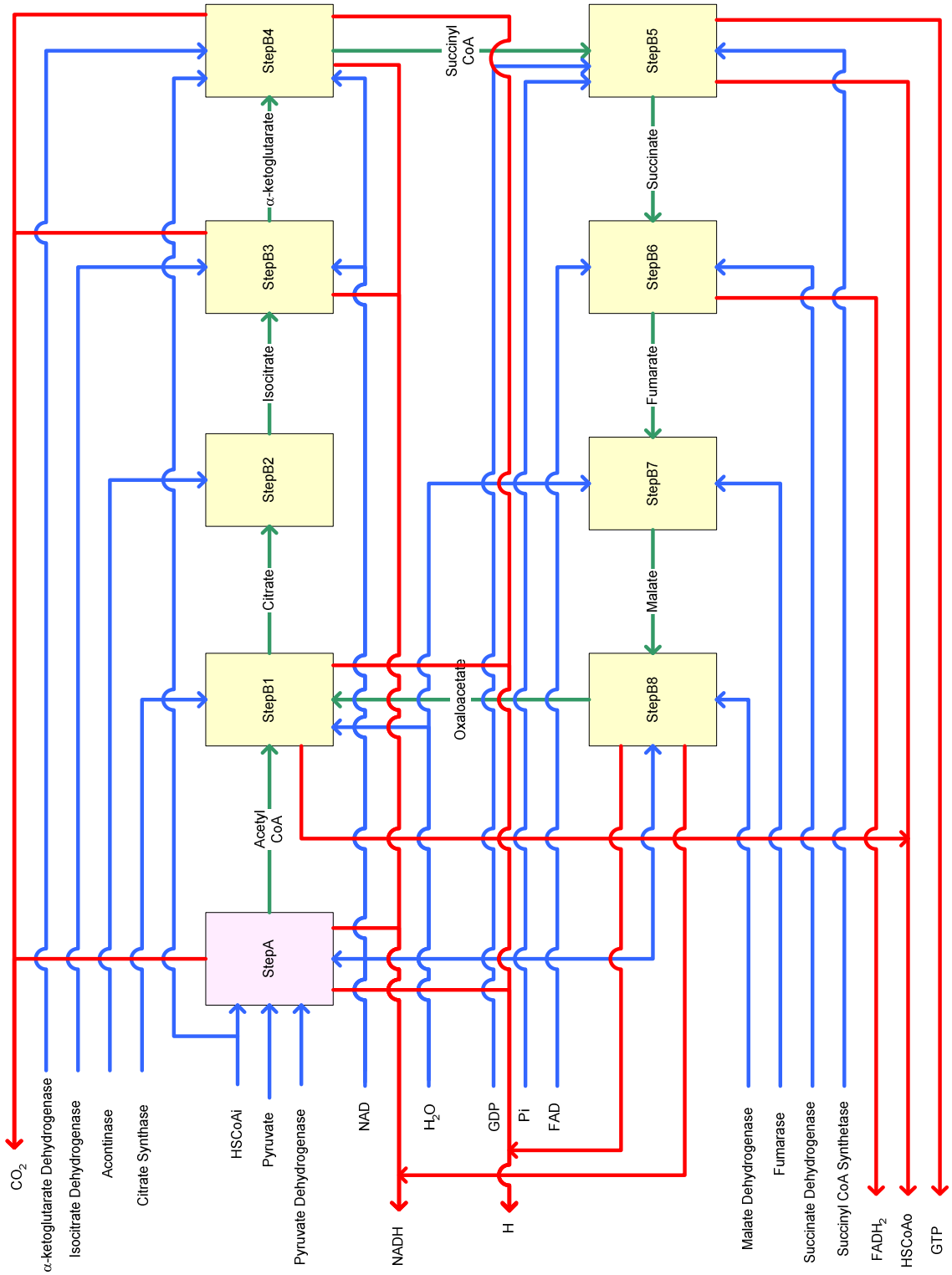

Figure 5.20 - Coupled Model of Kreb's Cycle

CD++ Modeler graphical user interface was used to represent the DEVS model for Kreb's cycle that shows the execution results for the model (see Figure 5.20).

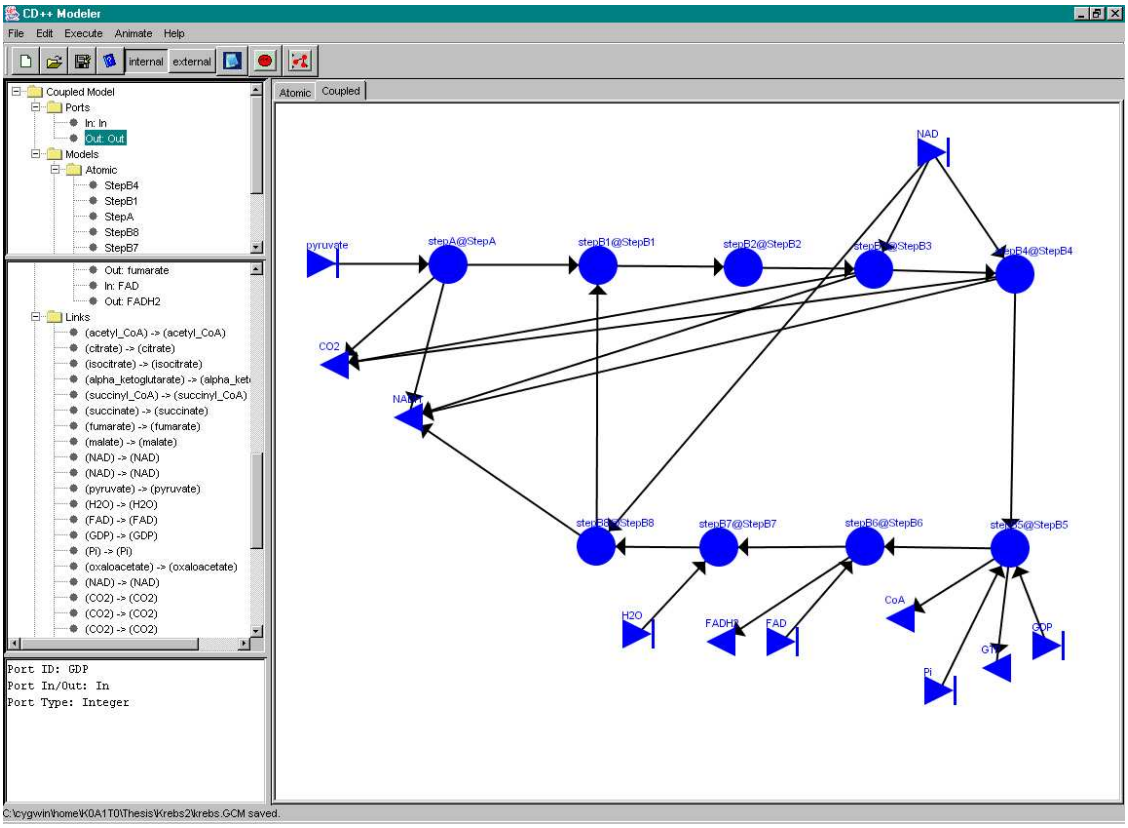

Figure 5.21 - krebs.GCM

5.4 SUMMARY

The CD++ tool, based on the formalism allows the definition of complex biological models using a high-level specification language. In this way, the construction of simulations can be improved greatly.

Krebs cycle is an aerobic process that oxidizes pyruvates to CO_2.

The sum of all reactions in the Kreb's cycle can be summarized as follow:

$$\text{Acetyl-CoA} + 3NAD^+ + FAD + ADP + P_i \rightarrow \text{CoA-SH} + 3NADH + H^+ + FADH_2 + ATP + 2CO_2 \quad (5.1)$$

Therefore, one molecule of glucose, results in two molecules of ATP and two molecules of NADH as energy.

Table 82 - Products of Kreb's Cycle

Product of StepA
1 NADH
Products of StepB1-B8
1 ATP
3 NADH
1 $FADH_2$

At this point, this pathway can be added to glycolytic pathway, and tested. This will present a coupled model of two completely tested sub-modules: Glycolysis, and Kreb's cycle. This is just the beginning. Our model can start growing to represent even more complex biological system.

CHAPTER 6-ELECTRON TRANSPORT CHAIN AND OXIDATIVE PHOSPHORYLATION

ATP production via Glycolysis discussed in the previous chapters is only part of the story, and everyone can see how much energy is wasted in this process. Converting Pyruvate (the product of Glycolysis) to CO_2 and trap the released chemical free energy in the form of ATP is the main challenge. This problem itself was decomposed into several problems: The pathway of breaking down carbon compounds, called the Krebs Cycle, solved by H. Krebs which was described in the previous chapter, and the oxidation of NADH and succinate by Electron Transport Chain, and the coupling of electron transport to ATP synthesis, called Oxidative Phosphorylation, resulting in the "Chemiosmotic Hypothesis" of P. Mitchell [11].

Electron Transport and oxidative phosphorylation reoxidize NADH and $FADH_2$ and trap the energy released as ATP. In eukaryotes, electron transport and oxidative phosphorylation occur in the inner membrane of mitochondria.

6.1 ELECTRON TRANSPORT CHAIN

The electron transport chain is a complex sequence found in the mitochondrial membrane that accepts electrons from electron donors such as NADH or succinate,

shuttles these electrons across the mitochondrial membrane creating an electrical and chemical gradient, and through the proton driven chemistry of the ATP synthase[11] (aka the F_0F_1 particle) generates Adenosine TriPhosphate (ATP).

All of these are proteolipid[12] complexes, with the first four containing either Flavins, Iron-Sulfur clusters, Copper centers, or Heme moieties. Coenzyme Q collects electrons from Complex I and II and delivers them by diffusion to Complex III. Cytochrome c is water soluble from Complex III to Complex IV. In the process of electron transport, proton gradient generated by electron transport creates an enormous source of potential energy, which is used to synthesize ATP as protons flow back into the matrix.

[11] An ATP synthase is a general term for an enzyme that can synthesize Adenosine Triphosphate (ATP) from Adenosine Diphoshate (ADP) and inorganic phosphate by utilizing some form of energy. The overall reaction sequence is: ADP + $P_i \rightarrow$ ATP. These enzymes are of crucial importance in almost all organisms, because ATP is the common "energy currency" of cells.

[12] A lipoprotein is a biochemical assembly that contains both proteins and lipids and maybe structural or catalytic in function. Lipoproteins may be enzymes, proton pumps, or some combination of these functions. Examples include the high density and low density lipoproteins of the blood and the transmembrane proteins of the mitochondrion and the chloroplast. The lipids are often an essential part of the complex, even if they seem to have no catalytic activity themselves. To isolate transmembrane lipoproteins from their associated membranes, detergents are often needed. [8]

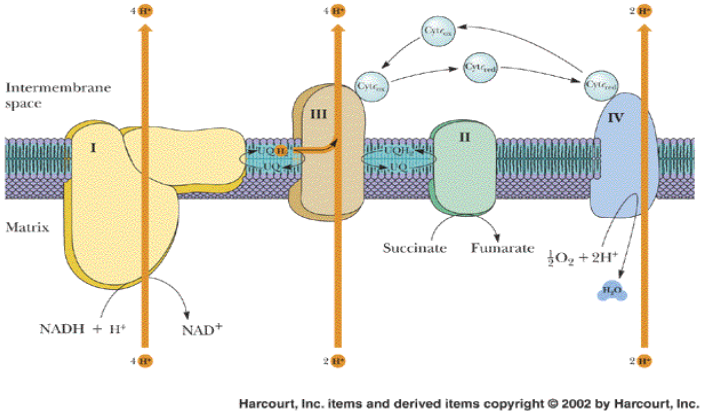

Figure 6.1 – Electron Transport Chain [Source: Harcourt, Inc. items and derived items copyright © 2002 by Harcourt, Inc.]

A large protein complex called ATP synthase is embedded in that membrane and enables protons to pass through in both directions; it generates ATP when the proton moves with (down) the gradient, and it costs ATP to pump a proton against (up) the gradient. Because protons have already been pumped into the intermembrane space against the gradient, they now can flow back into the mitochondrial matrix via the ATP synthase, generating ATP in the process.

The reaction is: $ADP^{3-} + H^+ + P_i \leftrightarrow ATP^{4-} + H_2O$

The *electrochemical potential* is a thermodynamic measure that reflects energy from entropy and electrostatics and is typically invoked in molecular processes that involve diffusion. It represents one of many interchangeable forms of potential energy through which energy may be conserved. In electrochemistry, it is typically applied in contexts where a chemical reaction is to take place, such as the transfer of an electron at a battery

electrode. The gradient of protons formed across the inner membrane by this process of active transport forms a miniature battery. The protons can flow back down this gradient, reentering the matrix, only through another complex of integral proteins in the inner membrane, the ATP synthase complex.

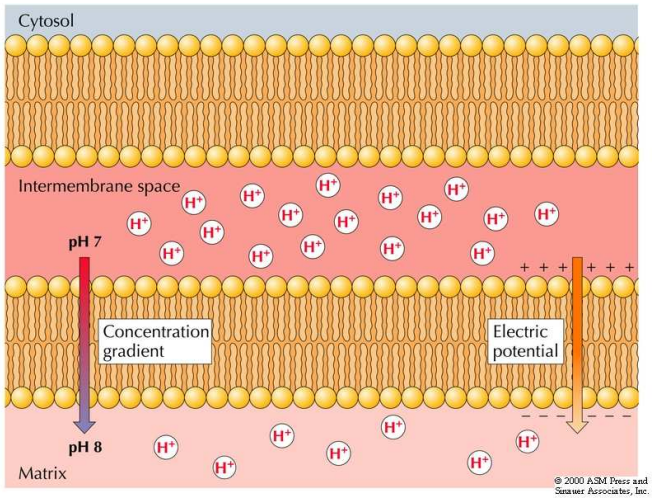

Figure 6.2 – Electrochemical Gradient, [Source: © 2000 ASM Press and Sinauer Associates, Inc.]

Adenosine Triphosphate (ATP) is the nucleotide known in biochemistry as the "molecular currency" of intracellular energy transfer. Chemically, ATP consists of adenosine and three phosphate groups. ATP is able to store and transport chemical energy within cells. ATP also plays an important role in the synthesis of nucleic acids.

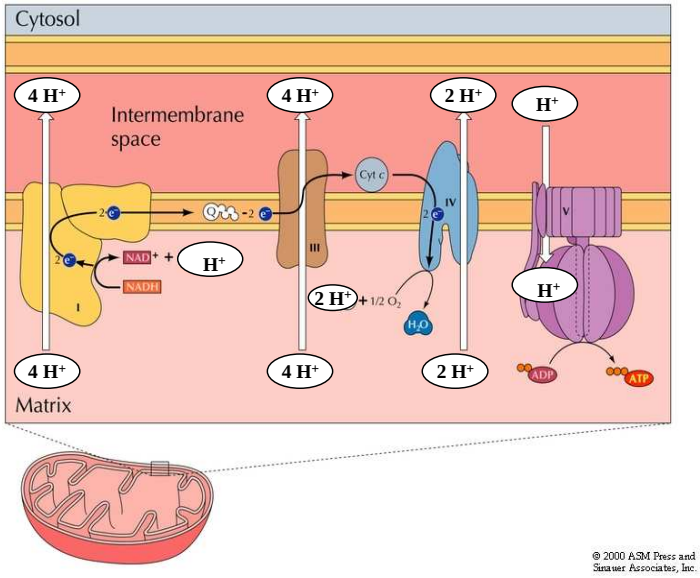

Figure 6.3 – Electron Transport Chain (Complex I, III, IV, and V), [Source: © 2000 ASM Press and Sinauer Associates, Inc.]

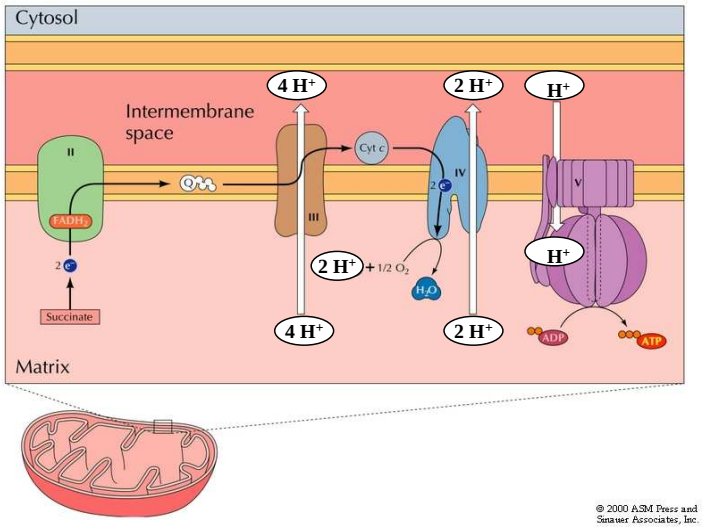

Figure 6.4 – Electron Transport Chain (Complex II, III, IV, and V), [Source: © 2000 ASM Press and Sinauer Associates, Inc.]

The respiratory chain accomplishes the stepwise transfer of electrons from NADH and $FADH_2$ to oxygen molecules to form H_2O with the use of protons. Cytochrome c

transfers only one electron at a time, so cytochrome c oxidase will wait till four electrons are accumulated to react with oxygen. And two freely diffused molecules that shuttle electrons from one complex to the next.

Electron Transport Chain happens on Cristae membrane, pumps H+ out of matrix, where NADH and $FADH_2$ being fed to it from Kreb's cycle, and large electrochemical proton gradient drives ATP Synthase (Complex V) to synthesise ATP.

6.2 CHEMIOSMOSIS IN MITOCHONDRIA

The energy released as electrons pass down the gradient from NADH to oxygen is harnessed by the three enzyme complexes of the respiratory chain to pump protons (H^+) against their concentration gradient from the matrix into the inter-membrane space of mitochondria. As their concentration increases (or their pH decreases) there, a strong diffusion gradient is set up. The only exit for these protons is through the ATP synthase complex (complex V). The energy released as these protons flow down their gradient is harnessed to the synthesis of ATP. This process is called chemiosmosis.

Chemiosmotic phosphorylation: The energized electrons released during the previous steps are used to concentrate hydrogen ions in one area to create a chemical gradient between positively and negatively charged ions (like a battery). The potential energy resulting from this osmotic gradient is used to resynthesize ATP from ADP. After electrons have been used, they must be transferred to O_2.

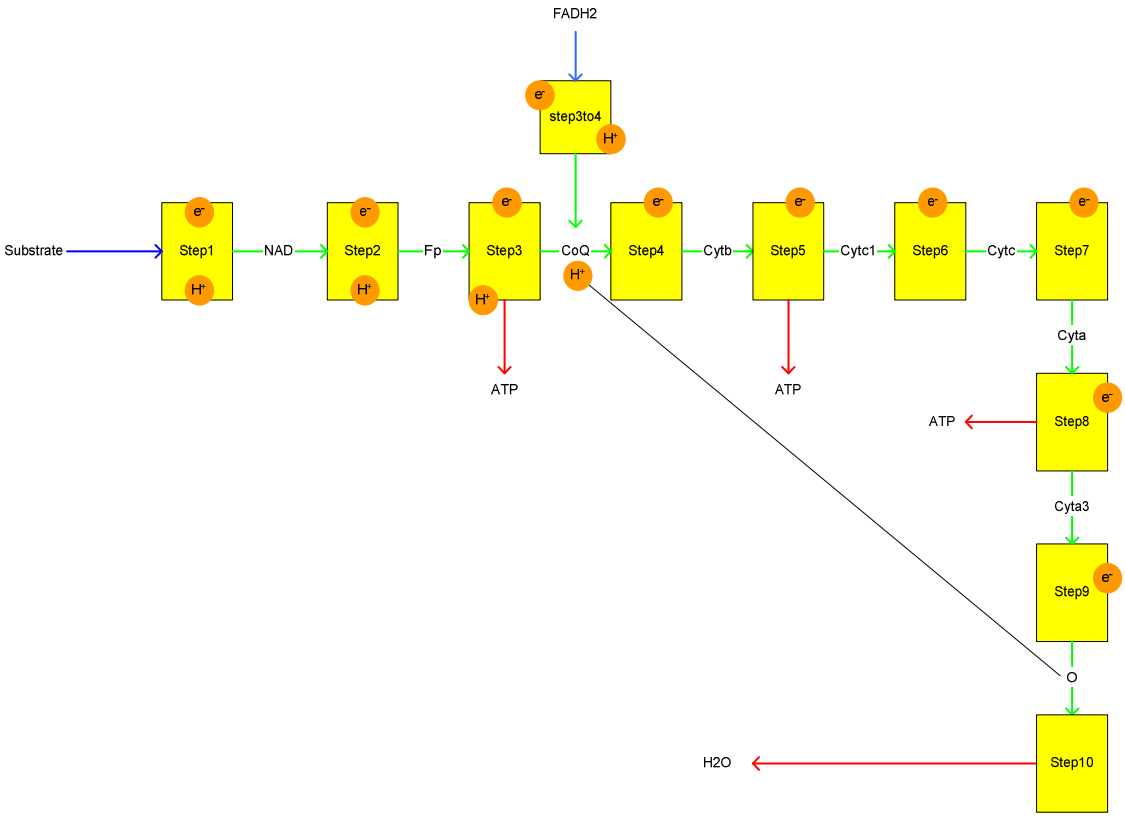

Figure 6.5 – Coupled Model of Electron Transport Chain

6.3 KEY POINTS IN ELECTRON TRANSPORT CHAIN

1. Protons are translocated across the membrane, from the matrix to the inter-membrane space.

2. Electrons are transported along the membrane, through a series of protein carriers.

3. Oxygen is the terminal electron acceptor, combining with electrons and H^+ ions to produce water.

4. As NADH delivers more H^+ and electrons into the ETS, the proton gradient increases, with H^+ building up outside the inner mitochondrial membrane, and OH^- inside the membrane.

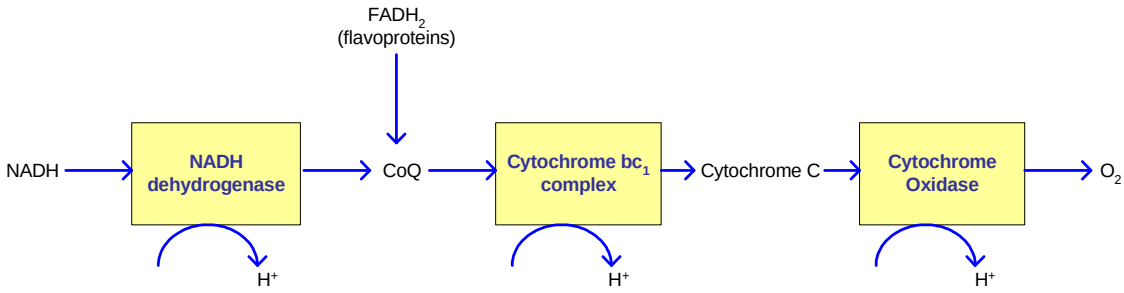

Figure 6.6 – Overview of Electron Transport Chain

6.4 SUMMARY

NAD and FAD remove the electrons that are donated during some of the steps of the Kreb's or Citric acid cycle. Then, they carry the electrons to the electron transport pumps and donate them to the pumps. So, NAD and FAD are "oxidized" because they lose the hydrogen ions to the pumps. The pumps then transport the hydrogens ions to the space between the two membranes where they accumulate in a high enough concentration to fuel the ATP pumps. With sufficient fuel, they "phosphorylate" the ADP. That is how oxidation is coupled to phosphorylation.

The hydrogen that gets pumped back into the matrix by the ATP pump then combine with the oxygen to make water. And that is very important because, without oxygen, they will accumulate and the concentration gradient needed to run the ATP pumps will not allow the pumps to work.

Electron transport and oxidative phosphorylation occur in the inner membrane of mitochondria. These processes reoxidize the NADH and FADH2 that are generated from Krebs cycle and glycolysis, and trap the energy released as ATP. Oxidative phosphorylation is by far the major source of ATP in the cell [11].

CHAPTER 7 - WEB BASED GRAPHICAL USER INTERFACE

This chapter is written in an attempt to explain the tools that visualize the results of DEVS simulation. CD++ allows the execution of DEVS models and to visualize them using a JAVA application.

7.1 VISUALIZING THE RESULTS GRAPHICALLY

The users can use the CD++ simulator either as a local application or a remote server. Once the simulation is done, the user can analyze the simulation results using different visualization tools. Visualization tools are now an integral part of the CD++ modeling and simulation toolkit [33].

The 2D visualization GUIs are used to visualize the results of atomic models, coupled DEVS models, and CELL-DEVS models. One of the visualization facilities enables users to analyze the input/output values transmitted from/into each ports of an atomic model by displaying these values on a graphical display. The information transmitted through each of them is collected in a result file during the simulation, storing all the messages sent between the DEVS components. The visualiztion routines extract all the messages related to the atomic models and their results, so that the user can select any of the atomic models for visualization.

7.2 CD++ MODELER

The CD Modeler is a GUI, which is used for creating atomic models and coupled models for the CD++ tool. It is a tool for designing and executing DEVS Models in an integrated environment. The basic function of the GUI includes create atomic model and coupled model, exporting the models to different formats and animating the simulations. The GUI also includes a simple text editor to modify Cell-DEVS models directly and the RUN options to execute de simulation with the cd++ and generate the draw information with the drawlog. The GUI is coded in Java, which enable it to run on various environments. The following sections explore the functions of the GUI with examples. This GUI is developed with JDK 1.4.1, so it is platform portable and can be run on various environments such as Eclipse, JBuilder with JDK1.4.1.

Once the GUI installed, DEVS models can be created. Before creating the models, the user should select the design space for corresponding atomic model or coupled model by clicking on the right tabs. Once the space selector is chosen, a title is chosen, uinits are added. Ports, links, and exportation come next.

The basic steps to create a coupled model are as same as creating an atomic model with the only difference that the coupled model needs to be created on the coupled model work place which can be selected by click on the “Coupled” tab above of the work place. The Coupled Models editor of CD++ Modeler application is composed by a central panel, 3 lateral panels and an upper button bar [33].

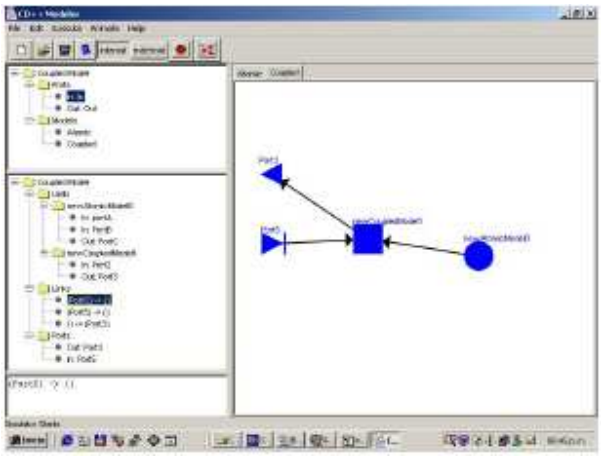

Figure 7.1 – CD Modeler Panel for a coupled Model

This Panel contains five sections:

Objects of the Coupled Model are on the *central panel*. Input Ports, Atomic Models, Coupled Models and Links between them can be added to this part. Objects to be added for the definition of this model can be seen on the *first lateral panel*. This part is divided into Input Ports, Output Ports, predefined Coupled Models, and predefined Atomic Models. Objects that form the Coupled Model that is being defined can be seen on the *middle lateral panel*. This part contains Units of the added Atomic or Coupled Models, Links between defined Models or Ports and Ports of the added Ports. On the third *lateral panel*, there is a description of the elements selected on the middle lateral panel. The *menu bar* with the following buttons: New, Open, Save, Help, Internal (Not used in Coupled Models), External (Not used in Coupled Models), Add new Atomic Model Unit, and Add new Coupled Model Unit (see figure 7.1).

To visualize the result files of Cell-Atomic models, Cell-DEVS models and Coupled models, the user must select the animation option from the main menu (see figure 7.2).

Figure 7.2 – Animation Menu for Visualization: From this pull-down menu, Atomic-DEVS model, Cell-DEVS model and Coupled-DEVS model can be specified [source: CD++, A tool for and DEVS and Cell-DEVS Modelling and Simulation User's Guide]

Every DEVS model includes at least one atomic model. Usually a coupled DEVS model includes many atomic models. After the simulation finishes, a log file will be generated. The log file records all the messages sent between DEVS components, that is, all the messages sent and received by all the atomic models have also been saved in the log file. Therefore, with this log file all the values in the messages sent and received by a specific atomic model can be extracted and visualized.

The visualization graph displays the model graph and the output value of all the output ports at the same time:

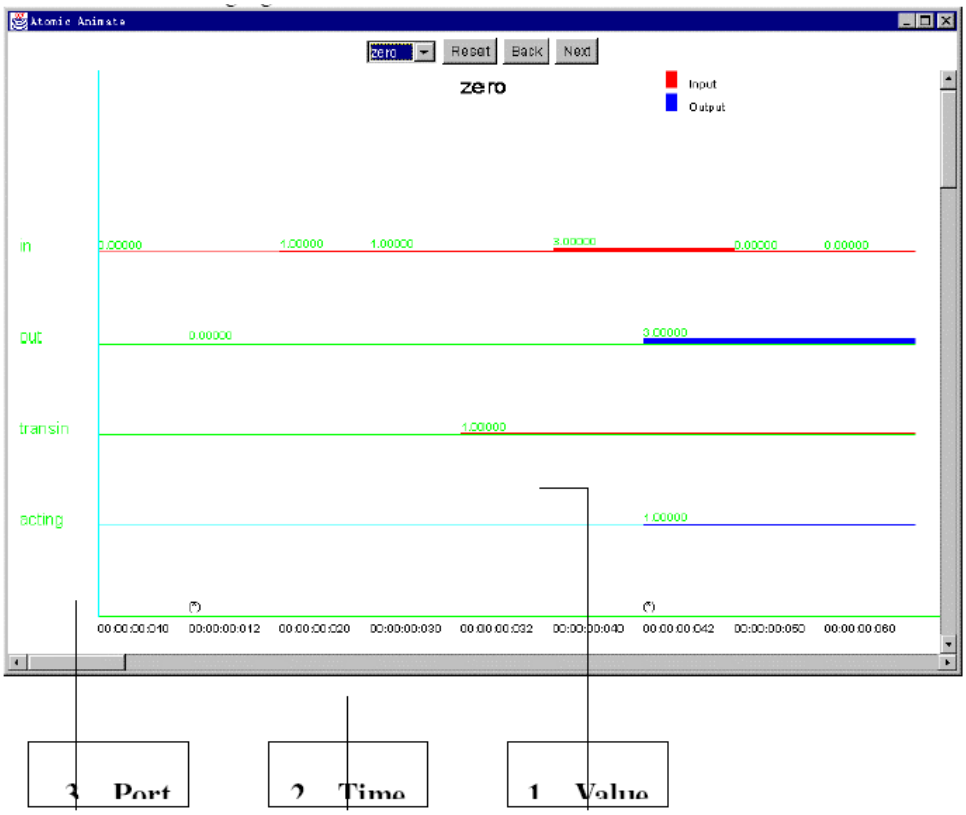

Figure 7.3 – The Graphic Display of the Output of Atomic Model Example [source: CD++, A tool for and DEVS and Cell-DEVS Modelling and Simulation User's Guide]

In the coupled-DEVS model, sometimes, it is useful to display the graph of the model, and the output values near the corresponding output port at the same time.

All the parameters are as follows: the coupled model (the graph file of the model), log file (the file used to record all the messages sent between DEVS components), and the delay between continuous displays (which can be used to control the speed of the visualization) [33].

The following two figures are illustrations of the choice components and the name of the visualized models (glycolysis and Krebs cycle), displayed by the buttons [33].

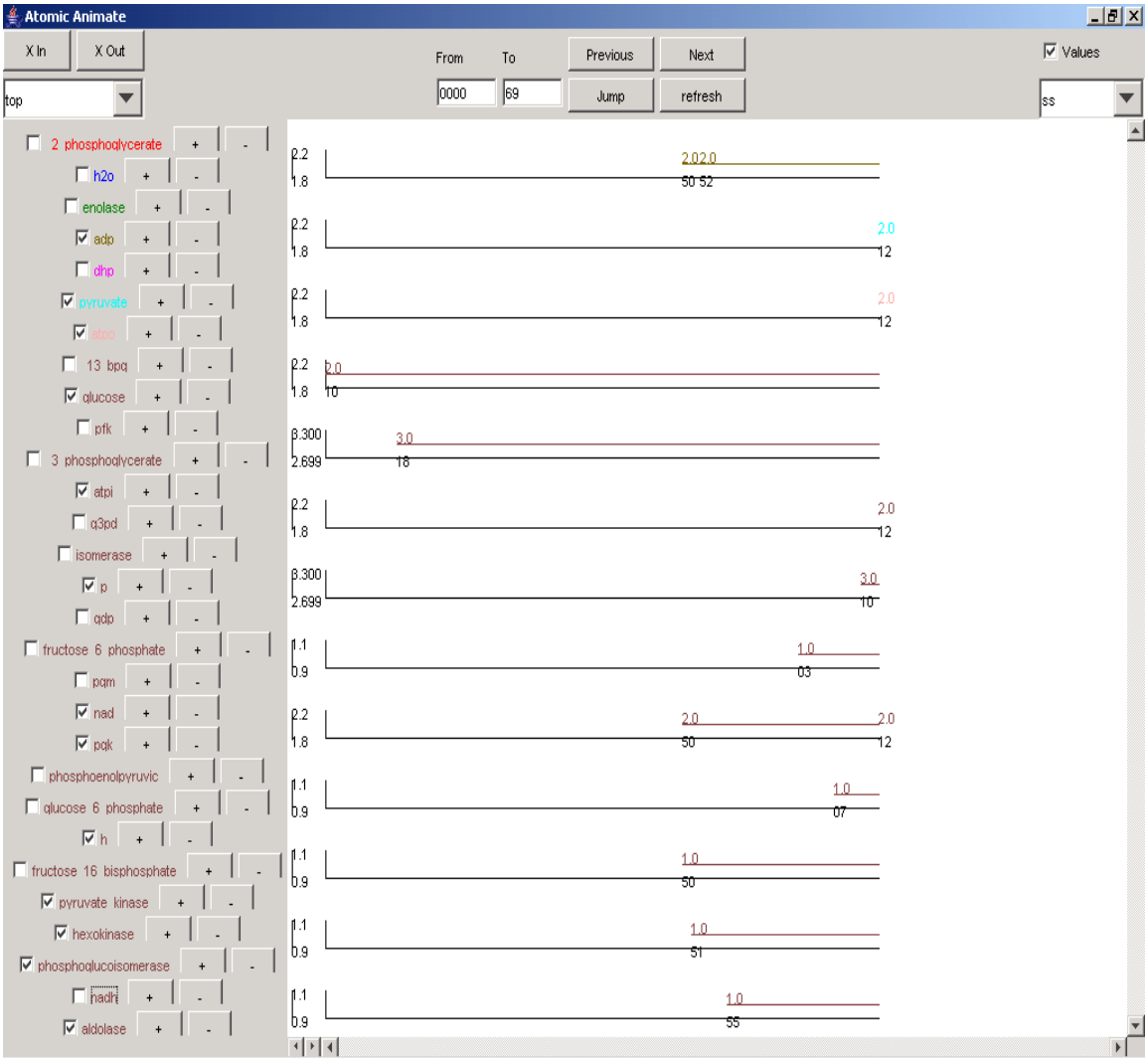

Figure 7.4 – Atomic Animation of Glycolysis

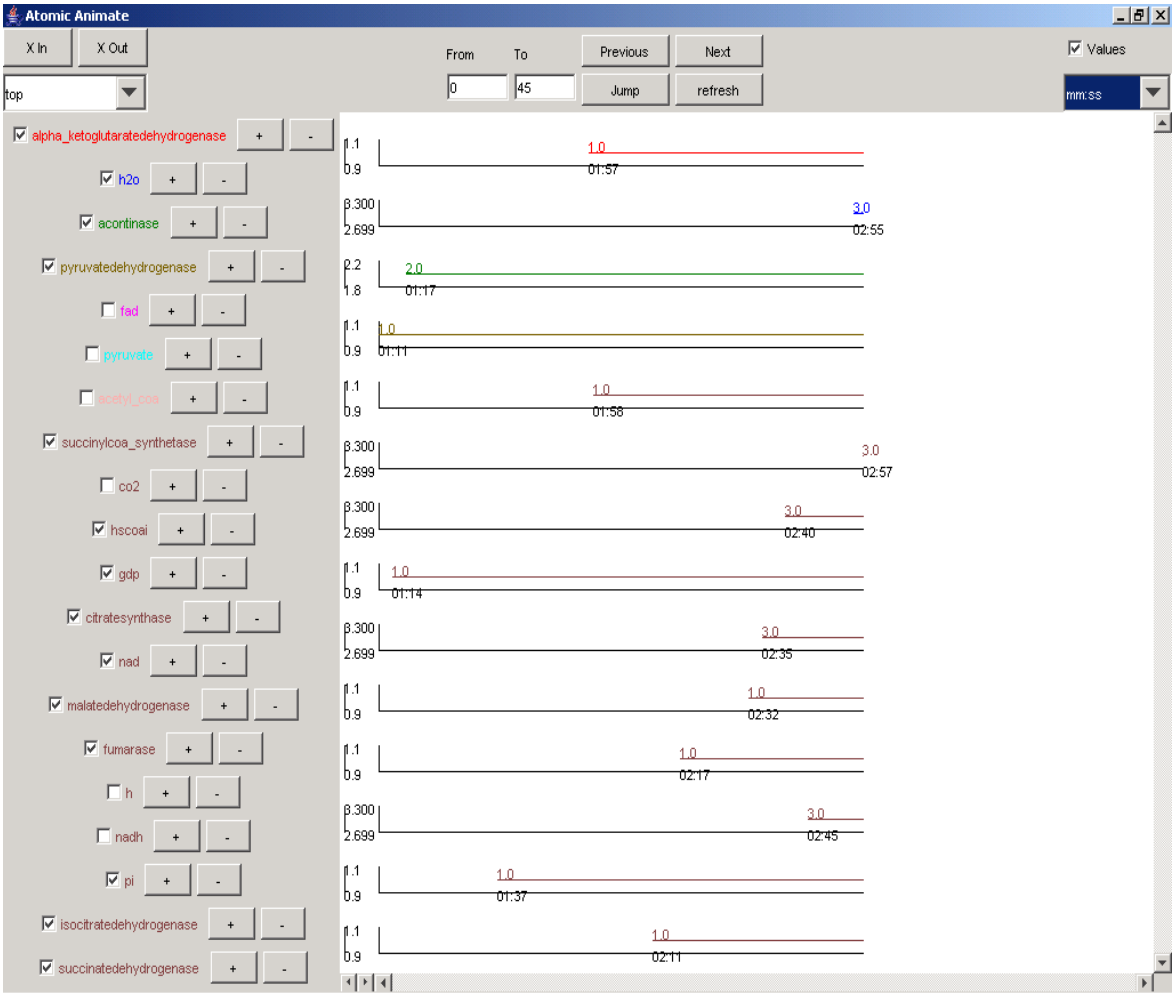

Figure 7.5 – Atomic Animation of Krebs Cycle

The users can select any of the models for visualization. The time-line lists all the port names on the left and the times on the botton. The value is shown as a piecewise constant signal whose height is related to the value displayed. Each signal starts when the input port receives, or the output port sends out that specific value, and ends when the model generates a new output. With this graphical display, the user can check all of the input/output values of an atomic model through the whole simulation process.

7.3 CD++ SIMULATION CLIENT

This tool is used for remote simulation, where the model and event files are sent to the simulation server. After the model is executed in the server, the results (log and output files) are sent back to the client machine and displayed in the window.

This GUI is a Client-Server approach with client graphical user interface:

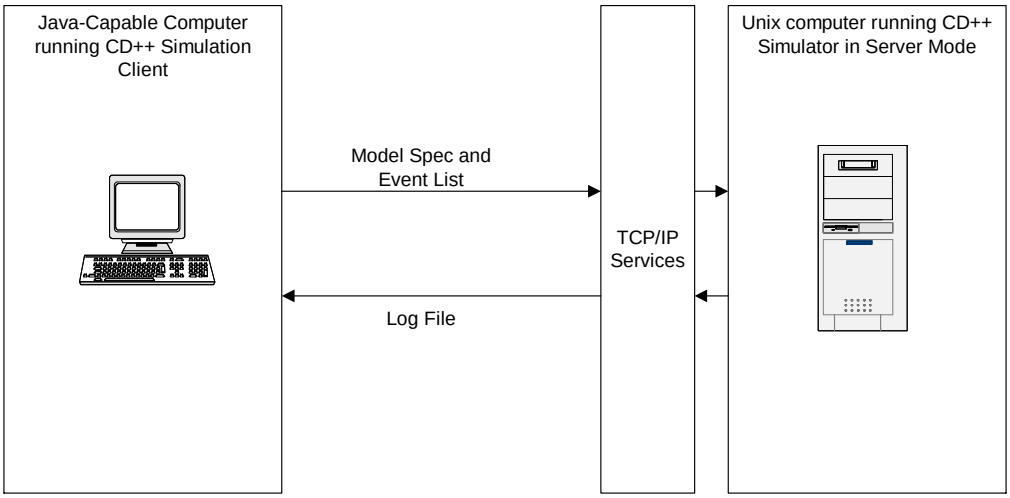

Figure 7.6 – Client-Server Approach [Source:CD++Simulation Client, Peter MacSween]

CD++ is sitting in server machine waiting for a model specification (.ma file) and and external event list (.ev file) from the client machine to simulate the model. Once those files received, simulator still binding to a TCP/IP port, will return the results (.log file) and (.out file) through that port to the client machine.

The GUI requirements for CD++ Simulation Client are as follow [33]:

- Model editor Panel (load, edit, and save tools for the model and input files)
- Simulation output panel
- Configuration dialog (server address and port)
- Menu and associated button bar (such as New File, Open, Close, Save, Cut, Copy, Edit, Simulate, and etc.)

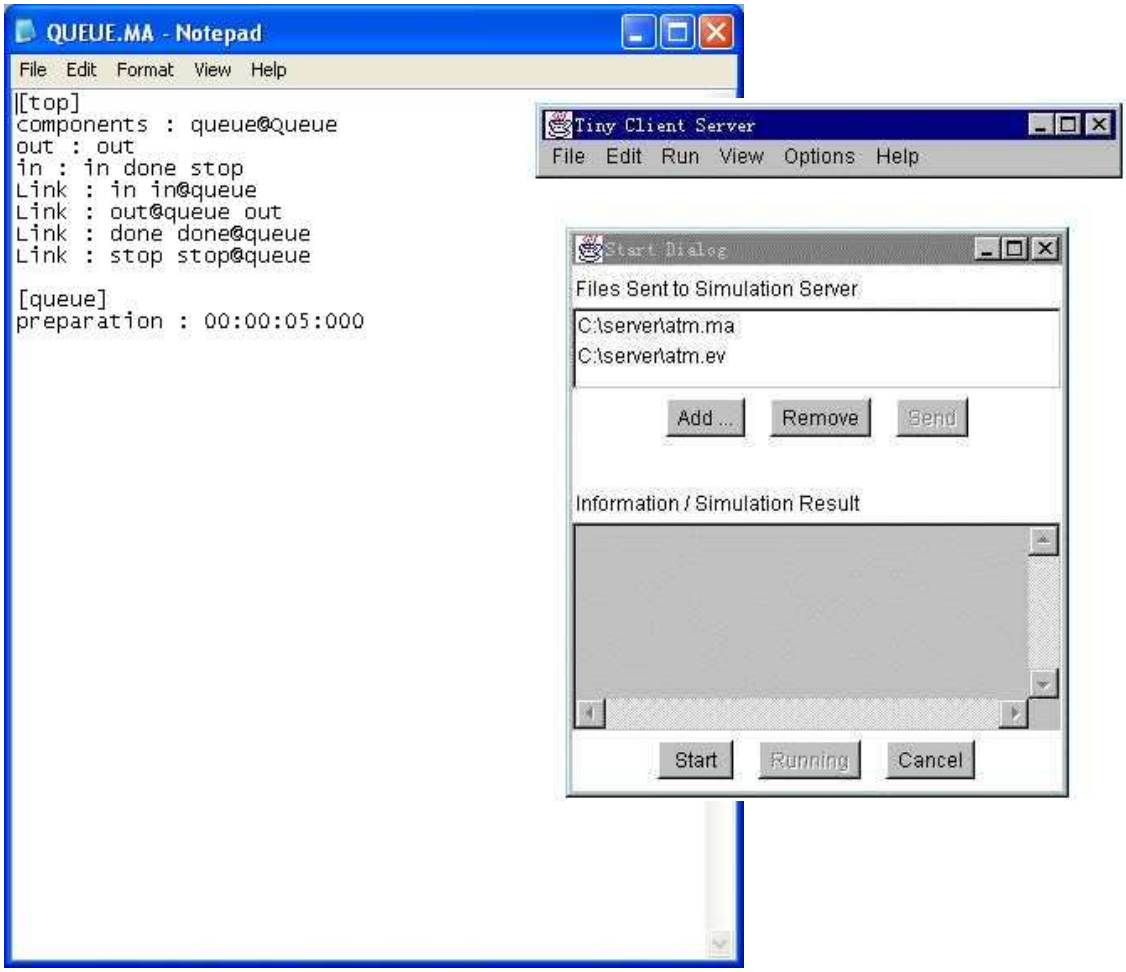

Figure 7.7 – CellDEVS Simulation Client, Implementation 1
[Source:CD++Simulation Client, Pete MacSween]

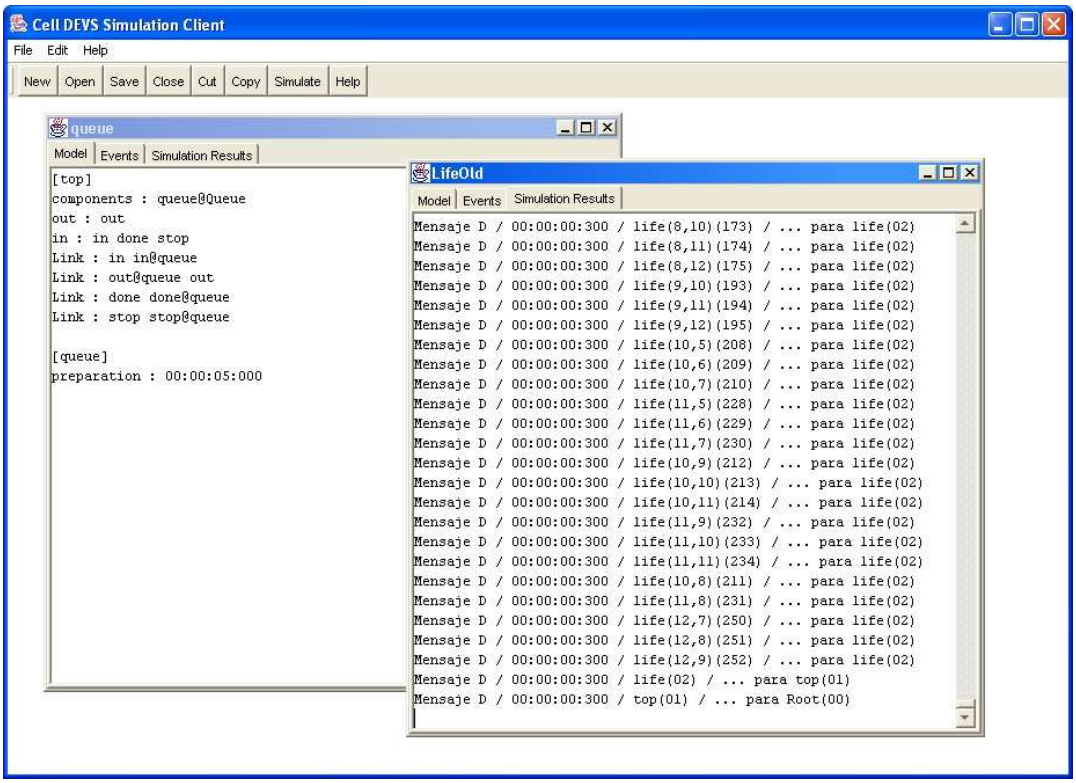

Figure 7.8 – CellDEVS Simulation Client, Implementation 2
[Source:CD++Simulation Client, Peter MacSween]

Client-Server approach is consistent with Model-Simulator where server tools are available. User Interface can always be improved to assist the modeller and the new users.

7.4 MAYA

MAYA is another application used as GUI here. The log files generated from simulation models can be fed into this application, and Maya can translate these models into 3D representations, and further animate them.

The graphics and video for the glycolysis model were created using this software application. Maya is one of the world's leading software applications for 3D digital animation and visual effects. It provides a suite of tools for 3D world creation. These tools include modeling, animation, rendering and dynamics etc. One can create convincing visual simulations of rigid and soft body objects interacting in the physical world and the rendering tool can allow animations to achieve photo realistic imagery.

The Maya software interface is fully customizable and it allows users to extend their functionality within Maya by providing access to Maya Embedded Language (MEL). With MEL, one can tailor the user interface and write scripts and macros to increase Maya functionality for ones particular purposes. A full Application Programmers Interface (API) is available to further enhance the power and possibilities for using Maya to translate Cell-DEVS and DEVS models into 3D representations.

Note that this work was done by a Coop student [Ayesha Khan], by taking the glycolysis model and feeding it to Maya software applicaiton.

The following figures are some snapshots taken from the animation of glycolysis log fed to Maya application:

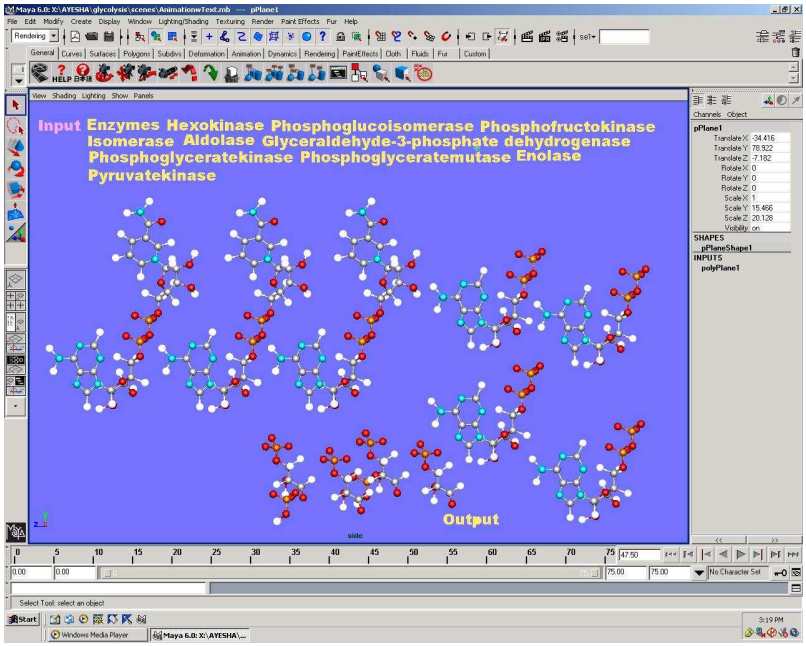

Figure 7.9 – Glycolysis.log being fed to MAYA software

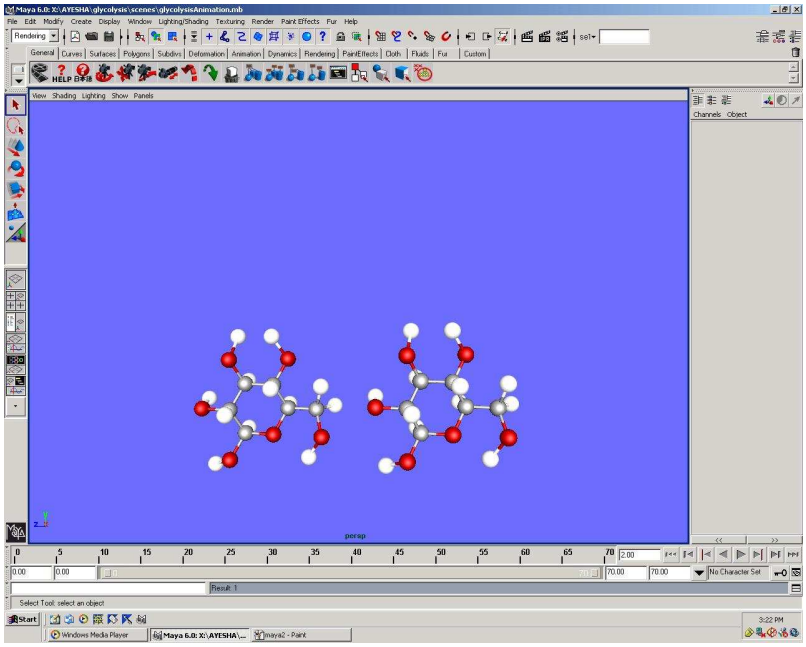

Figure 7.10 – A close up snapshot of Glycolysis presented with MAYA software

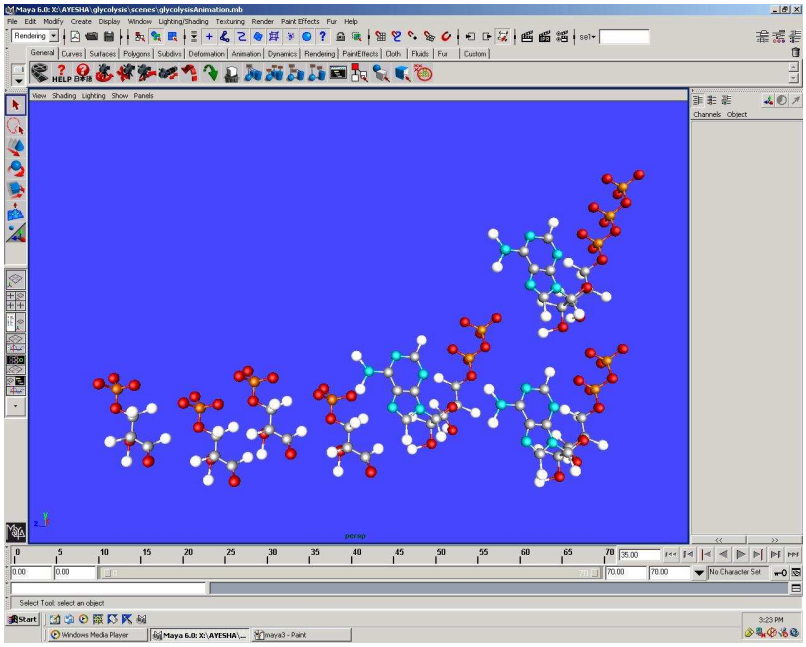

Figure 7.11 – Another snapshot of Glycolysis presentation with MAYA application

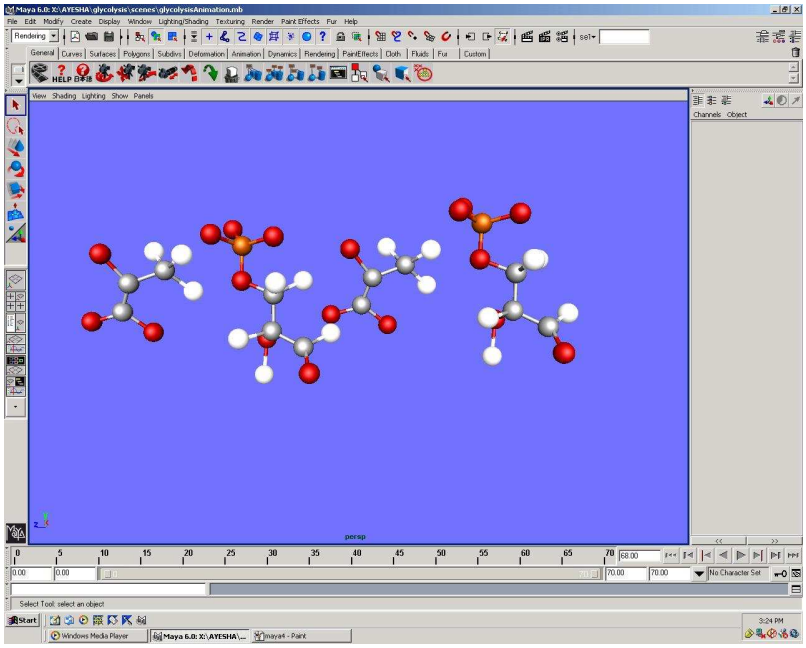

Figure 7.12 – A close up snapshot taken from animation result of glycolysis generated by the MAYA software

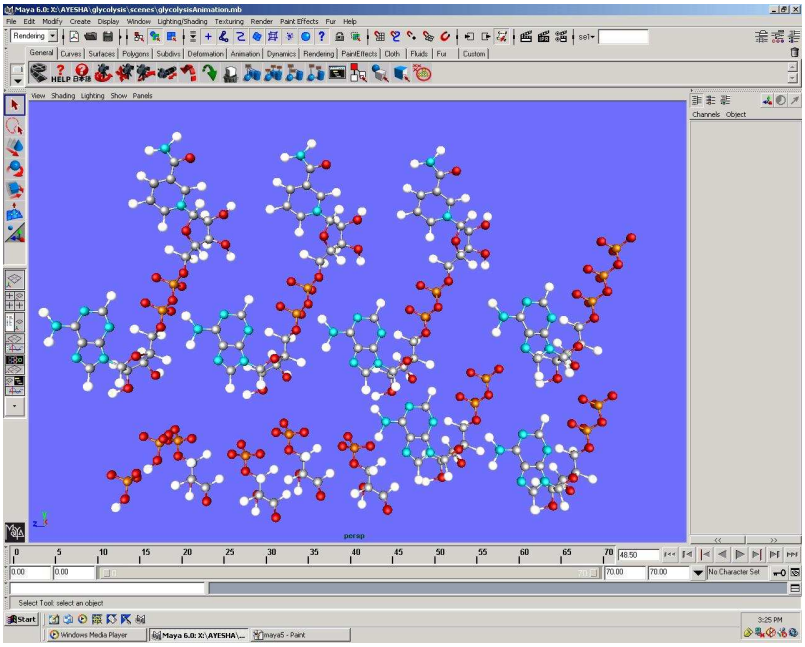

Figure 7.13 – A snapshot taken from animation result of glycolysis generated by the MAYA software

7.5 SUMMARY

Visualization is an important aspect of modeling and simulation. Different Graphical User Interfaces (GUIs) are available for CD++ tool were discussed earlier. Simulation results were used to construct useful 2D and 3D images, and animations via MAYA. To facilitate the users to use the CD++ simulator, CD++ simulation client was also introduced where it provides a number of services using a client/server approach. This tool provides a series of tools, including the CD++ modeler, 2D visualization tools, and an interface for remote simulation model execution. All these tools provide enough functions for a remote simulation environment in CD++ simulation.

CHAPTER 8 - CONCLUSION

During the last several decades, computer simulation has proved to be a powerful tool in basic and applied biological sciences, and has become an integral part of biological research in both basic and applied fields. High-performance computers are now more than ever essential research instruments in modern biomedicine. Advances in computer technology, along with the explosive growth in the size of biological data, keep increasing the role of computers in biomedical research even further. This thesis has shown a glimpse of that marriage – the use of computer simulation and basic biological field.

As illustrated in this thesis, the use of CD++ tool is presented to model and simulate biological pathways – glycolysis and Krebs's cycle - in an attempt to show how to apply a development technique to build a more complex simulation model – *mitochondria* - using a systematic method in which a model consists of a set of lower-level interactions. These models are implemented using the DEVS formalism. 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 CD++ reduces the algorithmic complexity for the modeler while allowing complex cellular timing behaviours.

The approach relies on the use of DEVS methodology and it is supported by the use of CD++, a DEVS tool that has been built following the formal definitions of DEVS models. Sharing and interoperability of model implementations, focusing in different model examples in the area of medicine, means of developing independent models that can be integrated at the level of DEVS interactions, and how CD++ models can be composed into simulations that can execute in distributed environments.

The DEVS formalism for modeling and simulation provides a framework for the construction of hierarchical models in a modular fashion, allowing model reuse, reducing development and testing time. The hierarchical and discrete event nature of DEVS makes it a good choice to achieve an efficient product development test. DEVS are timed models, which also enables us to define timing properties for the models under development. Each DEVS model can be built as a behavioural (atomic) or Structural (coupled) model.

The CD++ tool enables the description of discrete event models based on the DEVS formalism. The hierarchical nature of DEVS permitted to do this without modifying the original models, providing the base for enhanced system development in embedded platforms.

Here, a development technique, to build complex simulation evolving incrementally from simple subcomponents to complex simulations has been shown. The method relies on the use of DEVS, which provides an incremental, hierarchical view.

This view enables the reuse of simulations and components, where the integration of simulations and components is seamless.

This thesis showed how to develop several DEVS models using the CD++ toolkit, which provides a general framework to define and simulate complex generic models such as biomedical applications. This has been accomplished and introduced in this thesis in Chapter 5 and Chapter 6. The use of DEVS can improve the security and cost in the development of the simulations. The main gains are in the testing and maintenance phases, the more expensive for these systems. The use of a formal approach like DEVS made easy the development of the applications.

8.1 SUGGESTIONS FOR FURTHER RESEARCH

The future development could be done towards making large systems and exploring different and more complex biological systems to further enhance and formalizing real-time simulation using CD++.

As one can notice, the electon transport chain section has not been completely addressed by this thesis because of the time constrains, and it has only been introduced partially in Chapter 6 of this thesis where it can easily be developed to a Ph.D. thesis. Once this section complete, one can work her way up towards a complete organelle – Mitochondria – and then towards a complete cell.

Visualization is another important aspect of modeling and simulation. Chapter 6 explains different Graphical User Interfaces (GUIs) available for CD++ tool. Simulation

results were used to construct useful 2D and 3D images, and animations via MAYA. To facilitate the users to use the CD++ simulator, CD++ simulation client was also introduced where it provides a number of services using a client/server approach. This tool provides a series of tools, including the CD++ modeler, 2D visualization tools, and an interface for remote simulation model execution. All these tools provide enough functions for a remote simulation environment in CD++ simulation. Although a partially unresolved issue is how to best embed this tool into a web browser and make it available over the Internet to all the medical staff to be used in real time based on their needs. There has to be a way also to teach the Medical staff to develop their own simulation tools easily and/or modify whenever there is a new discovery or need to test a new drug or component. It is known that merely providing a tool for inputting values and getting the results won't be sufficient in the far future as everything is changing in a fast pace in nowadays competitive edge.

Other issues not addressed by this thesis involve a more in-depth analysis of this tool by real medical staff to see the real results and get the feedback for future developments.

APPENDIX A-CODE FOR GLYCOLYSIS

This appendix contains the code for glycolysis section.

A.1 ATOMIC MODELS

A.1.1 STEP1.H

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step1 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 21/10/2003
*  Modified: 12/11/2003
*
*******************************************************************/

#ifndef __STEP1_H
#define __STEP1_H

#include <list>
#include "atomic.h"     // class Atomic

class Step1 : public Atomic
{
public:
        Step1( const string &name = "Step1" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
```

```
        const Port &glucose;
        const Port &ATPi;
        const Port &hexokinase;
    // outputs
        Port &glucose_6_phosphate;
        Port &ADP;
        Port &H;

        Time preparationTime;
        double atpc;
        double glucosec;
        bool ifhex;
    double counter;

};      // class Step1

// ** inline ** //
inline
string Step1::className() const
{
        return "Step1" ;
}

#endif  //__STEP1_H
```

A.1.2 STEP1.CPP

```
/*****************************************************************
*
*  DESCRIPTION: Atomic Model Step1 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 21/10/2003
*  Modified: 24/11/2003
*  Modified: 08/12/2003
*
*****************************************************************/

/** include files **/
#include "step1.h"      // class step1
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*****************************************************************
* Function Name: Step1
* Description:
*****************************************************************/
```

```
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 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &Step1::initFunction()
{
        atpc = 0;
        glucosec = 0;
        ifhex = false;
        counter = 0;

        return *this ;
}

/*******************************************************************
* Function Name: externalFunction
* Description:
*******************************************************************/
Model &Step1::externalFunction( const ExternalMessage &msg )
{

        if( msg.port() == glucose )
        {
                glucosec = glucosec + msg.value() ;

                if ( (atpc > 0 ) && (ifhex == true ) )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }


        else if( msg.port() == ATPi )
        {
                atpc = atpc + msg.value() ;

                if ( (glucosec > 0 ) && (ifhex == true ) )
                        holdIn( active, Time::Zero );
```

```
                else
                        holdIn(passive, Time::Zero );
        }


        else if ( msg.port() == hexokinase )
        {
                ifhex = true ;

                if ( (glucosec > 0 ) && (atpc > 0) )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }

        return *this;
}

/*********************************************************************
* Function Name: internalFunction
* Description:
*********************************************************************/
Model &Step1::internalFunction( const InternalMessage & )
{
   counter = 0;
        if  ( state() == passive )
        {
                passivate();
        }
        else
        {

                if ( (atpc >= 1) && (glucosec >= 1 ) && (ifhex == 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 = 0;
                                glucosec = 0;
                        }
```

```
                holdIn(passive, Time::Zero );
        }

        else
        {
                passivate();
        }
    }

    return *this ;

}

/*******************************************************************
* Function Name: outputFunction
* Description:
*******************************************************************/
Model &Step1::outputFunction( const InternalMessage &msg )
{

    if ( counter != 0 )
    {
        sendOutput( msg.time(), ADP, counter );
  sendOutput( msg.time(), glucose_6_phosphate, counter );
        sendOutput( msg.time(), H, counter );
    }

    return *this ;
}
```

A.1.3 STEP2.H

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step2 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*                 mailto://roxana.djafarzadeh@worldheart.com
*
*  DATE: 21/10/2003
*  Modified: 13/11/2003
*
*******************************************************************/

#ifndef __STEP2_H
#define __STEP2_H

#include <list>
#include "atomic.h"     // class Atomic
```

```
class Step2 : public Atomic
{
public:
        Step2( const string &name = "Step2" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &glucose_6_phosphate;
        const Port &phosphoglucoisomerase;
   // outputs
        Port &fructose_6_phosphate;

        Time preparationTime;
        double g6pc;
        bool ifpgisomerase;
        double counter;


};      // class Step2

// ** inline ** //
inline
string Step2::className() const
{
        return "Step2" ;
}

#endif  //__STEP2_H
```

A.1.4 STEP2.CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step2 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE: 21/10/2003
*  Modified: 13/11/2003
*
*******************************************************************/
```

```
/** include files **/
#include "step2.h"      // class step2
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: Step2
* Description:
*******************************************************************/
Step2::Step2( const string &name )
: Atomic( name )
, glucose_6_phosphate( addInputPort( "glucose_6_phosphate" ) )
, phosphoglucoisomerase( addInputPort( "phosphoglucoisomerase" ) )
, fructose_6_phosphate( addOutputPort( "fructose_6_phosphate" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &Step2::initFunction()
{
        g6pc = 0;
        ifpgisomerase = false;
   counter = 0;

        return *this ;
}

/*******************************************************************
* Function Name: externalFunction
* Description:
*******************************************************************/
Model &Step2::externalFunction( const ExternalMessage &msg )
{
        if( msg.port() == glucose_6_phosphate )
        {
                g6pc = g6pc + msg.value() ;
                if ( ifpgisomerase == true )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }

        else if ( msg.port() == phosphoglucoisomerase )
```

```
	{
		ifpgisomerase = true ;

		if (g6pc > 0 )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	return *this;
}

/*******************************************************************
* Function Name: internalFunction
* Description:
*******************************************************************/
Model &Step2::internalFunction( const InternalMessage & )
{

	counter = 0;
	if  ( state() == passive )
	{
		passivate();
	}
	else
	{
		if ( (g6pc >= 1) && (ifpgisomerase == true ) )
		{
			counter = g6pc;
			g6pc = 0;

			holdIn(passive, Time::Zero );
		}

		else
		{
			passivate();
		}
	}

	return *this ;

}

/*******************************************************************
* Function Name: outputFunction
* Description:
*******************************************************************/
Model &Step2::outputFunction( const InternalMessage &msg )
{

	if ( counter != 0 )
	{
		sendOutput( msg.time(), fructose_6_phosphate, counter ) ;
	}
```

```
        return *this ;
}
```

A.1.5 STEP3.H

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step3 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 21/10/2003
*  modified: 13/11/2003
*  Modified: 08/12/2003
*
*******************************************************************/

#ifndef __STEP3_H
#define __STEP3_H

#include <list>
#include "atomic.h"     // class Atomic

class Step3 : public Atomic
{
public:
        Step3( const string &name = "Step3" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &fructose_6_phosphate;
        const Port &ATPi;
        const Port &PFK;
   // outputs
        Port &fructose_16_bisphosphate;
        Port &ADP;

        Time preparationTime;

        double f6pc;
        double atpc;
```

```
        bool   ifpfk;

        double counter;

};      // class Step3

// ** inline ** //
inline
string Step3::className() const
{
        return "Step3" ;
}

#endif  //__STEP3_H
```

A.1.6 STEP3.CPP

```
/*********************************************************************
*
*  DESCRIPTION: Atomic Model Step3 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 21/10/2003
*  Modified: 9/11/2003
*  Modifiel: 13/11/2003
*  Modified: 08/12/2003
*
*********************************************************************/

/** include files **/
#include "step3.h"      // class step3
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*********************************************************************
* Function Name: Step3
* Description:
*********************************************************************/
Step3::Step3( const string &name )
: Atomic( name )
, fructose_6_phosphate( addInputPort( "fructose_6_phosphate" ) )
, ATPi( addInputPort( "ATPi" ) )
, PFK( addInputPort( "PFK" ) )
, fructose_16_bisphosphate( addOutputPort( "fructose_16_bisphosphate" ) )
, ADP( addOutputPort( "ADP" ) )
, preparationTime( 0, 0, 10, 0 )
```

```
{
	string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

	if( time != "" )
		preparationTime = time ;
}

/*****************************************************************
* Function Name: initFunction
* Description: Resetea la lista
* Precondition: El tiempo del proximo evento interno es Infinito
*****************************************************************/
Model &Step3::initFunction()
{
	atpc = 0;
	f6pc = 0;
	ifpfk = false;
	counter = 0;

	return *this ;
}

/*****************************************************************
* Function Name: externalFunction
* Description:
*****************************************************************/
Model &Step3::externalFunction( const ExternalMessage &msg )
{

	if( msg.port() == fructose_6_phosphate )
	{
		f6pc = f6pc + msg.value() ;
		if ( (atpc > 0 ) && (ifpfk == true ) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	else if( msg.port() == ATPi )
	{
		atpc = atpc + msg.value() ;
		if ( (f6pc > 0 ) && (ifpfk == true ) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	else if ( msg.port() == PFK )
	{
		ifpfk = true ;

		if ( (f6pc > 0 ) && (atpc > 0) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
```

```
	}

	return *this;
}

/*********************************************************************
* Function Name: internalFunction
* Description:
*********************************************************************/
Model &Step3::internalFunction( const InternalMessage & )
{

	counter = 0;

	if  ( state() == passive )
	{
		passivate();
	}

	else
	{
		if ( (atpc >= 1) && (f6pc >= 1 ) && (ifpfk == true ) )
		{
			if (atpc > f6pc)
			{
				atpc = atpc - f6pc;
				counter = f6pc;
				f6pc = 0;
			}

			else if (atpc < f6pc)
			{
				f6pc = f6pc - atpc;
				counter = atpc;
				atpc = 0;
			}

			else if (atpc == f6pc)
			{
				counter = atpc;
				atpc = 0;
				f6pc = 0;

			}

			holdIn(passive, Time::Zero );
		}

		else
		{
			passivate();
		}
	}

	return *this ;
```

```
}
/*******************************************************************
* Function Name: outputFunction
* Description:
*******************************************************************/
Model &Step3::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
                sendOutput( msg.time(), ADP, counter ) ;
      sendOutput( msg.time(), fructose_16_bisphosphate, counter ) ;
        }
        return *this ;
}
```

A.1.7 STEP4.H

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step4 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*                  mailto://roxana.djafarzadeh@worldheart.com
*
*  DATE: 22/10/2003
*
*******************************************************************/

#ifndef __STEP4_H
#define __STEP4_H

#include <list>
#include "atomic.h"     // class Atomic

class Step4 : public Atomic
{
public:
        Step4( const string &name = "Step4" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
```

```
        const Port &fructose_16_bisphosphate;
        const Port &aldolase;
    // outputs
        Port &DHP;

        Time preparationTime;

        double f16pc;
        bool ifaldolase;

        double counter;


};      // class Step4

// ** inline ** //
inline
string Step4::className() const
{
        return "Step4" ;
}

#endif  //__STEP4_H
```

A.1.8 STEP4.CPP

```
/*************************************************************************
*
*  DESCRIPTION: Atomic Model Step4 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE: 22/10/2003
*  Modified: 13/11/2003
*
*************************************************************************/

/** include files **/
#include "step4.h"      // class step1
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*************************************************************************
* Function Name: Step4
* Description:
*************************************************************************/
Step4::Step4( const string &name )
: Atomic( name )
, fructose_16_bisphosphate( addInputPort( "fructose_16_bisphosphate" ) )
```

```
, aldolase( addInputPort( "aldolase" ) )
, DHP( addOutputPort( "DHP" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*********************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*********************************************************************/
Model &Step4::initFunction()
{
        f16pc = 0;
        ifaldolase = false;
   counter = 0;

        return *this ;
}

/*********************************************************************
* Function Name: externalFunction
* Description:
*********************************************************************/
Model &Step4::externalFunction( const ExternalMessage &msg )
{

        if( msg.port() == fructose_16_bisphosphate )
        {
                f16pc = f16pc + msg.value();

                if ( ifaldolase == true )
                        holdIn(active, Time::Zero);
                else
                        holdIn(passive, Time::Zero);
        }

        else if ( msg.port() == aldolase )
        {
                ifaldolase = true ;

                if ( f16pc > 0 )
                        holdIn(active, Time::Zero);
                else
                        holdIn(passive, Time::Zero);
        }

        return *this;
}

/*********************************************************************
```

```
* Function Name: internalFunction
* Description:
*******************************************************************/
Model &Step4::internalFunction( const InternalMessage & )
{
        counter = 0;

        if  ( state() == passive )
        {
                passivate();
        }

        else
        {
                if ( (f16pc >= 1) && (ifaldolase == true ) )
                {
                        counter = f16pc;
                        f16pc = 0;

                        holdIn(passive, Time::Zero );
                }
                else
                {
                        passivate();
                }
        }

        return *this;
}

/*******************************************************************
* Function Name: outputFunction
* Description:
*******************************************************************/
Model &Step4::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
                sendOutput( msg.time(), DHP, counter );
        }

        return *this;

}
```

A.1.9 STEP4TO5.H

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step4to5 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
```

```
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE: 18/02/2004
*
**********************************************************************/

#ifndef __STEP4to5_H
#define __STEP4to5_H

#include <list>
#include "atomic.h"     // class Atomic

class Step4to5 : public Atomic
{
public:
        Step4to5( const string &name = "Step4to5" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &DHP;
        const Port &isomerase;
   // outputs
        Port &GDP;

        Time preparationTime;

        double dhpc;
        bool ifisomerase;

        double counter;


};      // class Step4to5

// ** inline ** //
inline
string Step4to5::className() const
{
        return "Step4to5" ;
}

#endif  //__STEP4to5_H
```

A.1.10 STEP4TO5.CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step4to5 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE: 18/02/2004
*
*******************************************************************/

/** include files **/
#include "step4to5.h"      // class step4to5
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: Step4to5
* Description:
*******************************************************************/
Step4to5::Step4to5( const string &name )
: Atomic( name )
, DHP( addInputPort( "DHP" ) )
, isomerase( addInputPort( "isomerase" ) )
, GDP( addOutputPort( "GDP" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &Step4to5::initFunction()
{
        dhpc = 0;
        ifisomerase = false;
        counter = 0;

        return *this ;
}
```

```
/*******************************************************************
* Function Name: externalFunction
* Description:
*******************************************************************/
Model &Step4to5::externalFunction( const ExternalMessage &msg )
{

        if( msg.port() == DHP )
        {
                dhpc = dhpc + msg.value();

                if ( ifisomerase == true )
                        holdIn(active, Time::Zero);
                else
                        holdIn(passive, Time::Zero);
        }

        else if ( msg.port() == isomerase )
        {
                ifisomerase = true ;

                if ( dhpc > 0 )
                        holdIn(active, Time::Zero);
                else
                        holdIn(passive, Time::Zero);
        }

        return *this;
}

/*******************************************************************
* Function Name: internalFunction
* Description:
*******************************************************************/
Model &Step4to5::internalFunction( const InternalMessage & )
{
        counter = 0;

        if  ( state() == passive )
        {
                passivate();
        }

        else
        {
                if ( (dhpc >= 1) && (ifisomerase == true ) )
                {
                        counter = dhpc;
                        dhpc = 0;

                        holdIn(passive, Time::Zero );
                }
                else
                {
                        passivate();
```

```
                }
        }

        return *this;
}

/*******************************************************************
* Function Name: outputFunction
* Description:
*******************************************************************/
Model &Step4to5::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
                sendOutput( msg.time(), GDP, counter );
        }

        return *this;

}
```

A.1.11 STEP5.H

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step5 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*                       mailto://roxana.djafarzadeh@worldheart.com
*
*  DATE: 22/10/2003
*
*******************************************************************/

#ifndef __STEP5_H
#define __STEP5_H

#include <list>
#include "atomic.h"     // class Atomic

class Step5 : public Atomic
{
public:
        Step5( const string &name = "Step5" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
```

```
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &fructose_16_bisphosphate;
        const Port &aldolase;
   // outputs
        Port &GDP;

        Time preparationTime;

        double f16pc;
        bool ifaldolase;
        double counter;


};      // class Step5

// ** inline ** //
inline
string Step5::className() const
{
        return "Step5" ;
}

#endif  //__STEP5_H
```

A.1.12 STEP5.CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step5 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE: 22/10/2003
*  Modified: 15/11/2003
*
*******************************************************************/

/** include files **/
#include "step5.h"      // class step5
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: Step5
* Description:
```

```
*****************************************************************/
Step5::Step5( const string &name )
: Atomic( name )
, fructose_16_bisphosphate( addInputPort( "fructose_16_bisphosphate" ) )
, aldolase( addInputPort( "aldolase" ) )
, GDP( addOutputPort( "GDP" ) )
, preparationTime( 0, 0, 10, 0 )
{
	string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

	if( time != "" )
		preparationTime = time ;
}

/*****************************************************************
* Function Name: initFunction
* Description: Resetea la lista
* Precondition: El tiempo del proximo evento interno es Infinito
*****************************************************************/
Model &Step5::initFunction()
{

	f16pc = 0;
	ifaldolase = false;
	counter = 0;

	return *this ;
}

/*****************************************************************
* Function Name: externalFunction
* Description:
*****************************************************************/
Model &Step5::externalFunction( const ExternalMessage &msg )
{
	if( msg.port() == fructose_16_bisphosphate )
	{
		f16pc = f16pc + msg.value() ;
		if ( ifaldolase == true )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );

	}

	else if ( msg.port() == aldolase )
	{
		ifaldolase = true ;

		if (f16pc > 0 )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}
```

```
	return *this;
}

/*********************************************************************
* Function Name: internalFunction
* Description:
*********************************************************************/
Model &Step5::internalFunction( const InternalMessage & )
{
	counter = 0;

	if  ( state() == passive )
	{
		passivate();
	}

	else
	{
		if ( (f16pc >= 1) && (ifaldolase == true ) )
		{
			counter = f16pc;
			f16pc = 0;

			holdIn(passive, Time::Zero );
		}

		else
		{
			passivate();
		}
	}

	return *this ;
}

/*********************************************************************
* Function Name: outputFunction
* Description:
*********************************************************************/
Model &Step5::outputFunction( const InternalMessage &msg )
{

	if ( counter != 0 )
	{
		sendOutput( msg.time(), GDP, counter );
	}

	return *this ;
}
```

A.1.13 STEP6.H

```
/*********************************************************************
```

```
*
*  DESCRIPTION: Atomic Model Step6 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE: 27/10/2003
*  Modified: 15/11/2003
*
*******************************************************************/

#ifndef __STEP6_H
#define __STEP6_H

#include <list>
#include "atomic.h"     // class Atomic

class Step6 : public Atomic
{
public:
        Step6( const string &name = "Step6" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &GDP;
        const Port &NAD;
        const Port &P;
        const Port &G3PD;
   // outputs
        Port &_13_BPG;
        Port &NADH;
        Port &H;

        Time preparationTime;
        double gdpc;
        double nadc;
        double pc;
        bool ifg3pd;

        double counter;

};      // class Step6

// ** inline ** //
inline
string Step6::className() const
```

```
{
        return "Step6" ;
}

#endif  //__STEP6_H
```

A.1.14 STEP6.CPP

```
/*************************************************************************
*
*  DESCRIPTION: Atomic Model Step6 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE: 27/10/2003
*  Modified: 15/11/2003
*
*************************************************************************/

/** include files **/
#include "step6.h"      // class step6
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*************************************************************************
* Function Name: Step6
* Description:
*************************************************************************/
Step6::Step6( const string &name )
: Atomic( name )
, GDP( addInputPort( "GDP" ) )
, NAD( addInputPort( "NAD" ) )
, P( addInputPort( "P" ) )
, G3PD( addInputPort( "G3PD" ) )
, _13_BPG( addOutputPort( "_13_BPG" ) )
, NADH( addOutputPort( "NADH" ) )
, H( addOutputPort( "H" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*************************************************************************
* Function Name: initFunction
* Description: Resetea la lista
```

```
* Precondition: El tiempo del proximo evento interno es Infinito
*******************************************************************/
Model &Step6::initFunction()
{
	gdpc = 0;
	nadc = 0;
	pc = 0;
	ifg3pd = false;

	counter = 0;

	return *this ;
}

/*******************************************************************
* Function Name: externalFunction
* Description:
*******************************************************************/
Model &Step6::externalFunction( const ExternalMessage &msg )
{
	if( msg.port() == GDP )
	{
		gdpc = gdpc + msg.value() ;
		if ( (nadc > 0 ) && (pc > 0 ) && (ifg3pd == true ) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );

	}

	else if( msg.port() == NAD )
	{
		nadc = nadc + msg.value() ;
		if ( (gdpc > 0 ) && (pc > 0 ) && (ifg3pd == true ) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	else if( msg.port() == P )
	{
		pc = pc + msg.value() ;
		if ( (gdpc > 0 ) && (nadc > 0 ) && (ifg3pd == true ) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	else if ( msg.port() == G3PD )
	{
		ifg3pd = true ;

		if ( (gdpc > 0 ) && (nadc > 0 ) && (pc > 0 ) )
			holdIn( active, Time::Zero );
		else
```

```
                holdIn(passive, Time::Zero );
        }

        return *this;
}

/*********************************************************************
* Function Name: internalFunction
* Description:
*********************************************************************/
Model &Step6::internalFunction( const InternalMessage & )
{

        counter = 0;

        if  ( state() == passive )
        {
                passivate();
        }
        else
        {

                if ( (gdpc >= 1) && (nadc >= 1 ) && (pc >= 1 ) && (ifg3pd == true ) )
                {

                        if ( (gdpc>=nadc) && (nadc>=pc) )
                        {
                                gdpc = gdpc - pc;
                                nadc = nadc - pc;
                                counter = pc;
                                pc = 0;
                        }

                        else if ( (gdpc>=pc) && (pc>=nadc) )
                        {
                                gdpc = gdpc - nadc;
                                pc = pc - nadc;
                                counter = nadc;
                                nadc = 0;
                        }

                        else if ( (nadc>=gdpc) && (gdpc>=pc) )
                        {
                                gdpc = gdpc - pc;
                                nadc = nadc - pc;
                                counter = pc;
                                pc = 0;
                        }

                        else if ( (nadc>=pc) && (pc>=gdpc) )
                        {
                                nadc = nadc - gdpc;
                                pc = pc - gdpc;
                                counter = gdpc;
                                gdpc = 0;
```

```
                }

                else if ( (pc>=gdpc) && (gdpc>=nadc) )
                {
                        gdpc = gdpc - nadc;
                        pc = pc - nadc;
                        counter = nadc;
                        nadc = 0;
                }

                else if ( (pc>=nadc) && (nadc>=gdpc) )
                {
                        nadc = nadc - gdpc;
                        pc = pc - gdpc;
                        counter = gdpc;
                        gdpc = 0;
                }

                else if ( (gdpc == nadc) && (gdpc == pc) )
                {
                        counter = gdpc;
                        gdpc = 0;
                        nadc = 0;
                        pc = 0;
                }

                holdIn(passive, Time::Zero );
        }
        else
        {
                passivate();
        }

    }

    return *this ;
}

/*****************************************************************
* Function Name: outputFunction
* Description:
*****************************************************************/
Model &Step6::outputFunction( const InternalMessage &msg )
{

    if ( counter != 0 )
    {
            sendOutput( msg.time(), _13_BPG, counter );
  sendOutput( msg.time(), NADH, counter );
            sendOutput( msg.time(), H, counter );
    }

    return *this;

}
```

A.1.15 STEP7.H

```
/*********************************************************************
*
*  DESCRIPTION: Atomic Model Step7 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 23/10/2003
*  Modified: 15/11/2003
*  Modified: 09/12/2003
*
*********************************************************************/

#ifndef __STEP7_H
#define __STEP7_H

#include <list>
#include "atomic.h"     // class Atomic

class Step7 : public Atomic
{
public:
        Step7( const string &name = "Step7" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &_13_BPG; //1,3-bisphosphoglycerate
        const Port &ADP;
        const Port &PGK;          //phosphoglyceratekinase
   // outputs
        Port &_3_phosphoglycerate;      //3-phosphoglycerate
        Port &ATPo;

        Time preparationTime;

        double _13bpgc;
        double adpc;
        bool ifpgk;

        double counter;

};      // class Step7
```

```
// ** inline ** //
inline
string Step7::className() const
{
        return "Step7" ;
}

#endif  //__STEP7_H
```

A.1.16 STEP7.CPP

```
/*********************************************************************
*
*  DESCRIPTION: Atomic Model Step7 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE: 23/10/2003
*  Modified: 15/10/2003
*  Modified: 09/12/2003
*
*********************************************************************/

/** include files **/
#include "step7.h"      // class step7
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*********************************************************************
* Function Name: Step7
* Description:
*********************************************************************/
Step7::Step7( const string &name )
: Atomic( name )
, _13_BPG( addInputPort( "_13_BPG" ) )
, ADP( addInputPort( "ADP" ) )
, PGK( addInputPort( "PGK" ) )
, _3_phosphoglycerate( addOutputPort( "_3_phosphoglycerate" ) )
, ATPo( addOutputPort( "ATPo" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}
```

```
/*********************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*********************************************************************/
Model &Step7::initFunction()
{
	_13bpgc = 0;
	adpc = 0;
	ifpgk = false;
	counter = 0;

	return *this ;
}

/*********************************************************************
* Function Name: externalFunction
* Description:
*********************************************************************/
Model &Step7::externalFunction( const ExternalMessage &msg )
{
	if( msg.port() == _13_BPG )
	{
		_13bpgc = _13bpgc + msg.value() ;

		if ( (adpc > 0 ) && (ifpgk == true ) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	else if( msg.port() == ADP )
	{
		adpc = adpc + msg.value() ;

		if ( (_13bpgc > 0 ) && (ifpgk == true ) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	else if ( msg.port() == PGK )
	{
		ifpgk = true ;

		if ( (_13bpgc > 0 ) && (adpc > 0) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	return *this;
}
```

```
/*********************************************************************
* Function Name: internalFunction
* Description:
*********************************************************************/
Model &Step7::internalFunction( const InternalMessage & )
{

	counter = 0;

	if  ( state() == passive )
	{
		passivate();
	}

	else
	{
		if ( (_13bpgc >= 1) && (adpc >= 1 ) && (ifpgk == true ) )
		{
			if (_13bpgc > adpc)
			{
				_13bpgc = _13bpgc - adpc;
				counter = adpc;
				adpc = 0;
			}

			else if (_13bpgc < adpc)
			{
				adpc = adpc - _13bpgc;
				counter = _13bpgc;
				_13bpgc = 0;
			}

			else if (_13bpgc == adpc)
			{
				counter = adpc;
				adpc = 0;
				_13bpgc = 0;
			}

			holdIn(passive, Time::Zero );
		}

		else
		{
			passivate();
		}
	}

	return *this ;
}

/*********************************************************************
* Function Name: outputFunction
* Description:
*********************************************************************/
```

```
Model &Step7::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
                sendOutput( msg.time(), _3_phosphoglycerate, counter ) ;
    sendOutput( msg.time(), ATPo, counter ) ;
        }

        return *this ;

}
```

A.1.17 STEP8.H

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step8 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*                 mailto://roxana.djafarzadeh@worldheart.com
*
*  DATE: 23/10/2003
*  Modified 14/11/2003
*
*******************************************************************/

#ifndef __STEP8_H
#define __STEP8_H

#include <list>
#include "atomic.h"     // class Atomic

class Step8 : public Atomic
{
public:
        Step8( const string &name = "Step8" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &_3_phosphoglycerate;
        const Port &PGM;
   // outputs
```

```
        Port &_2_phosphoglycerate;

        Time preparationTime;

        double _3pgc;
        bool ifpgm;

        double counter;

};      // class Step8

// ** inline ** //
inline
string Step8::className() const
{
        return "Step8" ;
}

#endif  //__STEP8_H
```

A.1.18 STEP8.CPP

```
/*********************************************************************
*
*  DESCRIPTION: Atomic Model Step8 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE: 23/10/2003
*  Modified: 15/11/2003
*
*********************************************************************/

/** include files **/
#include "step8.h"      // class step2
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*********************************************************************
* Function Name: Step8
* Description:
*********************************************************************/
Step8::Step8( const string &name )
: Atomic( name )
, _3_phosphoglycerate( addInputPort( "_3_phosphoglycerate" ) )
, PGM( addInputPort( "PGM" ) )
, _2_phosphoglycerate( addOutputPort( "_2_phosphoglycerate" ) )
, preparationTime( 0, 0, 10, 0 )
```

```
{
	string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

	if( time != "" )
		preparationTime = time ;
}

/*********************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*********************************************************************/
Model &Step8::initFunction()
{
	_3pgc = 0;
	ifpgm = false;
	counter = 0;

	return *this ;
}

/*********************************************************************
* Function Name: externalFunction
* Description:
*********************************************************************/
Model &Step8::externalFunction( const ExternalMessage &msg )
{
	if( msg.port() == _3_phosphoglycerate )
	{
		_3pgc = _3pgc + msg.value() ;

		if ( ifpgm == true )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );

	}

	else if ( msg.port() == PGM )
	{
		ifpgm = true ;

		if (_3pgc > 0 )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	return *this;
}

/*********************************************************************
* Function Name: internalFunction
* Description:
*********************************************************************/
```

```
Model &Step8::internalFunction( const InternalMessage & )
{
        counter = 0;

        if  ( state() == passive )
        {
                passivate();
        }

        else
        {
                if ( (_3pgc >= 1) && (ifpgm == true ) )
                {
                        counter = _3pgc;
                        _3pgc = 0;

                        holdIn(passive, Time::Zero );
                }

                else
                {
                        passivate();
                }
        }

        return *this ;
}

/*******************************************************************
* Function Name: outputFunction
* Description:
*******************************************************************/
Model &Step8::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
      sendOutput( msg.time(), _2_phosphoglycerate, counter ) ;
        }

        return *this ;
}
```

A.1.19 STEP9.H

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step9 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
```

```
*                       mailto://roxana.djafarzadeh@worldheart.com
*
*  DATE: 23/10/2003
*  Modified: 14/11/2003
*
**********************************************************************/

#ifndef __STEP9_H
#define __STEP9_H

#include <list>
#include "atomic.h"     // class Atomic

class Step9 : public Atomic
{
public:
        Step9( const string &name = "Step9" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &_2_phosphoglycerate;
        const Port &enolase;
        // outputs
        Port &phosphoenolpyruvic;
        Port &H2O;

        Time preparationTime;
        double _2pgc;
        bool ifenolase;

        double counter;

};      // class Step9

// ** inline ** //
inline
string Step9::className() const
{
        return "Step9" ;
}

#endif  //__STEP9_H
```

A.1.20 STEP9.CPP

```
/*****************************************************************
*
*  DESCRIPTION: Atomic Model Step9 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE: 23/10/2003
*  Modified: 14/11/2003
*
*****************************************************************/

/** include files **/
#include "step9.h"      // class step1
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*****************************************************************
* Function Name: Step9
* Description:
*****************************************************************/
Step9::Step9( const string &name )
: Atomic( name )
, _2_phosphoglycerate( addInputPort( "_2_phosphoglycerate" ) )
, enolase( addInputPort( "enolase" ) )
, phosphoenolpyruvic( addOutputPort( "phosphoenolpyruvic" ) )
, H2O( addOutputPort( "H2O" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*****************************************************************
* Function Name: initFunction
* Description: Resetea la lista
* Precondition: El tiempo del proximo evento interno es Infinito
*****************************************************************/
Model &Step9::initFunction()
{
        _2pgc = 0;
        ifenolase = false;
        counter = 0;

        return *this ;
```

```
}

/*********************************************************************
* Function Name: externalFunction
* Description:
*********************************************************************/
Model &Step9::externalFunction( const ExternalMessage &msg )
{

        if( msg.port() == _2_phosphoglycerate )
        {
                _2pgc = _2pgc + msg.value() ;

                if ( ifenolase == true )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }

        else if ( msg.port() == enolase )
        {
                ifenolase = true ;

                if ( _2pgc > 0 )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }

        return *this;
}

/*********************************************************************
* Function Name: internalFunction
* Description:
*********************************************************************/
Model &Step9::internalFunction( const InternalMessage & )
{
   counter = 0;

        if  ( state() == passive )
        {
                passivate();
        }

        else
        {
                if ( (_2pgc >= 1) && (ifenolase == true ) )
                {
                        counter = _2pgc;
                        _2pgc = 0;

                        holdIn(passive, Time::Zero );
                }
```

```
                else
                {
                        passivate();
                }
        }

        return *this ;
}

/*******************************************************************
* Function Name: outputFunction
* Description:
*******************************************************************/
Model &Step9::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
                sendOutput( msg.time(), H2O, counter ) ;
        sendOutput( msg.time(), phosphoenolpyruvic, counter ) ;
        }

        return *this ;
}
```

A.1.21 STEP10.H

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step10 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*                  mailto://roxana.djafarzadeh@worldheart.com
*
*  DATE: 23/10/2003
*
*******************************************************************/

#ifndef __STEP10_H
#define __STEP10_H

#include <list>
#include "atomic.h"     // class Atomic

class Step10 : public Atomic
{
public:
        Step10( const string &name = "Step10" ); //Default constructor

        virtual string className() const ;
protected:
```

```
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &phosphoenolpyruvic;
        const Port &ADP;
        const Port &pyruvate_kinase;
   // outputs
        Port &pyruvate;
        Port &ATPo;

        Time preparationTime;

        double pepc;
        double adpc;
        bool ifpk;
   double counter;

};      // class Step10

// ** inline ** //
inline
string Step10::className() const
{
        return "Step10" ;
}

#endif  //__STEP10_H
```

A.1.22 STEP10CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model Step10 of Glycolysis
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE: 23/10/2003
*  Modified: 13/11/2003
*  Modified: 09/12/2003
*
*******************************************************************/

/** include files **/
#include "step10.h"     // class step10
```

```
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: Step10
* Description:
*******************************************************************/
Step10::Step10( const string &name )
: Atomic( name )
, phosphoenolpyruvic( addInputPort( "phosphoenolpyruvic" ) )
, ADP( addInputPort( "ADP" ) )
, pyruvate_kinase( addInputPort( "pyruvate_kinase" ) )
, pyruvate( addOutputPort( "pyruvate" ) )
, ATPo( addOutputPort( "ATPo" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &Step10::initFunction()
{
        pepc = 0;
        adpc = 0;
        ifpk = false;
        counter = 0;

        return *this ;
}

/*******************************************************************
* Function Name: externalFunction
* Description:
*******************************************************************/
Model &Step10::externalFunction( const ExternalMessage &msg )
{

        if( msg.port() == phosphoenolpyruvic )
        {
                pepc = pepc + msg.value() ;
                if ( (adpc > 0 ) && (ifpk == true ) )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }
```

```
        else if( msg.port() == ADP )
        {
                adpc = adpc + msg.value() ;
                if ( (pepc > 0 ) && (ifpk == true ) )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }

        else if ( msg.port() == pyruvate_kinase )
        {
                ifpk = true ;

                if ( (pepc > 0 ) && (adpc > 0) )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }

        return *this;
}

/*****************************************************************
* Function Name: internalFunction
* Description:
*****************************************************************/
Model &Step10::internalFunction( const InternalMessage & )
{
        counter = 0;

        if  ( state() == passive )
        {
                passivate();
        }

        else
        {
                if ( (pepc >= 1) && (adpc >= 1 ) && (ifpk == true ) )
                {
                        if (pepc > adpc)
                        {
                                pepc = pepc - adpc;
                                counter = adpc;
                                adpc = 0;
                        }

                        else if (pepc < adpc)
                        {
                                adpc = adpc - pepc;
                                counter = pepc;
                                pepc = 0;
                        }

                        else if (pepc == adpc)
                        {
```

```
                    counter = pepc;
                    pepc = 0;
                    adpc = 0;
                }

                holdIn(passive, Time::Zero );
            }

            else
            {
                passivate();
            }
        }

        return *this ;
}

/*****************************************************************
* Function Name: outputFunction
* Description:
*****************************************************************/
Model &Step10::outputFunction( const InternalMessage &msg )
{
        if ( counter != 0 )
        {
            sendOutput( msg.time(), ATPo, counter ) ;
    sendOutput( msg.time(), pyruvate, counter ) ;
        }

        return *this ;
}
```

A.2 COUPLED MODELS

A.2.1 GLYCOLYSIS.MA

```
[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 : phosphoglucoisomerase phosphoglucoisomerase@step2
Link : ATPi ATPi@step3
Link : PFK PFK@step3
```

Link : aldolase aldolase@step4
Link : isomerase isomerase@step4to5
Link : aldolase aldolase@step5
Link : NAD NAD@step6
Link : P P@step6
Link : G3PD G3PD@step6
Link : ADP ADP@step7
Link : PGK PGK@step7
Link : PGM PGM@step8
Link : enolase enolase@step9
Link : pyruvate_kinase pyruvate_kinase@step10
Link : glucose_6_phosphate@step1 glucose_6_phosphate@step2
Link : fructose_6_phosphate@step2 fructose_6_phosphate@step3
Link : fructose_16_bisphosphate@step3 fructose_16_bisphosphate@step4
Link : fructose_16_bisphosphate@step3 fructose_16_bisphosphate@step5
Link : DHP@step4 DHP@step4to5
Link : GDP@step4to5 GDP@step6
Link : GDP@step5 GDP@step6
Link : _13_BPG@step6 _13_BPG@step7
Link : _3_phosphoglycerate@step7 _3_phosphoglycerate@step8
Link : _2_phosphoglycerate@step8 _2_phosphoglycerate@step9
Link : phosphoenolpyruvic@step9 phosphoenolpyruvic@step10
Link : ADP@step1 ADP@step10
Link : H@step1 H
Link : ADP@step3 ADP@step7
Link : NADH@step6 NADH
Link : H@step6 H
Link : ATPo@step7 ATPo
Link : H2O@step9 H2O
Link : pyruvate@step10 pyruvate
Link : ATPo@step10 ATPo
[step1]
preparation : 00:00:00:000
[step2]
preparation : 00:00:05:000
[step3]
preparation : 00:00:15:000
[step4]
preparation : 00:00:05:000
[step4to5]
preparation : 00:00:05:000
[step5]
preparation : 00:00:05:000
[step6]
preparation : 00:00:05:000
[step7]
preparation : 00:00:05:000
[step8]
preparation : 00:00:05:000
[step9]
preparation : 00:00:05:000
[step10]
preparation : 00:00:05:000

A.2.2 GLYCOLYSIS.EV

```
00:00:10:00 glucose 8
00:00:18:00 ATPi 7
00:00:50:00 hexokinase 1
00:00:51:00 phosphoglucoisomerase 1
00:00:52:00 PFK 2
00:00:53:00 isomerase 1
00:00:55:00 aldolase 1
00:01:02:00 G3PD 1
00:01:03:00 PGK 1
00:01:04:00 PGM 1
00:01:05:00 enolase 1
00:01:07:00 pyruvate_kinase 1
00:01:10:00 NAD 5
00:01:12:00 P 6
00:08:40:00 glucose 8
00:08:50:00 hexokinase 1
00:09:52:00 PFK 2
00:11:18:00 ATPi 13
00:12:10:00 NAD 8
00:14:12:00 P 9
```

A.2.3 GLYCOLYSIS.OUT

```
00:00:50:000 h 7
00:01:12:000 nadh 5
00:01:12:000 h 5
00:01:12:000 atpo 5
00:01:12:000 h2o 5
00:01:12:000 atpo 5
00:01:12:000 pyruvate 5
00:11:18:000 h 9
00:12:10:000 nadh 1
00:12:10:000 h 1
00:12:10:000 atpo 1
00:12:10:000 h2o 1
00:12:10:000 atpo 1
00:12:10:000 pyruvate 1
00:14:12:000 nadh 7
00:14:12:000 h 7
00:14:12:000 atpo 7
00:14:12:000 h2o 7
00:14:12:000 atpo 7
00:14:12:000 pyruvate 7
```

APPENDIX B – CODE FOR KREB'S CYCLE

This appendix contains the code for Krebs section.

B.1 ATOMIC MODELS

B.1.1 STEPA.H

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model StepA of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 30/12/2003
*
*******************************************************************/

#ifndef __STEPA_H
#define __STEPA_H

#include <list>
#include "atomic.h"     // class Atomic

class StepA : public Atomic
{
public:
        StepA( const string &name = "StepA" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &pyruvate;
```

```
    // outputs
        Port &acetyl_CoA;
        Port &NADH;
        Port &CO2;

        Time preparationTime;
        double pyruvatec;
    double counter;

};      // class StepA

// ** inline ** //
inline
string StepA::className() const
{
        return "StepA" ;
}

#endif  //__STEPA_H
```

B.1.2 STEPA.CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model StepA of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 01/01/2004
*  Modified: 17/01/2004
*  Modified: 05/04/2004
*
*******************************************************************/

/** include files **/
#include "stepA.h"      // class stepA
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: StepA
* Description:
*******************************************************************/
StepA::StepA( const string &name )
: Atomic( name )
, pyruvate( addInputPort( "pyruvate" ) )
, acetyl_CoA( addOutputPort( "acetyl_CoA" ) )
```

```
, NADH( addOutputPort( "NADH" ) )
, CO2( addOutputPort( "CO2" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &StepA::initFunction()
{
        pyruvatec = 0;
        counter = 0;

        return *this ;
}

/*******************************************************************
* Function Name: externalFunction
* Description:
*******************************************************************/
Model &StepA::externalFunction( const ExternalMessage &msg )
{

        if( msg.port() == pyruvate )
        {
                pyruvatec = pyruvatec + msg.value();
                holdIn(active, Time::Zero);
        }
        else
        {
                holdIn(passive, Time::Zero);
        }

        return *this;
}

/*******************************************************************
* Function Name: internalFunction
* Description:
*******************************************************************/
Model &StepA::internalFunction( const InternalMessage & )
{

   counter = 0;

        if  ( state() == passive )

        {
```

```
                passivate();
        }

        else

        {
                if ( pyruvatec >=1 )
                {
                        counter = pyruvatec;

                        holdIn(passive, Time::Zero );
                }

                else
                {
                        passivate();
                }

        }

        return *this ;

}

/*****************************************************************
* Function Name: outputFunction
* Description:
*****************************************************************/
Model &StepA::outputFunction( const InternalMessage &msg )
{

   if ( counter != 0 )
        {
                sendOutput( msg.time(), acetyl_CoA, counter );
                sendOutput( msg.time(), NADH, counter );
                sendOutput( msg.time(), CO2, counter );
        }

        return *this ;

}
```

B.1.3 STEPB1.H

```
/*****************************************************************
*
*  DESCRIPTION: Atomic Model StepB1 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
```

```
*
*  DATE of Creation: 11/01/2004
*  Modified: 17/01/2004
*
*********************************************************************/

#ifndef __STEPB1_H
#define __STEPB1_H

#include <list>
#include "atomic.h"     // class Atomic

class StepB1 : public Atomic
{
public:
        StepB1( const string &name = "StepB1" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &acetyl_CoA;
        const Port &oxaloacetate;
   // outputs
        Port &citrate;

        Time preparationTime;
        double acetyl_CoAc;
        double oxaloacetatec;
   double counter;

};      // class StepB1

// ** inline ** //
inline
string StepB1::className() const
{
        return "StepB1" ;
}

#endif  //__STEPB1_H
```

B.1.4 STEPB1.CPP

```
/*********************************************************************
*
*  DESCRIPTION: Atomic Model StepB1 of Krebs Cycle
```

```
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of CREATION: 17/01/2004
*
**********************************************************************/

/** include files **/
#include "stepB1.h"      // class stepB1
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: StepB1
* Description:
*******************************************************************/
StepB1::StepB1( const string &name )
: Atomic( name )
, acetyl_CoA( addInputPort( "acetyl_CoA" ) )
, oxaloacetate( addInputPort( "oxaloacetate" ) )
, citrate( addOutputPort( "citrate" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &StepB1::initFunction()
{
        acetyl_CoAc = 0;
        oxaloacetatec = 0;
   counter = 0;

        return *this ;
}

/*******************************************************************
* Function Name: externalFunction
* Description:
*******************************************************************/
Model &StepB1::externalFunction( const ExternalMessage &msg )
{
        if( msg.port() == acetyl_CoA )
```

```
	{
		acetyl_CoAc = acetyl_CoAc + msg.value() ;

		if ( oxaloacetatec > 0 )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	else if ( msg.port() == oxaloacetate )
	{
		oxaloacetatec = oxaloacetatec + msg.value() ;

		if ( acetyl_CoAc > 0 )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	return *this;
}

/*****************************************************************
* Function Name: internalFunction
* Description:
*****************************************************************/
Model &StepB1::internalFunction( const InternalMessage & )
{

	counter = 0;

	if  ( state() == passive )
	{
		passivate();
	}

	else
	{
		if ( (acetyl_CoAc >= 1) && (oxaloacetatec >= 1) )
		{

			if ( acetyl_CoAc > oxaloacetatec )
			{
				acetyl_CoAc = acetyl_CoAc - oxaloacetatec;
				counter = oxaloacetatec;
				oxaloacetatec = 0;
			}

			else if ( acetyl_CoAc < oxaloacetatec )
			{
				oxaloacetatec = oxaloacetatec - acetyl_CoAc;
				counter = acetyl_CoAc;
				acetyl_CoAc = 0;
			}
```

```
                        else if ( acetyl_CoAc == oxaloacetatec )
                        {
                                counter = acetyl_CoAc;
                                acetyl_CoAc = 0;
                                oxaloacetatec = 0;
                        }

                        holdIn(passive, Time::Zero );
                }

                else
                {
                        passivate();
                }
        }

        return *this ;

}

/*****************************************************************
* Function Name: outputFunction
* Description:
*****************************************************************/
Model &StepB1::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
                sendOutput( msg.time(), citrate, counter ) ;
        }

        return *this ;
}
```

B.1.5 STEPB2.H

```
/*****************************************************************
*
*  DESCRIPTION: Atomic Model StepB2 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 11/01/2004
*  Modifications: 17/01/2004
*
*****************************************************************/

#ifndef __STEPB2_H
```

```
#define __STEPB2_H

#include <list>
#include "atomic.h"     // class Atomic

class StepB2 : public Atomic
{
public:
        StepB2( const string &name = "StepB2" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &citrate;
   // outputs
        Port &isocitrate;

        Time preparationTime;
        double citratec;
   double counter;

};      // class StepB2

// ** inline ** //
inline
string StepB2::className() const
{
        return "StepB2" ;
}

#endif  //__STEPB2_H
```

B.1.6 STEPB2.CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model StepB2 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 11/01/2004
*
```

```
**********************************************************************/

/** include files **/
#include "stepB2.h"      // class stepB2
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: StepB2
* Description:
*******************************************************************/
StepB2::StepB2( const string &name )
: Atomic( name )
, citrate( addInputPort( "citrate" ) )
, isocitrate( addOutputPort( "isocitrate" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &StepB2::initFunction()
{
        citratec = 0;
        counter = 0;

        return *this ;
}

/*******************************************************************
* Function Name: externalFunction
* Description:
*******************************************************************/
Model &StepB2::externalFunction( const ExternalMessage &msg )
{

        if( msg.port() == citrate )
        {
                citratec = citratec + msg.value() ;

                //cout << "Roxana: External Function = citratec:" << citratec << "\n";

                holdIn(passive, Time::Zero);
        }

        return *this;
```

```
}

/*******************************************************************
* Function Name: internalFunction
* Description:
*******************************************************************/
Model &StepB2::internalFunction( const InternalMessage & )
{

        if  ( state() == passive )
        {
                passivate();
                //cout << "Roxana: Internal function = Passive State!\n";
        }
        else
        {
                counter = citratec;
                //cout << "Roxana: Internal Function = counter:" << counter << "\n";

                holdIn(passive, Time::Zero );

                //cout << "Roxana: Internal Function = citratec:" << citratec << "\n";
                //cout << "Roxana: Internal Function = counter:" << counter << "\n";

        }

        //cout << "Roxana: Internal Function = counter:" << counter << "\n";
        //cout << "Roxana: Internal Function = citratec:" << citratec << "\n";

        return *this ;

}

/*******************************************************************
* Function Name: outputFunction
* Description:
*******************************************************************/
Model &StepB2::outputFunction( const InternalMessage &msg )
{

        sendOutput( msg.time(), isocitrate, citratec );

        return *this ;
}
```

B.1.7 STEPB3.H

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model StepB3 of Krebs Cycle
*
```

```
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 11/01/2004
*  Modified: 17/01/2004
*
*********************************************************************/

#ifndef __STEPB3_H
#define __STEPB3_H

#include <list>
#include "atomic.h"     // class Atomic

class StepB3 : public Atomic
{
public:
        StepB3( const string &name = "StepB3" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &isocitrate;
        const Port &NAD;
   // outputs
        Port &alpha_ketoglutarate;
        Port &NADH;
        Port &CO2;

        Time preparationTime;
        double isocitratec;
        double nadc;
   double counter;

};      // class StepB3

// ** inline ** //
inline
string StepB3::className() const
{
        return "StepB3" ;
}

#endif  //__STEPB3_H
```

B.1.8 STEPB3.CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model StepB3 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 11/01/2004
*  Modified: 18/01/2004
*
*******************************************************************/

/** include files **/
#include "stepB3.h"      // class stepB3
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: StepB3
* Description:
*******************************************************************/
StepB3::StepB3( const string &name )
: Atomic( name )
, isocitrate( addInputPort( "isocitrate" ) )
, NAD( addInputPort( "NAD" ) )
, alpha_ketoglutarate( addOutputPort( "alpha_ketoglutarate" ) )
, NADH( addOutputPort( "NADH" ) )
, CO2( addOutputPort( "CO2" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &StepB3::initFunction()
{
        isocitratec = 0;
        nadc = 0;
        counter = 0;
```

```
	return *this ;
}

/*********************************************************************
* Function Name: externalFunction
* Description:
*********************************************************************/
Model &StepB3::externalFunction( const ExternalMessage &msg )
{

	if( msg.port() == isocitrate )
	{
		isocitratec = isocitratec + msg.value() ;
		if ( nadc > 0 )
			holdIn(active, Time::Zero);
		else
			holdIn(passive, Time::Zero);
	}


	else if( msg.port() == NAD )
	{
		nadc = nadc + msg.value() ;
		if ( isocitratec > 0 )
			holdIn(active, Time::Zero);
		else
			holdIn(passive, Time::Zero);
	}


	return *this;
}

/*********************************************************************
* Function Name: internalFunction
* Description:
*********************************************************************/
Model &StepB3::internalFunction( const InternalMessage & )
{

   counter = 0;
	if  ( state() == passive )
	{
		passivate();
	}
	else
	{
		if ( ( isocitratec >= 1 ) && ( nadc >= 1 ) )
		{

			if ( isocitratec > nadc )
			{
				isocitratec = isocitratec - nadc;
				counter = nadc;
				nadc = 0;
```

```
                }

                else if ( isocitratec < nadc )
                {
                        nadc = nadc - isocitratec;
                        counter = isocitratec;
                        isocitratec = 0;
                }

                else if ( isocitratec == nadc )
                {
                        counter = isocitratec;
                        isocitratec = 0;
                        nadc = 0;
                }

                holdIn(passive, Time::Zero );
        }

        else
        {
                passivate();
        }
    }


    return *this ;

}

/*********************************************************************
* Function Name: outputFunction
* Description:
*********************************************************************/
Model &StepB3::outputFunction( const InternalMessage &msg )
{

    if ( counter != 0 )
    {
        sendOutput( msg.time(), alpha_ketoglutarate, counter );
        sendOutput( msg.time(), NADH, counter );
        sendOutput( msg.time(), CO2, counter );
    }

    return *this ;
}
```

B.1.9 STEPB4.H

```
/*********************************************************************
*
*  DESCRIPTION: Atomic Model StepB4 of Krebs Cycle
```

```
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 11/01/2004
*  Modified: 17/01/2004
*
*********************************************************************/

#ifndef __STEPB4_H
#define __STEPB4_H

#include <list>
#include "atomic.h"     // class Atomic

class StepB4 : public Atomic
{
public:
        StepB4( const string &name = "StepB4" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &alpha_ketoglutarate;
        const Port &NAD;
   // outputs
        Port &succinyl_CoA;
        Port &NADH;
        Port &CO2;

        Time preparationTime;
        double alpha_ketoglutaratec;
        double nadc;
   double counter;

};      // class StepB4

// ** inline ** //
inline
string StepB4::className() const
{
        return "StepB4" ;
}

#endif  //__STEPB4_H
```

B.1.10 STEPB4.CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model StepB4 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 11/01/2004
*  Modified: 18/01/2004
*
*******************************************************************/

/** include files **/
#include "stepB4.h"      // class stepB4
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: StepB4
* Description:
*******************************************************************/
StepB4::StepB4( const string &name )
: Atomic( name )
, NAD( addInputPort( "NAD" ) )
, alpha_ketoglutarate( addInputPort( "alpha_ketoglutarate" ) )
, succinyl_CoA( addOutputPort( "succinyl_CoA" ) )
, NADH( addOutputPort( "NADH" ) )
, CO2( addOutputPort( "CO2" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &StepB4::initFunction()
{
        alpha_ketoglutaratec = 0;
        nadc = 0;
        counter = 0;
```

```
	return *this ;
}

/*********************************************************************
* Function Name: externalFunction
* Description:
*********************************************************************/
Model &StepB4::externalFunction( const ExternalMessage &msg )
{

	if( msg.port() == alpha_ketoglutarate )
	{
		alpha_ketoglutaratec = alpha_ketoglutaratec + msg.value() ;
		if ( nadc > 0 )
			holdIn(active, Time::Zero);
		else
			holdIn(passive, Time::Zero);
	}


	else if( msg.port() == NAD )
	{
		nadc = nadc + msg.value() ;
		if ( alpha_ketoglutaratec > 0 )
			holdIn(active, Time::Zero);
		else
			holdIn(passive, Time::Zero);
	}


	return *this;
}

/*********************************************************************
* Function Name: internalFunction
* Description:
*********************************************************************/
Model &StepB4::internalFunction( const InternalMessage & )
{

   counter = 0;
	if  ( state() == passive )
	{
		passivate();
	}
	else
	{
		if ( ( alpha_ketoglutaratec >= 1 ) && ( nadc >= 1 ) )
		{

			if ( alpha_ketoglutaratec > nadc )
			{
				alpha_ketoglutaratec = alpha_ketoglutaratec - nadc;
				counter = nadc;
				nadc = 0;
```

```
                        }

                        else if ( alpha_ketoglutaratec < nadc )
                        {
                                nadc = nadc - alpha_ketoglutaratec;
                                counter = alpha_ketoglutaratec;
                                alpha_ketoglutaratec = 0;
                        }

                        else if ( alpha_ketoglutaratec == nadc )
                        {
                                counter = alpha_ketoglutaratec;
                                alpha_ketoglutaratec = 0;
                                nadc = 0;
                        }

                        holdIn(passive, Time::Zero );
                }

                else
                {
                        passivate();
                }
        }


        return *this ;

}

/*****************************************************************
* Function Name: outputFunction
* Description:
*****************************************************************/
Model &StepB4::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
                sendOutput( msg.time(), succinyl_CoA, counter );
                sendOutput( msg.time(), NADH, counter );
                sendOutput( msg.time(), CO2, counter );
        }

        return *this ;
}
```

B.1.11 STEPB5.H

```
/*****************************************************************
*
*  DESCRIPTION: Atomic Model StepB5 of Krebs Cycle
```

```
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 11/01/2004
*  Modified: 18/01/2004
*
*********************************************************************/

#ifndef __STEPB5_H
#define __STEPB5_H

#include <list>
#include "atomic.h"     // class Atomic

class StepB5 : public Atomic
{
public:
        StepB5( const string &name = "StepB5" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &GDP;
        const Port &Pi;
        const Port &succinyl_CoA;
   // outputs
        Port &succinate;
        Port &CoA;
        Port &GTP;

        Time preparationTime;

        double succinyl_CoAc;
        double Pic;
        double gdpc;
   double counter;

};      // class StepB5

// ** inline ** //
inline
string StepB5::className() const
{
        return "StepB5" ;
}
```

```
#endif   //__STEPB5_H
```

B.1.12 STEPB5.CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model StepB5 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 18/01/2004
*
*******************************************************************/

/** include files **/
#include "stepB5.h"      // class stepB5
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: StepB5
* Description:
*******************************************************************/
StepB5::StepB5( const string &name )
: Atomic( name )
, GDP( addInputPort( "GDP" ) )
, Pi( addInputPort( "Pi" ) )
, succinyl_CoA( addInputPort( "succinyl_CoA" ) )
, succinate( addOutputPort( "succinate" ) )
, CoA( addOutputPort( "CoA" ) )
, GTP( addOutputPort( "GTP" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &StepB5::initFunction()
{
        succinyl_CoAc = 0;
        gdpc = 0;
```

```
	Pic = 0;

	return *this ;
}

/*********************************************************************
* Function Name: externalFunction
* Description:
*********************************************************************/
Model &StepB5::externalFunction( const ExternalMessage &msg )
{
	if( msg.port() == succinyl_CoA )
	{
		succinyl_CoAc = succinyl_CoAc + msg.value() ;
		if ( ( gdpc > 0 ) && ( Pic > 0 ) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );

	}

	else if( msg.port() == GDP )
	{
		gdpc = gdpc + msg.value() ;
		if ( ( succinyl_CoAc > 0 ) && ( Pic > 0 ) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	else if( msg.port() == Pi )
	{
		Pic = Pic + msg.value() ;
		if ( ( succinyl_CoAc > 0 ) && ( gdpc > 0 ) )
			holdIn( active, Time::Zero );
		else
			holdIn(passive, Time::Zero );
	}

	return *this;
}

/*********************************************************************
* Function Name: internalFunction
* Description:
*********************************************************************/
Model &StepB5::internalFunction( const InternalMessage & )
{

	counter = 0;

	if  ( state() == passive )
	{
		passivate();
	}
```

```
else
{
        if ( (succinyl_CoAc >= 1) && (gdpc >= 1 ) && (Pic >= 1 ) )
        {
                if ( ( succinyl_CoAc >= gdpc ) && ( gdpc >= Pic ) )
                {
                        succinyl_CoAc = succinyl_CoAc - Pic;
                        gdpc = gdpc - Pic;
                        counter = Pic;
                        Pic = 0;
                }

                else if ( ( succinyl_CoAc >= Pic ) && ( Pic >= gdpc ) )
                {
                        succinyl_CoAc = succinyl_CoAc - gdpc;
                        Pic = Pic - gdpc;
                        counter = gdpc;
                        gdpc = 0;
                }

                else if ( ( gdpc >= succinyl_CoAc ) && ( succinyl_CoAc >= Pic ) )
                {
                        gdpc = gdpc - Pic;
                        succinyl_CoAc = succinyl_CoAc - Pic;
                        counter = Pic;
                        Pic = 0;
                }

                else if ( ( gdpc >= Pic ) && ( Pic >= succinyl_CoAc ) )
                {
                        gdpc = gdpc - succinyl_CoAc;
                        Pic = Pic - succinyl_CoAc;
                        counter = succinyl_CoAc;
                        succinyl_CoAc = 0;
                }

                else if ( ( Pic >= succinyl_CoAc ) && ( succinyl_CoAc >= gdpc ) )
                {
                        Pic = Pic - gdpc;
                        succinyl_CoAc = succinyl_CoAc - gdpc;
                        counter = gdpc;
                        gdpc = 0;
                }

                else if ( ( Pic >= gdpc ) && ( gdpc >= succinyl_CoAc ) )
                {
                        Pic = Pic - succinyl_CoAc;
                        gdpc = gdpc - succinyl_CoAc;
                        counter = succinyl_CoAc;
                        succinyl_CoAc= 0;
                }

                else if ( (succinyl_CoAc == gdpc) && (gdpc == Pic) )
```

```
                        {
                                counter = succinyl_CoAc;
                                succinyl_CoAc = 0;
                                gdpc = 0;
                                Pic = 0;
                        }

                        holdIn(passive, Time::Zero );
                }
                else
                {
                        passivate();
                }

        }

        return *this ;
}

/*****************************************************************
* Function Name: outputFunction
* Description:
*****************************************************************/
Model &StepB5::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
                sendOutput( msg.time(), succinate, counter );
                sendOutput( msg.time(), CoA, counter );
                sendOutput( msg.time(), GTP, counter );
        }

        return *this;

}
```

B.1.13 STEPB6.H

```
/*****************************************************************
*
*  DESCRIPTION: Atomic Model StepB6 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 11/01/2004
*  Modified: 18/01/2004
*
*****************************************************************/
```

```
#ifndef __STEPB6_H
#define __STEPB6_H

#include <list>
#include "atomic.h"     // class Atomic

class StepB6 : public Atomic
{
public:
        StepB6( const string &name = "StepB6" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &succinate;
        const Port &FAD;
   // outputs
        Port &fumarate;
        Port &FADH2;

        Time preparationTime;
        double succinatec;
        double fadc;
   double counter;

};      // class StepB6

// ** inline ** //
inline
string StepB6::className() const
{
        return "StepB6" ;
}

#endif  //__STEPB6_H
```

B.1.14 STEPB6.CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model StepB6 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
```

```
*
*  DATE of CREATION: 18/01/2004
*
**********************************************************************/

/** include files **/
#include "stepB6.h"      // class stepB6
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: StepB6
* Description:
*******************************************************************/
StepB6::StepB6( const string &name )
: Atomic( name )
, succinate( addInputPort( "succinate" ) )
, FAD( addInputPort( "FAD" ) )
, fumarate( addOutputPort( "fumarate" ) )
, FADH2( addOutputPort( "FADH2" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &StepB6::initFunction()
{
        succinatec = 0;
        fadc = 0;
   counter = 0;

        return *this ;
}

/*******************************************************************
* Function Name: externalFunction
* Description:
*******************************************************************/
Model &StepB6::externalFunction( const ExternalMessage &msg )
{
        if( msg.port() == succinate )
        {
                succinatec = succinatec + msg.value() ;

                if ( fadc > 0 )
```

```
                holdIn( active, Time::Zero );
        else
                holdIn(passive, Time::Zero );
    }

    else if ( msg.port() == FAD )
    {
        fadc = fadc + msg.value() ;

        if ( succinatec > 0 )
                holdIn( active, Time::Zero );
        else
                holdIn(passive, Time::Zero );
    }

    return *this;
}

/*****************************************************************
* Function Name: internalFunction
* Description:
*****************************************************************/
Model &StepB6::internalFunction( const InternalMessage & )
{

    counter = 0;

    if  ( state() == passive )
    {
        passivate();
    }

    else
    {
        if ( (succinatec >= 1) && (fadc >= 1) )
        {

                if ( succinatec > fadc )
                {
                        succinatec = succinatec - fadc;
                        counter = fadc;
                        fadc = 0;
                }

                else if ( succinatec < fadc )
                {
                        fadc = fadc - succinatec;
                        counter = succinatec;
                        succinatec = 0;
                }

                else if ( succinatec == fadc )
                {
                        counter = succinatec;
                        succinatec = 0;
```

```
                        fadc = 0;
                }

                holdIn(passive, Time::Zero );
        }

        else
        {
                passivate();
        }
    }

    return *this ;

}

/*****************************************************************
* Function Name: outputFunction
* Description:
*****************************************************************/
Model &StepB6::outputFunction( const InternalMessage &msg )
{

    if ( counter != 0 )
    {
        sendOutput( msg.time(), fumarate, counter ) ;
        sendOutput( msg.time(), FADH2, counter ) ;
    }

    return *this ;
}
```

B.1.15 STEPB7.H

```
/*****************************************************************
*
*  DESCRIPTION: Atomic Model StepB7 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 11/01/2004
*  Modified: 18/01/2004
*
*****************************************************************/

#ifndef __STEPB7_H
#define __STEPB7_H

#include <list>
#include "atomic.h"     // class Atomic
```

```
class StepB7 : public Atomic
{
public:
        StepB7( const string &name = "StepB7" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &fumarate;
        const Port &H2O;
   // outputs
        Port &malate;

        Time preparationTime;
        double fumaratec;
        double h2oc;
   double counter;

};      // class StepB7

// ** inline ** //
inline
string StepB7::className() const
{
        return "StepB7" ;
}

#endif  //__STEPB7_H
```

B.1.16 STEPB7.CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model StepB7 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE of CREATION: 17/01/2004
*
*******************************************************************/

/** include files **/
#include "stepB7.h"     // class stepB7
```

```
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: StepB7
* Description:
*******************************************************************/
StepB7::StepB7( const string &name )
: Atomic( name )
, fumarate( addInputPort( "fumarate" ) )
, H2O( addInputPort( "H2O" ) )
, malate( addOutputPort( "malate" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/*******************************************************************
* Function Name: initFunction
* Description:
* Precondition:
*******************************************************************/
Model &StepB7::initFunction()
{
        fumaratec = 0;
        h2oc = 0;
   counter = 0;

        return *this ;
}

/*******************************************************************
* Function Name: externalFunction
* Description:
*******************************************************************/
Model &StepB7::externalFunction( const ExternalMessage &msg )
{
        if( msg.port() == fumarate )
        {
                fumaratec = fumaratec + msg.value() ;

                if ( h2oc > 0 )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }

        else if ( msg.port() == H2O )
        {
                h2oc = h2oc + msg.value() ;
```

```
                if ( fumaratec > 0 )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }

        return *this;
}

/*********************************************************************
* Function Name: internalFunction
* Description:
*********************************************************************/
Model &StepB7::internalFunction( const InternalMessage & )
{

        counter = 0;

        if  ( state() == passive )
        {
                passivate();
        }

        else
        {
                if ( (fumaratec >= 1) && (h2oc >= 1) )
                {

                        if ( fumaratec > h2oc )
                        {
                                fumaratec = fumaratec - h2oc;
                                counter = h2oc;
                                h2oc = 0;
                        }

                        else if ( fumaratec < h2oc )
                        {
                                h2oc = h2oc - fumaratec;
                                counter = fumaratec;
                                fumaratec = 0;
                        }

                        else if ( fumaratec == h2oc )
                        {
                                counter = fumaratec;
                                fumaratec = 0;
                                h2oc = 0;
                        }

                        holdIn(passive, Time::Zero );
                }

                else
                {
```

```
                        passivate();
                }
        }

        return *this ;

}

/*****************************************************************
* Function Name: outputFunction
* Description:
*****************************************************************/
Model &StepB7::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
                sendOutput( msg.time(), malate, counter ) ;
        }

        return *this ;
}
```

B.1.17 STEPB8.H

```
/*****************************************************************
*
*  DESCRIPTION: Atomic Model StepB8 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*         mailto://rdjafar@site.uottawa.ca
*
*  DATE of Creation: 11/01/2004
*  Modified: 18/01/2004
*
*****************************************************************/

#ifndef __STEPB8_H
#define __STEPB8_H

#include <list>
#include "atomic.h"     // class Atomic

class StepB8 : public Atomic
{
public:
        StepB8( const string &name = "StepB8" ); //Default constructor

        virtual string className() const ;
protected:
        Model &initFunction();
```

```
        Model &externalFunction( const ExternalMessage & );
        Model &internalFunction( const InternalMessage & );
        Model &outputFunction( const InternalMessage & );

private:
   // inputs
        const Port &malate;
        const Port &NAD;
   // outputs
        Port &oxaloacetate;
        Port &NADH;

        Time preparationTime;
        double malatec;
        double nadc;

   double counter;

};      // class StepB8

// ** inline ** //
inline
string StepB8::className() const
{
        return "StepB8" ;
}

#endif  //__STEPB8_H
```

B.1.18 STEPB8.CPP

```
/*******************************************************************
*
*  DESCRIPTION: Atomic Model StepB8 of Krebs Cycle
*
*  AUTHOR: Roxana Djafarzadeh
*
*  EMAIL: mailto://roxanadj@hotmail.com
*       mailto://rdjafar@site.uottawa.ca
*
*  DATE of CREATION: 18/01/2004
*
*******************************************************************/

/** include files **/
#include "stepB8.h"     // class stepB8
#include "message.h"    // class ExternalMessage, InternalMessage
#include "mainsimu.h"   // MainSimulator::Instance().getParameter( ... )

/** public functions **/

/*******************************************************************
* Function Name: StepB8
```

```
* Description:
**********************************************************************/
StepB8::StepB8( const string &name )
: Atomic( name )
, malate( addInputPort( "malate" ) )
, NAD( addInputPort( "NAD" ) )
, oxaloacetate( addOutputPort( "oxaloacetate" ) )
, NADH( addOutputPort( "NADH" ) )
, preparationTime( 0, 0, 10, 0 )
{
        string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;

        if( time != "" )
                preparationTime = time ;
}

/**********************************************************************
* Function Name: initFunction
* Description:
* Precondition:
**********************************************************************/
Model &StepB8::initFunction()
{
        malatec = 0;
        nadc = 0;
   counter = 0;

        return *this ;
}

/**********************************************************************
* Function Name: externalFunction
* Description:
**********************************************************************/
Model &StepB8::externalFunction( const ExternalMessage &msg )
{
        if( msg.port() == malate )
        {
                malatec = malatec + msg.value() ;

                if ( nadc > 0 )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }

        else if ( msg.port() == NAD )
        {
                nadc = nadc + msg.value() ;

                if ( malatec > 0 )
                        holdIn( active, Time::Zero );
                else
                        holdIn(passive, Time::Zero );
        }
```

```
	return *this;
}

/*****************************************************************
* Function Name: internalFunction
* Description:
*****************************************************************/
Model &StepB8::internalFunction( const InternalMessage & )
{

	counter = 0;

	if  ( state() == passive )
	{
		passivate();
	}

	else
	{
		if ( (malatec >= 1) && (nadc >= 1) )
		{

			if ( malatec > nadc )
			{
				malatec = malatec - nadc;
				counter = nadc;
				nadc = 0;
			}

			else if ( malatec < nadc )
			{
				nadc = nadc - malatec;
				counter = malatec;
				malatec = 0;
			}

			else if ( malatec == nadc )
			{
				counter = malatec;
				malatec = 0;
				nadc = 0;
			}

			holdIn(passive, Time::Zero );
		}

		else
		{
			passivate();
		}
	}

	return *this ;
```

```
}

/*******************************************************************
* Function Name: outputFunction
* Description:
*******************************************************************/
Model &StepB8::outputFunction( const InternalMessage &msg )
{

        if ( counter != 0 )
        {
                sendOutput( msg.time(), oxaloacetate, counter ) ;
                sendOutput( msg.time(), NADH, counter ) ;
        }

        return *this ;
}
```

B.2 COUPLED MODELS

B.2.1 KREBS

```
#!/bin/sh
./simu -mkrebs.ma -ekrebs.ev -lkrebs.log -okrebs.out
```

B.2.2 KREBS.MA

```
[top]
components : stepA@StepA stepB1@StepB1 stepB2@StepB2 stepB3@StepB3 stepB4@StepB4
stepB5@StepB5 stepB6@StepB6 stepB7@StepB7 stepB8@StepB8

out : FADH2 GTP HSCoAo H NADH CO2
in : pyruvate pyruvateDehydrogenase HSCoAi NAD oxaloacetate H2O citrateSynthase
acontinase isocitrateDehydrogenase alpha_ketoglutarateDehydrogenase GDP Pi
succinylCoA_Synthetase FAD succinateDehydrogenase fumarase malateDehydrogenase

Link : pyruvate pyruvate@stepA
Link : pyruvateDehydrogenase pyruvateDehydrogenase@stepA
Link : HSCoAi HSCoAi@stepA
Link : NAD NAD@stepA
Link : citrateSynthase citrateSynthase@stepB1
Link : H2O H2O@stepB1
Link : oxaloacetate oxaloacetate@stepB1
Link : acontinase acontinase@stepB2
Link : NAD NAD@stepB3
Link : isocitrateDehydrogenase isocitrateDehydrogenase@stepB3
Link : HSCoAi HSCoAi@stepB4
Link : NAD NAD@stepB4
```

Link : alpha_ketoglutarateDehydrogenase alpha_ketoglutarateDehydrogenase@stepB4
Link : GDP GDP@stepB5
Link : Pi Pi@stepB5
Link : succinylCoA_Synthetase succinylCoA_Synthetase@stepB5
Link : FAD FAD@stepB6
Link : succinateDehydrogenase succinateDehydrogenase@stepB6
Link : H2O H2O@stepB7
Link : fumarase fumarase@stepB7
Link : NAD NAD@stepB8
Link : malateDehydrogenase malateDehydrogenase@stepB8

Link : NADH@stepA NADH
Link : CO2@stepA CO2
Link : H@stepA H
Link : HSCoAo@stepB1 HSCoAo
Link : H@stepB1 H
Link : NADH@stepB3 NADH
Link : CO2@stepB3 CO2
Link : NADH@stepB4 NADH
Link : H@stepB4 H
Link : CO2@stepB4 CO2
Link : HSCoAo@stepB5 HSCoAo
Link : GTP@stepB5 GTP
Link : FADH2@stepB6 FADH2
Link : NADH@stepB8 NADH
Link : H@stepB8 H

Link : acetyl_CoA@stepA acetyl_CoA@stepB1
Link : citrate@stepB1 citrate@stepB2
Link : isocitrate@stepB2 isocitrate@stepB3
Link : alpha_ketoglutarate@stepB3 alpha_ketoglutarate@stepB4
Link : succinyl_CoA@stepB4 succinyl_CoA@stepB5
Link : succinate@stepB5 succinate@stepB6
Link : fumarate@stepB6 fumarate@stepB7
Link : malate@stepB7 malate@stepB8
Link : oxaloacetate@stepB8 oxaloacetate@stepB1

[stepA]
preparation : 00:00:05:000

[stepB1]
preparation : 00:00:10:000

[stepB2]
preparation : 00:00:25:000

[stepB3]
preparation : 00:00:40:000

[stepB4]
preparation : 00:00:45:000

[stepB5]
preparation : 00:00:55:000

[stepB6]
preparation : 00:01:05:000

[stepB7]
preparation : 00:01:20:000

[stepB8]
preparation : 00:01:35:000

B.2.3 KREBS.EV

00:00:10:00 pyruvate 1
00:00:25:00 NAD 2
00:00:40:00 H2O 2
00:00:57:00 HSCoAi 2
00:01:11:00 pyruvateDehydrogenase 1
00:01:14:00 citrateSynthase 1
00:02:01:00 oxaloacetate 1
00:02:14:00 acontinase 1
00:02:30:00 isocitrateDehydrogenase 1
00:02:30:00 alpha_ketoglutarateDehydrogenase 1
00:02:50:00 succinylCoA_Synthetase 1
00:03:l0:00 GDP 1
00:03:11:00 Pi 1
00:03:15:00 FAD 1
00:03:28:00 succinateDehydrogenase 1
00:03:40:00 fumarase 1
00:03:45:00 malateDehydrogenase 1

B.2.4 KREBS.OUT

00:01:11:000 nadh 1
00:01:11: 000 co2 1
00:01:11:000 h 1
00:02:01:000 hscoao 1
00:02:01:000 h 1
00:02:30:000 nadh 1
00:02:30:000 co2 1
00:02:30:000 nadh 1
00:02:30:000 co2 1
00:02:30:000 h 1
00:03:11:000 hscoao 1
00:03:11:000 gtp 1
00:03:28:000 fadh2 1
00:03:45:000 nadh 1
00:03:45:000 h 1

B.2.5 KREBS.LOG

Mensaje I / 00:00:00:000 / Root(00) para top(01)
Mensaje I / 00:00:00:000 / top(01) para stepa(02)

Mensaje I / 00:00:00:000 / top(01) para stepb1(03)
Mensaje I / 00:00:00:000 / top(01) para stepb2(04)
Mensaje I / 00:00:00:000 / top(01) para stepb3(05)
Mensaje I / 00:00:00:000 / top(01) para stepb4(06)
Mensaje I / 00:00:00:000 / top(01) para stepb5(07)
Mensaje I / 00:00:00:000 / top(01) para stepb6(08)
Mensaje I / 00:00:00:000 / top(01) para stepb7(09)
Mensaje I / 00:00:00:000 / top(01) para stepb8(10)
Mensaje D / 00:00:00:000 / stepa(02) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb1(03) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb2(04) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb3(05) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb4(06) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb5(07) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb6(08) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb7(09) / ... para top(01)
Mensaje D / 00:00:00:000 / stepb8(10) / ... para top(01)
Mensaje D / 00:00:00:000 / top(01) / ... para Root(00)
Mensaje X / 00:00:10:000 / Root(00) / pyruvate / 1.00000 para top(01)
Mensaje X / 00:00:10:000 / top(01) / pyruvate / 1.00000 para stepa(02)
Mensaje D / 00:00:10:000 / stepa(02) / 00:00:00:000 para top(01)
Mensaje D / 00:00:10:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:10:000 / Root(00) para top(01)
Mensaje * / 00:00:10:000 / top(01) para stepa(02)
Mensaje D / 00:00:10:000 / stepa(02) / ... para top(01)
Mensaje D / 00:00:10:000 / top(01) / ... para Root(00)
Mensaje X / 00:00:25:000 / Root(00) / nad / 2.00000 para top(01)
Mensaje X / 00:00:25:000 / top(01) / nad / 2.00000 para stepa(02)
Mensaje X / 00:00:25:000 / top(01) / nad / 2.00000 para stepb3(05)
Mensaje X / 00:00:25:000 / top(01) / nad / 2.00000 para stepb4(06)
Mensaje X / 00:00:25:000 / top(01) / nad / 2.00000 para stepb8(10)
Mensaje D / 00:00:25:000 / stepa(02) / 00:00:00:000 para top(01)
Mensaje D / 00:00:25:000 / stepb3(05) / 00:00:00:000 para top(01)
Mensaje D / 00:00:25:000 / stepb4(06) / 00:00:00:000 para top(01)
Mensaje D / 00:00:25:000 / stepb8(10) / 00:00:00:000 para top(01)
Mensaje D / 00:00:25:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:25:000 / Root(00) para top(01)
Mensaje * / 00:00:25:000 / top(01) para stepa(02)
Mensaje D / 00:00:25:000 / stepa(02) / ... para top(01)
Mensaje D / 00:00:25:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:25:000 / Root(00) para top(01)
Mensaje * / 00:00:25:000 / top(01) para stepb3(05)
Mensaje D / 00:00:25:000 / stepb3(05) / ... para top(01)
Mensaje D / 00:00:25:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:25:000 / Root(00) para top(01)
Mensaje * / 00:00:25:000 / top(01) para stepb4(06)
Mensaje D / 00:00:25:000 / stepb4(06) / ... para top(01)
Mensaje D / 00:00:25:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:25:000 / Root(00) para top(01)
Mensaje * / 00:00:25:000 / top(01) para stepb8(10)
Mensaje D / 00:00:25:000 / stepb8(10) / ... para top(01)
Mensaje D / 00:00:25:000 / top(01) / ... para Root(00)
Mensaje X / 00:00:40:000 / Root(00) / h2o / 2.00000 para top(01)
Mensaje X / 00:00:40:000 / top(01) / h2o / 2.00000 para stepb1(03)
Mensaje X / 00:00:40:000 / top(01) / h2o / 2.00000 para stepb7(09)

Mensaje D / 00:00:40:000 / stepb1(03) / 00:00:00:000 para top(01)
Mensaje D / 00:00:40:000 / stepb7(09) / 00:00:00:000 para top(01)
Mensaje D / 00:00:40:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:40:000 / Root(00) para top(01)
Mensaje * / 00:00:40:000 / top(01) para stepb1(03)
Mensaje D / 00:00:40:000 / stepb1(03) / ... para top(01)
Mensaje D / 00:00:40:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:40:000 / Root(00) para top(01)
Mensaje * / 00:00:40:000 / top(01) para stepb7(09)
Mensaje D / 00:00:40:000 / stepb7(09) / ... para top(01)
Mensaje D / 00:00:40:000 / top(01) / ... para Root(00)
Mensaje X / 00:00:57:000 / Root(00) / hscoai / 2.00000 para top(01)
Mensaje X / 00:00:57:000 / top(01) / hscoai / 2.00000 para stepa(02)
Mensaje X / 00:00:57:000 / top(01) / hscoai / 2.00000 para stepb4(06)
Mensaje D / 00:00:57:000 / stepa(02) / 00:00:00:000 para top(01)
Mensaje D / 00:00:57:000 / stepb4(06) / 00:00:00:000 para top(01)
Mensaje D / 00:00:57:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:57:000 / Root(00) para top(01)
Mensaje * / 00:00:57:000 / top(01) para stepa(02)
Mensaje D / 00:00:57:000 / stepa(02) / ... para top(01)
Mensaje D / 00:00:57:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:00:57:000 / Root(00) para top(01)
Mensaje * / 00:00:57:000 / top(01) para stepb4(06)
Mensaje D / 00:00:57:000 / stepb4(06) / ... para top(01)
Mensaje D / 00:00:57:000 / top(01) / ... para Root(00)
Mensaje X / 00:01:11:000 / Root(00) / pyruvatedehydrogenase / 1.00000 para top(01)
Mensaje X / 00:01:11:000 / top(01) / pyruvatedehydrogenase / 1.00000 para stepa(02)
Mensaje D / 00:01:11:000 / stepa(02) / 00:00:00:000 para top(01)
Mensaje D / 00:01:11:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:01:11:000 / Root(00) para top(01)
Mensaje * / 00:01:11:000 / top(01) para stepa(02)
Mensaje D / 00:01:11:000 / stepa(02) / 00:00:00:000 para top(01)
Mensaje D / 00:01:11:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:01:11:000 / Root(00) para top(01)
Mensaje * / 00:01:11:000 / top(01) para stepa(02)
Mensaje Y / 00:01:11:000 / stepa(02) / acetyl_coa / 1.00000 para top(01)
Mensaje Y / 00:01:11:000 / stepa(02) / nadh / 1.00000 para top(01)
Mensaje Y / 00:01:11:000 / stepa(02) / co2 / 1.00000 para top(01)
Mensaje Y / 00:01:11:000 / stepa(02) / h / 1.00000 para top(01)
Mensaje D / 00:01:11:000 / stepa(02) / ... para top(01)
Mensaje X / 00:01:11:000 / top(01) / acetyl_coa / 1.00000 para stepb1(03)
Mensaje Y / 00:01:11:000 / top(01) / nadh / 1.00000 para Root(00)
Mensaje Y / 00:01:11:000 / top(01) / co2 / 1.00000 para Root(00)
Mensaje Y / 00:01:11:000 / top(01) / h / 1.00000 para Root(00)
Mensaje D / 00:01:11:000 / stepb1(03) / 00:00:00:000 para top(01)
Mensaje D / 00:01:11:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:01:11:000 / Root(00) para top(01)
Mensaje * / 00:01:11:000 / top(01) para stepb1(03)
Mensaje D / 00:01:11:000 / stepb1(03) / ... para top(01)
Mensaje D / 00:01:11:000 / top(01) / ... para Root(00)
Mensaje X / 00:01:14:000 / Root(00) / citratesynthase / 1.00000 para top(01)
Mensaje X / 00:01:14:000 / top(01) / citratesynthase / 1.00000 para stepb1(03)
Mensaje D / 00:01:14:000 / stepb1(03) / 00:00:00:000 para top(01)
Mensaje D / 00:01:14:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:01:14:000 / Root(00) para top(01)

Mensaje * / 00:01:14:000 / top(01) para stepb1(03)
Mensaje D / 00:01:14:000 / stepb1(03) / ... para top(01)
Mensaje D / 00:01:14:000 / top(01) / ... para Root(00)
Mensaje X / 00:02:01:000 / Root(00) / oxaloacetate / 1.00000 para top(01)
Mensaje X / 00:02:01:000 / top(01) / oxaloacetate / 1.00000 para stepb1(03)
Mensaje D / 00:02:01:000 / stepb1(03) / 00:00:00:000 para top(01)
Mensaje D / 00:02:01:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:02:01:000 / Root(00) para top(01)
Mensaje * / 00:02:01:000 / top(01) para stepb1(03)
Mensaje D / 00:02:01:000 / stepb1(03) / 00:00:00:000 para top(01)
Mensaje D / 00:02:01:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:02:01:000 / Root(00) para top(01)
Mensaje * / 00:02:01:000 / top(01) para stepb1(03)
Mensaje Y / 00:02:01:000 / stepb1(03) / citrate / 1.00000 para top(01)
Mensaje Y / 00:02:01:000 / stepb1(03) / hscoao / 1.00000 para top(01)
Mensaje Y / 00:02:01:000 / stepb1(03) / h / 1.00000 para top(01)
Mensaje D / 00:02:01:000 / stepb1(03) / ... para top(01)
Mensaje X / 00:02:01:000 / top(01) / citrate / 1.00000 para stepb2(04)
Mensaje Y / 00:02:01:000 / top(01) / hscoao / 1.00000 para Root(00)
Mensaje Y / 00:02:01:000 / top(01) / h / 1.00000 para Root(00)
Mensaje D / 00:02:01:000 / stepb2(04) / 00:00:00:000 para top(01)
Mensaje D / 00:02:01:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:02:01:000 / Root(00) para top(01)
Mensaje * / 00:02:01:000 / top(01) para stepb2(04)
Mensaje D / 00:02:01:000 / stepb2(04) / ... para top(01)
Mensaje D / 00:02:01:000 / top(01) / ... para Root(00)
Mensaje X / 00:02:14:000 / Root(00) / acontinase / 1.00000 para top(01)
Mensaje X / 00:02:14:000 / top(01) / acontinase / 1.00000 para stepb2(04)
Mensaje D / 00:02:14:000 / stepb2(04) / 00:00:00:000 para top(01)
Mensaje D / 00:02:14:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:02:14:000 / Root(00) para top(01)
Mensaje * / 00:02:14:000 / top(01) para stepb2(04)
Mensaje D / 00:02:14:000 / stepb2(04) / 00:00:00:000 para top(01)
Mensaje D / 00:02:14:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:02:14:000 / Root(00) para top(01)
Mensaje * / 00:02:14:000 / top(01) para stepb2(04)
Mensaje Y / 00:02:14:000 / stepb2(04) / isocitrate / 1.00000 para top(01)
Mensaje D / 00:02:14:000 / stepb2(04) / ... para top(01)
Mensaje X / 00:02:14:000 / top(01) / isocitrate / 1.00000 para stepb3(05)
Mensaje D / 00:02:14:000 / stepb3(05) / 00:00:00:000 para top(01)
Mensaje D / 00:02:14:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:02:14:000 / Root(00) para top(01)
Mensaje * / 00:02:14:000 / top(01) para stepb3(05)
Mensaje D / 00:02:14:000 / stepb3(05) / ... para top(01)
Mensaje D / 00:02:14:000 / top(01) / ... para Root(00)
Mensaje X / 00:02:30:000 / Root(00) / isocitratedehydrogenase / 1.00000 para top(01)
Mensaje X / 00:02:30:000 / top(01) / isocitratedehydrogenase / 1.00000 para stepb3(05)
Mensaje D / 00:02:30:000 / stepb3(05) / 00:00:00:000 para top(01)
Mensaje D / 00:02:30:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje X / 00:02:30:000 / Root(00) / alpha_ketoglutaratedehydrogenase / 1.00000 para top(01)
Mensaje X / 00:02:30:000 / top(01) / alpha_ketoglutaratedehydrogenase / 1.00000 para stepb4(06)
Mensaje D / 00:02:30:000 / stepb4(06) / 00:00:00:000 para top(01)
Mensaje D / 00:02:30:000 / top(01) / 00:00:00:000 para Root(00)

Mensaje * / 00:02:30:000 / Root(00) para top(01)
Mensaje * / 00:02:30:000 / top(01) para stepb3(05)
Mensaje D / 00:02:30:000 / stepb3(05) / 00:00:00:000 para top(01)
Mensaje D / 00:02:30:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:02:30:000 / Root(00) para top(01)
Mensaje * / 00:02:30:000 / top(01) para stepb3(05)
Mensaje Y / 00:02:30:000 / stepb3(05) / alpha_ketoglutarate / 1.00000 para top(01)
Mensaje Y / 00:02:30:000 / stepb3(05) / nadh / 1.00000 para top(01)
Mensaje Y / 00:02:30:000 / stepb3(05) / co2 / 1.00000 para top(01)
Mensaje D / 00:02:30:000 / stepb3(05) / ... para top(01)
Mensaje X / 00:02:30:000 / top(01) / alpha_ketoglutarate / 1.00000 para stepb4(06)
Mensaje Y / 00:02:30:000 / top(01) / nadh / 1.00000 para Root(00)
Mensaje Y / 00:02:30:000 / top(01) / co2 / 1.00000 para Root(00)
Mensaje D / 00:02:30:000 / stepb4(06) / 00:00:00:000 para top(01)
Mensaje D / 00:02:30:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:02:30:000 / Root(00) para top(01)
Mensaje * / 00:02:30:000 / top(01) para stepb4(06)
Mensaje D / 00:02:30:000 / stepb4(06) / 00:00:00:000 para top(01)
Mensaje D / 00:02:30:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:02:30:000 / Root(00) para top(01)
Mensaje * / 00:02:30:000 / top(01) para stepb4(06)
Mensaje Y / 00:02:30:000 / stepb4(06) / succinyl_coa / 1.00000 para top(01)
Mensaje Y / 00:02:30:000 / stepb4(06) / nadh / 1.00000 para top(01)
Mensaje Y / 00:02:30:000 / stepb4(06) / co2 / 1.00000 para top(01)
Mensaje Y / 00:02:30:000 / stepb4(06) / h / 1.00000 para top(01)
Mensaje D / 00:02:30:000 / stepb4(06) / ... para top(01)
Mensaje X / 00:02:30:000 / top(01) / succinyl_coa / 1.00000 para stepb5(07)
Mensaje Y / 00:02:30:000 / top(01) / nadh / 1.00000 para Root(00)
Mensaje Y / 00:02:30:000 / top(01) / co2 / 1.00000 para Root(00)
Mensaje Y / 00:02:30:000 / top(01) / h / 1.00000 para Root(00)
Mensaje D / 00:02:30:000 / stepb5(07) / 00:00:00:000 para top(01)
Mensaje D / 00:02:30:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:02:30:000 / Root(00) para top(01)
Mensaje * / 00:02:30:000 / top(01) para stepb5(07)
Mensaje D / 00:02:30:000 / stepb5(07) / ... para top(01)
Mensaje D / 00:02:30:000 / top(01) / ... para Root(00)
Mensaje X / 00:02:50:000 / Root(00) / succinylcoa_synthetase / 1.00000 para top(01)
Mensaje X / 00:02:50:000 / top(01) / succinylcoa_synthetase / 1.00000 para stepb5(07)
Mensaje D / 00:02:50:000 / stepb5(07) / 00:00:00:000 para top(01)
Mensaje D / 00:02:50:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:02:50:000 / Root(00) para top(01)
Mensaje * / 00:02:50:000 / top(01) para stepb5(07)
Mensaje D / 00:02:50:000 / stepb5(07) / ... para top(01)
Mensaje D / 00:02:50:000 / top(01) / ... para Root(00)
Mensaje X / 00:03:00:000 / Root(00) / gdp / 1.00000 para top(01)
Mensaje X / 00:03:00:000 / top(01) / gdp / 1.00000 para stepb5(07)
Mensaje D / 00:03:00:000 / stepb5(07) / 00:00:00:000 para top(01)
Mensaje D / 00:03:00:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:00:000 / Root(00) para top(01)
Mensaje * / 00:03:00:000 / top(01) para stepb5(07)
Mensaje D / 00:03:00:000 / stepb5(07) / ... para top(01)
Mensaje D / 00:03:00:000 / top(01) / ... para Root(00)
Mensaje X / 00:03:11:000 / Root(00) / pi / 1.00000 para top(01)
Mensaje X / 00:03:11:000 / top(01) / pi / 1.00000 para stepb5(07)
Mensaje D / 00:03:11:000 / stepb5(07) / 00:00:00:000 para top(01)

Mensaje D / 00:03:11:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:11:000 / Root(00) para top(01)
Mensaje * / 00:03:11:000 / top(01) para stepb5(07)
Mensaje D / 00:03:11:000 / stepb5(07) / 00:00:00:000 para top(01)
Mensaje D / 00:03:11:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:11:000 / Root(00) para top(01)
Mensaje * / 00:03:11:000 / top(01) para stepb5(07)
Mensaje Y / 00:03:11:000 / stepb5(07) / succinate / 1.00000 para top(01)
Mensaje Y / 00:03:11:000 / stepb5(07) / hscoao / 1.00000 para top(01)
Mensaje Y / 00:03:11:000 / stepb5(07) / gtp / 1.00000 para top(01)
Mensaje D / 00:03:11:000 / stepb5(07) / ... para top(01)
Mensaje X / 00:03:11:000 / top(01) / succinate / 1.00000 para stepb6(08)
Mensaje Y / 00:03:11:000 / top(01) / hscoao / 1.00000 para Root(00)
Mensaje Y / 00:03:11:000 / top(01) / gtp / 1.00000 para Root(00)
Mensaje D / 00:03:11:000 / stepb6(08) / 00:00:00:000 para top(01)
Mensaje D / 00:03:11:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:11:000 / Root(00) para top(01)
Mensaje * / 00:03:11:000 / top(01) para stepb6(08)
Mensaje D / 00:03:11:000 / stepb6(08) / ... para top(01)
Mensaje D / 00:03:11:000 / top(01) / ... para Root(00)
Mensaje X / 00:03:15:000 / Root(00) / fad / 1.00000 para top(01)
Mensaje X / 00:03:15:000 / top(01) / fad / 1.00000 para stepb6(08)
Mensaje D / 00:03:15:000 / stepb6(08) / 00:00:00:000 para top(01)
Mensaje D / 00:03:15:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:15:000 / Root(00) para top(01)
Mensaje * / 00:03:15:000 / top(01) para stepb6(08)
Mensaje D / 00:03:15:000 / stepb6(08) / ... para top(01)
Mensaje D / 00:03:15:000 / top(01) / ... para Root(00)
Mensaje X / 00:03:28:000 / Root(00) / succinatedehydrogenase / 1.00000 para top(01)
Mensaje X / 00:03:28:000 / top(01) / succinatedehydrogenase / 1.00000 para stepb6(08)
Mensaje D / 00:03:28:000 / stepb6(08) / 00:00:00:000 para top(01)
Mensaje D / 00:03:28:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:28:000 / Root(00) para top(01)
Mensaje * / 00:03:28:000 / top(01) para stepb6(08)
Mensaje D / 00:03:28:000 / stepb6(08) / 00:00:00:000 para top(01)
Mensaje D / 00:03:28:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:28:000 / Root(00) para top(01)
Mensaje * / 00:03:28:000 / top(01) para stepb6(08)
Mensaje Y / 00:03:28:000 / stepb6(08) / fumarate / 1.00000 para top(01)
Mensaje Y / 00:03:28:000 / stepb6(08) / fadh2 / 1.00000 para top(01)
Mensaje D / 00:03:28:000 / stepb6(08) / ... para top(01)
Mensaje X / 00:03:28:000 / top(01) / fumarate / 1.00000 para stepb7(09)
Mensaje Y / 00:03:28:000 / top(01) / fadh2 / 1.00000 para Root(00)
Mensaje D / 00:03:28:000 / stepb7(09) / 00:00:00:000 para top(01)
Mensaje D / 00:03:28:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:28:000 / Root(00) para top(01)
Mensaje * / 00:03:28:000 / top(01) para stepb7(09)
Mensaje D / 00:03:28:000 / stepb7(09) / ... para top(01)
Mensaje D / 00:03:28:000 / top(01) / ... para Root(00)
Mensaje X / 00:03:40:000 / Root(00) / fumarase / 1.00000 para top(01)
Mensaje X / 00:03:40:000 / top(01) / fumarase / 1.00000 para stepb7(09)
Mensaje D / 00:03:40:000 / stepb7(09) / 00:00:00:000 para top(01)
Mensaje D / 00:03:40:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:40:000 / Root(00) para top(01)
Mensaje * / 00:03:40:000 / top(01) para stepb7(09)

Mensaje D / 00:03:40:000 / stepb7(09) / 00:00:00:000 para top(01)
Mensaje D / 00:03:40:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:40:000 / Root(00) para top(01)
Mensaje * / 00:03:40:000 / top(01) para stepb7(09)
Mensaje Y / 00:03:40:000 / stepb7(09) / malate / 1.00000 para top(01)
Mensaje D / 00:03:40:000 / stepb7(09) / ... para top(01)
Mensaje X / 00:03:40:000 / top(01) / malate / 1.00000 para stepb8(10)
Mensaje D / 00:03:40:000 / stepb8(10) / 00:00:00:000 para top(01)
Mensaje D / 00:03:40:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:40:000 / Root(00) para top(01)
Mensaje * / 00:03:40:000 / top(01) para stepb8(10)
Mensaje D / 00:03:40:000 / stepb8(10) / ... para top(01)
Mensaje D / 00:03:40:000 / top(01) / ... para Root(00)
Mensaje X / 00:03:45:000 / Root(00) / malatedehydrogenase / 1.00000 para top(01)
Mensaje X / 00:03:45:000 / top(01) / malatedehydrogenase / 1.00000 para stepb8(10)
Mensaje D / 00:03:45:000 / stepb8(10) / 00:00:00:000 para top(01)
Mensaje D / 00:03:45:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:45:000 / Root(00) para top(01)
Mensaje * / 00:03:45:000 / top(01) para stepb8(10)
Mensaje D / 00:03:45:000 / stepb8(10) / 00:00:00:000 para top(01)
Mensaje D / 00:03:45:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:45:000 / Root(00) para top(01)
Mensaje * / 00:03:45:000 / top(01) para stepb8(10)
Mensaje Y / 00:03:45:000 / stepb8(10) / oxaloacetate / 1.00000 para top(01)
Mensaje Y / 00:03:45:000 / stepb8(10) / nadh / 1.00000 para top(01)
Mensaje Y / 00:03:45:000 / stepb8(10) / h / 1.00000 para top(01)
Mensaje D / 00:03:45:000 / stepb8(10) / ... para top(01)
Mensaje X / 00:03:45:000 / top(01) / oxaloacetate / 1.00000 para stepb1(03)
Mensaje Y / 00:03:45:000 / top(01) / nadh / 1.00000 para Root(00)
Mensaje Y / 00:03:45:000 / top(01) / h / 1.00000 para Root(00)
Mensaje D / 00:03:45:000 / stepb1(03) / 00:00:00:000 para top(01)
Mensaje D / 00:03:45:000 / top(01) / 00:00:00:000 para Root(00)
Mensaje * / 00:03:45:000 / Root(00) para top(01)
Mensaje * / 00:03:45:000 / top(01) para stepb1(03)
Mensaje D / 00:03:45:000 / stepb1(03) / ... para top(01)
Mensaje D / 00:03:45:000 / top(01) / ... para Root(00)

REFERENCES

[1] "Biochemistry: 3. Cell Energetics, the Mitochondrion and the Chloroplast 4. Metabolic Pathways", A Second Level Course prepared by an Open University course team.

[2] "The Silicon Cell: Computing the Living Cell", { http://www.bio.vu.nl/hwconf/Silicon/index.html}

[3] Ameghino J., Wainer G., and Glinsky E., "Applying Cell-DEVS in Models of Complex Systems", Dept. of Systems and Computing Engineering, Carleton University, Ottawa.

[4] Benedikt Westermann, "Merging Mitochondria Matters: Cellular Role and Molecular Machinery of Mitochondrial Fusion", Institut für physiologische Chemie der Universität München, München, Germany, April 5 2002.

[5] Bruce Alberts, Dennis Bray, Julian Lewis, Martin Raff, Keith Roberts and James D. Watson, "Molecular Biology of THE CELL", Third Edition.

[6] Cell Based Discrete-Event Simulation Website, http://www.sce.carleton.ca/faculty/wainer/wbgraf/index.html

[7] Christen G., Dobniewski A., and Wainer G. "Defining DEVS models with the CD++ toolkit". In *Proceedings of European Simulation Symposium*. Marseilles, France. 2001.

[8] Dennis Normile, "Complex Systems: Building Wokring Cells 'in Silico'", Volume 284, Number 5411 Issue of 2 Apr 1999, pp.80-81, ©1999 by the American Association for the advancement of Science.

[9] Glinsky J. E. and Wainer G., "Definition of Real-Time Simulation in the CD++ Toolkit", Dept. of Systems and Computer Engineering, Carleton University.

[10] Glinsky, E.; Wainer, G. 2002. "Performance Analysis of Real-Time DEVS models". In *Proceedings of 2002 Winter Simulation Conference*. San Diego, U.S.A.

[11] Hames B.D. and Hooper N.M., "Biochemistry", *Instant Notes*, 2nd Edition.

[12] Helena Curtis and N. Sue Barnes, “Biology”, Fifth Edition.

[13] http://encarta.msn.com

[14] I. Edward Alcamo, Ph.D., “Anatomy and Physiology, the Easy Way”, Barron’s Educational Series, Inc.

[15] Immo E. Scheffler, “Mitochondria”.

[16] Jerry Banks, John S. Carson I, Barry L. Nelson and David M. Nicol, "Discrete-Event System Simulation", Prentice-Hall International Series.

[17] Joanna Poulton, University of Oxford, Oxford, UK, and Laurence Bindoff, Haukeland Hospital, Bergen, Norway, “Mitochondrial Respiratory Chain Disorders”.

[18] Johannes Herrmann, Tom Lutz, Stephan Meier, Marc Preuss, Gregor Szyrach, and Frank Baumann, “Protein insertion into the inner membrane of mitochondria”, Proceedings (poster or lecture-abstract) of the joint annual fall meeting, German Society for Biochemistry and Molecular Biology (GBM) & German Society for Experimental and Clinical Pharmacology and Toxicology (DGPT), Halle (Saale), Germany, September 7-10, 2002 (www.gbm-online.de)

[19] Lidan Li, Gabriel A. Wainer, and Trevor Pearce, “Hardware in the Loop Simulation Using Real-Time CD++”, Dept. of Systems and Computer Engineering, Carleton University.

[20] MacDougall M. H., “Simulating Computer Systems, Techniques and Tools”, the MIT Press 1987.

[21] Marie Aimar-Beurton, Bernard Korzenierwski, Thierry Letekkier, Stephane Ludinard, Jean-Pierre Mazat and Christine Nazaret, “Virtual Mitochondria: Metabolic Modelling and Control”, Molecular Biology Reports 29: 227-232, 2002.

[22] Naim A. Kheir, “Systems Modeling and Computer Simulation”, 2nd Edition, 1996.

[23] Peter E Thorsness and Theodor Hanekamp, University of Wyoming, Laramie, Wyoming, USA, “Mitochondria: Origin”.

[24] Rodriquez A. Daniel and Wainer A. Gabriel, “CD++, User’s Guide”, Departmento de Computacion, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Argentina, 1999.

[25] Stefan Krauss, Beth Israel Deaconess Medical Center and Harvard Medical School, USA, “Mitochondria: Structure and Role in Respiration”.

[26] Takahashi K, Kaizu K, Hu B, Tomita M., "A Multi-algorithm, Multi-timescale Method for Cell Simulation", Institute for Advanced Biosciences, Keio University, Fujisawa, Kanagawa, 252-8520, Japan, Bioinformatics. 2004 Mar 1; 20(4):538-46. Epub 2004 Jan 22.

[27] Tomita M., "E-Cell Project: Towards Whole Cell Simulation", Institute fro Advanced Biosciences, Keio University.

[28] Tomita, M., Hashimoto, K., Takahashi, K, Shimizu, T., Matsyzki, Y. Miyoshi, F. Saito, K., Tanida, S., Yugi, K., Venter, J.C, & Hutchison, C., "E-Cell: Software Environment for Whole Cell Simulation", Bioinformatics 15, 72-84 (1999).

[29] Troccoli, A.; Wainer, G. "Implementing Parallel Cell-DEVS". In *Proceedings of Annual Simulation Symposium.* Orlando, FL. U.S.A. 2003.

[30] Wenhong Chen, M. Eng., "A Facility for Remote Execution and Visualization of DEVS and CELL-DEVS Models", Master of Science Thesis for Information and System Science, May 2003.

[31] Wainer A. Gabriel, "Discrete Events Cellular Models with Explicit Delays", Doctorate Thesis in Computer Engineering, Université D'Aix-Marseille III, France. December 1998.

[32] Wainer A. Gabriel, Wu Jason and Wang Yolanda, "Manual of the GUI for the CD++ Tool", Computer Science Department, Carleton University, Ottawa, 2001.

[33] Wainer A. Gabriel, Chen Wenhong, Cidre Juan Ignacio, Glinsky Ezequiel, Leon Steve, Monadi Ali and Troccoli Alejandro, "CD++: A tool for and DEVS and Cell-DEVS Modelling and Simulation User's Guide", Department of Systems and Computer Engineering, Carleton University, Draft – August 2004.

[34] Wainer G., "CD++: a toolkit to define discrete-event models". Wainer. *Software, Practice and Experience*. Wiley. Vol. 32, No.3. pp. 1261-1306. November 2002.

[35] Yu H and Wainer G. "Embedded CD++". Carleton University Technical Report SCE-03-04. 2004.

[36] Yugi, K., and Tomita, M., "A General Computational Model of Mitochondrial Metabolism in a Whole Organelle Scale", Institute for Advanced Biosciences, Keio University, Endo 5322, Fujisawa, Kanagawa, 252-8520, Japan Bioinformatics, 2004 Feb 12.

[37] Yugi K., Kanai y. and Tomita M., "Quantitative Modeling of Mitochondrial Energy Metabolism using E-CELL Simulation Environment", Genome Informatics 2000, 456-457 (Universal Academy Press;2000)

[38] Yugi K., Kanai y. and Tomita M., “Kinetic Modeling of Energy Metabolism in Mitochondria”, Genome Informatics 1999, 368-369 (Universal Academy Press;1999)

[39] Zeigler B., “Theory of Modeling and Simulation”, Wiley, 1976.

FURTHER READING

[1] Edwards M. Mills, Matthew L. Banks, Jon E. Sprangue, snf Yotrn Ginkrl, "Uncoupling the agony from ectasy: A mitochondrial protein may mediate a dangerous side-effect of some recreational drugs." Cardiovascular Branch, NHLBI, National Institutes of Health, Bethesda, Maryland 10892-1622, and Department of Pharmaceutical and Biomedical Sciences, Ohio Northern University, ADA, OHIA 45810, USA. Nature, Vol 426, Nov 2003.

[2] Gottfried Schatz, President, Swiss Science and Technology Council, "What Mitochondria have told me".

[3] Joubert, F., Hoerer, J.A. And Mazert, J.–L "Modeling the energy transfer pathways. Creatine kinase activities and heterogeneous distribution of ADP in the perfused heart", U446 INSERM, Université Paris, 92260, Chatenay-Malabry, France.

[4] Katrin Henze and William Martin, "Essence of Mitochondria".

[5] E-cell Project, Project Cybercell, and Silicon Cell Project websites.