NOTE TO USERS

uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Roxana Djafarzadeh
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Modeling and Simulation of Cellular Metabolism and Energy Production by Mitochondria
TITRE DE LA THÈSE / TITLE OF THESIS

T. Mussivand
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

G. Wainer
CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

P. Liu

S. Tavoularis

Gary W. Slater
LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

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

FEBRUARY 2005

Your file Votre référence
ISBN: 0-494-11257-3
Our file Notre référence
ISBN: 0-494-11257-3

ABSTRACT

It is the intent of this thesis to investigate and develop a simulation model for metabolic pathways in the cells, namely, Glycolysis and Krebs cycle, using the DEVS formalism and the CD++ tool, then to further improve it to complete virtual mitochondrion. The hierarchical nature of DEVS makes it ideal for describing naturally hierarchical systems as the Cell, while its discrete-event approach improves performance due to the asynchronous nature of the events occurring in the cell. Simultaneously, as DEVS is a timed-based modeling approach, timing of the chemical reactions can be adequately represented. The models were developed using the CD++ toolkit, a modeling tool for simulation of complex physical systems that can be used to simulate a variety of models. CD++ server permits the execution and visualization of the results with sophisticated and easy-to-use Graphical User Interfaces. A precise and easy to use simulation environment for biological models of glycolysis and Krebs cycle was created, and the results presented show the potential of this approach.

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. Thank you.

I also thank Miss A. Khan for her help to create the animation of my models using MAYA software application.

Many thanks to my brother, Siamak Djafarzadeh (Ph. D., Faculty of Medicine in Bern, Switzerland), for always encouraging me and helping me in biology and biochemistry fields.

Special thanks to Dr. G. Schatz (The President of Swiss Science and Technology Council), the father of mitochondrial research, whose technical and non-technical papers were of great help.

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

And last, but not least, I thank my cats, Jasmine and Poupée, for their unconditional love, Sr. Dolores Lepage for always believing in me, Dr. Sitwell (my neurologist) for taking care of my migraines, TuNhi Tran for always being there for a cup of coffee and spiritual talk, Yan Robichaud for preparing me for my defence, and everyone else who helped me in this process.

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

LIST OF ABBREVIATIONS

ADP	Adenosine DiPhosphate
API	Application Program Interface
ATP	Adenosine TriPhosphate
CD++	A Modeling and Simulation Toolkit
DEVS	Discrete EVent 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

LIST OF TABLES

CHAPTER 1 – INTRODUCTION

Simulation is becoming increasingly important in the analysis and design of complex systems such as biological processes. Biological model's abstract representation of a real system could be defined with mathematical notations and can be analyzed using mathematical techniques. However, the complexity of biological systems sometimes makes it impossible to use such analytical methods. When studying these complex systems, simulation can be a useful technique, often the only one available practically.

Modeling and simulation of any biological entity can be useful for pharmaceutical companies in designing new drugs. This plays a key role as different cases can be tested saving time and therefore resources, whereas this possibility to explore is not always doable in wet-lab environment, or with real subjects, which makes simulation even more appealing. By using simulation and modeling, a pharmaceutical researcher could explore the impact of a new drug on a specific cell with varying the proportion between various compounds in that drug. In designing a particular drug, the researchers can simulate and see the end result immediately, and change the model accordingly. Simulation can help them to understand how different proportions can be, or in case of studying a disease, the researcher can examine what the absence of a chemical could result in. In this thesis, the energy production aspect of mitochondria was studied.

Mitochondrion was chosen as the organelle to be simulated as it represents a mini-cell and can be later expanded to a complete cell simulation. Mitochondria, the powerhouses of

the cell, are the only organelles in the cell to contain their own DNA making them good candidates for simulation. Moreover, they have been found to have a major role in diseases such as Parkinson, Alzheimer, Diabetes Mellitus, and weight loss [17].

There have already been some efforts in the past to simulate different metabolic pathways, biological entities such as cell, and even mitochondria. Nevertheless, none of the available tools has had real application in real life. As there are already proven classical models of different metabolic pathways, one can easily verify the results of the simulations with those proven results. To achieve this, glycolysis was studied and modeled as the pathway feeding to mitochondria. Afterwards, mitochondrion was broken down into Krebs cycle and Electron Transport Chain modules. The Krebs cycle was modeled and simulated thoroughly but electron transport chain was left as research in progress.

1.1 OBJECTIVES

The goal of this research is to study the applicability of recent advanced simulation and modeling techniques to analyze biological systems. We use the Discrete Event Systems specification (DEVS) formalism [30] to define several models that represent the biological systems. This thesis addresses one solution for simulating complex biological models: designing and validating a DEVS biological model taking advantage of the DEVS formalism. To develop and test our biological simulation framework, we modeled and simulated the dynamics of mitochondrial metabolism in a whole organelle scale. For this purpose, a mitochondrial model was constructed which included the biological pathways – the glycolysis and the Krebs cycle. Currently, the model is examined with emphasis on cellular

metabolism and energy production aspect of mitochondria based on the two mentioned biological pathways but later on, it can be expanded to include every aspect of it.

The end goal is to deliver a practical tool that biologists can use to model their own problems. We can use all available DEVS simulation tools, create reusable building blocks that can be integrated with already existing DEVS models, and be easily used by biologists.

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. The delivered system meets the following goals: robustness, easy to use and test, and abstraction level that it offers to the biologists. One of the main goals making this system as easy to use as possible for researchers in medical field, to study the appropriateness of this method, expandability to more complex systems, and finally, to ensure that these tools are capable of meeting all the mentioned objectives.

1.2 APPROACH

DEVS is used to attack this complex biological model, allowing for the definition and integration of individual components. DEVS also provides a formal specification framework, which allowed reduction of testing time and improvement of the development process. With this in mind, we have approached this project considering the ability to describe the multiple abstraction levels, the capacity of defining different components using a unique approach, the extensibility of the components, their modifiability, and good testing facilities.

There are already many DEVS simulation tools to choose from. The DEVS formalism allows designing hierarchical models. It also allows creating reusable building blocks that can be integrated with already existing DEVS models, where these blocks can be easily used or modified by biologists [30].

The application of this paradigm is shown through modeling of these complex biological systems developed in CD++. The CD++ tool was used to implement the DEVS models because it provides a general framework to define and simulate complex generic models. The use of this simulation toolkit provided good results for most of our goals. Our proposed model can be extended or modified easily, tested and validated. Nevertheless, a significant computational challenge is how we can integrate such sub-cellular models running on different types of algorithms to construct higher order models.

1.3 THESIS PLAN

The thesis is structured as follows:

- 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, related work, DEVS formalism, and then CD++ toolkit, which was used as the simulation tool.

- Chapter 4 presents the proposed model and simulation results of Glycolysis using DEVS formalism and CD++ tool.

- Chapter 5 presents the proposed Krebs cycle model and simulation results using DEVS formalism and CD++ tool.

- Chapter 6 describes all the graphical user interfaces already available and the ones developed for our purpose.

- Chapter 7 concludes 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 ribosome (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 and brain cells, have more mitochondria than other cells [9, 17].

2.1.1 MITOCHONDRIAL STRUCTURE

A mitochondrion is typically ranging in size from 0.5 to 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).

[1] Eukaryote: Cells with a nucleus

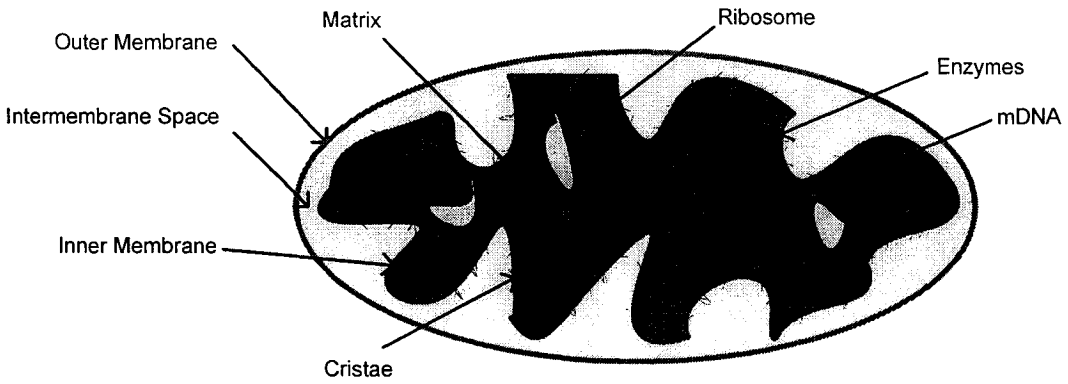


Figure 2.1 – 2-D Model of Mitochondria

The smooth outer membrane holds numerous transport proteins, which shuttle materials in and out of the mitochondrion. The components between the outer and inner membranes have important roles in electron transport and oxidative phosphorylation. The inner membrane has many folds called cristae. 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 [9, 11, and 13].

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].

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], each molecule of 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.

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 [7, 9, and 13].

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 [19].

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 [19].

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 [19].

---

[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.

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 co-activator. Those mitochondria also serve as a source for reactive oxygen species that contribute to cell death during apoptosis [13, 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 [17].

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 [7].

The role of glycolysis (glycol: 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 two ATP at the end of glycolysis. Glycolysis itself does not require oxygen [7].

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-diphosphate). At this point, the molecule splits into two 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) [7].

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

Table 1 – First Phase: Preparatory Steps [8]

Step 1	Glucose enters the cell and is phosphorylated on the number six carbon.	• 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.	• 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.	• 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 glyceraldehyde 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) [7].

Table 2 – Second Phase: Oxidative Steps [8]

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. [7]

Table 3 – Inputs, Outputs, and Enzymes Involved in Glycolysis [8]

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

Table 3 gives a summary of ten steps of Glycolysis, the inputs and outputs of each step, and enzymes involved in each step. 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 four 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 Krebs' 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 [8].

Table 4 – Summary of Krebs' Cycle [8]

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				

Table 4 gives a summary of different steps in Krebs' cycle, the inputs and outputs of each step, and enzymes involved in each step.

2.3.1 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 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 accumulate [5, 8].

2.3.2 ENERGY YIELD

From each turn of Krebs cycle, 12 ATP molecules are produced: one directly from the cycle, nine from three NADH, and two 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 membranes 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 $FADH_2$ [9].

Electron transport and oxidative phosphorylation occur in the inner membrane of mitochondria. These processes reoxidize the NADH and FADH2 that are generated from

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

Krebs cycle and glycolysis, and trap the energy released as ATP. Oxidative phosphorylation is by far the major source of ATP in the cell [9].

NAD and FAD remove the electrons that are donated during some of the steps of the Krebs' or Citric acid cycle. Then, they carry the electrons to the electron transport pumps and donate them to the pumps. Therefore, NAD and FAD are "oxidized" because they lose the hydrogen ions to the pumps. The pumps then transport the hydrogen 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. 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.

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

Table 5 – Five 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. Figure 2.2 shows the five complexes across the inner membrane of mitochondria [9].

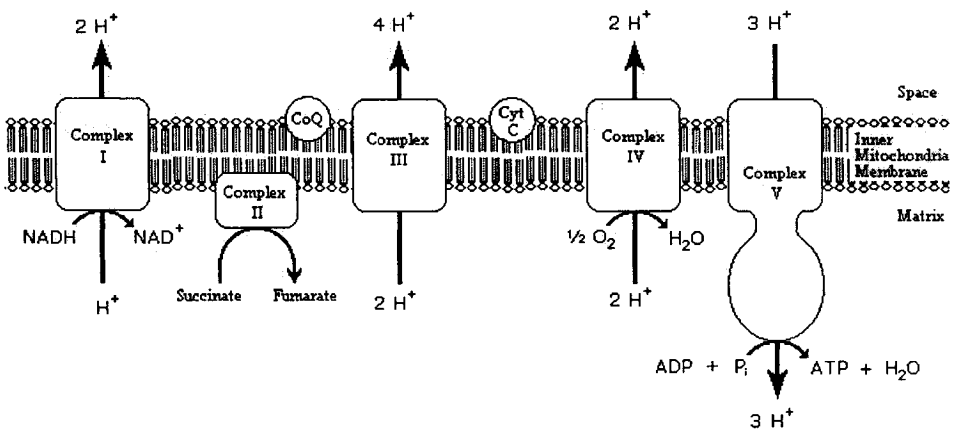


Figure 2.2 – Oxidative Phosphorylation Using 5 Complexes [Source: http://www.gwu.edy/~mpb/oxidativephos.htm]

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 [9, 11].

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 and tending to establish an electrochemical potential [9].

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 V is the terminus of the electron transfer chain, consuming oxygen and making water. Cytochrome c^5 is also an

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 [9].

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 [9].

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 Krebs' cycle runs, hydrogen ions (or electrons) are donated to the two carrier molecules in four 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 [9].

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 [9].

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

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.

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 [9, 11]. Figure 2.3 shows the flow of electrons during Oxidative Phosphorylation.

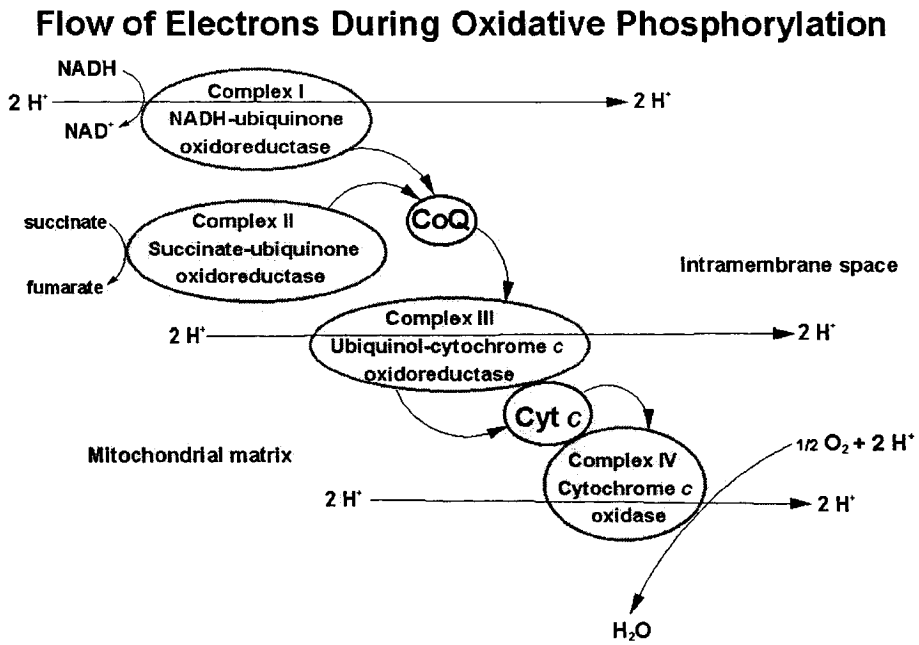


Figure 2.3 – Flow of Electrons During Oxidative Phosphorylation [Source: 2001 M. W. King]

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[7] (aka the F_0F_1 particle) generates Adenosine TriPhosphate (ATP) [9].

[7] 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 → ATP. These enzymes are of crucial importance in almost all organisms, because ATP is the common "energy currency" of cells.

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 inter-membrane space against the gradient, they now can flow back into the mitochondrial matrix via the ATP synthase, generating ATP in the process [9, 11].

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

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.

Electron Transport and oxidative phosphorylation reoxidize NADH and $FADH_2$ and trap the energy released as ATP by the stepwise transfer of electrons from these molecules to oxygen 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.

2.4.3 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 [9, 11].

2.4.4 KEY POINTS IN ELECTRON TRANSPORT CHAIN [8]

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 ETC, the proton gradient increases with H^+ building up outside the inner mitochondrial membrane and OH^- inside the membrane.

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. The function of mitochondria is to get as much ATP out of glucose or other food products as possible. If we have no oxygen, we only get four molecules of ATP energy packets for each glucose molecule during 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 four 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 (see Table 6).

Table 6 – Total ATPs Generated in the Process of the Oxidation of One Molecule of Glucose [7]

In the Cytoplasm		
Glycolysis:	2 ATP	2 ATP
In the Mitochondrion		
Glycolysis:	2 NADH → 6 ATP	6 ATP[8]
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

Mitochondria's main function is converting the potential energy of food molecules into ATP. Figure 2.4 shows the location of the metabolic pathways:

[8] 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.

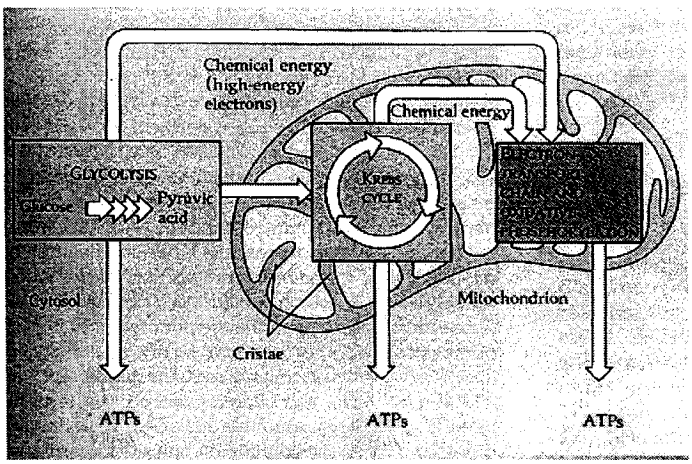


Figure 2.4 – Metabolic Pathways from Glycolysis to Electron Transport Chain [Source: Natural Toxins Research Center at Texas A&M University – Kingsville]

The following table summarizes metabolic pathways needed to produce energy for the body. Glycolysis, the bridging step and Krebs cycle are modeled, while electron transport chain is left as a research in progress.

Table 7 – A Summary of Steps, Location, Inputs, and Outputs

Phase	Location	Summary	Inputs	Outputs
Glycolysis	Cytoplasm	Glucose is degraded to pyruvate, producing 2 ATP molecules and hydrogens; can proceed anaerobically	Glucose, ATP, NAD^+	Pyruvate, ATP, NADH
Bridging Step	Mitochondria	Pyruvate is degraded and combined with coenzyme A to form acetyl CoA; hydrogens and carbon dioxide are released	Pyruvate, CoA	Acetyl CoA, carbon dioxide, NADH
Krebs Cycle	Mitochondria	Series of reactions in which the acetyl portion of acetyl CoA is degraded to carbon dioxide; hydrogens are released	Acetyl CoA, H_2O	CO_2, NADH, $FADH_2$, GTP
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, $FADH_2$, O_2	ATP, H_2O

CHAPTER 3 – MODELING AND SIMULATION TOOLS

This chapter begins with a general discussion of simulation and bio-simulation. Then it discusses to the DEVS formalism. Later, the CD++ toolkit is introduced with the reasons why it is chosen as our development toolkit for this 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 [10, 16].

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 [14].

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 [30].

A system 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 (biological, 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) [14].

3.3 BIO-SIMULATION

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 behaviour 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 behaviour. 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 codes 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 tetrahedral. 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 [10].

3.4 OVERVIEW OF RELATED EFFORTS

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 [18, 20, and 21] from Keio University, Japan (http://www.e-cell.org), the Virtual Mitochondria project [4] from University of Bordeaux, France, the Cyber-cell [1] (University of Alberta, Alberta, Canada) (http://www.projectcybercell.ca), and the Silicon cell [3] (BioCentrum Amsterdam and IMBS, Vrije Universiteit, Amsterdam, CWI, and the Institute for Informatics of the University of Amsterdam).

3.4.1 E-CELL PROJECT

The E-cell project [18, 20, and 21] was started at Keio University, Japan, at 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 [20].

This model combines the Gillespie-Gibson stochastic algorithm and deterministic differential equations. Dramatic improvements in performance were obtained without significant accuracy drawbacks [20]. 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 [21].

A kinetic model of mitochondrial energy metabolism using E-cell system (a simulation environment for E-cell simulation) has also 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.

Their final goal is to apply this system for analyses of human mitochondrial diseases [27,28, and 29].

3.4.2 VIRTUAL MITOCHONDRIA: METABOLIC MODELLING AND CONTROL

In this Model [4], 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 parameter 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 developed for whole cells in the post-genomic area, i.e. to construct and to analyse the metabolic maps from the genes [4].

3.4.3 THE SILICON CELL: COMPUTING THE LIVING CELL

This project of the University of Amsterdam [3] applies a computationally based methodology to modeling cellular metabolism and regulation that captures the physical and chemical constraints inherently placed on cells, which limit cellular behaviour. The *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 are 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, behaviour, fitness and performance characteristics of cells under changing environmental, physiological and genetic conditions [3].

3.5 THE DEVS FORMALISM

DEVS (Discrete EVent System Specification) is a theoretical approach, which allows the definition of hierarchical and modular models that can be easily reused [30]. Each DEVS model can be behavioural (atomic) or structural (coupled), consisting of inputs, outputs, state variables, and functions to compute the next states and outputs. One of the main advantages of DEVS framework is that it separates modeling from simulation. The DEVS formalism is used to provide enhanced execution speed. A real system modeled using DEVS can be described as a set of communicating atomic or coupled sub-models. 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.

3.5.1 ATOMIC DEVS MODEL

A DEVS atomic model is defined by input ports, output ports, state variable, state transition functions, output function, and time advance function. Atomic models can be used to interact directly with other models or to be part of a DEVS coupled model. These models are added to the tool at compile time, and if a new atomic models need to be defined; they must be coded in C++ and incorporated into CD++ model hierarchy. In the

DEVS formalism, one must specify basic models from which larger ones are built, and the way these models are connected together in hierarchical fashion. In this formalism, basic models are defined by the structure:

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

Where

M: an atomic DEVS model defined by the following:

X: external input events set;

S: sequential state set;

Y: external output events set;

δ_{int}: S $\rightarrow$ S, is the internal transition function;

δ_{ext}: Q * X $\rightarrow$ S, is the external transition function; where Q = { (s, e) | s $\in$ S, and e $\in$ [0, $t_a(s)$]} is the total state set and e: the elapsed time since last state transition;

λ: S $\rightarrow$ Y, is the output function; and

t_a: S $\rightarrow$ $R_0^+ \cup \infty$, is the time advance function.

Each atomic model is provided with an interface consisting of input and output ports that allow the model to communicate with other models. The input events set (X) is made up of all possible inputs that may occur on the input ports through which external events are received. Similarly, the output events set (Y) consists of all possible outputs the atomic model may have and through which external events are sent. In the set of state variables and parameters (S), two state variables are usually present: *phase* and *sigma*. In the absence of external events, the system stays in the current *phase* for the time given by *sigma*. The time advance function (ta) controls the timing of internal transitions. This function returns the value of *sigma* when the *sigma* state variable is present. The internal transition

function (δ_{int}) specifies to which next state system will transit after the time given by the time advance function has elapsed. The external function (δ_{ext}) specifies how the system changes state when an input is received. This function will put the system in a new *phase* and *sigma* scheduling it for a next internal transition. The next state is computed based on the present state, the value of the external event, and the time that has elapsed in the current state. The output function (λ) generates an external output just before an internal transition takes place [23, 24, and 30].

3.5.2 COUPLED DEVS MODEL

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

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

Where:

CM: the Coupled Model;

X: external input event set;

Y: external output event set;

$D \in N, D < \infty$: 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 i and $\forall$ j $\in$ Ii (the models that can be influenced by outputs of model i),

Z_{ij}: $Y_i \rightarrow X_j$, $\forall$ j $\in$ D, Z_{ij} is the i to j output translation function; and

Select: is a function, the tie-breaking selector.

Note that the bracket notation signifies a set of objects. For example, {Mi} denotes that we have a set of atomic models (e.g. {M1, M2, M3 ...}). Each component in a coupled model is identified by an index number. Each model is associated with a set of influencees, defined as those models to which output values must be sent. The translation function uses an index of influencees, created for each model (Ii). This function defines which outputs of model Mi will be converted into inputs for model Mj. When two sub-models have simultaneous events, the *select* function defines which of them should be activated first [30].

Multi-component models are implemented in DEVS as coupled models. Coupled models are defined by a set of basic components, which are interconnected through their model interfaces (input/output ports) together to form a new model. The latter model can itself be a part of a larger coupled model, constructing a hierarchical model. A coupled model can be expressed as an equivalent basic model in the DEVS formalism. Such a basic model can itself be employed in a larger coupled model. This shows that the formalism is closed under coupling as required for hierarchical model construction. Expressing a coupled model (CM) as an equivalent basic model explains how the basic models interact to yield the overall behaviour [30].

A coupled model contains a set of components with a set of influences associated with each component (defined as those models to which output values must be sent), a set of input ports through which external events are receive, and a set of output ports through which external events are sent out. At any event time t, each component i am in a state si and have been there for an elapsed time e_i. The time advance in state si is ta_i (s_i) so that component i is scheduled for an internal event at time t + (ta_i (s_i) - e_i). The next event in the

system will occur at a time which is the minimum of these scheduled times, namely, at time t + σ, where σ is the minimum of the residual times, $(ta_i (s_i) - e_i)$, over the components i in D. Of those components whose remaining times $(ta_i (s_i) - e_i)$ are equal to the minimum, we choose one using the tie breaking select function. Let i* be this selected, or imminent, component. At time t + σ, just before i* changes state, it computes its output $y^* = \lambda_i^*(s_i^*)$. This output is sent to each of the influencees of i* in the form of a translated input: for influencee j, the input, x_{i^*j} is $Z_{i^*j}(y^*)$. The elapsed time at any component i at time t + σ is $e_i' = e_i + \sigma$. An influencee, j responds to the external event generated by i* by applying its external transition function to obtain the next state and to reset its elapsed time to zero. Other components not in the influencee set are unaffected by the activation of i* except that their elapsed time clock is incremented by σ. Finally, the i* executes its internal transition by going to state $s_{i^*}' = \delta_{int}(s_i^*)$ and resetting its elapsed time to zero [30].

3.6 THE CD++ TOOLKIT

The CD++ toolkit was developed in order to implement the theoretical concepts already specified by the DEVS formalism. 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 [25, 26].

3.6.1 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. To generate a new atomic model, a new class that is derived from class Atomic is designed, and the new type of atomic model is added to the method **MainSimulator.registerNewAtomics()**. 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 [24].

- ***initFunction***: method invoked by the simulator at the beginning of the simulation. Before calling this method, the sigma value is infinite and the state is passive.

- ***externalFunction*** **(ExternalMessage &)**: method invoked when an external event arrives in one of the model's output port. It corresponds to the δ_{ext} function of the DEVS formalism.

- ***internalFunction*** **(InternalMessage &)**: method defining the δ_{int} function of the DEVS formalism. Before invoking to this method, the value of *sigma* is zero, since the interval has been completed for the internal transition.

- ***outputFunction*** **(const CollectMessage &)**: Called when an internal event arrives, before calling the internal transition function. It is in charge of transmitting the output events of the model. It corresponds to the λ function of the DEVS formalism. Before invoking to the method the value of *sigma* is zero, since the interval has been completed for the internal transition.

These methods can invoke certain predefined primitives. These primitives are used to define the Atomic model's behaviour and allow the interaction with the abstract simulator [24]:

- ***holdIn*****(state, time):** The model changes its state into *state* by time *time*. It instructs the model to remain in that state for the specified time. When this interval expires, a change of state has to occur (the state could be *active* or *passive*), following which the output and internal transition methods will be invoked.

- ***passivate()***: It is equivalent to ***holdIn*****(passive, infinity)**. It sets the next internal transition time to infinity. It indicates to the simulator that the model enters in passive mode and it will only be activated again if an external event is received.

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

- ***nextChange*****()**: It informs the time remaining before the next change of state (*sigma*).

- ***lastChange*()**: It informs the time of the last change of state.

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

- ***getParameter*(modelName, parameterName)**: It gets the parameter *parameterName* value.

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

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 multi-component model using a specification language specially defined for this purpose [24].

Since Model is an abstract base class, it defines the interface for message exchange, Atomic and Coupled classes are the only ones, which can receive and send messages. The derived classes are responsible for overloading the initialization, internal transition, external transition and output methods. The Atomic derived classes should not send any kind of message, except for the output values informed through the sendOuput method. The Atomic class is responsible for sending the Y and D messages to their parents using the sigma and state values [24].

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 (see table 8 for an example). The following syntax is used [24]:

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 [24].

3.7 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 presents an abstract way of thinking about the world with independence of the simulation mechanisms, underlying hardware and midware [22].

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 existing techniques such as cellular automata, Petri Nets, State Charts, and Queuing models have been mapped to DEVS. This permits sharing information at the level of the model, and different sub-models 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 [23, 30].

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

4.1 THE PATHWAY

As mentioned in chapter 2, the first step in glycolysis is phosphorylation[9] 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-diphosphate). At this point, the molecule splits into two 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 [10, 8]].

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 (1,3-Diphosphoglycerate or DPGA). Next, phosphoglycerate kinase generates a molecule of ATP while forming 3- phosphoglycerate (3-PGA). 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. Phosphoglycerate mutase then converts 3-PGA to 2-phosphoglycerate (2-PGA). Phosphoenolpyruvate (PEP) is then formed in the presence of enolase. Finally, pyruvate and another molecule of ATP are produced with the help of pyruvate kinase. This serves as an additional regulatory step [7].

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 [8].

4.2 ATOMIC MODEL DEFINITION

The glycolysis pathway presented in Figure 4.1 was defined as DEVS model and implemented using CD++. In this section we will show how the model was defined, using

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

as an example, the definition of Step1 of glycolysis in which glucose is phosphorylated by ATP to form glucose 6-phosphate and ADP. The source code for the step one of glycolysis is included in the next section, while the source code for the steps two to ten are documented in appendix A of this paper.

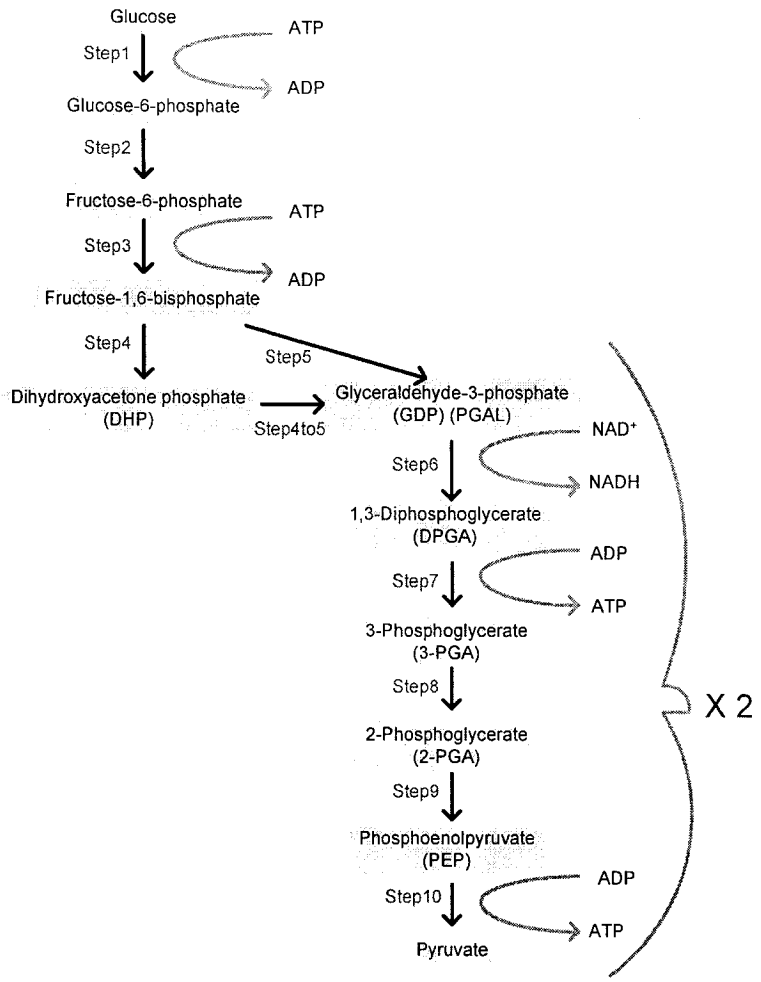


Figure 4.1 – Glycolysis Reactions

4.2.2 STEP 1

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

Step1 modeled using DEVS can be described as an atomic model defined by:

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

where X is the set of external input event types, S is the sequential state set, Y is the set of external event types generated as output, δ_{int} (δ_{ext}) is the internal (external) transition function dictating state transitions due to internal (external input) events, λ is the output function generating external events at the output, and ta is the time-advance function [30]:

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

X = {glucose, ATPi, hexokinase}

Y = {glucose_6_phosphate, ADP, H}

δ_{int} = Internal function (See table 11)

δ_{ext} = External function (See table 9)

ta = Sigma, a real variable that holds the time remaining to the next internal event (in tables 9 and 11, this is represented by the ***holdIn()*** function).

λ = Output function (See table 10)

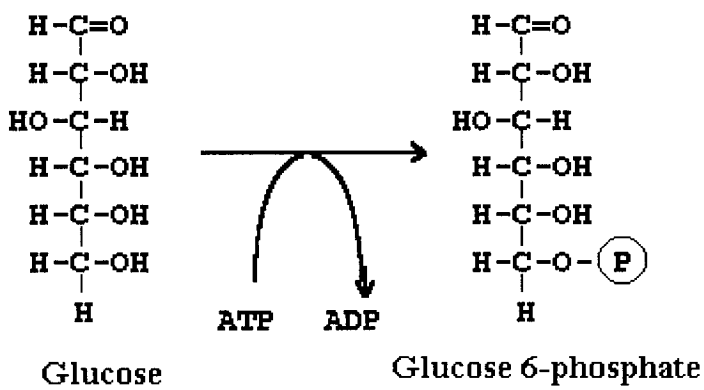


Figure 4.2 – Step1 of Glycolysis [24]

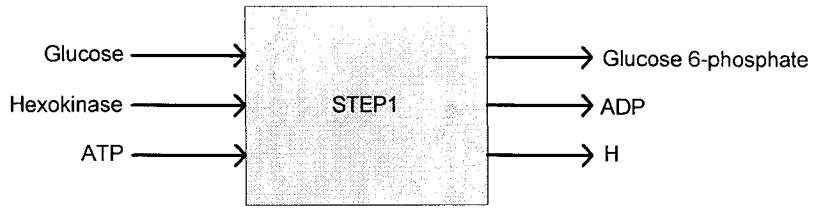


Figure 4.3 - Atomic Model for Step1 of Glycolysis

The external transition function is invoked every time an external event arrives. Here, glucose, ATPi, and hexokinase are the external input events arriving and invoking this method. This function describes the reaction of the step1 to each of the enzymes or glucose, as described in Table 9. When an event arrives at message port, first it is checked against one of the valid input events. Once validated, its value is added to the counter kept for that input event by the Step1 atomic model. Later, this atomic model checks for other requirements of the reaction. Once it receives any of the valid inputs, it increments its value by one of the following counters: ***glucosec*** (the counter representing the number of glucose molecules) and ***atpc*** (the counter representing the number of ATP molecules). In case of ***ifhex*** (the counter for hexokinase), since it is an enzyme and only its presence is needed for the reaction to happen, its value is set to true by its first arrival. Once the value of the counter of the input event is incremented, the condition for the other requirements is checked. For example, if the input event is ***glucose***, the glucose counter value (***glucosec***) is incremented by the value of the glucoses entered, and then it is checked against other input counters: ***atpc*** and ***ifhex***. If there is an ***ATPi*** in the system and ***hexokinase*** as well, then the reaction will happen [`if msg.port() == ATPi ) then atpc = atpc + msg.value();`]. An internal transition will be scheduled after a preparation time, which represents the reaction time.

Table 9 – Step1.cpp: External Transition Function (δ_{ext})

```
Model &Step1::externalFunction( const ExternalMessage &msg ) {
  if( msg.port() == glucose ) {
       glucosec = glucosec + msg.value() ;
       if ( (atpc > 0 ) && (ifhex == true ) )
              holdIn( active, Prep_time );
       else
              holdIn(idle, Time::Zero );
  }
  else if( msg.port() == ATPi ) {
       atpc = atpc + msg.value() ;

       if ( (glucosec>0) && (ifhex==true) )
              holdIn( active, Prep_time );
       else
              holdIn(idle, Time::Zero );
  }
  else if ( msg.port() == hexokinase ) {
       ifhex = true ;

       if ( (glucosec > 0 ) && (atpc > 0) )
              holdIn( active, Prep_time );
       else
              holdIn(idle, Time::Zero );
  }
  return *this;
}
```

When the preparation time interval expires, the output function (λ) is invoked and the first value in Step1 is sent through the corresponding output port (see Table 10). The output must always return a content structure. In this case, the output function is activated when all the conditions of the external function have been satisfied, that is, all three input events are in, and the reaction can happen. As a result, ***ADP***, ***glucose_6_phosphate***, and ***H*** will be sent out through the corresponding output ports.

Table 10 – Step1.cpp: Output Function (λ)

```
Model &Step1::outputFunction(InternalMessage &msg ) {
  if ( counter != 0 )       {
    sendOutput( msg.time(), ADP, counter );
    sendOutput( msg.time(), glucose_6_phosphate, counter );
    sendOutput( msg.time(), H, counter );    }
  return *this ;
}
```

After calling the output function, the internal transition function is invoked (see table 11). The internal transition function first checks the phase of the processor. If ***phase*** equals busy the macro ***passivate*** is used to set the ***phase*** to passive and sigma to infinity. The internal transition function will produce an internal state change according to the substances available in the mitochondria. The function updates the number of substances available according to the reaction, and it then passivates.

Table 11 - Step1.cpp: Internal Transition Function (δ_{int})

```
Model &Step1::internalFunction( const InternalMessage & )   {
    counter = 0;
    if  ( state() == idle ) {
        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 = glucosec = 0;
            }
            holdIn(passive, Time::Zero );
        }
        else {
        passivate();
        }
    }
    return *this ;
}
```

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

Table 12 - Step1.h: Model Definition

```
class Step1 : public Atomic {

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

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

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

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

The constructor performs initialization of parameters and state. It creates the input and output ports and sets the variable *preparationTime*, and initializes the variables to zero (See table 13).

Table 13 - Step1.cpp: Model Constructor

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

The initialization function sets the sequential state set to zero (see table 14).

Table 14 – Step1.cpp: Model Initialization Function

```
Model &Step1::initFunction()
{
        atpc = 0;
        glucosec = 0;
        ifhex = false;
        counter = 0;

        return *this ;
}
```

A new atomic model is created as a new class that inherits from the class Atomic. To tell CD++ that a new atomic definition has been added, the model must be registered in the method ***MainSimulator.registerNewAtomics()***. This method is located in the ***register.cpp***, which is shown in the next Table:

Table 15 – Content of Register.cpp

```
void MainSimulator::registerNewAtomics()
{
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Step10>(), "Step10" ) ;
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Step9>(), "Step9" ) ;
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Step8>(), "Step8" ) ;
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Step7>(), "Step7" ) ;
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Step6>(), "Step6" ) ;
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Step5>(), "Step5" ) ;
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Step4to5>(),"Step4to5");
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Step4>(), "Step4" ) ;
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Step3>(), "Step3" ) ;
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Step2>(), "Step2" ) ;
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Step1>() , "Step1" ) ;
}
```

4.2.3 STEP 2

In this step, Glucose 6-phosphate is converted to fructose 6-phosphate by phosphoglucoisomerase. The atomic model of step2 has two inputs: ***glucose_6_phosphate*** and ***phosphoglucoisomerase*** (enzyme), and one output ***fructose_6_phosphate*** (see Figure 4.5).

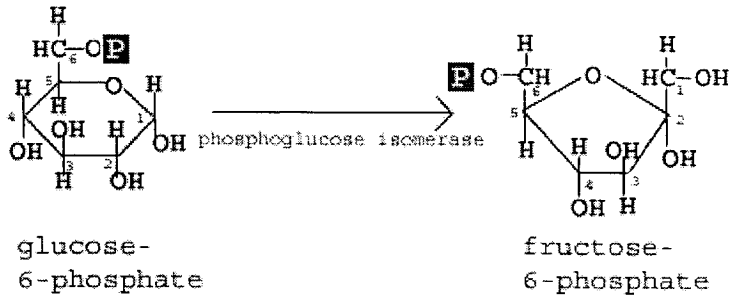


Figure 4.4 - Step2 of Glycolysis [2]

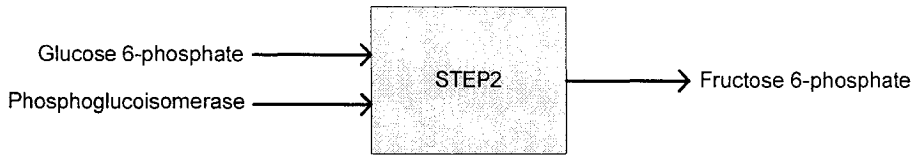


Figure 4.5 - Atomic Model for Step2 of Glycolysis

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

Where

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

X = {glucose_6_phosphate, phosphoglucoisomerase }

Y = {fructose_6_phosphate}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix A).

4.2.4 STEP 3

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

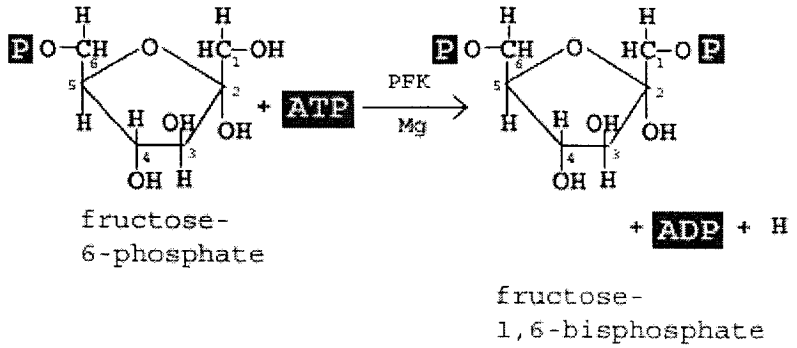


Figure 4.6 – Step3 of Glycolysis [2]

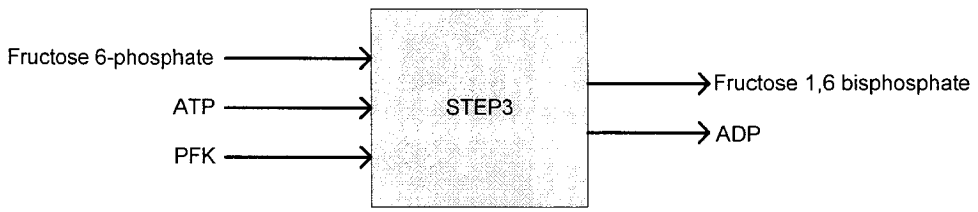


Figure 4.7 - Atomic Model for Step3 of Glycolysis

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

Where

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

X = {fructose_6_phosphate, ATP, PFK}

Y = {fructose_16_bisphosphate, ADP}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix A).

4.2.5 STEP 4 AND 5

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). The atomic models for these steps have two inputs: ***fructose_16_bisphosphate*** (fructose 1,6-bisphosphate) and ***aldolase*** (enzyme).

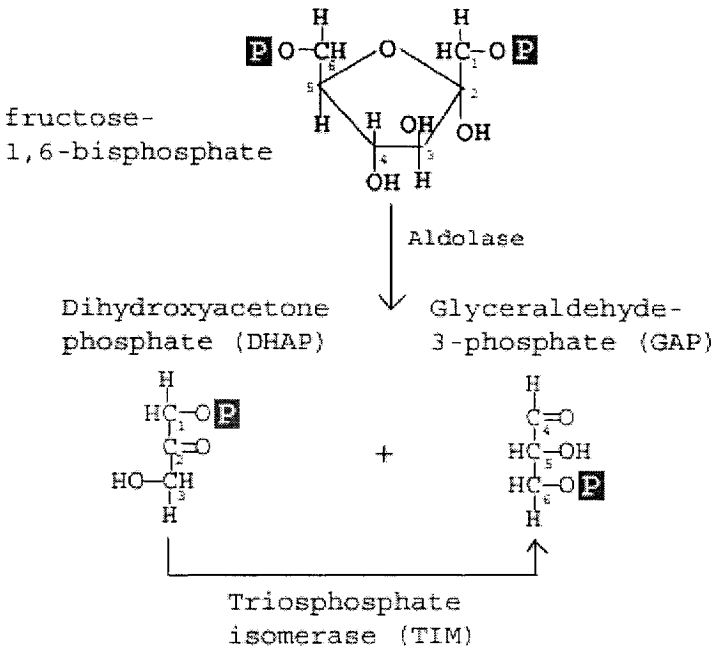


Figure 4.8 – Step 4 and 5 of Glycolysis [2]

Two separate atomic models are defined for this reaction. Step4 produces the output DHP (DiHydroxyacetone Phosphate), while the atomic model for step5 produces GDP (Glyceraldehydes 3-phosphate) (See Figure 4.8).

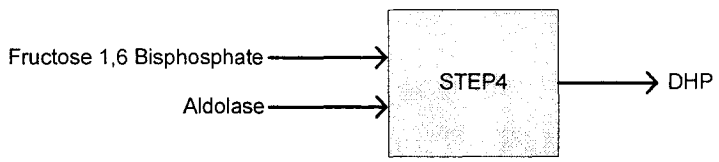


Figure 4.9 - Atomic Model for Step4 of Glycolysis

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

Where

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

X = {fructose_16_bisphosphate, aldolase}, and Y = {DHP}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix A).

Glyceraldehydes 3-phosphate (GDP) is the other product of fructose 1,6-bisphosphate. The atomic model of step5 has two inputs: ***fructose_6_phosphate*** (fructose 1,6-bisphosphate) and ***aldolase*** (enzyme), and one output: ***GDP*** (Glyceraldehydes 3-phosphate).

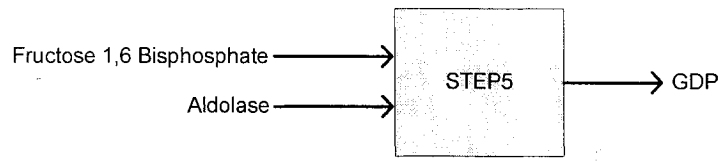


Figure 4.10 - Atomic Model for Step5 of Glycolysis

$Step5 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

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

X = {fructose_6_phosphate, aldolase}

Y = {GDP}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix A).

4.2.6 STEP 4 TO 5

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

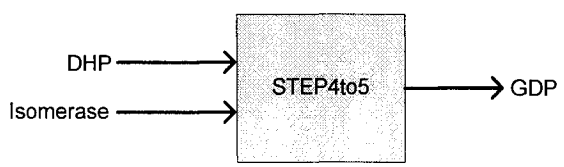


Figure 4.11 - Atomic Model for Step4to5 of Glycolysis

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

Where

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

X = {DHP, isomerase}

Y = {GDP}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix A).

4.2.7 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^+.

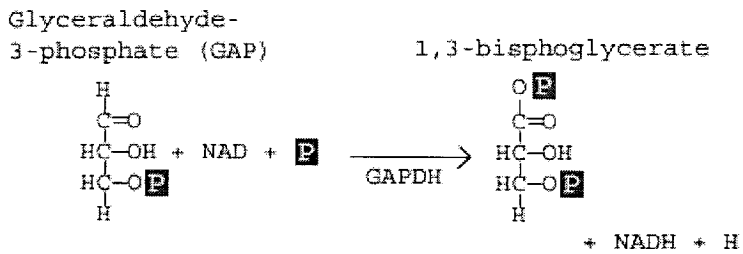


Figure 4.12 – Step6 of Glycolysis [2]

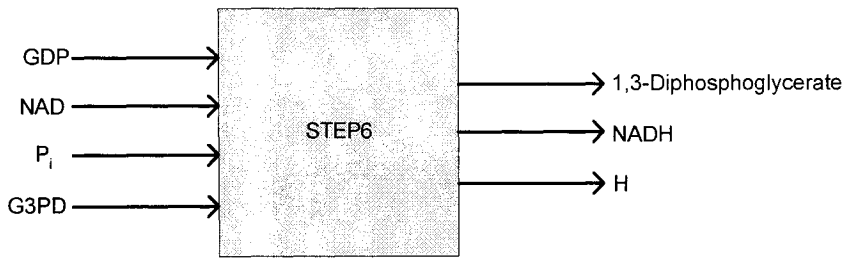


Figure 4.13 - Atomic Model for Step6 of Glycolysis

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

Where

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

X = {GDP, NAD, Pi, G3PD}

Y = {_13_BPG, NADH, H}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix A).

4.2.8 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).

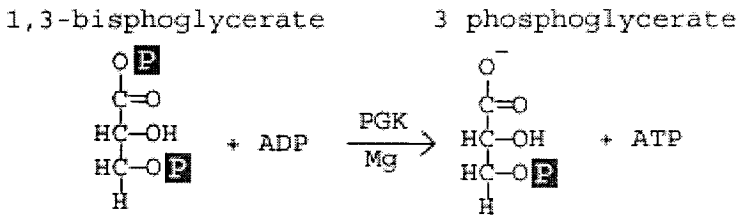


Figure 4.14 – Step7 of Glycolysis [2]

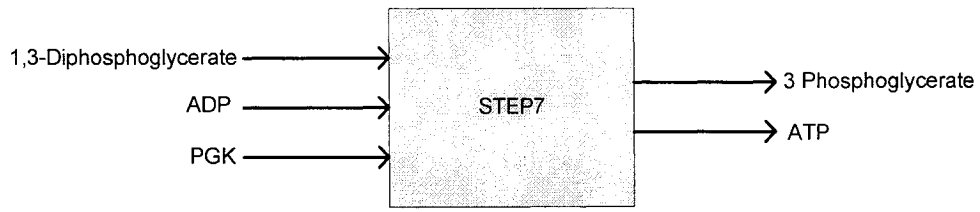


Figure 4.15 - Atomic Model for Step7 of Glycolysis

$Step7 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

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

X = {_13_BPG, ADP, PGK}

Y = {_3_phosphoglycerate, ATPo}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix A).

4.2.9 STEP 8

3-phosphoglycerate is converted to 2-phosphoglycerate by phosphoglycerate mutase (PGM). As shown in Figure 4.17, ***_3_phosphoglycerate*** and ***PGM*** enter the system generating ***_2_phosphoglycerate***.

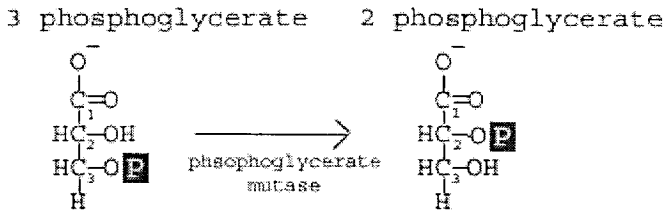


Figure 4.16 – Step8 of Glycolysis [2]

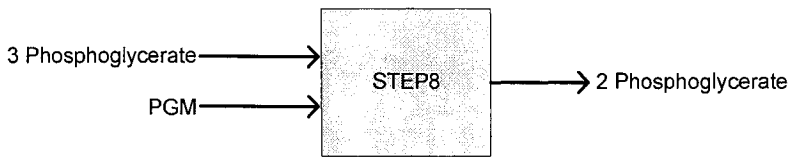


Figure 4.17 - Atomic Model for Step8 of Glycolysis

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

Where

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

X = {_3_phosphoglycerate, PGM}

Y = {_2_phosphoglycerate}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix A).

4.2.10 STEP 9

Enolase catalyzes the dehydration of 2-phosphoglycerate to form phosphoenolpyruvic (PEP). Water is released in this process. This model has two inputs: ***_2_phosphoglycerate*** and ***enolase***, generating two outputs ***phosphoenolpyruvic*** and ***H2O*** (see Figure 4.19).

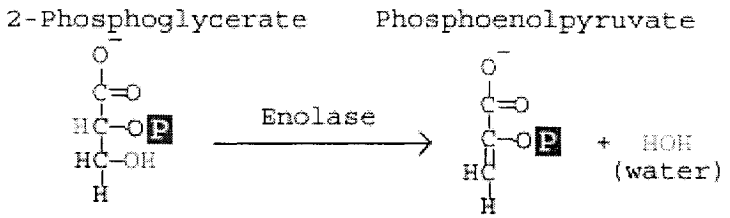


Figure 4.18 – Step9 of Glycolysis [2]

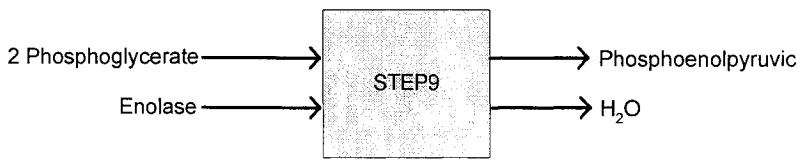


Figure 4.19 - Atomic Model for Step9 of Glycolysis

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

Where

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

X = {_2_phosphoglycerate, enolase}

Y = {phosphoenolpyruvic, H2O}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix A).

4.2.11 STEP 10

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

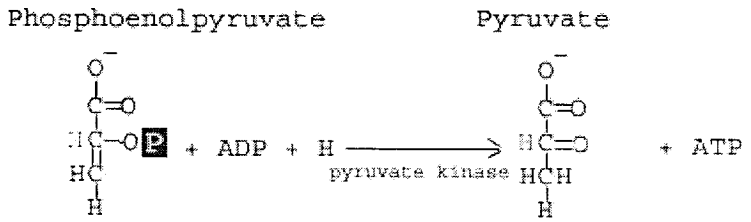


Figure 4.20 - Step10 of Glycolysis [2]

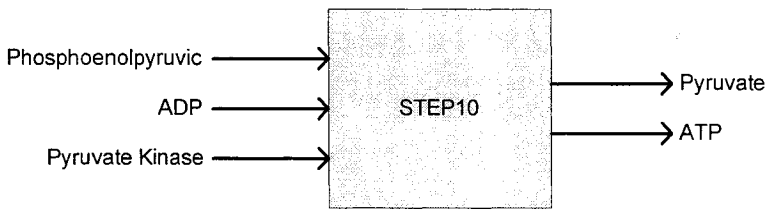


Figure 4.21 - Atomic Model for Step10 of Glycolysis

$Step10 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

Where

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

X = {phosphoenolpyruvic, ADP, pyruvate_kinase}

Y = {pyruvate, ATP}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix A).

4.3 COUPLED MODEL DEFINITION

All the steps explained in the previous section were implemented in CD++, as shown in section 4.2.1 and the appendix A. To do this, first the behaviour of each component was carefully specified with an analysis of inputs and outputs for each step as shown in sections 4.2.1 to 4.2.11. 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 integrating all the sub-models previously defined. This model follows the design presented in Figure 4.23, and its detailed definition can be found in the following table. The top model here is composed of one coupled model *(Glycolysis)* and sixteen atomic components *(step1 – step10)*. The input and output ports define the model's interface, and the links between components define the model's coupling, following the structural description in Table 16. Once this model is completely defined, we can execute it within a given experimental framework, and analyze the simulation results, which are provided in a log file with the format shown in Table 17.

Table 16 – 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 shows a snapshot of the glycolysis simulation. 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 simulation where the inputs and outputs happen. Only a fragment of glycolysis.log has been shown in Table 17 (refer to Appendix A for a full description of Glycolysis.log).

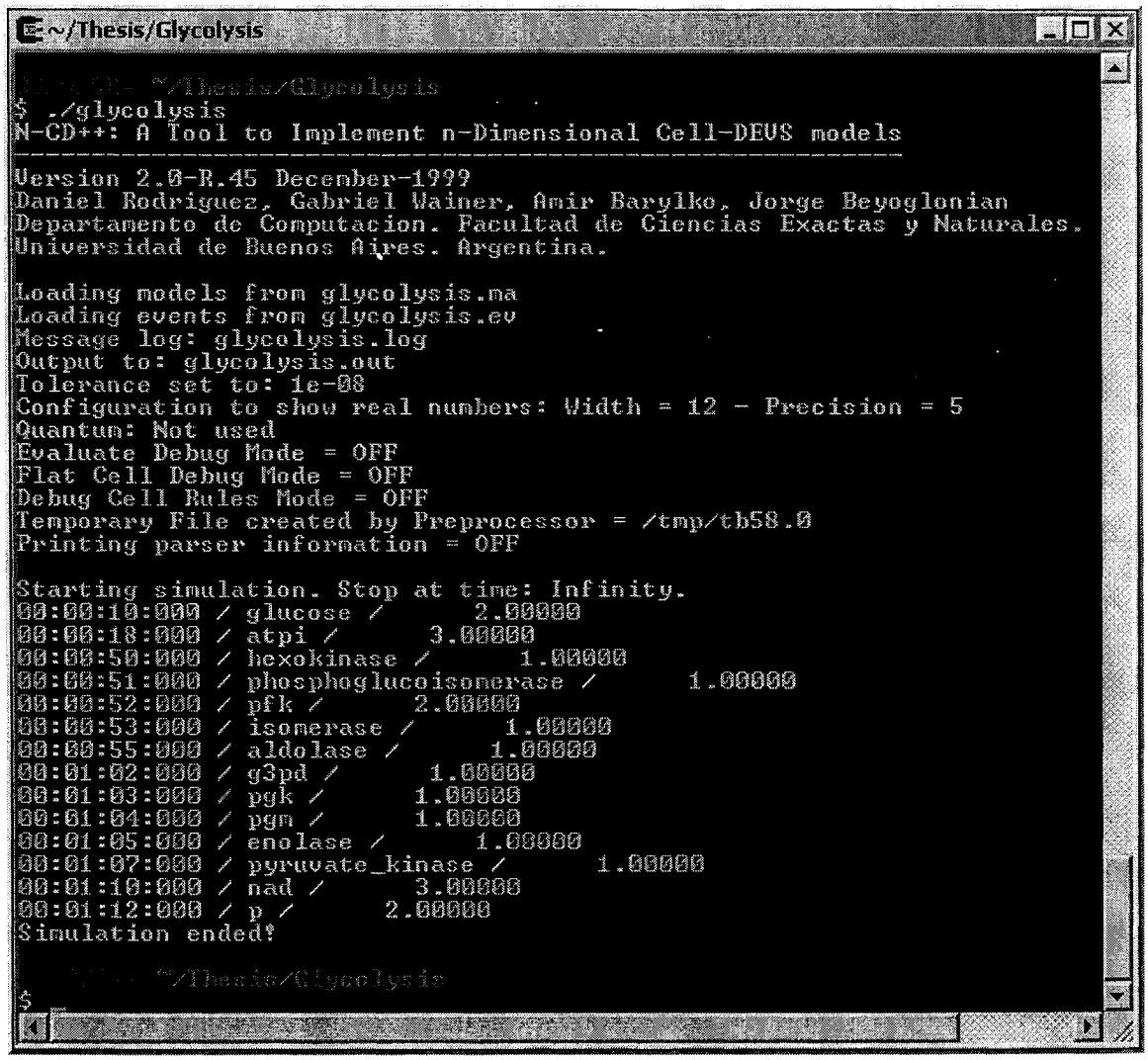


Figure 4.22 – Snapshot of Simulation Run for Glycolysis

The log file registers the flow of messages between the models that participates in the simulation. Inter-process interaction is carried out through messages passing. Each message includes information of the source (or destination), the event simulated time, and a content (consisting of a port and a value). Each line of the file shows the message type, the time in which occur, the emitter and the destiny. Message is the base class that defines the different messages: ***** (internal event), ***X*** (external event), ***Y*** (model's output), and ***D*** (a model has finished with its task). This information is common to all the messages. In addition, if the message is type of ***X*** or ***Y***, then it will include the port and the value. For the messages or type ***D*** it will include the time of the next event, or '...' in case that this time is infinite. The numbers that figure next to the name of the simulator associated to each model only are for information for the developer. By analyzing the log file, we can see the activation of the different parts in Glycolysis model, and each of the elements involved in this simulation (see Table 17) [24].

Table 17 – A Fragment of Glycolysis.log File

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

```
Message D / 00:00:51:000 / step3(04) / 00:00:00:000 to top(01)
Message D / 00:00:51:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:51:000 / Root(00) to 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 18 – 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 4.23 for a drawing of the glycolysis model.

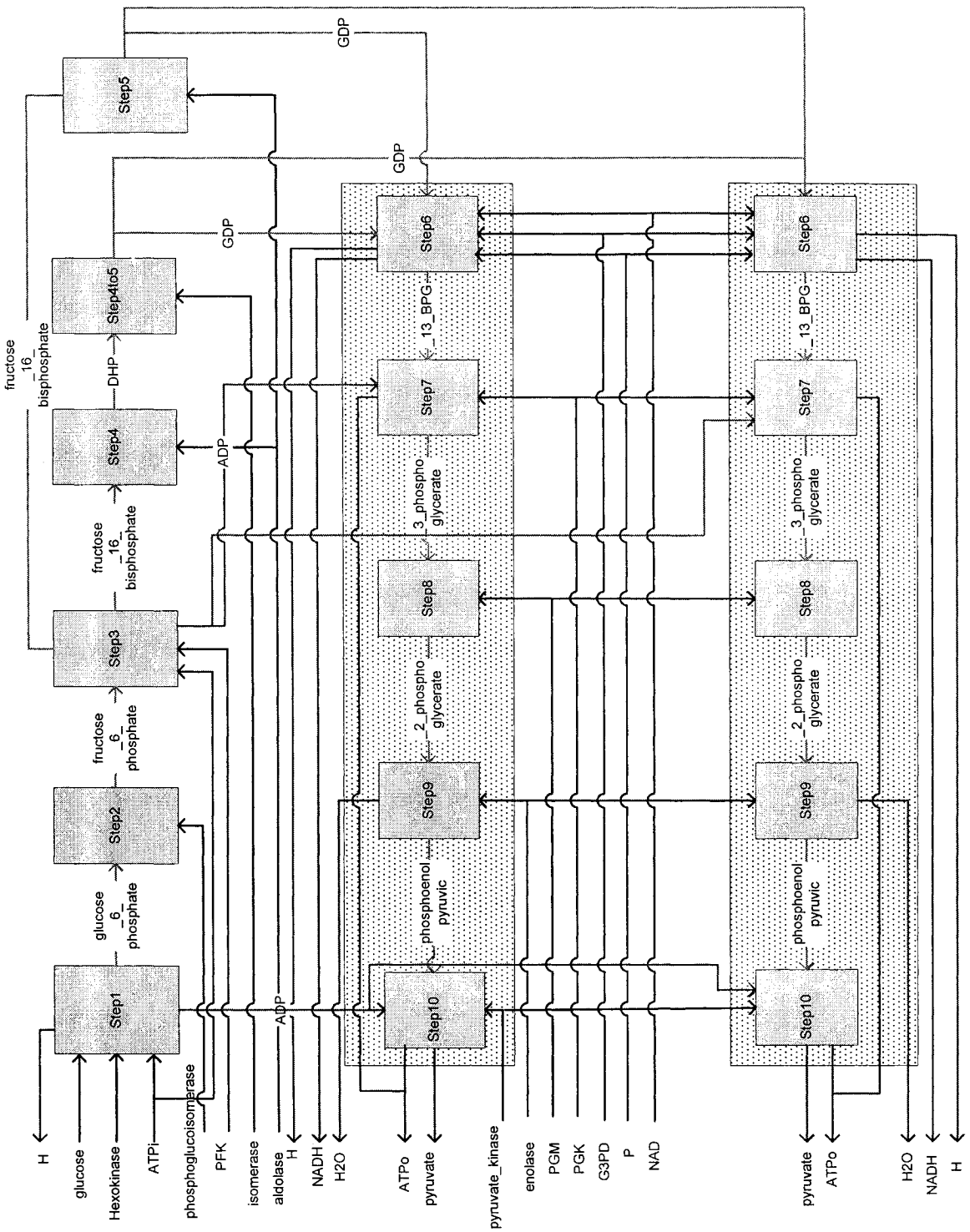


Figure 4.23 – Coupled Model of Glycolysis

4.4 SIMULATION RESULTS

The present section is devoted to show some of the results obtained when the models previously presented are simulated. The various atomic models contained in the previously defined coupled model were tested using different scenarios. One of the most important features of DEVS is that models at any level of complexity can be independently tested.

In the first case, we show the results obtained simulating Step1 of glycolysis. Table 19 shows the definition for atomic model step1 (***step1.ma***).

Table 19 – Atomic Model Definition for Step1: Step1.ma

```
[top]
components : step1@Step1
out : glucose_6_phosphate ADP H
in : glucose ATPi hexokinase
Link : glucose glucose@step1
Link : ATPi ATPi@step1
Link : hexokinase hexokinase@step1
Link : ADP@step1 ADP
Link : glucose_6_phosphate@step1 glucose_6_phosphate
Link : H@step1 H

[step1]
Preparation : 00:00:00:000
```

Table 20 shows the log file generated by scenario 1 of step1 (see Table 21). This file includes all the messages interchanged in this step. As explained earlier, there are four kinds of messages: * (used to signal a state change due to an internal event), **X** (used when an external event arrives), **Y** (the model's output), and **D** (indicating that a model is finished with its task). The **I** messages initialize the corresponding model. For each message, we show its type, timestamp, value, origin/destination, and the port used for the transmission. The execution cycle starts by initializing the higher level models. When

the initialization cycle finishes, the imminent model is executed. We are able to follow the execution flow of any program by analyzing this log file. As one can see, the first two lines are the initialization of the system at time 00:00:00:000. The next two lines are D messages indicating that the model is done with initialization. The lines after indicate the arrival of an external event "***hexokinse***" with value 5 at time 00:00:15:000, and another external event at time 00:00:30:000 with a value 2 from port glucose, and another external event at time 00:00:30:000 with a value 6 from port ATPi. Following the log file, one can find 3 sets of outputs (indicated by Message Y) generated at time 00:00:30:000 each with a value 2 from output ports: ***adp***, ***glucose_6_phosphate***, and ***h*** (`Message Y/00:00:30:000/ step1(02)/ adp/ 2.00000 to top(01), Message Y/00:00:30:000/ step1(02)/ glucose_6_phosphate/ 2.00000 to top(01), and Message Y/ 00:00:30:000/ step1(02)/ h/ 2.00000 to top(01)`). This means that at time 00:00:30:00 all the inputs required to complete the reaction were present in the system, and the system was ready to generate the proper outputs. We are able to follow the execution flow of any model the same way by analyzing its log file.

Table 20 – Log File for Scenario 2 of Step1: Step1.log

```
Message I / 00:00:00:000 / Root(00) to top(01)
Message I / 00:00:00:000 / top(01) to step1(02)
Message D / 00:00:00:000 / step1(02) / ... to top(01)
Message D / 00:00:00:000 / top(01) / ... to Root(00)
Message X / 00:00:15:000 / Root(00) / hexokinase /      1.00000 to top(01)
Message X / 00:00:15:000 / top(01) / hexokinase /      1.00000 to step1(02)
Message D / 00:00:15:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:00:15:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:15:000 / Root(00) to top(01)
Message * / 00:00:15:000 / top(01) to step1(02)
Message D / 00:00:15:000 / step1(02) / ... to top(01)
Message D / 00:00:15:000 / top(01) / ... to Root(00)
Message X / 00:00:30:000 / Root(00) / glucose /      2.00000 to top(01)
Message X / 00:00:30:000 / top(01) / glucose /      2.00000 to step1(02)
Message D / 00:00:30:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:00:30:000 / top(01) / 00:00:00:000 to Root(00)
Message X / 00:00:30:000 / Root(00) / atpi /      6.00000 to top(01)
Message X / 00:00:30:000 / top(01) / atpi /      6.00000 to step1(02)
Message D / 00:00:30:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:00:30:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:30:000 / Root(00) to top(01)
```

```
Message * / 00:00:30:000 / top(01) to step1(02)
Message D / 00:00:30:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:00:30:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:30:000 / Root(00) to top(01)
Message * / 00:00:30:000 / top(01) to step1(02)
Message Y / 00:00:30:000 / step1(02) / adp /      2.00000 to top(01)
Message Y / 00:00:30:000 / step1(02) / glucose_6_phosphate /      2.00000 to top(01)
Message Y / 00:00:30:000 / step1(02) / h /      2.00000 to top(01)
Message D / 00:00:30:000 / step1(02) / ... to top(01)
Message Y / 00:00:30:000 / top(01) / adp /      2.00000 to Root(00)
Message Y / 00:00:30:000 / top(01) / glucose_6_phosphate /      2.00000 to Root(00)
Message Y / 00:00:30:000 / top(01) / h /      2.00000 to Root(00)
Message D / 00:00:30:000 / top(01) / ... to Root(00)
Message X / 00:00:40:000 / Root(00) / glucose /      4.00000 to top(01)
Message X / 00:00:40:000 / top(01) / glucose /      4.00000 to step1(02)
Message D / 00:00:40:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:00:40:000 / top(01) / 00:00:00:000 to Root(00)
Message X / 00:00:40:000 / Root(00) / atpi /      1.00000 to top(01)
Message X / 00:00:40:000 / top(01) / atpi /      1.00000 to step1(02)
Message D / 00:00:40:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:00:40:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:40:000 / Root(00) to top(01)
Message * / 00:00:40:000 / top(01) to step1(02)
Message D / 00:00:40:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:00:40:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:40:000 / Root(00) to top(01)
Message * / 00:00:40:000 / top(01) to step1(02)
Message Y / 00:00:40:000 / step1(02) / adp /      4.00000 to top(01)
Message Y / 00:00:40:000 / step1(02) / glucose_6_phosphate /      4.00000 to top(01)
Message Y / 00:00:40:000 / step1(02) / h /      4.00000 to top(01)
Message D / 00:00:40:000 / step1(02) / ... to top(01)
Message Y / 00:00:40:000 / top(01) / adp /      4.00000 to Root(00)
Message Y / 00:00:40:000 / top(01) / glucose_6_phosphate /      4.00000 to Root(00)
Message Y / 00:00:40:000 / top(01) / h /      4.00000 to Root(00)
Message D / 00:00:40:000 / top(01) / ... to Root(00)
Message X / 00:00:55:000 / Root(00) / glucose /      1.00000 to top(01)
Message X / 00:00:55:000 / top(01) / glucose /      1.00000 to step1(02)
Message D / 00:00:55:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:00:55:000 / top(01) / 00:00:00:000 to Root(00)
Message X / 00:00:55:000 / Root(00) / atpi /      1.00000 to top(01)
Message X / 00:00:55:000 / top(01) / atpi /      1.00000 to step1(02)
Message D / 00:00:55:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:00:55:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:55:000 / Root(00) to top(01)
Message * / 00:00:55:000 / top(01) to step1(02)
Message D / 00:00:55:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:00:55:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:55:000 / Root(00) to top(01)
Message * / 00:00:55:000 / top(01) to step1(02)
Message Y / 00:00:55:000 / step1(02) / adp /      1.00000 to top(01)
Message Y / 00:00:55:000 / step1(02) / glucose_6_phosphate /      1.00000 to top(01)
Message Y / 00:00:55:000 / step1(02) / h /      1.00000 to top(01)
Message D / 00:00:55:000 / step1(02) / ... to top(01)
Message Y / 00:00:55:000 / top(01) / adp /      1.00000 to Root(00)
Message Y / 00:00:55:000 / top(01) / glucose_6_phosphate /      1.00000 to Root(00)
Message Y / 00:00:55:000 / top(01) / h /      1.00000 to Root(00)
Message D / 00:00:55:000 / top(01) / ... to Root(00)
Message X / 00:01:05:000 / Root(00) / glucose /      1.00000 to top(01)
Message X / 00:01:05:000 / top(01) / glucose /      1.00000 to step1(02)
Message D / 00:01:05:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:01:05:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:01:05:000 / Root(00) to top(01)
Message * / 00:01:05:000 / top(01) to step1(02)
Message D / 00:01:05:000 / step1(02) / 00:00:00:000 to top(01)
Message D / 00:01:05:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:01:05:000 / Root(00) to top(01)
Message * / 00:01:05:000 / top(01) to step1(02)
Message Y / 00:01:05:000 / step1(02) / adp /      1.00000 to top(01)
```

```
Message Y / 00:01:05:000 / step1(02) / glucose_6_phosphate /      1.00000 to top(01)
Message Y / 00:01:05:000 / step1(02) / h /      1.00000 to top(01)
Message D / 00:01:05:000 / step1(02) / ... to top(01)
Message Y / 00:01:05:000 / top(01) / adp /      1.00000 to Root(00)
Message Y / 00:01:05:000 / top(01) / glucose_6_phosphate /      1.00000 to Root(00)
Message Y / 00:01:05:000 / top(01) / h /      1.00000 to Root(00)
Message D / 00:01:05:000 / top(01) / ... to Root(00)
```

Table 21 shows Step1's input and output files with their timestamps, obtained from five different experimentation. When the simulation starts, the components have initial value of zero. As we can see, in scenario 1, the first set of outputs are generated in time 00:00:30:000 only when all the inputs necessary for the reaction to take place are entered to the system [`00:00:30:000 adp 2, 00:00:30:000 glucose_6_phosphate 2, and 00:00:30:000 h 2`]. At this point, hexokinase is present in the system, and we have four molecules of Atpi left. At time 00:00:40:000, as soon as four molecules of glucose enter the system, another set of outputs are generated each with a value 4. Later, at time 00:00:40:000, another input of ATPi enters the system, but nothing happens till time 00:00:55:000 when a glucose enters the system and invokes the generation of another set of outputs each with value 1. Another ATPi enters the sytem at time 00:00:55:000, but nothing happens because there is no more glucose left in the system. At time 00:00:65:000, one glucose enters the system and another set of outputs are generated with time stamp 00:01:05:000. (note that times 00:00:65:000 are 00:01:05:000 are the same). In scenario 2, we test our system by omitting the first input hexokinase from the first scenario's input events set. As we can see, no output is generated as the presence of hexokinase is essential in this reaction, and it has to be present for any output to be generated. We changed the order of inputs and their values (see scenarios 3, 4, and 5 in Table 21), trying to make sure that we still get the correct results. Scenario 3, 4, and 5 further approves the validity of our model.

Table 21 – Input Event Files and Corresponding Output Files Resulted from Simulating Step1 of Glycolysis Five Different Times

Scenario	Step1.ev	Step1.out
1	00:00:15:00 hexokinase 1 00:00:30:00 glucose 2 00:00:30:00 ATPi 6 00:00:40:00 glucose 4 00:00:40:00 ATPi 1 00:00:55:00 glucose 1 00:00:55:00 ATPi 1 00:00:65:00 glucose 1	00:00:30:000 adp 2 00:00:30:000 glucose_6_phosphate 2 00:00:30:000 h 2 00:00:40:000 adp 4 00:00:40:000 glucose_6_phosphate 4 00:00:40:000 h 4 00:00:55:000 adp 1 00:00:55:000 glucose_6_phosphate 1 00:00:55:000 h 1 00:01:05:000 adp 1 00:01:05:000 glucose_6_phosphate 1 00:01:05:000 h 1
2	00:00:30:00 glucose 2 00:00:30:00 ATPi 6 00:00:40:00 glucose 4 00:00:40:00 ATPi 1 00:00:55:00 glucose 1 00:00:55:00 ATPi 1 00:00:65:00 glucose 1	
3	00:00:30:00 glucose 2 00:00:30:00 ATPi 6 00:00:40:00 glucose 4 00:00:45:00 hexokinase 4 00:00:40:00 ATPi 2 00:00:55:00 glucose 1	00:00:45:000 adp 6 00:00:45:000 glucose_6_phosphate 6 00:00:45:000 h 6 00:00:55:000 adp 1 00:00:55:000 glucose_6_phosphate 1 00:00:55:000 h 1
4	00:00:30:00 glucose 2 00:00:46:00 ATPi 6 00:01:45:00 hexokinase 1 00:02:40:00 ATPi 2 00:03:55:00 glucose 1	00:01:45:000 adp 2 00:01:45:000 glucose_6_phosphate 2 00:01:45:000 h 2 00:03:55:000 adp 1 00:03:55:000 glucose_6_phosphate 1 00:03:55:000 h 1
5	00:00:03:00 glucose 2 00:00:04:00 ATPi 3 00:00:04:00 hexokinase 1 00:01:11:00 ATPi 2 00:03:55:00 glucose 1	00:00:04:000 adp 2 00:00:04:000 glucose_6_phosphate 2 00:00:04:000 h 2 00:03:55:000 adp 1 00:03:55:000 glucose_6_phosphate 1 00:03:55:000 h 1

The next table corresponds to the definition of step 2 of glycolysis model:

Table 22 – Atomic Model for Step2 of Glycolysis: step2.ma

```
[top]
components : step2@Step2
out : fructose_6_phosphate
in : glucose_6_phosphate phosphoglucoisomerase

Link : glucose_6_phosphate glucose_6_phosphate@step2
Link : phosphoglucoisomerase phosphoglucoisomerase@step2
Link : fructose_6_phosphate@step2 fructose_6_phosphate

[step2]
preparation : 00:00:05:000
```

In Table 23, we include five different scenarios of input events for this step. The output files are generated for each of the five scenarios, simulating the atomic model for this step (see Table 23).

Table 23 – Input Event Files and Corresponding Output Files Resulted from Simulating Step2 of Glycolysis Five Different Times

Scenario	Step2.ev	Step2.out
1	00:00:15:00 glucose_6_phosphate 2 00:00:30:00 phosphoglucoisomerase 1 00:00:30:00 glucose_6_phosphate 1 00:00:45:00 glucose_6_phosphate 4 00:00:55:00 glucose_6_phosphate 5	00:00:30:000 fructose_6_phosphate 3 00:00:45:000 fructose_6_phosphate 4 00:00:55:000 fructose_6_phosphate 5
2	00:00:15:00 glucose_6_phosphate 2 00:00:30:00 phosphoglucoisomerase 1 00:01:55:00 glucose_6_phosphate 5	00:00:30:000 fructose_6_phosphate 2 00:01:55:000 fructose_6_phosphate 5
3	00:00:15:00 glucose_6_phosphate 2 00:01:55:00 glucose_6_phosphate 5	
4	00:00:15:00 glucose_6_phosphate 2 00:01:55:00 glucose_6_phosphate 5 00:02:05:00 phosphoglucoisomerase 1	00:02:05:000 fructose_6_phosphate 7
5	00:00:15:00 glucose_6_phosphate 2 00:01:55:00 ADP 1 00:02:05:00 phosphoglucoisomerase 1	

In scenario 1, at time 00:00:15:00, 1 glucose_6_phosphate enters the system but there is no output. At time 00:00:30:00, 1 unit of phosphoglucoisomerase and another unit of glucose_6_phosphate enter the system. At this point, an output of fructose_6_phosphate with a value of 3 is generated: `.00:00:30:000 fructose_6_phosphate 3`. At time 00:00:45:00, 4 units of glucose_6_phosphate enter the system, and immediately we can see 4 units of fructose_6_phosphate being generated. At time 00:00:55:00, 5 units of glucose_6_phosphate enter the system, and 5 units of fructose_6_phosphate are generated. This shows that only the presence of phosphoglucoisomerase is needed in the system. The value of the outputs are totally depended on the values of glucose_6_phosphate.

Scenario 2 is written to further test the timing of the outputs. In scenario 3, we omit the input phosphoglucoisomerase and we can see that no output is generated. This proves again that the presence of phosphoglucoisomerase is necessary for this reaction to be completed. In scenario 4, we can see that 2 units of glucose_6_phosphate are entered at time 00:00:15:000 and another 5 units are entered the system at time 00:00:30:000. No output is generated till time 00:02:05:000, when 1 unit of phosphogluisomerase enters the system. At this time, 7 units of fructose_6_phosphate are generated. This further proves that the presence of the enzyme is needed for this reaction to happen. In scenario 5, we introduce an input which is not defined in our atomic model for this step. This results in an error and no outputs are generated for this case.

Once steps 1 and 2 are tested separately, we couple them together and test it to make sure that the two are running correctly when coupled together. Table 24 defines the coupled model for Step1 and Step2 of glycolysis.

Table 24 – Definition for Coupled Model Step12: step12.ma

```
[top]
components : step1@Step1 step2@Step2

out : fructose_6_phosphate ADP H
in : glucose ATPi hexokinase phosphoglucoisomerase

Link : glucose glucose@step1
Link : ATPi ATPi@step1
Link : hexokinase hexokinase@step1
Link : phosphoglucoisomerase phosphoglucoisomerase@step2


Link : ADP@step1 ADP
Link : H@step1 H
Link : fructose_6_phosphate@step2 fructose_6_phosphate

Link : glucose_6_phosphate@step1 glucose_6_phosphate@step2

[step1]
preparation : 00:00:00:000

[step2]
preparation : 00:00:05:000
```

Once a coupled model is defined, we can write test cases for it. The next table shows five scenarios tested for this coupled model:

Table 25 –Input Event Files and Corresponding Output Files Resulted from Simulating Coupled Model Step12

Scenario	Step12.ev	Step12.out
1	00:00:15:00 hexokinase 1 00:00:30:00 glucose 2 00:00:55:00 glucose 1 00:00:55:00 ATPi 2 00:01:55:00 glucose 1 00:02:00:00 atpi 1 00:02:02:00 glucose 1 00:02:05:00 atpi 1 00:02:13:00 phosphoglucoisomerase 1	00:00:55:000 adp 2 00:00:55:000 h 2 00:02:00:000 adp 1 00:02:00:000 h 1 00:02:05:000 adp 1 00:02:05:000 h 1 00:02:13:000 fructose_6_phosphate 4
2	00:00:30:00 glucose 2 00:00:55:00 glucose 1 00:02:00:00 atpi 1 00:02:02:00 glucose 1 00:02:05:00 atpi 1 00:02:13:00 hexokinase 1 00:02:15:00 phosphoglucoisomerase 1	00:02:13:000 adp 2 00:02:13:000 h 2 00:02:15:000 fructose_6_phosphate 2
3	00:00:30:00 glucose 2 00:00:55:00 glucose 1 00:01:01:00 hexokinase 1 00:02:00:00 atpi 1 00:02:02:00 glucose 1 00:02:05:00 atpi 1	00:02:00:000 adp 1 00:02:00:000 h 1 00:02:05:000 adp 1 00:02:05:000 h 1
4	00:00:30:00 glucose 2 00:00:55:00 glucose 1 00:00:55:00 ATPi 2 00:01:55:00 glucose 1 00:02:00:00 atpi 1 00:02:02:00 glucose 1 00:02:05:00 atpi 1 00:02:13:00 phosphoglucoisomerase 1	
5	00:00:55:00 glucose 1 00:02:02:00 glucose 1 00:02:05:00 atpi 1 00:02:13:00 phosphoglucoisomerase 1 00:03:15:00 hexokinase 1	00:03:15:000 adp 1 00:03:15:000 h 1 00:03:15:000 fructose_6_phosphate 1

In scenario 1, no output is generated till time 00:00:55:00. This indicates that for the first reaction to happen, the variables *hexokinase*, *glucose*, and *ATPi* are needed. One unit of hexokinase is entered at time 00:00:15:00, 2 units of glucose are entered at time 00:00:30:00, followed by another unit of it at time 00:00:55:00. The entrance of 2 units of ATPi at time 00:00:55:00 causes the generation of 2 units of ADP and H at this time.

Fructose_6_phosphate is not generated till time 00:02:13:00, when 4 units of it altogether are generated. This proves that for ADP and H to be generated, only ATPi and maybe the presence of hexokinase are needed, while for the generation of fructose_6_phosphate, the presence of glucose and phosphoglucoisomerase are needed.

In scenario 2, 2 and 1 unit of glucose are entered at times 00:00:30:00 and 00:00:55:00. At this point, there is no output yet. At time 00:02:00:00, 1 unit of ATPi enters the system but still nothing is generated. This explains that for the outputs ADP and H to be generated, the presence of hexokinase is definitely needed. At time 00:02:02:00, another glucose enters the system, increasing the glucose counter to 4. At time 00:02:05:00, 1 unit of ATPi enters the system and still no output is generated. At time 00:02:13:00, 1 unit of hexokinase enters the system, and this results in the immediate production of 2 units of ADP and 2 units of H. At time 00:02:15:00, 1 unit of phosphoglucoisomerase enters the systems resulting in generation of output fructose_6_phosphate with a value 2. Note that we had 4 glucose in the system but only 2 fructose_6_phosphate were generated. This proves that 1 ATPi is needed for each fructose_6_phosphate generation and since we had only 2 ATPi entering the system, only 2 molecules of fructose_6_phosphate are generated. This proves that the heat energy generated by breaking the molecule ATPi is needed for this reaction to be completed making this reaction endergonic.

In Scenario 3, since there is no phosphoglucoisomerase, therefore no fructose_6_phosphate was generated. Only ADP and H are generated at times 00:02:00:00 and 00:02:05:00, which corresponds to the entrance of ATPi at those times. In scenario 4, no output is generated at all. This is because hexokinase is missing in the system. This proves

that hexokinase is essential for any of these reactions to be completed. In scenario 5, no output is generated till time 00:03:15:00, when a hexokinase enters this system. This further proves the role of hexokinase in this model.

Finally, we show 3 scenarios with 3 set of different inputs and the ouputs generated from execution of the complete glycolysis coupled model. The coupled model definition was shown in the previous section in Table 16. Next table shows the input event and output files for coupled model Glycolysis:

Table 26 –Input Event Files and Corresponding Output Files Resulted from Simulating Coupled Model Glycolysis

Scenario	glycolysis.ev	glycolysis.out
1	00:00:10:00 glucose 2 00:00:18:00 ATPi 2 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 2 00:01:12:00 P 2	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
2	00:00:10:00 glucose 5 00:00:18:00 ATPi 6 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 3 00:01:04:00 PGM 1 00:01:05:00 enolase 1 00:01:10:00 NAD 4 00:01:12:00 P 5 00:00:10:00 glucose 5 00:01:03:00 PGK 3 00:01:23:00 pyruvate_kinase 1	00:00:50:000 h 6 00:01:12:000 nadh 4 00:01:12:000 h 4 00:01:12:000 atpo 4 00:01:12:000 h2o 4 00:01:23:000 atpo 4 00:01:23:000 pyruvate 4
3	00:00:10:00 glucose 5 00:00:18:00 ATPi 6 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 3 00:01:04:00 PGM 1	

	00:01:05:00 enolase 1 00:01:10:00 NAD 4 00:01:12:00 P 5 00:00:10:00 glucose 5 00:01:03:00 PGK 3 00:01:23:00 pyruvate_kinase 1	

In scenario 1, as we can see, 2 units of glucose enter the system at time 00:00:10:00, 2 units of ATPi enter the system at time 00:00:18:00, and 1 unit of hexokinase enter the system at time 00:00:50:00. At this time 2 units of H are generated. Nothing is generated till time 00:01:12:00. This is when the last input P is entered the system with value 2. This can show that either all the inputs ought to be present in the system for the outputs NADH, ATPo, and pyruvate to be produced, or P plays an important role in completion of one of intermediate reactions. This can be proven by further experimenting with the inputs and changing their timestamps. Scenario 2 is designed with this in mind. This case proves that P is needed for the production of ATPo. Pyruvate is produced only when pyruvate_kinase is present in the system. Scenario 3 shows that no input is generated if no hexokinase is present in the system. This makes sense because hexokinase is needed for the first atomic model coupled to this system.

Next, we can create a batch file with a sequence of thousand different input event files for each case. These test sequences should cover the different combinations of inputs and states in which the model must respond correctly. The expected responses could be then compared with the actual ones. If all comparisons succeed then we may gain full confidence in the model and place it into the model base as a verified model. The stored test file can be reused when modifications are made in the model. If an error occurs in a coupled model, and all components have been verified in the foregoing way, we can look for the source to

be in the coupling or tie-breaking specifications of the coupled model. This bottom-up testing makes it possible to confidently build up successively higher level sub-components until the final model is achieved.

4.5 SUMMARY

The CD++ tool, based on the DEVS 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 architecture presented in the previous sections was completely implemented using CD++.

First, the behaviour of each component was carefully specified, with an analysis of inputs and outputs. The specification also provided test cases. Then, each component was defined as a DEVS model following the specification. Afterwards, each model was implemented in CD++, including an experimental framework following the test cases defined in the specification. Finally, the main model was built as a coupled model connecting all the sub-models previously defined. This model follows the design presented in Figure 4.23, and its detailed definition found in Table 16.

In this thesis, we are using a constant (deterministic) time advance for the chemical reactions. It is possible to use a probabilistic (stochastic) function for the time advance to model more precisely various steps of chemical reactions. The focus of this work being the development of a flexible framework, there was no need to include probabilistic time functions. Deterministic ones were sufficient for our purpose.

We have presented the use of DEVS in simulating a biological model. The model is based on the metabolic pathway of glycolysis. This tool can be used in any biological model to analyze and understand the basic behaviour of different pathways. The use of DEVS allowed us to have reusable models (in this case, steps 6 to 10 of glycolysis). The concept of internal transition functions can be used to improve the definition of the timing properties of each component, permitting the definition of more complex biological systems. Nevertheless, in this case, most timing delays were represented as simple input/output relations. A complete analysis of the execution flow at the instruction level can be achieved by tracing the execution in the log file, where one can study the flow of a program and each instruction with detail. One also can include new components, change existing ones, or implement new ones. The hierarchical nature of DEVS provides the means to go deeper into the hierarchy.

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 turn 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: [8]

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 27 – Summary of Krebs' Cycle [8]

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				

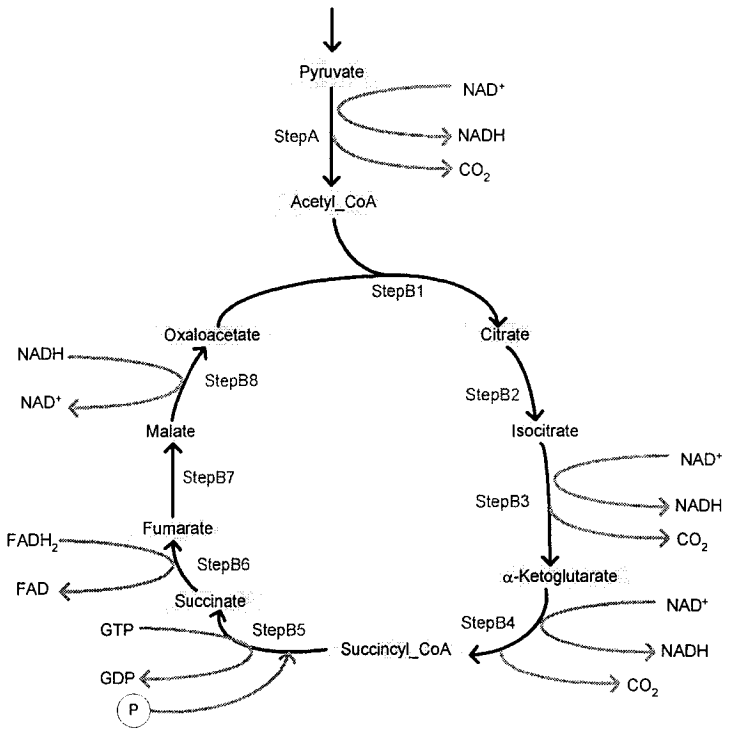


Figure 5.1 – Krebs' Cycle Reactions

5.2 ATOMIC MODELS

The Kreb's cycle was completely implemented using CD++. The source code for the stepA of Krebs cycle is included in section 5.2.1 along with pseudocode, while the source codes 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:

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

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'.

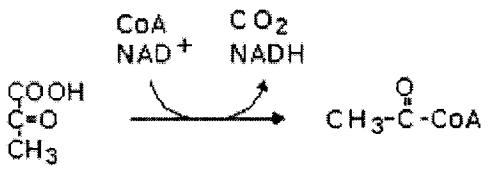


Figure 5.2 - StepA of Krebs' Cycle [24]

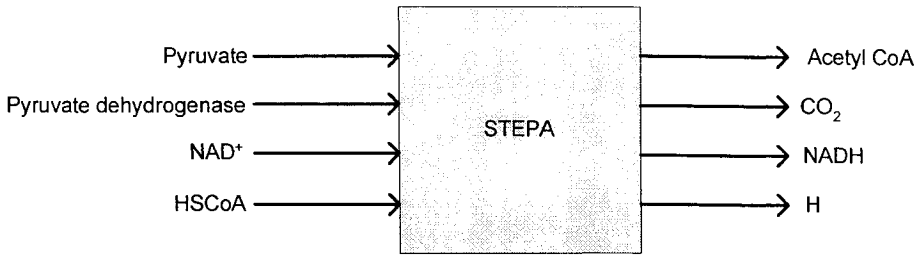


Figure 5.3 - Atomic Model for StepA of Krebs' Cycle

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 (See table 29)

δ_{ext} = external function (See table 28)

ta = Sigma, a real variable that holds the time remaining to the next internal event (in tables 28 and 29, this is represented by the ***holdIn()*** function).

λ = output function (See table 30)

The following table shows the external transition function. This function will be invoked everytime an external event arrives. Here, pyruvate, pyruvateDehydrogenase, HSCoAi, and NAD are the external input events arriving and invoking this method, following the description in Figure 5.1. When an event arrives at any of the input ports, its value is added to a counter kept for that kind of input (*pyruvatec, ifpyruvateDehydrogenase, hscoaic and nadc).*

Once the value of the counter of the input event is incremented, the condition for the other requirements is checked. For example, if the input event is pyruvate, the pyruvate counter value is incremented by the value of the pyruvates entered, and then it is checked against other input counters: hscoaic, nadc and ifpyruvateDehydrogenase. If there is an NAD, an HSCoAi, and pyruvateDehydrogenase are in the system, then the reaction will happen. In the case of pyruvateDehydrogenase, since it is an enzyme and its presence is only needed for the reaction, its value is set to true. The internal transition function then passivates the model, waiting for the next input.

Table 28 – External Function for StepA of Krebs' Cycle (δ_{ext})

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

The following table shows the internal function for stepA of Krebs' cycle. Before calling this method, the sigma value is zero because the interval to the internal transition has expired. When the preparation time interval expires and after calling the output function, this method is invoked (see table 29).

Table 29 – Internal Function for StepA of Krebs' Cycle (δ_{int})

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

Table 30 shows the output function for stepA of Krebs' cycle. When the preparation time interval expires, this method is invoked and the Acetyl_CoA, NADH, CO2, and H have to be sent out through the output ports.

Table 30 – Output Function for StepA of Krebs' Cycle (λ)

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

All the remaining steps were developed using a similar approach. To do this, the behaviour of each component was carefully specified with an analysis of inputs and outputs. Each step was defined as a DEVS model following the specification. Afterwards, each model was implemented in CD++, and tested separately. Once every model was thoroughly tested, the main model was built as a coupled model connecting all the sub-models previously defined. The source codes for all the steps of Krebs' cycle are documented in Appendix B.

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).

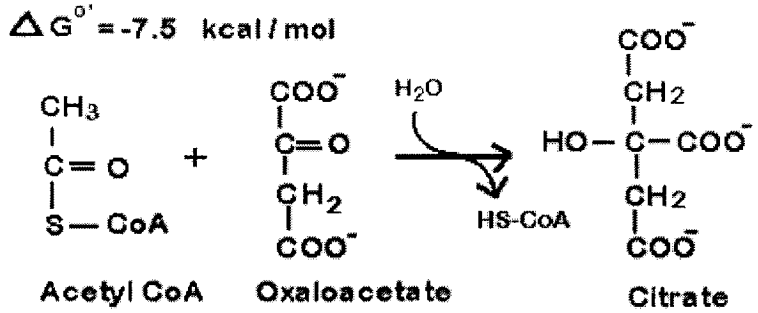


Figure 5.4 - StepB1 of Krebs' Cycle [24]

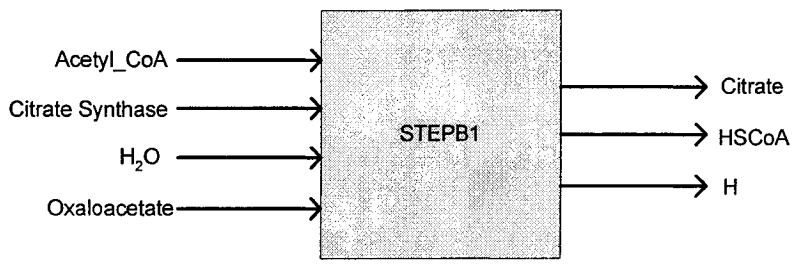


Figure 5.5 - Atomic Model for StepB1 of Krebs' Cycle

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

where

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

X = {acetyl_CoA, oxaloacetate, H2O, citrateSynthase}

Y = {citrate, HSCoAo, H}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix).

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. This reaction was translated to atomic model stepB2 as shown in Figure 5.7, with two input event ports for Citrate and Acontinase, and one output Isocitrate.

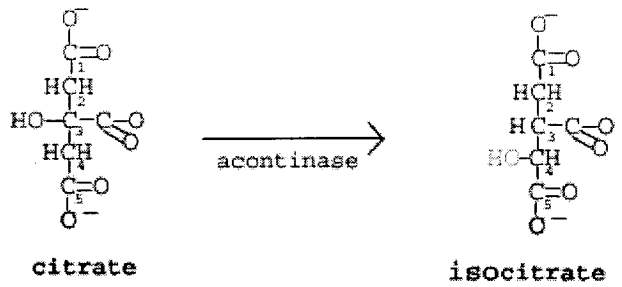


Figure 5.6 – StepB2 of Krebs' Cycle [2]

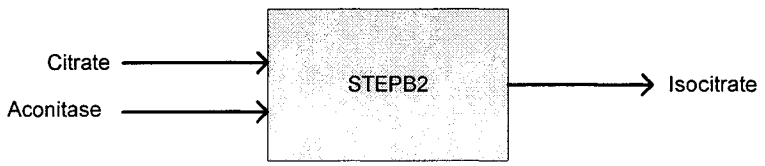


Figure 5.7 - Atomic Model for StepB2 of Krebs' Cycle

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

where

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

X = {citrate, acontinase}

Y = {isocitrate}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix B).

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.

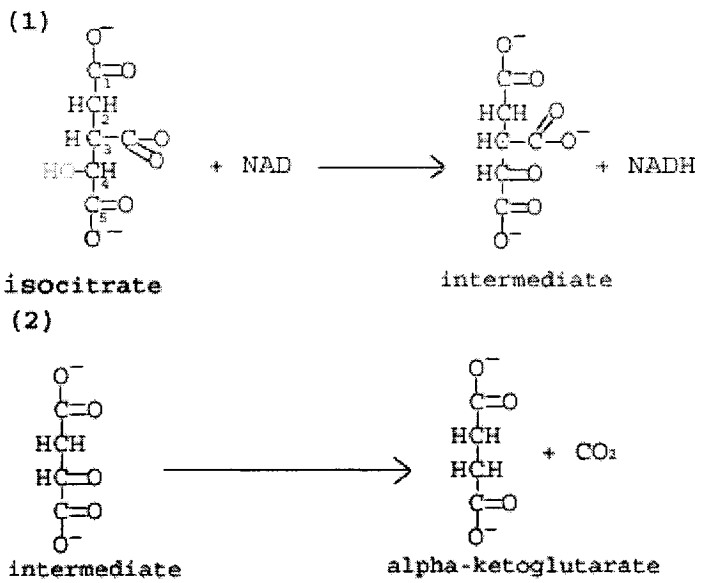


Figure 5.8 – StepB3 of Krebs' Cycle [2]

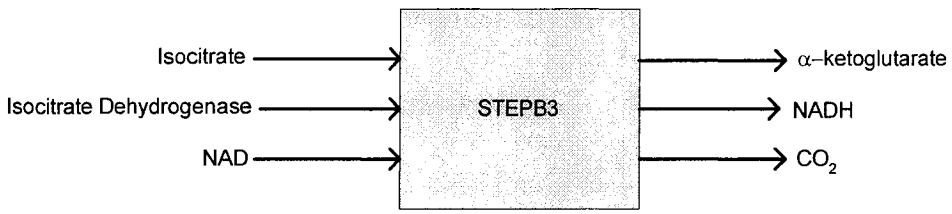


Figure 5.9 - Atomic Model for StepB3 of Krebs' Cycle

$StepB3 = < S, X, Y, \delta_{int}, \delta_{ext}, ta, \lambda >$

where

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

X = {isocitrate, isocitrateDehydrogenase, NAD}

Y = {alpha_ketuglutarate, NADH, CO2}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix B).

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.

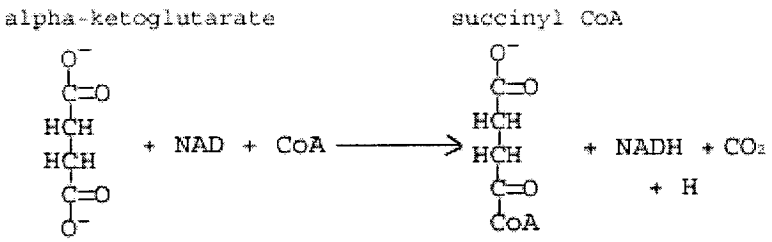


Figure 5.10 – StepB4 of Krebs' Cycle [2]

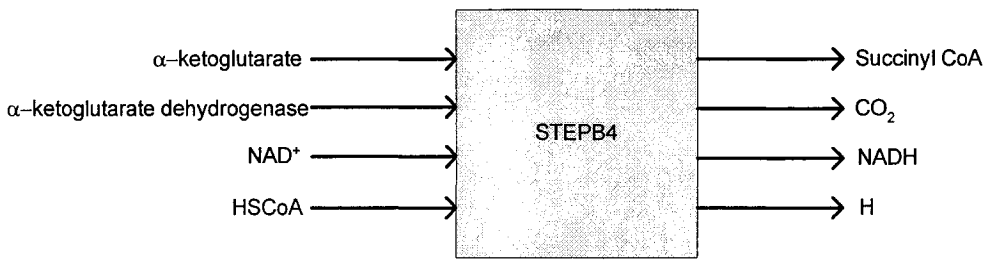


Figure 5.11 - Atomic Model for StepB4 of Krebs' Cycle

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

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}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix B).

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.

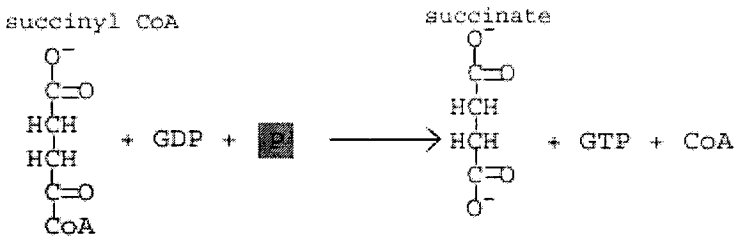


Figure 5.12 – StepB5 of Krebs' Cycle [46]

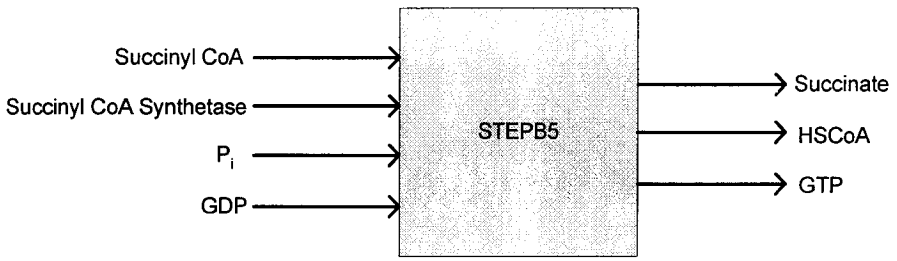


Figure 5.13 – Atomic Model for StepB5 of Krebs' Cycle

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

where

S = {succinyl_CoAc, ifsuccinylCoA_Synthetase, pic, gdpc, counter, phase, sigma}

X = {succinyl_CoA, succinylCoA_Synthetase, Pi, GDP}

Y = {succinate, HSCoAo, GTP}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix B).

5.2.7 STEP B6

Succinate is oxidized to fumarate by succinate dehydrogenase. FAD is tightly bound to this enzyme and is reduced to $FADH_2$. As shown in Figure 5.15, the atomic model of this reaction has three inputs: succinate, succinateDehydrogenase, and FAD, with two outputs : fumarate, and FADH2.

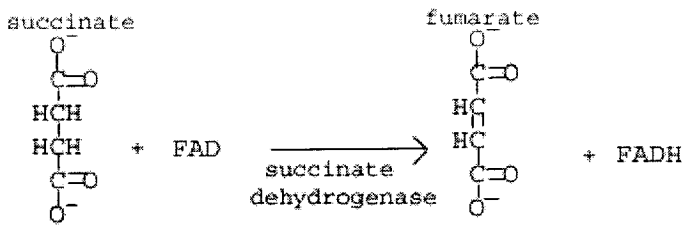


Figure 5.14 – StepB6 of Krebs' Cycle [46]

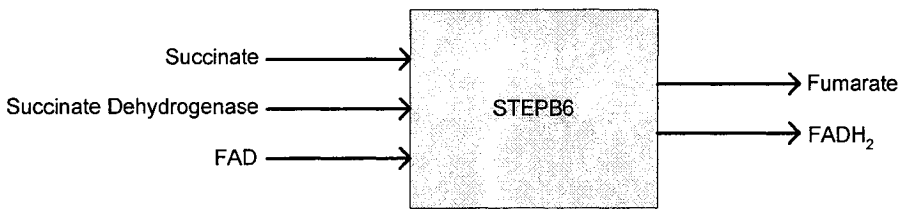


Figure 5.15 - Atomic Model for StepB6 of Krebs' Cycle

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

where

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

X = {succinate, succinateDehydrogenase, FAD}

Y = {fumarate, FADH2}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix B).

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.

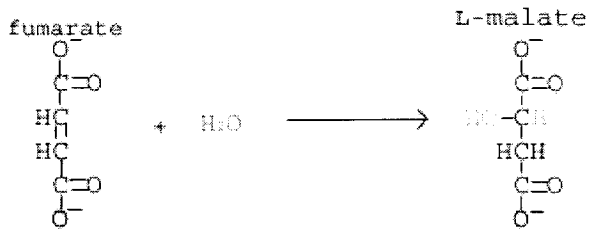


Figure 5.16 – StepB7 of Krebs' Cycle [2]

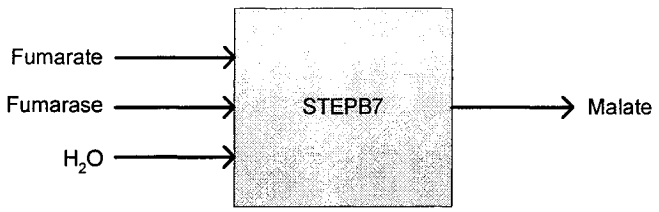


Figure 5.17 - Atomic Model for StepB7 of Krebs' Cycle

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

where

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

X = {fumarate, fumarase, H2O}

Y = {malate}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix B).

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.

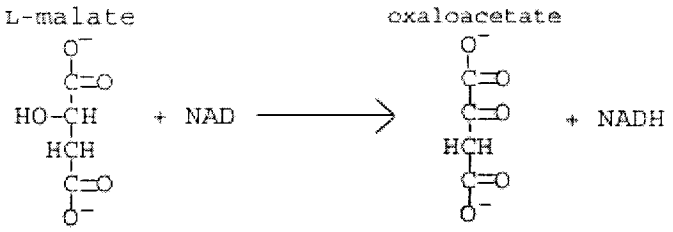


Figure 5.18 – StepB8 of Krebs' Cycle [2]

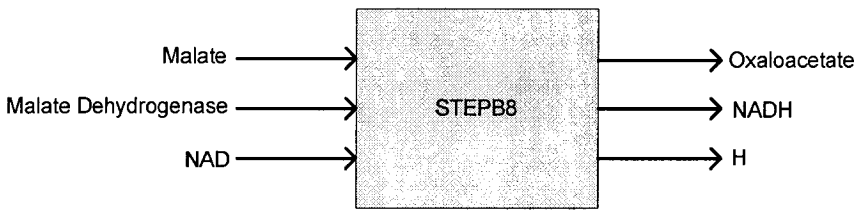


Figure 5.19 - Atomic Model for StepB8 of Krebs' Cycle

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

where

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

X = {malate, malateDehydrogenase, NAD}

Y = {oxaloacetate, NADH, H}

δ_{int}, δ_{ext}, ta and λ are defined as before (CD++ implementation in Appendix B).

5.3 COUPLED MODEL DEFINITION

All the steps explained in the previous section were 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 sub-models previously defined. This model follows the design presented in Figure 5.20, and its detailed definition can be found in Table 31.

Table 31 – 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. The log file keeps a record of all the messages sent between DEVS

processors. Each line of the file shows the message type, the time of the even, the sender and the receiver. In addition, messages of type X or Y will include the port through which the messages was received and the vale received. For messages of type D, the remaining type for the next transition will be shown. A '...' for this field indicates infinity. The numbers between brackets show the ID of the DEVS processor and are provided only for debugging purposes. Only a fragment of the krebs.log is shown in Table 32 (for a full file, refer to Appendix B).

Table 32 – A Section of Krebs.log File

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

The output file from Krebs model (see Table 33) can be checked to verify the validity of the model.

Table 33 - 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 5.20 for a drawing of the Krebs coupled model.

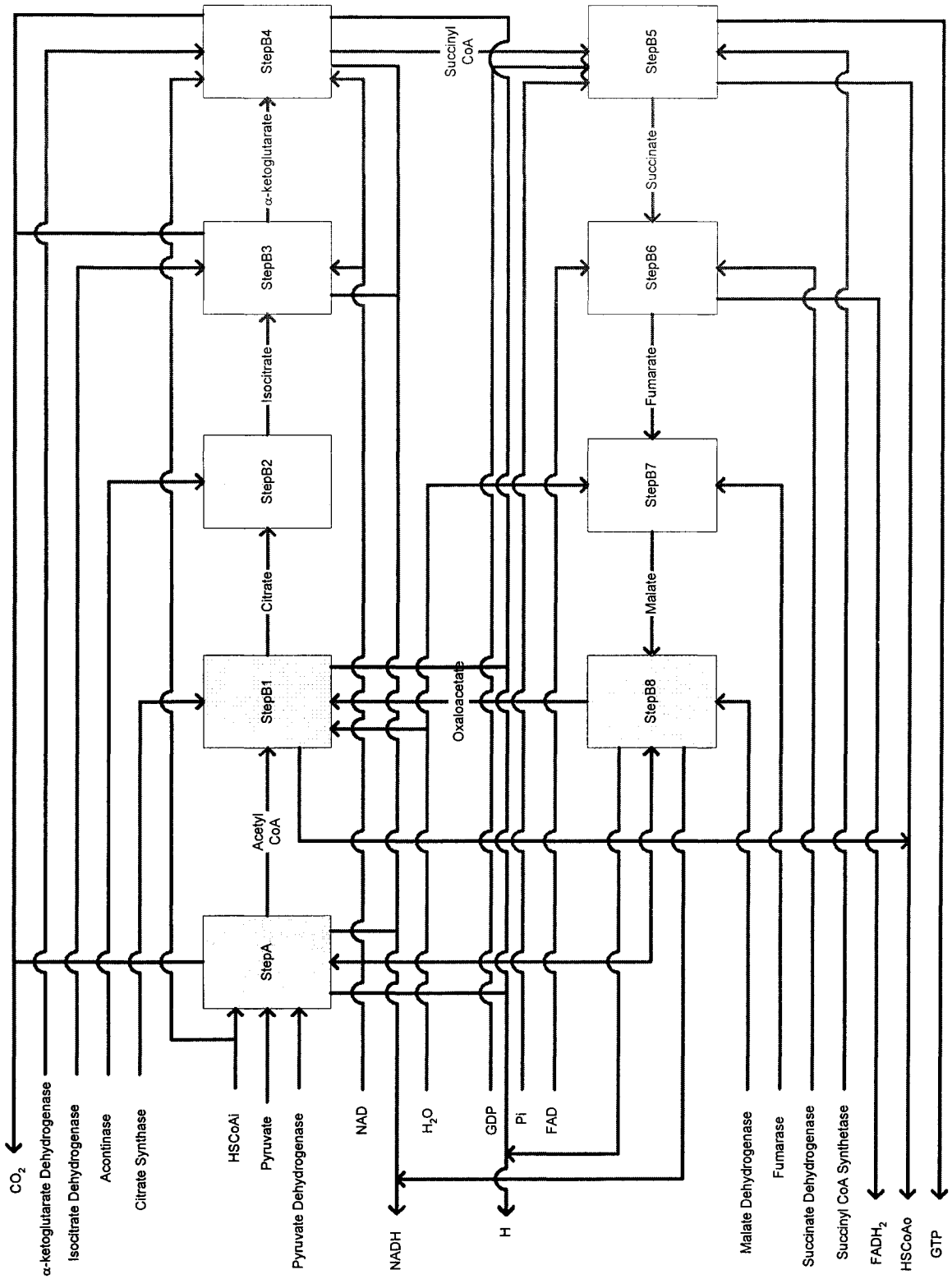


Figure 5.20 – Coupled Model of Krebs' Cycle

5.4 SIMULATION RESULTS

The atomic models for step A, and steps B1 to B8 were tested one by one and then connected to each other to form coupled model and tested again. This section presents the test results of some of these atomic and coupled models described in previous sections.

In the first case, we show the results obtained from simulating the atomic model 'StepA'. Table 34 defines the atomic model for this step. The `[top]` in the first line defines the coupled model at the top level. For atomic components, an instance name and a class name is specified (`components : stepA@StepA`). **Out** enumerates model's output ports (`out : acetyl_CoA NADH CO2 H`), while **In** enumerates the input ports (`in : pyruvate pyruvateDehydrogenase HSCoAi NAD`). **Link** defines the links between the components, and between the components and the coupled model (i.e. `Link: pyruvate pyruvate@stepA`).

Table 34 – Atomic Model Definition for StepA

```
[top]
components : stepA@StepA

out : acetyl_CoA NADH CO2 H
in : pyruvate pyruvateDehydrogenase HSCoAi NAD

Link : pyruvate pyruvate@stepA
Link : pyruvateDehydrogenase pyruvateDehydrogenase@stepA
Link : HSCoAi HSCoAi@stepA
Link : NAD NAD@stepA
Link : acetyl_CoA@stepA acetyl_CoA
Link : NADH@stepA NADH
Link : CO2@stepA CO2
Link : H@stepA H

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

To simulate StepA, the model must be compiled first. To simulate the StepA model manually (without batch file) the following command is typed in the command line:

```
simu -mstep1.ma -estep1.ev -lstep1.log -ostep1.out
```

Once the simulation is finished, a log file is created and can be viewed. Table 35 shows the log file including the messages interchanged for scenario one of StepA (Table 37). For each message, its type, timestamp, value, origin/destination, and the port used for the transmission are shown. We are able to follow the execution flow of this step by analyzing its log file. As one can see, the first two lines are the initialization of the system at time 00:00:00:000 (indicated by letter I). The next two lines are D messages indicating that the model is done with initialization. The lines after indicate the arrival of external events ***Pyruvate*** with value 5 at time 00:00:15:000, ***hscoi*** at time 00:00:25:000 with a value 1, ***nad*** at time 00:00:25:000 with a value 2, and ***pyruvateDehydrogenase*** at time 00:00:42:000 (letter X indicates that these messages are external input events). Towards the end of the log file, one can find the outputs ***nadh***, ***co2***, and ***h*** all with value 1 generated at times 00:00:42:000 (letter Y indicates that these messages represent output events).

Table 35 – Log File for stepA: stepA.log

```
Message I / 00:00:00:000 / Root(00) para top(01)
Message I / 00:00:00:000 / top(01) to stepa(02)
Message D / 00:00:00:000 / stepa(02) / ... to top(01)
Message D / 00:00:00:000 / top(01) / ... to Root(00)
Message X / 00:00:15:000 / Root(00) / pyruvate /      5.00000 to top(01)
Message X / 00:00:15:000 / top(01) / pyruvate /      5.00000 to stepa(02)
Message D / 00:00:15:000 / stepa(02) / 00:00:00:000 to top(01)
Message D / 00:00:15:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:15:000 / Root(00) to top(01)
Message * / 00:00:15:000 / top(01) to stepa(02)
Message D / 00:00:15:000 / stepa(02) / ... to top(01)
Message D / 00:00:15:000 / top(01) / ... to Root(00)
Message X / 00:00:25:000 / Root(00) / hscoai /      1.00000 to top(01)
Message X / 00:00:25:000 / top(01) / hscoai /      1.00000 to stepa(02)
Message D / 00:00:25:000 / stepa(02) / 00:00:00:000 to top(01)
Message D / 00:00:25:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:25:000 / Root(00) to top(01)
Message * / 00:00:25:000 / top(01) to stepa(02)
Message D / 00:00:25:000 / stepa(02) / ... to top(01)
Message D / 00:00:25:000 / top(01) / ... to Root(00)
Message X / 00:00:35:000 / Root(00) / nad /      2.00000 to top(01)
Message X / 00:00:35:000 / top(01) / nad /      2.00000 to stepa(02)
Message D / 00:00:35:000 / stepa(02) / 00:00:00:000 to top(01)
Message D / 00:00:35:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:35:000 / Root(00) to top(01)
```

```
Message * / 00:00:35:000 / top(01) to stepa(02)
Message D / 00:00:35:000 / stepa(02) / ... to top(01)
Message D / 00:00:35:000 / top(01) / ... to Root(00)
Message X / 00:00:42:000 / Root(00) / pyruvatedehydrogenase /      1.00000 to top(01)
Message X / 00:00:42:000 / top(01) / pyruvatedehydrogenase /      1.00000 to stepa(02)
Message D / 00:00:42:000 / stepa(02) / 00:00:00:000 to top(01)
Message D / 00:00:42:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:42:000 / Root(00) to top(01)
Message * / 00:00:42:000 / top(01) to stepa(02)
Message D / 00:00:42:000 / stepa(02) / 00:00:00:000 to top(01)
Message D / 00:00:42:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:42:000 / Root(00) to top(01)
Message * / 00:00:42:000 / top(01) to stepa(02)
Message Y / 00:00:42:000 / stepa(02) / acetyl_coa /      1.00000 to top(01)
Message Y / 00:00:42:000 / stepa(02) / nadh /      1.00000 to top(01)
Message Y / 00:00:42:000 / stepa(02) / co2 /      1.00000 to top(01)
Message Y / 00:00:42:000 / stepa(02) / h /      1.00000 to top(01)
Message D / 00:00:42:000 / stepa(02) / ... to top(01)
Message Y / 00:00:42:000 / top(01) / acetyl_coa /      1.00000 to Root(00)
Message Y / 00:00:42:000 / top(01) / nadh /      1.00000 to Root(00)
Message Y / 00:00:42:000 / top(01) / co2 /      1.00000 to Root(00)
Message Y / 00:00:42:000 / top(01) / h /      1.00000 to Root(00)
Message D / 00:00:42:000 / top(01) / ... to Root(00)
```

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

Table 36 – Input Event Files and Corresponding Output Files Resulted from Simulating StepA of Krebs' Cycle

Scenario	stepA.ev	stepA.out
1	00:00:15:00 pyruvate 5 00:00:25:00 HSCoAi 1 00:00:35:00 NAD 2 00:00:42:00 pyruvateDehydrogenase 1	00:00:42:000 acetyl_coa 1 00:00:42:000 nadh 1 00:00:42:000 co2 1 00:00:42:000 h 1
2	00:00:15:00 pyruvate 5 00:00:25:00 HSCoAi 1 00:00:35:00 NAD 2	
3	00:00:15:00 pyruvate 1 00:00:35:00 NAD 2 00:00:42:00 pyruvateDehydrogenase 1 00:01:25:00 HSCoAi 1 00:02:15:00 pyruvate 5 00:02:35:00 NAD 4 00:03:01:00 HSCoAi 1	00:01:25:000 acetyl_coa 1 00:01:25:000 nadh 1 00:01:25:000 co2 1 00:01:25:000 h 1 00:03:01:000 acetyl_coa 1 00:03:01:000 nadh 1 00:03:01:000 co2 1 00:03:01:000 h 1

In scenario one, the first input event pyruvate enters at time 00:00:15:00 and has a value 5. The second input is HSCOi with a value 1 and it enters at time 00:00:25:00. The third

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

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

The next table shows the model definition for StepB1 of Krebs' cycle:

Table 37 – Model Definition for StepB1 of Krebs' Cycle: stepB1.ma

```
[top]
components : stepB1@StepB1

out : citrate HSCoAo H
in : acetyl_CoA oxaloacetate H2O citrateSynthase

Link : acetyl_CoA acetyl_CoA@stepB1
Link : oxaloacetate oxaloacetate@stepB1
Link : H2O H2O@stepB1
Link : citrateSynthase citrateSynthase@stepB1
Link : citrate@stepB1 citrate
Link : HSCoAo@stepB1 HSCoAo
Link : H@stepB1 H

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

Input and output files for this step are shown for three different scenarios in the following table:

Table 38 – StepA Input Event Files and Corresponding Output Files Resulted from Simulating StepB1 of Krebs' Cycle

Scenario	stepB1.ev	stepB1.out
1	00:00:15:00 acetyl_CoA 5 00:00:20:00 oxaloacetate 3 00:00:25:00 citrateSynthase 1 00:00:35:00 H2O 2	00:00:35:000 citrate 2 00:00:35:000 hscoao 2 00:00:35:000 h 2
2	00:00:15:00 acetyl_CoA 2 00:00:20:00 oxaloacetate 3 00:00:25:00 citrateSynthase 1 00:00:35:00 H2O 2 00:00:55:00 acetyl_CoA 2 00:01:20:00 oxaloacetate 1 00:02:35:00 H2O 3	00:00:35:000 citrate 2 00:00:35:000 hscoao 2 00:00:35:000 h 2 00:02:35:000 citrate 2 00:02:35:000 hscoao 2 00:02:35:000 h 2
3	00:00:15:00 acetyl_CoA 2 00:00:20:00 oxaloacetate 3 00:00:25:00 citrateSynthase 1 00:00:55:00 acetyl_CoA 2 00:01:20:00 oxaloacetate 1 00:02:15:00 H2O 3 00:02:20:00 oxaloacetate 1	00:02:15:000 citrate 3 00:02:15:000 hscoao 3 00:02:15:000 h 3

In the first scenario, there are four inputs ***acetyl_CoA***, ***oxaloacetate***, ***citateSynthase***, and ***H2O*** entering at times 00:00:15:00, 00:00:20:00, 00:00:25:00, and 00:00:35:00. As we can see in the output file, all the outputs are generated at time 00:00:35:00 with a value 2. This means that all the inputs are needed for the reaction in this step to be completed. There are only two of each because the smallest value from the input list, other than the enzymes ***oxaloacetate*** and ***citrateSynthase***, has a value 2 (`00:00:35:00 H2O`). In the second scenario, there are 2 sets of outputs with a total number of 4. This is because there are 4 units of ***acetyl_CoA***, and 5 units of ***H2O***, and the output units will match the lower value of the two. One can verify the results of the scenario three in the same way.

Next, Atomic models of steps A and B1 are coupled together to form a coupled model that we have named StepAB1. This coupled model is tested again to make sure that when

two atomic models tested separately will still work when coupled together. Table 39 shows the coupled model defined for StepAB1. As we can see, this model has two components defined in the components section (components : stepA@StepA stepB1@StepB1).

Table 39 – Definition for Coupled Model StepAB1: stepAB1.ma

```
[top]
components : stepA@StepA stepB1@StepB1

out : citrate HSCoAo H NADH CO2
in : pyruvate pyruvateDehydrogenase HSCoAi NAD oxaloacetate H2O
citrateSynthase

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 : NADH@stepA NADH
Link : CO2@stepA CO2
Link : H@stepA H
Link : HSCoAo@stepB1 HSCoAo
Link : H@stepB1 H
Link : citrate@stepB1 citrate

Link : acetyl_CoA@stepA acetyl_CoA@stepB1

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

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

The following table lists the input event files and the corresponding output files for the coupled model StepAB1. Note that one way to verify the results are to check the list of the outputs and make sure that all the outputs defined in StepAB1.ma are listed (out : citrate HSCoAo H NADH CO2). In scenario one, we can see that all the expected outputs are listed. Three outputs ***nadh***, ***co2***, and ***h*** are generated at time 00:00:01:11:000, indicating that only ***pyruvate***, ***NAD***, ***H2O***, ***HSCoAi***, and ***pyruvateDehydrogenase*** are required for these outputs to be generated. The outputs ***citrate***, ***hscoao***, and ***h*** are generated at time 00:02:01:000 indicating that ***citrateSynthase*** and ***oxaloacetate*** are needed for these

components to be generated. One can further verify this coupled model by examining the scenario 2 and 3.

Table 40 – Input Event Files and Corresponding Output Files Resulted from Simulating Coupled Model StepAB1

Scenario	KrebsAB1.ev	KrebsAB1.out
1	00:00:10:00 pyruvate 3 00:00:25:00 NAD 3 00:00:40:00 H2O 4 00:00:57:00 HSCoAi 2 00:01:11:00 pyruvateDehydrogenase 2 00:01:14:00 citrateSynthase 1 00:02:01:00 oxaloacetate 1	00:01:11:000 nadh 1 00:01:11:000 co2 1 00:01:11:000 h 1 00:02:01:000 citrate 1 00:02:01:000 hscoao 1 00:02:01:000 h 1
2	00:00:10:00 pyruvate 3 00:00:25:00 NAD 3 00:00:40:00 H2O 4 00:00:57:00 HSCoAi 2 00:01:14:00 citrateSynthase 1 00:02:01:00 oxaloacetate 1 00:03:11:00 pyruvateDehydrogenase 2 00:03:57:00 HSCoAi 1	00:03:11:000 nadh 1 00:03:11:000 co2 1 00:03:11:000 h 1 00:03:11:000 citrate 1 00:03:11:000 hscoao 1 00:03:11:000 h 1 00:03:57:000 nadh 1 00:03:57:000 co2 1 00:03:57:000 h 1
3	00:00:25:00 NAD 3 00:00:40:00 H2O 1 00:00:57:00 HSCoAi 2 00:01:14:00 citrateSynthase 1 00:02:01:00 oxaloacetate 1 00:03:11:00 pyruvateDehydrogenase 2 00:03:40:00 H2O 4 00:03:57:00 HSCoAi 1 00:04:10:00 pyruvate 1	00:04:10:000 nadh 1 00:04:10:000 co2 1 00:04:10:000 h 1 00:04:10:000 citrate 1 00:04:10:000 hscoao 1 00:04:10:000 h 1

After testing all the atomic models one by one, and adding them on top of each other and testing them again, we can make sure that all the sub-models are free of error. Once all the basic atomic models are tested, the Krebs' coupled model is formed by connecting all the previously tested atomic models. (See Table 31 for a definition of Krebs coupled model).

The following table shows the input and output files for 3 different scenarios for the Krebs' coupled model:

Table 41 – Input Event and Output Files of Krebs' Coupled Model

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

As we can see in scenario 1, the outputs CoA-SH (***hscoa***), NADH, H, FADH2, GTP, and CO2 can be found in the list. To verify these outputs, we can compare them with the proven classical model taken from any biology book [7]. The sum of all reactions in the Krebs' cycle is summarized as the following reaction:

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

Comparing the two, we can see that we have the outputs listed in formula 5.1. The value associated with these outputs depends on the amount of input gradients. The same thing is true with the scenarios 2 and 3.

Once each of the basic models was tested and each output generated was checked, one could couple them to build a more complicated model with confidence. The procedure was developed using the verification facilities of DEVS, defining a batch file with thousand test cases. The mechanism consists of creating an experimental framework, which executes different input event sets. The execution results are then checked. These values can be easily checked against the values obtained from the classical model. This kind of testing allows us to see any unexpected behaviour. In this way, we could find errors and fix them as we build up the model. Once the execution is over, we take into account the number of times correct outputs were obtained and the number of times incorrect outputs resulted, in order to compute the percentage of success for each simulation. The worse case response times enables a more comprehensive study of the real time performance. This analysis showed good performance in all the cases.

5.5 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. We have presented the use of DEVS in simulating a biological

model. The model is based on the metabolic pathway of Krebs' cycle. This tool can be used in any biological model to analyze and understand the basic behaviour of the different pathways. The use of DEVS allowed us to have reusable models. We are able to follow the execution flow of any program by analyzing its log file.

The concept of internal transition functions can be used to improve the definition of the timing properties of each component, permitting the definition of more complex biological systems. Here, constant (deterministic) time advance for the chemical reactions of Krebs cycle was used. As mentioned before, it is possible to use a probabilistic (stochastic) function for the time advance to model more precisely various steps of chemical reactions. The focus of this work being the development of a flexible framework, there was no need to include probabilistic time functions. Deterministic ones were sufficient for our purpose. An experimental framework was also built for the final integrated model. A complete analysis of the execution flow at the instruction level can be achieved by tracing the execution in the log file, where one can study the flow of a program and each instruction with detail. One also can include new components, change existing ones, or implement new ones. The hierarchical nature of DEVS provides the means to go deeper into the hierarchy. 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.

At this point, this pathway can be connected to glycolytic pathway, and tested. This will present a coupled model of two completely tested sub-modules: Glycolysis, and Krebs'

cycle. This is just the beginning. Our model can start growing to represent even more complex biological systems.

CHAPTER 6 – GRAPHICAL USER INTERFACE

Visualization tools are crucial in helping to better understand the behaviour of complex systems, facilitate thinking, problem solving, and decision-making. Scientific visualization tools create visual displays, in which numeric values in data sets are represented visually as colors, shapes, or symbols. Theis chapter is written in an attempt to explain the tools that visualize the results of DEVS simulation. To facilitate the users to use the CD++ simulator, its design is extended to provide a number of services.

6.1 VISUALIZING THE RESULTS GRAPHICALLY

CD++ was recently provided with facilities for 2D and 3D visualization using VRML and Java. Visualization tools are now an integral part of the CD++ modeling and simulation toolkit. The users can use the CD++ simulator as either a local application or a remote server. Once the simulation is done, the user can analyze the simulation results using different visualization tools [24].

The 2D visualization GUIs are used to visualize the results of atomic models, and coupled 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 visualization 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 [24].

6.2 CD++ MODELER

The CD Modeler is a graphical user interface, which is used to create atomic and coupled models for the CD++ tool. It is a tool for designing and executing DEVS Models in an integrated environment. The basic functions of the GUI include creating atomic and coupled models, exporting the models to different formats and animating the simulations. The GUI is coded in JAVA, which enables it to run on various environments [24].

The CD++ Modeler graphical user interface was used to represent the DEVS model for glycolysis that shows the execution results of the model. Using CD++ modeler, one can define the atomic models and the ports, or import a coupled model. Following figure shows the interface of CD++ Modeler with the glycolysis model being designed in. This figure has two views: the left side of the GUI contains a sorted tree diagram, and the right side contains a visual representation of the model. The atomic models of glycolysis are apparent as the circles. The input and output ports are visible in the tree diagram.

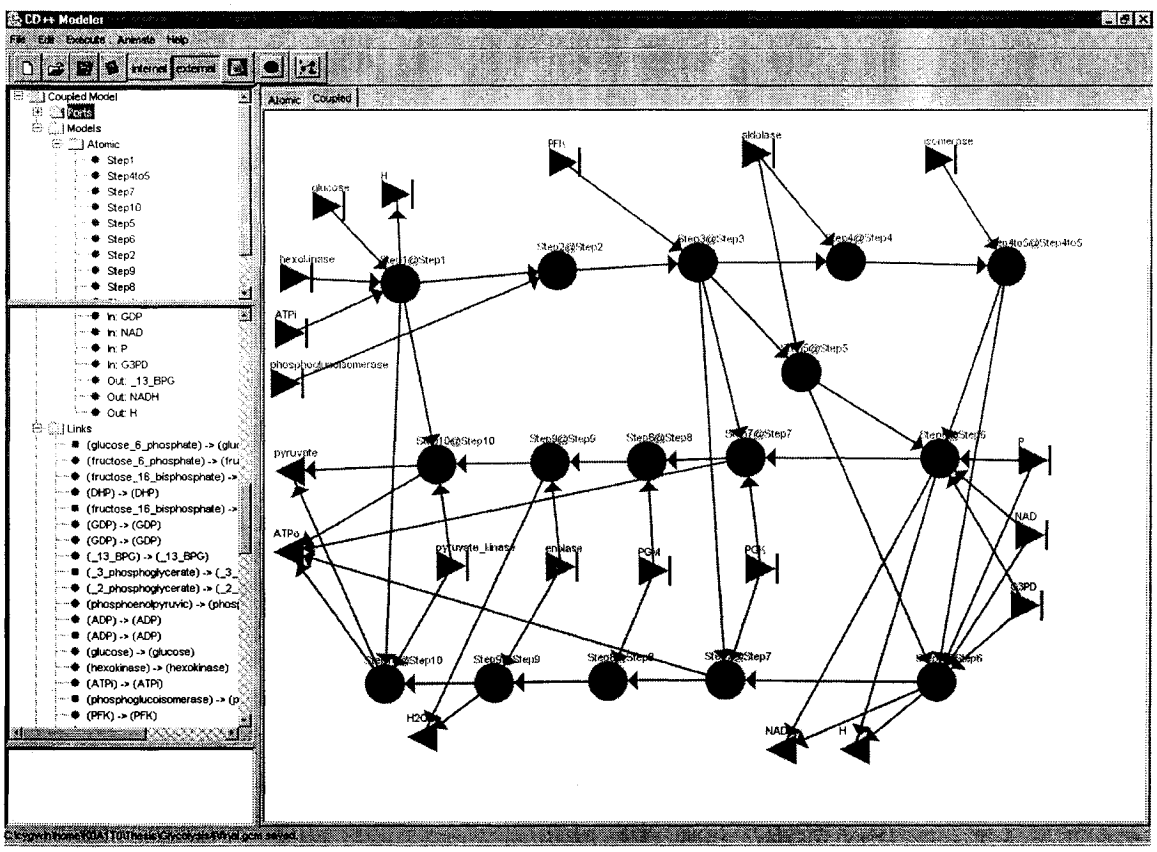


Figure 6.1 – Glycolysis.GCM

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 is generated. Instead of studying the log file records in text files, as shown in chapters 4 and 5, we can visualize the values messages sent and received using the visualization facilities. A graphic display of the output of this model will be displayed as shown in Figure 6.2. The visualization graph displays the model graph and the output value of all the output ports at the same time. The following figure shows a snapshot of coupled animation for glycolysis model.

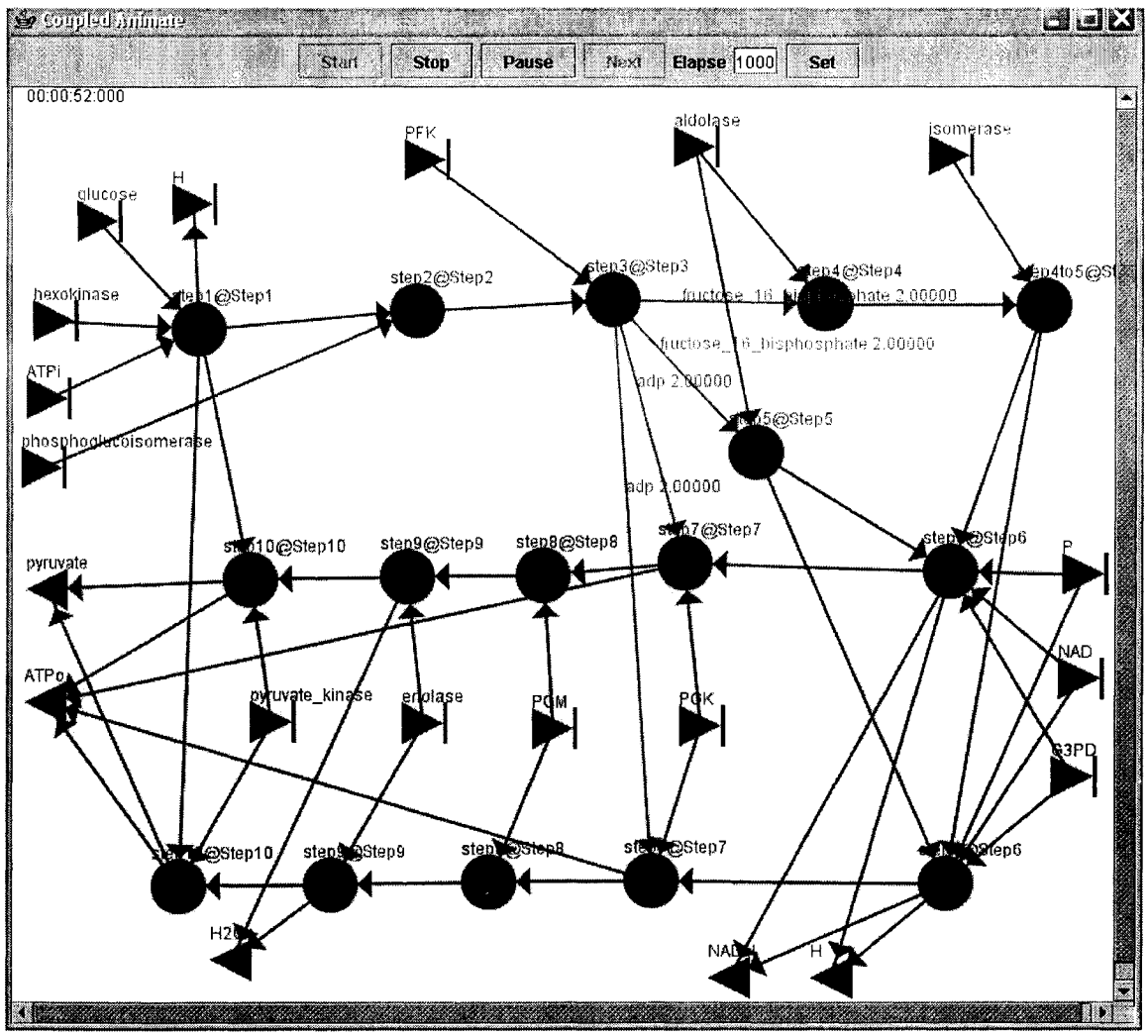


Figure 6.2 – Coupled Animation of Glycolysis

Figures 6.3 and 6.5 are illustrations of the choice components and the name of the visualized models (glycolysis and 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 bottom. 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.

The following table shows the input and output trajectories for the atomic model Step1. Figure 6.3 shows the execution results for Step1 of this model using CD++'s GUI.

Table 42 - Inputs and Outputs for Step1

INPUTS (step1.ev)	OUTPUTS (step1.out)
00:00:15:00 hexokinase 1	00:00:30:000 adp 2
00:00:30:00 glucose 2	00:00:30:000 glucose_6_phosphate 2
00:00:30:00 ATPi 6	00:00:30:000 h 2
00:00:40:00 glucose 4	00:00:40:000 adp 4
00:00:40:00 ATPi 1	00:00:40:000 glucose_6_phosphate 4
00:00:55:00 glucose 1	00:00:40:000 h 4
00:00:55:00 ATPi 1	00:00:55:000 adp 1
00:00:65:00 glucose 1	00:00:55:000 glucose_6_phosphate 1
	00:00:55:000 h 1
	00:01:05:000 adp 1
	00:01:05:000 glucose_6_phosphate 1
	00:01:05:000 h 1

As we can see in Table 44, by time **00:00:30:00**, we have all the three inputs required to generate the outputs. Since hexokinase is an enzyme, only its presence required. At time **00:00:30:00**, two ***glucoses*** and six ***ATPi*** enter the system generating two ***ADP***, 2 ***glucose_6_phosphate***, and two ***H*** molecules. This can be verified by Figure 6.4 where it demonstrates the atomic animation of Step1. At time 00:**00:40:00**, four ***glucose*** molecules enter the system generating four more outputs of each of the ***ADP***, ***glucose_6_phosphate***, and ***H*** molecules. This can be again verified by following the output events timelines with the values shown on top of each timeline. Therefore, the atomic animation makes it easy to verify the values of each output on top of each timeline.

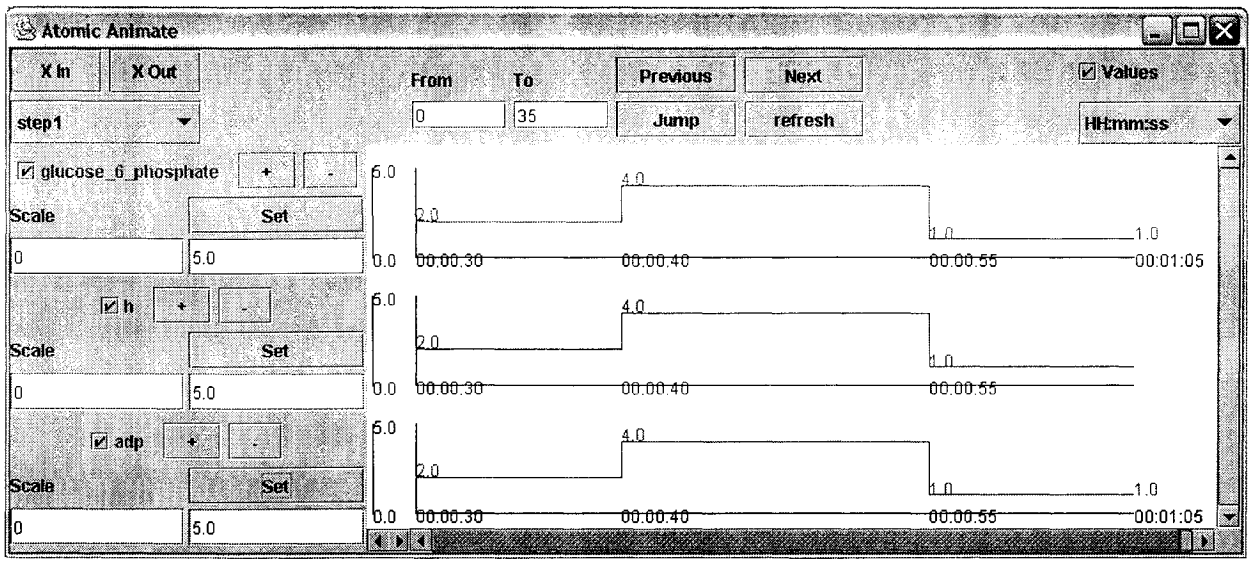


Figure 6.3 – Atomic Animation of Step1 of Glycolysis

The following figure shows the CD++ Modeler graphical user interface to represent the DEVS model for Krebs cycle.

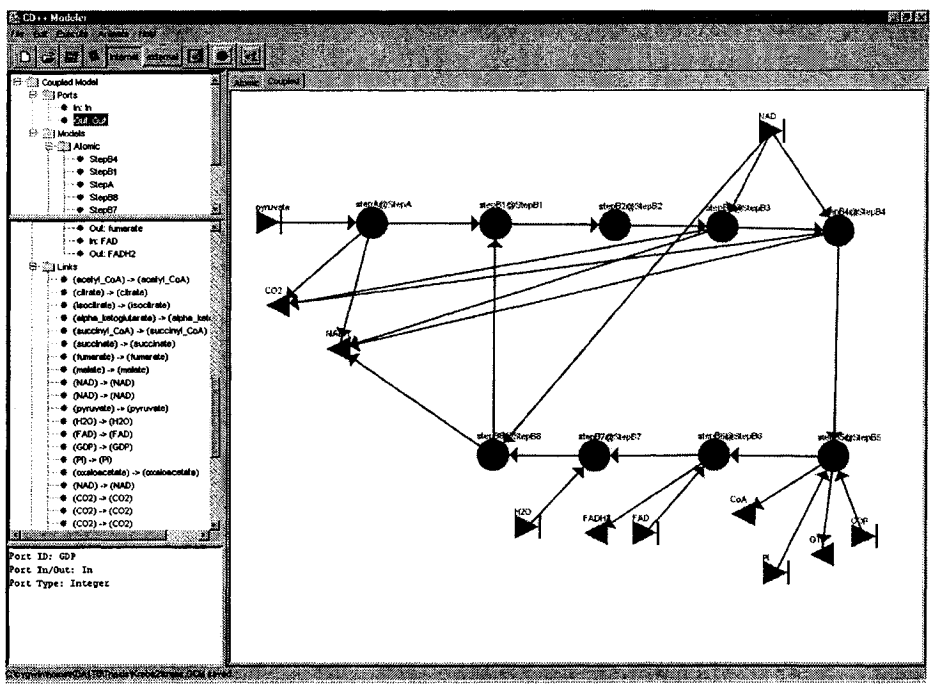


Figure 6.4 – Krebs.GCM

The following table shows a list of the input and output events for Krebs' Model.

Table 43 – Inputs and Outputs for Krebs' Model

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

Figure 6.5 demonstrates the atomic animation for this set of inputs and outputs. For each event chosen in input event file, a timeline starting exactly based on the time specified in table 32 is drawn, with the value of the event marked on the line. Starting from the top, the first time line shows ***isocitrateDehydrogenase*** entering at time **00:02:30:00**, with the value **1** indicated on top of it. The second line is drawn for the input event ***fumarase*** at time **00:03:40:00** with the value **1**. The third line is the timeline for ***FAD*** at time **00:03:15:00** with the value **1**. The rest of the lines are the exact translation of the input event file to timeline visualization.

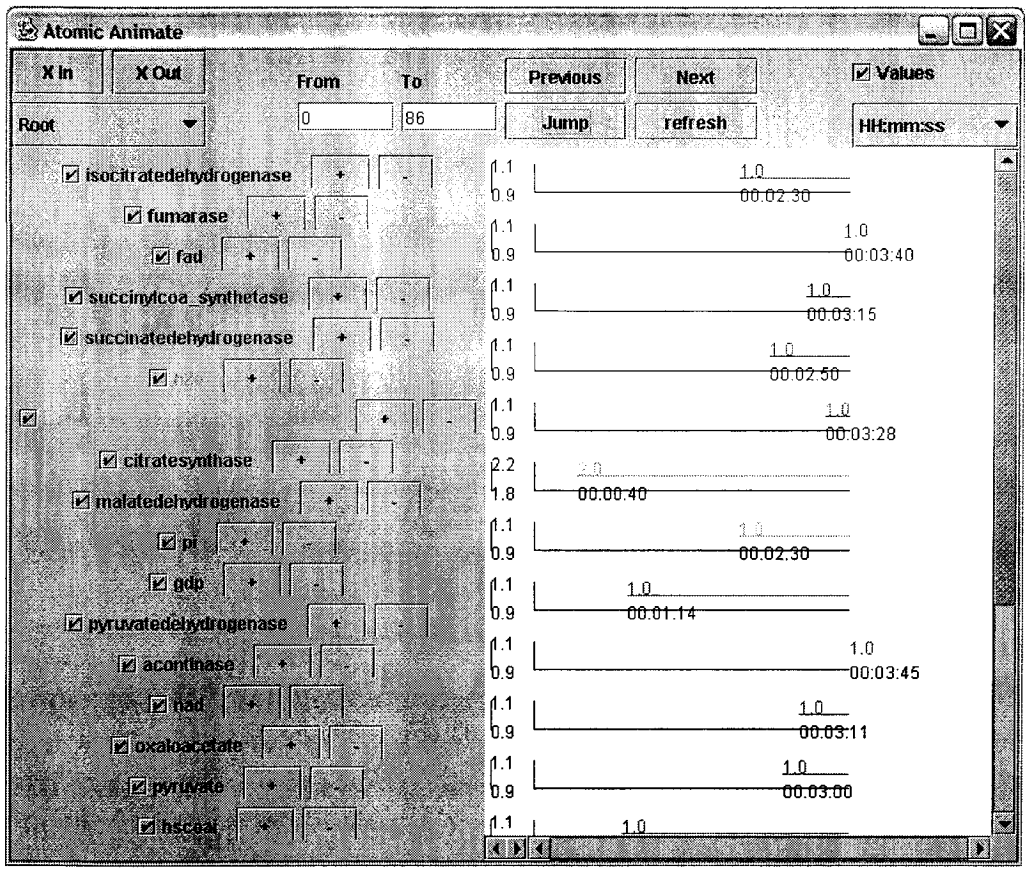


Figure 6.5 – Atomic Animation of Krebs' Cycle

6.3 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 (see Figure 6.6).

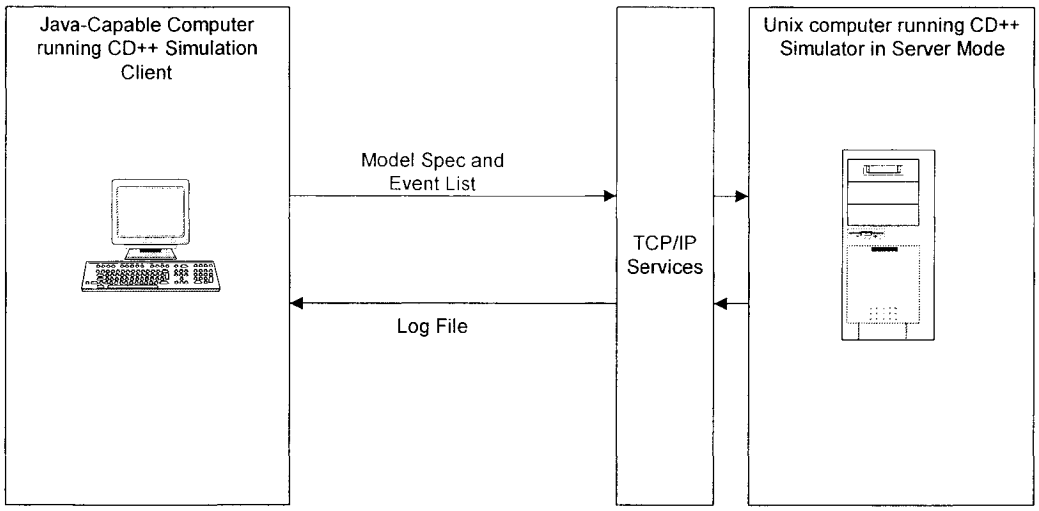


Figure 6.6 – Client-Server Approach

CD++ is sitting in server machine waiting for a model specification (.ma file) 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 [24]:

- 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, etc.)

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.

6.4 CD++/MAYA

CD++/MAYA is another application used as GUI here. Maya [6] is a powerful application for three dimensional modeling and animation, using special effects and rendering. It allows one to create digital imagery, three dimensional animation and visual effects. The Maya software interface is fully customizable and it allows users to extend their functionality within Maya by providing access to the Maya Embedded Language (MEL).

MAYA's modeling and animation tools were used to create three-dimensional environments for each DEVS model. The user must create visual scene files in Maya, while an application, written in MEL permits to create a user interface that allows parsing CD++ model and log files, and to visualize the corresponding model in a 3D visual environment. This instantiates a MEL script specific to a particular model, and animates the three-dimensional world (scene file) in accordance with the CD++ log file. Finally, to facilitate viewing these animations in easily available media players such as Quick Time, Real Player and Windows Media Player, Maya's software renderer can be used create an animation in AVI format [11].

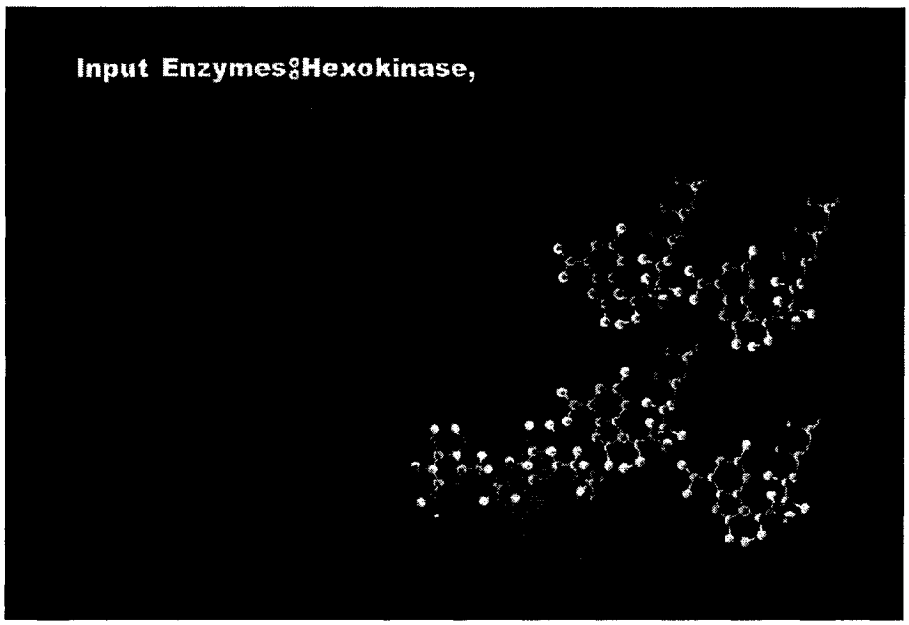
This application allows virtual worlds to be developed using the MAYA visualization environment, and permits interaction with DEVS models built in CD++. The final result is an enhanced simulation environment, which permits improved experimentation. Using these facilities, the users can now develop and test their models in local workstations, and send them to be simulated in a remote CD++ server executing in a high performance platform. Then, they can receive, visualize and analyze the result on the local computer, improving model definition and execution.

The graphics and video for the glycolysis model were created using this software application. 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. 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 DEVS models into 3D representations [11].

6.4.1 3D VISUALIZATION OF GLYCOLYSIS AND KREBS MODELS

In this section, we introduce the results of 3D visualization of models of glycolysis and Krebs' cycle. Simulations of these models allow the researchers to develop a complete picture of the system they study. CD++/MAYA visualization environment can help with this task.

The following snapshots show the visual results of Glycolysis, the sequence of reactions occurring in the cells that permit to break down one glucose molecule into two molecules of pyruvate. Figure 6.7 shows the visual results of the execution of two of the steps. Figure 6.7 a) shows the end of Glycolysis step one, where two G-6-P and two ADP are formed. Figure 6.7 b) shows step 6, which begins at the appearance of three NAD^+ molecules.



(a)

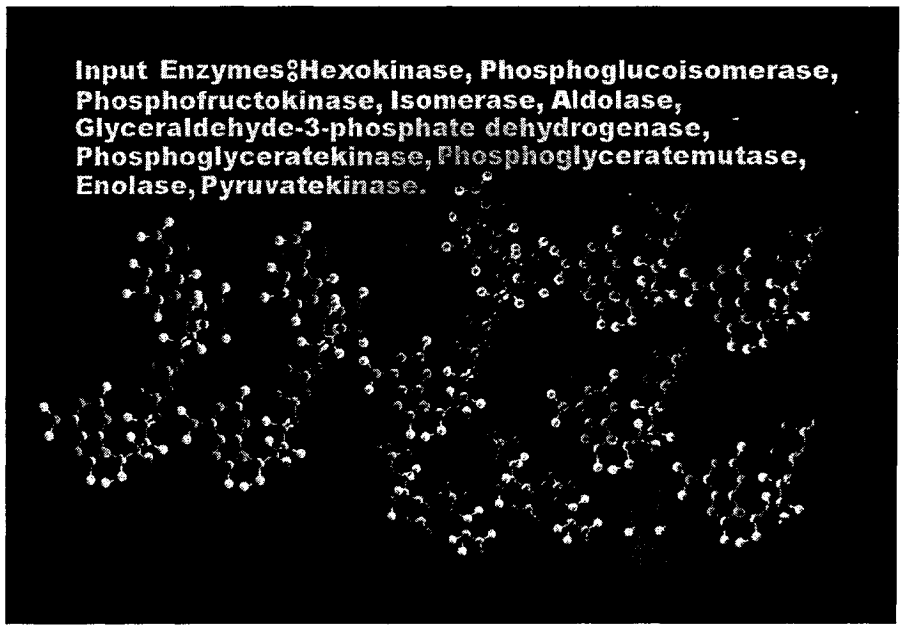
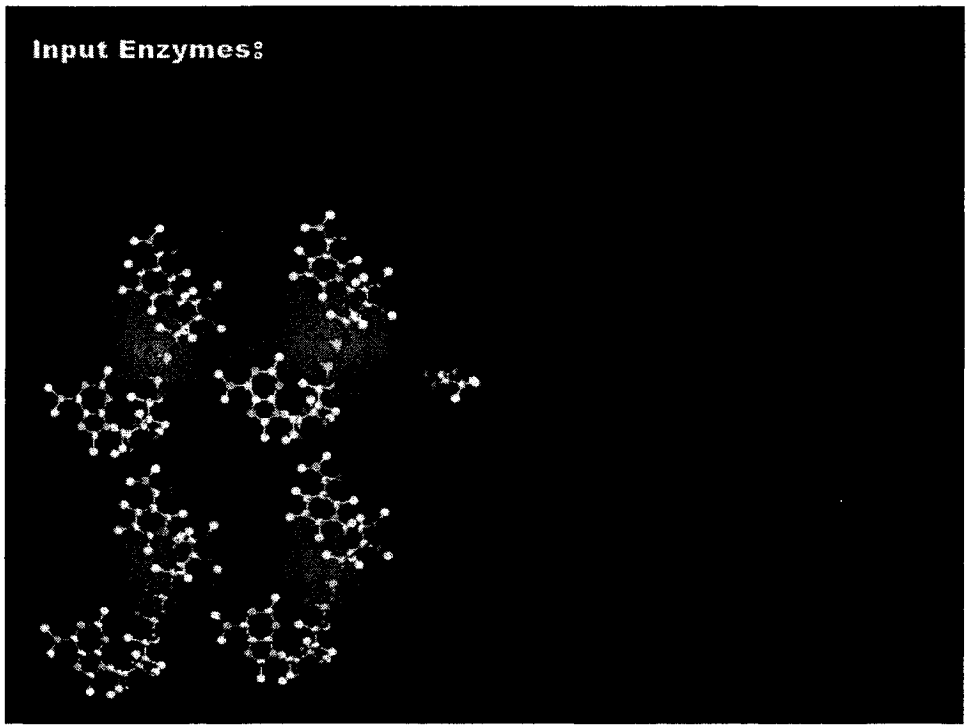


Figure 6.7- (a) Step 1 (b) Step 6

Figure 6.8 shows snapshots of some of the reactions in the Krebs Cycle Animation done in CD++/MAYA. Figure 6.8 a) shows the beginning of the reaction, in which one pyruvate and four NAD^+ appear. Figure 6.8 b) shows the formation of Acetyl CoA, and the production of Carbon dioxide and NADH as by products.



(a)

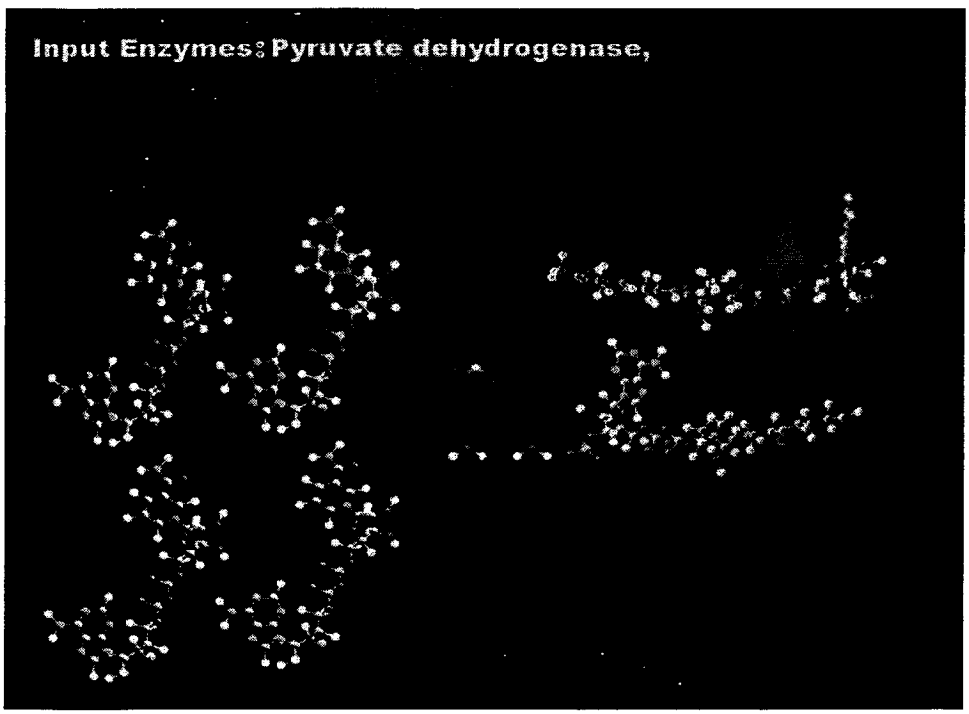


Figure 6.8– (a) The Krebs Cycle Begins (b) Acetyl CoA is Formed

6.5 SUMMARY

Visualization is an important aspect of modeling and simulation. To facilitate the users to use the CD++ simulator, its design is extended to provide a number of services. Different Graphical User Interfaces (GUIs) 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.

The current facilities have highly improved the use of the previously existing tools, thus enhancing the analysis experience of the modelers using the toolkit. 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. DEVS allows independence of the simulator, the models developed, the experiment conducted and the visual engine, while maintaining unity in the model specification and tool interoperation. All these tools provide enough functions for a remote simulation environment in CD++ simulation.

CHAPTER 7 – CONCLUSION

During the last several decades, computer simulation has proved to be a powerful tool 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 an important application of the use of computer simulation applied to the biological field.

The main objective of this thesis was to study the applicability of recent advanced simulation modelling techniques to complex biological systems, specifically the mitochondria. The first step was to use the DEVS (Discrete EVent Systems specification) mathematical formalism to create models of biological pathways, the glycolysis and Krebs's cycle. By combining these models, we then created a higher-order coupled model of the biological pathways of the mitochondria to show how to build a more complex simulation model using a systematic method in which a model consists of a set of lower-level interactions.

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 testing process. DEVS models

are timed, which 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 use of DEVS permits to model the problem even by a non-computer science specialist.

The CD++ tool enables the description, implementation and execution of discrete event models based on the DEVS formalism. The hierarchical nature of DEVS permitted to do this without modifying the original base models. CD++ was used to run the simulations of the biological DEVS models. The high-level language of CD++ reduces the algorithmic complexity for the modeller while allowing complex cellular timing behaviours.

One advantage of the DEVS formalism is the potential of sharing models designed by various people, possibly in different fields and to make them interoperate easily. A second advantage is the possibility that CD++ gives to run simulations in a distributed environment.

This thesis proposed several biological DEVS models and tested them using the CD++ toolkit. This has been accomplished and introduced in Chapters 4 and 5. The use of DEVS can improve the security and reduce the cost in the development of the simulations. The main gains are in the testing and maintenance phases, the more expensive for these systems. More specifically, the main contributions of this thesis are the translation of the biological processes to atomic and coupled DEVS models, the design of the glycolysis and Krebs models using DEVS formalism, the validation of the models with respect to wet-lab results, and to ensure that the proposed models and existing DEVS tools can be used easily by biologists and researcher in medical field. Finally, we showed that we could build complex simulations, evolving incrementally from simple subcomponents to complex systems. This methodology enables the reuse of simulations and components, where the integration of

simulations and components is seamless, makes the testing process more efficient, and makes the testing process more efficient.

7.1 SUGGESTIONS FOR FURTHER RESEARCH

Future research 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++.

The electron transport chain section has not been completely addressed by this thesis because of time constrains, and it has only been introduced partially in Chapter 2 where it can be developed in a Ph.D. thesis. Once this section is completed, 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 make the CD++ simulator easier to use, a CD++ simulation client was introduced that provides a number of services using a client/server approach. It 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 for CD++ models. A partially unresolved issue that can be further studied 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. An appropriate approach to teach well 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 is required. Indeed, merely providing a tool for inputting values and getting the results will not be sufficient in the long run as the technology is improving.

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.

CHAPTER 8 – REFERENCES

[1] "Project Cybercell", {http://www.projectcybercell.ca/}.

[2] "SparkNotes LLC, Part of the Barnes & Noble Learning Network", {http://www.sparknotes.com/biology/}.

[3] "The Silicon Cell: Computing the Living Cell", {http://www.bio.vu.nl/hwconf/Silicon/index.html}.

[4] AIMAR-BEURTON M., KORZENIEWSKI B., LETELLIER T., LUDINARD S., MAZAT J., and NAZARET C., "Virtual Mitochondria: Metabolic Modelling and Control", Molecular Biology Reports 29: 227-232, 2002.

[5] ALBERTS, B.; BRAY, D.; LEWIS, J.; RAFF, M.;ROBERTS, K.; WATSON, J. "Molecular Biology of the cell", Third Edition. Garland Publishing. 1994.

[6] ALIAS Corp. "Maya 6 Features in Detail", {http://www.alias.com/eng/products-services/maya/file/maya6_features _in_detail.pdf}.

[7] CURTIS HELENA and BARNES N. SUE, "Biology", Fifth Edition. W. H. Freeman. 1989.

[8] HAMES B. D. and HOOPER N. M., "Instant Notes in Biochemistry", 2nd Edition. Springer. 2000.

[9] IMMO E. SCHEFFLER, "Mitochondria".

[10] JERRY BANKS, JOHN S. CARSON I, BARRY L. NELSON and DAVID M. NICOL, "Discrete-Event System Simulation", Prentice-Hall International Series.

[11] KATRIN H. and WILLIAM M., "Essence of Mitochondria".

[12] KHAN A. and WAINER G., "A visualization engine based on Maya for DEVS models", Accepted for publication in Proceedings of SISO Fall Interoperability Workshop. Orlando, FL. U.S.A. 2004.

[13] KRAUSS S. "Mitochondria: Structure and Role in Respiration". In: Nature Encyclopedia of Life Sciences. Nature Publishing Group. June 2001. http://www.els.net/ [doi:10.1038/npg.els.0001380]

[14] MACDOUGALL M. H., "Simulating Computer Systems, Techniques and Tools", the MIT Press 1987.

[15] MABER, J., "Step by step Glycolysis", Dept. Biochemistry & Molecular Biology, the University of Leeds, {http://www.jonmaber.demon.co.uk/glysteps}

[16] NAIM A. KHEIR, "Systems Modeling and Computer Simulation", 2nd Edition, 1996.

[17] POULTON J. and BINDOFF L. "Mitochondrial Respiratory Chain Disorders". In: Nature Encyclopedia of Life Sciences. Nature Publishing Group. May 2000. http://www.els.net/ [doi:10.1038/npg.els.0002287]

[18] TAKAHASHI K., KAIZU K., HU B., and 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.

[19] THORSNESS P. and HANEKAMP T. "Mitochondria: Origin". In: Nature Encyclopedia of Life Sciences. Nature Publishing Group. January 2000. http://www.els.net/ [doi:10.1038/npg.els.0001381]

[20] TOMITA M., "E-Cell Project: Towards Whole Cell Simulation", Institute for Advanced Biosciences, Keio University.

[21] 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).

[22] TROCCOLI A., and WAINER G., "Implementing Parallel Cell-DEVS". In Proceedings of Annual Simulation Symposium. Orlando, FL. U.S.A. 2003.

[23] WAINER G., "Discrete Events Cellular Models with Explicit Delays", Doctorate Thesis in Computer Engineering, Université D'Aix-Marseille III, France. December 1998.

[24] WAINER G., CHEN W., CIDRE J., GLINSKY E., LEON S., MONADI A. and TROCCOLI A., "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.

[25] WAINER G., "CD++: a toolkit to define discrete-event models". Software, Practice and Experience. Vol. 32, No.3. pp. 1261-1306. November 2002.

[26] WAINER G. and CHEN W., "A framework for remote execution and visualization of Cell-DEVS models", In Simulation: Transactions of the Society for Modeling and Simulation International. November 2003. pp. 626-647.

[27] 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.

[28] 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)

[29] YUGI K., KANAI Y. and TOMITA M., "Kinetic Modeling of Energy Metabolism in Mitochondria", Genome Informatics 1999, 368-369 (Universal Academy Press;1999)

[30] ZEIGLER B., KIM T., and PRAEHOFER H., "Theory of Modeling and Simulation". Academic Press. 2000.

```
        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;
```

```
                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
                {
```

```
        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;
```

```
* 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: 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();
        }
```

```
/*******************************************************************
* 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 );
        }
```

```
	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)
			{
```

```
/*****************************************************************
* 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();
                }
        }
```

```
        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 ;
}
```

```
                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

```
/*******************************************************************
* 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 );
```

```
        }

        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: 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

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

```
* 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 )
        {
```

```
                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;
                        }
```

```
/** 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
```

```
/** 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 )
```

```
* 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

Message I / 00:00:00:000 / Root(00) to top(01)
Message I / 00:00:00:000 / top(01) to stepa(02)
Message I / 00:00:00:000 / top(01) to stepb1(03)

Message I / 00:00:00:000 / top(01) to stepb2(04)
Message I / 00:00:00:000 / top(01) to stepb3(05)
Message I / 00:00:00:000 / top(01) to stepb4(06)
Message I / 00:00:00:000 / top(01) to stepb5(07)
Message I / 00:00:00:000 / top(01) to stepb6(08)
Message I / 00:00:00:000 / top(01) to stepb7(09)
Message I / 00:00:00:000 / top(01) to stepb8(10)
Message D / 00:00:00:000 / stepa(02) / ... to top(01)
Message D / 00:00:00:000 / stepb1(03) / ... to top(01)
Message D / 00:00:00:000 / stepb2(04) / ... to top(01)
Message D / 00:00:00:000 / stepb3(05) / ... to top(01)
Message D / 00:00:00:000 / stepb4(06) / ... to top(01)
Message D / 00:00:00:000 / stepb5(07) / ... to top(01)
Message D / 00:00:00:000 / stepb6(08) / ... to top(01)
Message D / 00:00:00:000 / stepb7(09) / ... to top(01)
Message D / 00:00:00:000 / stepb8(10) / ... to top(01)
Message D / 00:00:00:000 / top(01) / ... to Root(00)
Message X / 00:00:10:000 / Root(00) / pyruvate / 1.00000 to top(01)
Message X / 00:00:10:000 / top(01) / pyruvate / 1.00000 to stepa(02)
Message D / 00:00:10:000 / stepa(02) / 00:00:00:000 to top(01)
Message D / 00:00:10:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:10:000 / Root(00) to top(01)
Message * / 00:00:10:000 / top(01) to stepa(02)
Message D / 00:00:10:000 / stepa(02) / ... to top(01)
Message D / 00:00:10:000 / top(01) / ... to Root(00)
Message X / 00:00:25:000 / Root(00) / nad / 2.00000 to top(01)
Message X / 00:00:25:000 / top(01) / nad / 2.00000 to stepa(02)
Message X / 00:00:25:000 / top(01) / nad / 2.00000 to stepb3(05)
Message X / 00:00:25:000 / top(01) / nad / 2.00000 to stepb4(06)
Message X / 00:00:25:000 / top(01) / nad / 2.00000 to stepb8(10)
Message D / 00:00:25:000 / stepa(02) / 00:00:00:000 to top(01)
Message D / 00:00:25:000 / stepb3(05) / 00:00:00:000 to top(01)
Message D / 00:00:25:000 / stepb4(06) / 00:00:00:000 to top(01)
Message D / 00:00:25:000 / stepb8(10) / 00:00:00:000 to top(01)
Message D / 00:00:25:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:25:000 / Root(00) to top(01)
Message * / 00:00:25:000 / top(01) to stepa(02)
Message D / 00:00:25:000 / stepa(02) / ... to top(01)
Message D / 00:00:25:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:25:000 / Root(00) to top(01)
Message * / 00:00:25:000 / top(01) to stepb3(05)
Message D / 00:00:25:000 / stepb3(05) / ... to top(01)
Message D / 00:00:25:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:25:000 / Root(00) to top(01)
Message * / 00:00:25:000 / top(01) to stepb4(06)
Message D / 00:00:25:000 / stepb4(06) / ... to top(01)
Message D / 00:00:25:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:25:000 / Root(00) to top(01)
Message * / 00:00:25:000 / top(01) to stepb8(10)
Message D / 00:00:25:000 / stepb8(10) / ... to top(01)
Message D / 00:00:25:000 / top(01) / ... to Root(00)
Message X / 00:00:40:000 / Root(00) / h2o / 2.00000 to top(01)
Message X / 00:00:40:000 / top(01) / h2o / 2.00000 to stepb1(03)
Message X / 00:00:40:000 / top(01) / h2o / 2.00000 to stepb7(09)
Message D / 00:00:40:000 / stepb1(03) / 00:00:00:000 to top(01)
Message D / 00:00:40:000 / stepb7(09) / 00:00:00:000 to top(01)
Message D / 00:00:40:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:40:000 / Root(00) to top(01)
Message * / 00:00:40:000 / top(01) to stepb1(03)
Message D / 00:00:40:000 / stepb1(03) / ... to top(01)
Message D / 00:00:40:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:40:000 / Root(00) to top(01)
Message * / 00:00:40:000 / top(01) to stepb7(09)
Message D / 00:00:40:000 / stepb7(09) / ... to top(01)
Message D / 00:00:40:000 / top(01) / ... to Root(00)
Message X / 00:00:57:000 / Root(00) / hscoai / 2.00000 to top(01)
Message X / 00:00:57:000 / top(01) / hscoai / 2.00000 to stepa(02)
Message X / 00:00:57:000 / top(01) / hscoai / 2.00000 to stepb4(06)

Message D / 00:00:57:000 / stepa(02) / 00:00:00:000 to top(01)
Message D / 00:00:57:000 / stepb4(06) / 00:00:00:000 to top(01)
Message D / 00:00:57:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:57:000 / Root(00) to top(01)
Message * / 00:00:57:000 / top(01) to stepa(02)
Message D / 00:00:57:000 / stepa(02) / ... to top(01)
Message D / 00:00:57:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:00:57:000 / Root(00) to top(01)
Message * / 00:00:57:000 / top(01) to stepb4(06)
Message D / 00:00:57:000 / stepb4(06) / ... to top(01)
Message D / 00:00:57:000 / top(01) / ... to Root(00)
Message X / 00:01:11:000 / Root(00) / pyruvatedehydrogenase / 1.00000 to top(01)
Message X / 00:01:11:000 / top(01) / pyruvatedehydrogenase / 1.00000 to stepa(02)
Message D / 00:01:11:000 / stepa(02) / 00:00:00:000 to top(01)
Message D / 00:01:11:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:01:11:000 / Root(00) to top(01)
Message * / 00:01:11:000 / top(01) to stepa(02)
Message D / 00:01:11:000 / stepa(02) / 00:00:00:000 to top(01)
Message D / 00:01:11:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:01:11:000 / Root(00) to top(01)
Message * / 00:01:11:000 / top(01) to stepa(02)
Message Y / 00:01:11:000 / stepa(02) / acetyl_coa / 1.00000 to top(01)
Message Y / 00:01:11:000 / stepa(02) / nadh / 1.00000 to top(01)
Message Y / 00:01:11:000 / stepa(02) / co2 / 1.00000 to top(01)
Message Y / 00:01:11:000 / stepa(02) / h / 1.00000 to top(01)
Message D / 00:01:11:000 / stepa(02) / ... to top(01)
Message X / 00:01:11:000 / top(01) / acetyl_coa / 1.00000 to stepb1(03)
Message Y / 00:01:11:000 / top(01) / nadh / 1.00000 to Root(00)
Message Y / 00:01:11:000 / top(01) / co2 / 1.00000 to Root(00)
Message Y / 00:01:11:000 / top(01) / h / 1.00000 to Root(00)
Message D / 00:01:11:000 / stepb1(03) / 00:00:00:000 to top(01)
Message D / 00:01:11:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:01:11:000 / Root(00) to top(01)
Message * / 00:01:11:000 / top(01) to stepb1(03)
Message D / 00:01:11:000 / stepb1(03) / ... to top(01)
Message D / 00:01:11:000 / top(01) / ... to Root(00)
Message X / 00:01:14:000 / Root(00) / citratesynthase / 1.00000 to top(01)
Message X / 00:01:14:000 / top(01) / citratesynthase / 1.00000 to stepb1(03)
Message D / 00:01:14:000 / stepb1(03) / 00:00:00:000 to top(01)
Message D / 00:01:14:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:01:14:000 / Root(00) to top(01)
Message * / 00:01:14:000 / top(01) to stepb1(03)
Message D / 00:01:14:000 / stepb1(03) / ... to top(01)
Message D / 00:01:14:000 / top(01) / ... to Root(00)
Message X / 00:02:01:000 / Root(00) / oxaloacetate / 1.00000 to top(01)
Message X / 00:02:01:000 / top(01) / oxaloacetate / 1.00000 to stepb1(03)
Message D / 00:02:01:000 / stepb1(03) / 00:00:00:000 to top(01)
Message D / 00:02:01:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:01:000 / Root(00) to top(01)
Message * / 00:02:01:000 / top(01) to stepb1(03)
Message D / 00:02:01:000 / stepb1(03) / 00:00:00:000 to top(01)
Message D / 00:02:01:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:01:000 / Root(00) to top(01)
Message * / 00:02:01:000 / top(01) to stepb1(03)
Message Y / 00:02:01:000 / stepb1(03) / citrate / 1.00000 to top(01)
Message Y / 00:02:01:000 / stepb1(03) / hscoao / 1.00000 to top(01)
Message Y / 00:02:01:000 / stepb1(03) / h / 1.00000 to top(01)
Message D / 00:02:01:000 / stepb1(03) / ... to top(01)
Message X / 00:02:01:000 / top(01) / citrate / 1.00000 to stepb2(04)
Message Y / 00:02:01:000 / top(01) / hscoao / 1.00000 to Root(00)
Message Y / 00:02:01:000 / top(01) / h / 1.00000 to Root(00)
Message D / 00:02:01:000 / stepb2(04) / 00:00:00:000 to top(01)
Message D / 00:02:01:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:01:000 / Root(00) to top(01)
Message * / 00:02:01:000 / top(01) to stepb2(04)
Message D / 00:02:01:000 / stepb2(04) / ... to top(01)
Message D / 00:02:01:000 / top(01) / ... to Root(00)
Message X / 00:02:14:000 / Root(00) / acontinase / 1.00000 to top(01)

Message X / 00:02:14:000 / top(01) / acontinase / 1.00000 to stepb2(04)
Message D / 00:02:14:000 / stepb2(04) / 00:00:00:000 to top(01)
Message D / 00:02:14:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:14:000 / Root(00) to top(01)
Message * / 00:02:14:000 / top(01) to stepb2(04)
Message D / 00:02:14:000 / stepb2(04) / 00:00:00:000 to top(01)
Message D / 00:02:14:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:14:000 / Root(00) to top(01)
Message * / 00:02:14:000 / top(01) to stepb2(04)
Message Y / 00:02:14:000 / stepb2(04) / isocitrate / 1.00000 to top(01)
Message D / 00:02:14:000 / stepb2(04) / ... to top(01)
Message X / 00:02:14:000 / top(01) / isocitrate / 1.00000 to stepb3(05)
Message D / 00:02:14:000 / stepb3(05) / 00:00:00:000 to top(01)
Message D / 00:02:14:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:14:000 / Root(00) to top(01)
Message * / 00:02:14:000 / top(01) to stepb3(05)
Message D / 00:02:14:000 / stepb3(05) / ... to top(01)
Message D / 00:02:14:000 / top(01) / ... to Root(00)
Message X / 00:02:30:000 / Root(00) / isocitratedehydrogenase / 1.00000 to top(01)
Message X / 00:02:30:000 / top(01) / isocitratedehydrogenase / 1.00000 to stepb3(05)
Message D / 00:02:30:000 / stepb3(05) / 00:00:00:000 to top(01)
Message D / 00:02:30:000 / top(01) / 00:00:00:000 to Root(00)
Message X / 00:02:30:000 / Root(00) / alpha_ketoglutaratedehydrogenase / 1.00000 to top(01)
Message X / 00:02:30:000 / top(01) / alpha_ketoglutaratedehydrogenase / 1.00000 to stepb4(06)
Message D / 00:02:30:000 / stepb4(06) / 00:00:00:000 to top(01)
Message D / 00:02:30:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:30:000 / Root(00) to top(01)
Message * / 00:02:30:000 / top(01) to stepb3(05)
Message D / 00:02:30:000 / stepb3(05) / 00:00:00:000 to top(01)
Message D / 00:02:30:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:30:000 / Root(00) to top(01)
Message * / 00:02:30:000 / top(01) to stepb3(05)
Message Y / 00:02:30:000 / stepb3(05) / alpha_ketoglutarate / 1.00000 to top(01)
Message Y / 00:02:30:000 / stepb3(05) / nadh / 1.00000 to top(01)
Message Y / 00:02:30:000 / stepb3(05) / co2 / 1.00000 to top(01)
Message D / 00:02:30:000 / stepb3(05) / ... to top(01)
Message X / 00:02:30:000 / top(01) / alpha_ketoglutarate / 1.00000 to stepb4(06)
Message Y / 00:02:30:000 / top(01) / nadh / 1.00000 to Root(00)
Message Y / 00:02:30:000 / top(01) / co2 / 1.00000 to Root(00)
Message D / 00:02:30:000 / stepb4(06) / 00:00:00:000 to top(01)
Message D / 00:02:30:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:30:000 / Root(00) to top(01)
Message * / 00:02:30:000 / top(01) to stepb4(06)
Message D / 00:02:30:000 / stepb4(06) / 00:00:00:000 to top(01)
Message D / 00:02:30:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:30:000 / Root(00) to top(01)
Message * / 00:02:30:000 / top(01) to stepb4(06)
Message Y / 00:02:30:000 / stepb4(06) / succinyl_coa / 1.00000 to top(01)
Message Y / 00:02:30:000 / stepb4(06) / nadh / 1.00000 to top(01)
Message Y / 00:02:30:000 / stepb4(06) / co2 / 1.00000 to top(01)
Message Y / 00:02:30:000 / stepb4(06) / h / 1.00000 to top(01)
Message D / 00:02:30:000 / stepb4(06) / ... to top(01)
Message X / 00:02:30:000 / top(01) / succinyl_coa / 1.00000 to stepb5(07)
Message Y / 00:02:30:000 / top(01) / nadh / 1.00000 to Root(00)
Message Y / 00:02:30:000 / top(01) / co2 / 1.00000 to Root(00)
Message Y / 00:02:30:000 / top(01) / h / 1.00000 to Root(00)
Message D / 00:02:30:000 / stepb5(07) / 00:00:00:000 to top(01)
Message D / 00:02:30:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:30:000 / Root(00) to top(01)
Message * / 00:02:30:000 / top(01) to stepb5(07)
Message D / 00:02:30:000 / stepb5(07) / ... to top(01)
Message D / 00:02:30:000 / top(01) / ... to Root(00)
Message X / 00:02:50:000 / Root(00) / succinylcoa_synthetase / 1.00000 to top(01)
Message X / 00:02:50:000 / top(01) / succinylcoa_synthetase / 1.00000 to stepb5(07)
Message D / 00:02:50:000 / stepb5(07) / 00:00:00:000 to top(01)
Message D / 00:02:50:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:02:50:000 / Root(00) to top(01)
Message * / 00:02:50:000 / top(01) to stepb5(07)

Message D / 00:02:50:000 / stepb5(07) / ... to top(01)
Message D / 00:02:50:000 / top(01) / ... to Root(00)
Message X / 00:03:00:000 / Root(00) / gdp / 1.00000 to top(01)
Message X / 00:03:00:000 / top(01) / gdp / 1.00000 to stepb5(07)
Message D / 00:03:00:000 / stepb5(07) / 00:00:00:000 to top(01)
Message D / 00:03:00:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:00:000 / Root(00) to top(01)
Message * / 00:03:00:000 / top(01) to stepb5(07)
Message D / 00:03:00:000 / stepb5(07) / ... to top(01)
Message D / 00:03:00:000 / top(01) / ... to Root(00)
Message X / 00:03:11:000 / Root(00) / pi / 1.00000 to top(01)
Message X / 00:03:11:000 / top(01) / pi / 1.00000 to stepb5(07)
Message D / 00:03:11:000 / stepb5(07) / 00:00:00:000 to top(01)
Message D / 00:03:11:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:11:000 / Root(00) to top(01)
Message * / 00:03:11:000 / top(01) to stepb5(07)
Message D / 00:03:11:000 / stepb5(07) / 00:00:00:000 to top(01)
Message D / 00:03:11:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:11:000 / Root(00) to top(01)
Message * / 00:03:11:000 / top(01) to stepb5(07)
Message Y / 00:03:11:000 / stepb5(07) / succinate / 1.00000 to top(01)
Message Y / 00:03:11:000 / stepb5(07) / hscoao / 1.00000 to top(01)
Message Y / 00:03:11:000 / stepb5(07) / gtp / 1.00000 to top(01)
Message D / 00:03:11:000 / stepb5(07) / ... to top(01)
Message X / 00:03:11:000 / top(01) / succinate / 1.00000 to stepb6(08)
Message Y / 00:03:11:000 / top(01) / hscoao / 1.00000 to Root(00)
Message Y / 00:03:11:000 / top(01) / gtp / 1.00000 to Root(00)
Message D / 00:03:11:000 / stepb6(08) / 00:00:00:000 to top(01)
Message D / 00:03:11:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:11:000 / Root(00) to top(01)
Message * / 00:03:11:000 / top(01) to stepb6(08)
Message D / 00:03:11:000 / stepb6(08) / ... to top(01)
Message D / 00:03:11:000 / top(01) / ... to Root(00)
Message X / 00:03:15:000 / Root(00) / fad / 1.00000 to top(01)
Message X / 00:03:15:000 / top(01) / fad / 1.00000 to stepb6(08)
Message D / 00:03:15:000 / stepb6(08) / 00:00:00:000 to top(01)
Message D / 00:03:15:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:15:000 / Root(00) to top(01)
Message * / 00:03:15:000 / top(01) to stepb6(08)
Message D / 00:03:15:000 / stepb6(08) / ... to top(01)
Message D / 00:03:15:000 / top(01) / ... to Root(00)
Message X / 00:03:28:000 / Root(00) / succinatedehydrogenase / 1.00000 to top(01)
Message X / 00:03:28:000 / top(01) / succinatedehydrogenase / 1.00000 to stepb6(08)
Message D / 00:03:28:000 / stepb6(08) / 00:00:00:000 to top(01)
Message D / 00:03:28:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:28:000 / Root(00) to top(01)
Message * / 00:03:28:000 / top(01) to stepb6(08)
Message D / 00:03:28:000 / stepb6(08) / 00:00:00:000 to top(01)
Message D / 00:03:28:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:28:000 / Root(00) to top(01)
Message * / 00:03:28:000 / top(01) to stepb6(08)
Message Y / 00:03:28:000 / stepb6(08) / fumarate / 1.00000 to top(01)
Message Y / 00:03:28:000 / stepb6(08) / fadh2 / 1.00000 to top(01)
Message D / 00:03:28:000 / stepb6(08) / ... to top(01)
Message X / 00:03:28:000 / top(01) / fumarate / 1.00000 to stepb7(09)
Message Y / 00:03:28:000 / top(01) / fadh2 / 1.00000 to Root(00)
Message D / 00:03:28:000 / stepb7(09) / 00:00:00:000 to top(01)
Message D / 00:03:28:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:28:000 / Root(00) to top(01)
Message * / 00:03:28:000 / top(01) to stepb7(09)
Message D / 00:03:28:000 / stepb7(09) / ... to top(01)
Message D / 00:03:28:000 / top(01) / ... to Root(00)
Message X / 00:03:40:000 / Root(00) / fumarase / 1.00000 to top(01)
Message X / 00:03:40:000 / top(01) / fumarase / 1.00000 to stepb7(09)
Message D / 00:03:40:000 / stepb7(09) / 00:00:00:000 to top(01)
Message D / 00:03:40:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:40:000 / Root(00) to top(01)
Message * / 00:03:40:000 / top(01) to stepb7(09)

Message D / 00:03:40:000 / stepb7(09) / 00:00:00:000 to top(01)
Message D / 00:03:40:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:40:000 / Root(00) to top(01)
Message * / 00:03:40:000 / top(01) to stepb7(09)
Message Y / 00:03:40:000 / stepb7(09) / malate / 1.00000 to top(01)
Message D / 00:03:40:000 / stepb7(09) / ... to top(01)
Message X / 00:03:40:000 / top(01) / malate / 1.00000 to stepb8(10)
Message D / 00:03:40:000 / stepb8(10) / 00:00:00:000 to top(01)
Message D / 00:03:40:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:40:000 / Root(00) to top(01)
Message * / 00:03:40:000 / top(01) to stepb8(10)
Message D / 00:03:40:000 / stepb8(10) / ... to top(01)
Message D / 00:03:40:000 / top(01) / ... to Root(00)
Message X / 00:03:45:000 / Root(00) / malatedehydrogenase / 1.00000 to top(01)
Message X / 00:03:45:000 / top(01) / malatedehydrogenase / 1.00000 to stepb8(10)
Message D / 00:03:45:000 / stepb8(10) / 00:00:00:000 to top(01)
Message D / 00:03:45:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:45:000 / Root(00) to top(01)
Message * / 00:03:45:000 / top(01) to stepb8(10)
Message D / 00:03:45:000 / stepb8(10) / 00:00:00:000 to top(01)
Message D / 00:03:45:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:45:000 / Root(00) to top(01)
Message * / 00:03:45:000 / top(01) to stepb8(10)
Message Y / 00:03:45:000 / stepb8(10) / oxaloacetate / 1.00000 to top(01)
Message Y / 00:03:45:000 / stepb8(10) / nadh / 1.00000 to top(01)
Message Y / 00:03:45:000 / stepb8(10) / h / 1.00000 to top(01)
Message D / 00:03:45:000 / stepb8(10) / ... to top(01)
Message X / 00:03:45:000 / top(01) / oxaloacetate / 1.00000 to stepb1(03)
Message Y / 00:03:45:000 / top(01) / nadh / 1.00000 to Root(00)
Message Y / 00:03:45:000 / top(01) / h / 1.00000 to Root(00)
Message D / 00:03:45:000 / stepb1(03) / 00:00:00:000 to top(01)
Message D / 00:03:45:000 / top(01) / 00:00:00:000 to Root(00)
Message * / 00:03:45:000 / Root(00) to top(01)
Message * / 00:03:45:000 / top(01) to stepb1(03)
Message D / 00:03:45:000 / stepb1(03) / ... to top(01)
Message D / 00:03:45:000 / top(01) / ... to Root(00)