# A THREE-DIMENSIONAL MODELING APPROACH TO PETRI NETWORK DESIGN AND MODELING

By

LINDA KAYE DANCE

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING

UNIVERSITY OF FLORIDA

2001

I dedicate this work to my father, Gilbert Dance, and to my mother, Ruth Dance.

ACKNOWLEDGMENTS

I wish to express my deepest gratitude and appreciation to my mentor and advisor, Dr. Paul A. Fishwick. He instilled within me an enthusiasm and appreciation for the field of modeling and simulation and provided me tremendous motivation, encouragement, and the opportunity for rewarding challenges. I am indebted to him for his time, effort, and patience in teaching and guiding me through the simulation courses, the research work in the simulation lab, and this thesis. I am extremely grateful for the opportunity to work and study under him and I value his knowledge, insight, and vision. I thank him for having confidence in me. I would also like to thank Dr. Douglas Dankel for showing confidence in me from the beginning of my efforts in this program. His support on my behalf is greatly appreciated. Also, I am grateful for the opportunity to be a part of the graduate program and I appreciate all of his efforts as the graduate coordinator. His diligence as both professor and graduate coordinator is to be commended. He has been most helpful and caring and has made my experience throughout the program positive. I would also like to thank Dr. Jörg Peters for his continuous interest in my work through our encounters in the lab. His commitment and enthusiasm to the field of engineering have been an inspiration to me. I am grateful to all three committee members for their time and effort in serving on my committee.

I am grateful to the Computer and Information Science and Engineering Department for the education that I received through my studies and my research and the opportunity to earn this degree. I appreciate the opportunity for both teaching

assistantship and research assistantship. These positions provided financial support but, more importantly, they provided a valuable educational experience. Additionally, I appreciate the understanding and support with my personal family challenges as I completed this goal.

I would also like to thank my coworkers in the Simulation Lab, Robert Cubert, John Hopkins, Andrew Reddish, and Taewoo Kim, for all of their help and support, the many inspiring intellectual conversations, and also for the working relationships and friendships developed. I would like to express appreciation to Renee Lucas and Faye Sheppard for their time and effort in helping me with the grammar in this thesis and I admire their abilities.

I would like to express appreciation to my family for their love, support and encouragement through my graduate studies and for sharing in the successes and struggles of this important goal that I set out to achieve. To both of my parents, I am grateful for the qualities they instilled in me: honesty, strong work ethic, yearning for achievement, to always do my best, and to trust in God. I am grateful to my father for teaching me to be the individual that I am and to set my own goals. Special thanks goes to my mother for her interest, encouragement and prayers during this time. I am grateful to have a very special brother, Thom, in whom I can always find a source of inspiration. I am also very grateful for a very special sister, Faye, who has always shown interest in my academic endeavors and the goals that I set out to achieve. I would also like to acknowledge a special nephew, Joe IV, for providing me inspiration through his enthusiasm and interest in learning.

I am grateful to my friends for all their interest, help, support, encouragement, and understanding while I was so intense and focused on my studies and research that school became my life. I extend my deepest gratitude and appreciation to Gordon and Karen Freas who are very special to me and I thank them for always being there for me and helping me to achieve this goal. Also special thanks go to Bob and Betty Kottman, and Jeff, Kim, and Jodi Knack for being the greatest neighbors and for their interest in my schoolwork and my family.

Ultimately, I thank God for giving me the courage to embark on this journey and the strength to endure throughout.

TABLE OF CONTENTS

page

LIST OF FIGURES

Figure Page

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Engineering

A THREE-DIMENSIONAL MODELING APPROACH TO PETRI NETWORK DESIGN AND MODELING

By

Linda Kaye Dance

May 2001

Chairman: Dr. Paul A. Fishwick
Major Department: Computer and Information Science and Engineering

Models are designed and constructed for the purpose of studying either real or hypothetical systems. Each system is a dynamic system and consists of both static physical and dynamic behavioral relationships. The goal of model design is to communicate these relationships effectively and efficiently.

Visualization is a powerful communications tool, and the use of 3D is magnitudes stronger than its counterpart 2D representation. This research demonstrates that the use of 3D and aesthetics in the model design results in a more powerful model and describes how this is achieved through the rube™ methodology.

The rube™ methodology incorporates the use of metaphors to bring together the two domains, i.e., the real system and the underlying model, into a more meaningful model resulting in high memory retention of the relationships within the model. Metaphors are used for both the static physical objects as well as for the dynamic

behaviors. The use of behavioral metaphors is a powerful concept making the abstract behavior concrete through visual representations. The rube™ research encompasses 3D models with dynamic behaviors, incorporation of aesthetics in model design, utilization of metaphors, web based modeling and simulation, and model reuse and repositories.

Petri net is the dynamic model type upon which this thesis work is based. Petri nets are powerful tools for modeling concurrency and resource contention. The underlying Petri net mathematical formalism is implemented in a model template that includes the static physical structure and the rules of interaction of the Petri net model, resulting in the model behaviors. The classical dining philosophers problem has been implemented as a 3D aesthetic dynamic behavioral model that is effectively communicated and understood. A movie is included to demonstrate the execution of this model.

# CHAPTER 1
# INTRODUCTION

This chapter includes an overview of the general field of study known as modeling and computer simulation. The chapter then presents the key ideas and the purposes of this research. An introduction of the language utilized for the implementation in this research is then presented. An overview of Petri nets, the specific area of interest of this thesis work, is next. The chapter then gives a description of objectives that this research sets out to achieve and the organization of the following chapters.

## Modeling and Computer Simulation

Modeling and computer simulation is a powerful tool useful for problem solving and decision making in almost any area or discipline. Models are created to represent a real or hypothetical system constructed for the purpose of study. To learn about a system, a model is designed and built, and then the output of the model's execution is studied. Computer simulation is the discipline of modeling a system, executing the model on a computer, and analyzing the output from the model execution [1].

All models are objects, and objects are perfect models of themselves [2]. The real system is the target object. The model is the source object, and it contains the attributes and behaviors of the target object, the object being modeled [3]. Modeling is the relationship between the target object and the source object [3]. Figure 1-1 illustrates this relationship.

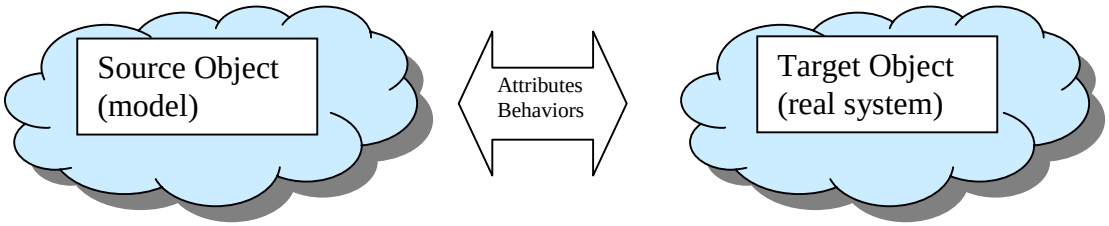



Figure 1-1. Source object and target object relationship.

A model captures the system's static properties, i.e., the physical characteristics, and its dynamic properties, i.e., the behaviors. A good model captures the information and relationships that are important for the intended purpose of the model and that purpose can be functional or aesthetic [4]. A good model, demonstrated in this research, is a model that has a functional purpose along with an equally important aesthetic purpose to enhance the meaning and understanding of the model's functional value.

**3D and Aesthetics in Modeling**

Digital 3D is readily available for everyone today because computer hardware continues to increase in power and speed and decrease in cost. Although 3D is ubiquitous today because of the technological advances, this thesis will answer the question "Why would you design a model in 3D and incorporate aesthetics?" The answer in not that the technology is available to do it, but this section will provide some insight into the motivation and justification for this direction in the field of modeling and simulation. It is the technological advances that allow this direction to proceed and the lack of these technological advances cannot be used for an excuse not to advance the field of modeling and simulation.

From an education perspective, visualization and animation greatly enhance the user's comprehension. Computer visualization is one of the most innovative means of classroom teaching delivery as well as self-learning aids, which is used for all disciplines and at all levels of education. In addition, a realistic, real-time animated visualization has an entertainment value that generates both excitement and enthusiasm for the learning experience. This entertainment value can be key to a better educational experience that encourages curiosity and stimulates exploration to gain more knowledge [5]. The implementation model in Chapter 4, rube™ Model Implementation, demonstrates that this entertainment value also has the same effect in a model.

From a business perspective E-commerce sites that use 3D have experienced increases in sales. ATI's Internet marketing manager, Bal Sahjpaul, describes the impact of 3D with a comparative analogy in stating that, "If a picture is worth a thousand words, then the 3D Web is worth a billion words" [6]. With the belief in the power that 3D has in a business application, it seems clear that applying 3D to modeling and simulation will result in a much more powerful and meaningful model that is ultimately worth multiple times more than its counterpart 2D model.

From an artist's perspective, the relationship of 3D and aesthetics is like a completely new world. John Klima, a New York artist, states that "the appealing aspect of 3D is that it gives the artist tons of artistic elbowroom" [6]. If an artist's perspective were applied to modeling and simulation, the aesthetics that could be incorporated into models would be limitless. The important thing is not to let the aesthetics detract from the model's meaning and understanding but to enhance it and increase the memory retention of the model. The freedom in representing a model in 3D is also like a

completely new world from the traditional 2D representation. This will be seen in the rube™ implementation section of this thesis in Chapter 4.

An *immersive* system allows the viewer to get inside the model, and feel as if he is part of the model. This immersion comes from viewing in 3D and being able to navigate within the 3D space. Examples of the benefits of using 3D and an immersion into the environment are demonstrated by the entertainment industry and the Department of Defense with allowing a game player or a soldier to enter and navigate simulated environments [7]. This feeling of participating within a simulated environment has increased benefits over feeling removed from the observation.

One motivation behind using 3D and aesthetics in modeling is for sensory appeal and immersion into the model. This results in enjoyment and ultimately both increased understanding and better memory retention of the model through the use of metaphors. This may not seem intuitive because of the belief that use of minimalism of the model or within the model results in the best communication. Minimalism in the model can still be obtained with 3D and aesthetics but not to the degree of the 2D model with traditional scientific symbols that lack meaning to the system being modeled and therefore the model is not as effective as the 3D model. The minimalism of a model can be obtained with the use of metaphors because of the communication power of a metaphor within a model. The metaphor minimizes the outward appearance of the model but at the same time adds information to the model that is inherent to the user. The implementation model in Chapter 4 demonstrates this principle.

A key motivation for the use of 3D and aesthetics is that humans think graphically with an innate ability to process graphical information resulting in fast and effective

communication. Further, human information processing of graphical information is involuntary, leaving more conscious problem solving abilities available. This is a strong argument that supports the design of the visualization for the model using 3D and aesthetics. The graphical information of the model will transfer automatically, even more so than 2D graphical information because of the realism added, increasing the automatic transfer so the brain can focus on understanding and problem solving. A good visual design is critical in the communication and understanding of the model [8].

**Visualization of the Behavioral Model**

The behavioral model is an abstract model. It is not as easy to represent behaviors or dynamics as it is to represent static structures or concrete objects. This provides an opportunity for the use of metaphors to have a far greater impact in the model design than the metaphors used in the static structure. This opportunity is also explored in the implementation of a model and successfully demonstrated in Chapter 4, rube™ Model Implementation.

Animation and visual changes of the model to represent the behavioral model through the use of metaphors play an important role in the model in terms of understanding and memory retention. In a hypothetical system, animation provides one of the best methods to validate the system design [8]. Animation provides insight into the model that might otherwise be overlooked. Visualization of “what is happening” provides an understanding of the dynamic behaviors of the system and how they affect the results of the simulation. The animation displays a working model that a user will gain acceptance to because it does the job of communication and displays the results in

an easy to understand form. The communication of results comes with less effort and more enjoyment. This enjoyment factor aids in a deeper understanding of the model.

**Web Based Simulation**

The web is a powerful tool since it provides communication around the world. Because the web is inexpensive, it is accessible to almost everyone. The web has become popular for efficient communications in all sectors, i.e., education, business and government. With this method for sharing information in a fast, cost effective manner, it suggests that web based simulations should be seriously considered as the ideal mode of communication.

Incorporation of 3D graphics, animations, and sound into a web document is needed to provide a model and simulation that incorporates the desired 3D and aesthetics in the model. This is now realizable with the advancements in computer technology. For modeling and simulation, the benefits of web based simulation are (1) fast and efficient accessibility of models for collaboration and education, (2) model and object repositories for reuse, (3) the ability to incorporate real time input into a simulation, (4) and implementation of distributed systems with input from different sites.

The web allows the field of modeling and simulation to make use of the infrastructure already in place to provide fast and effective worldwide access and distribution. The fact that 3D on the web is growing at an incredible rate of speed [9] as the web alone has done indicates that the 3D web based simulations should be the direction for the field of modeling and simulation.

## VRML

VRML, Virtual Reality Modeling Language, is the 3D analog to the 2D HTML, Hyper Text Markup Language. VRML is a multi-platform language that is used as the implementation tool to achieve the objective of designing models in 3D with aesthetics that can dynamically output the behavior as the model is executing.

VRML provides the physical aspect needed for modeling in 3D and the use of a scripting language provides the behavioral aspect to accomplish the dynamic execution of the simulation. VRML allows sound to be incorporated in the world. This results in an enhanced understanding of the behaviors of a model. VRML enables users to feel as though they were inside the model with its navigational abilities within the VRML browser. The user can interact and navigate to a submodel within a multimodel, thus allowing a mechanism to implement a hierarchical presentation of a multimodel by use of models inside objects of a multimodel.

VRML provides a powerful tool for the development and presentation of 3D web based models with more meaning and the ability for reuse via the web. To view a VRML world, a VRML browser plug-in is necessary to enable viewing of the 3D file within an HTML web browser window. The VRML browser plug-in provides visualization, sound, navigation, and interaction in the 3D world that has been created.

## Petri Nets

Petri nets are the specific area in the field of study of modeling and simulation for this thesis work and a Petri net model implementation will be presented. Petri nets are classified as one of the dynamic model types in the field of modeling and simulation defined by Fishwick [1].

A Petri net is a tool that can be applied to study systems by designing a Petri net model. Petri net theory allows the system to be modeled with a Petri net with a mathematical representation of the system [10]. A Petri net is a mathematical tool but the power comes from the utilization of a Petri net as a graphical tool. The graphical side of Petri nets makes it easy to represent different interactions between discrete events so it is an excellent tool to study concurrent events of a system.

A Petri net is classified as a declarative model. Declarative models contain states and events, and the model is viewed as sequences of change in state [1]. A Petri net is a system composed of events and conditions. The conditions of the system define the state of the system at any time. The conditions control the occurrence of the events. The occurrence of events changes the conditions of the system, resulting in a new state. In summary, the conditions control the event, and event occurrence changes the conditions. This describes the high-level view of a Petri net, but it is the component level of the Petri net that the conditions and events are properties of, and the connection of components and their relationships makes it a powerful tool for modeling concurrent events within the system.

The precondition of an event is the condition that allowed the event occurrence and it changes after the event occurs. The post condition of an event is where a condition changes after an event takes place. The concept of flow through the Petri net represents the conditions changing and moving through the structure. Tokens are used to represent the conditions, and they are the component of the Petri net that flows indicating the changing conditions. The state of the system at any point in time is represented by the distribution of tokens in the places of the Petri net. Any Petri net that has a finite number

of states can also be represented as a finite state machine, as demonstrated with the example problem implemented in Chapter 4, Petri Nets Relationship To Other Dynamic Model Types.

## Objectives of Research

The objective of this study is to demonstrate that a model can be designed to effectively and efficiently communicate the static structure and, at the same time, the dynamic behavior while achieving the understanding of relationships within the model. The benefits of using metaphors, 3D, aesthetics, and sound are explored through a model implementation.

## Organization of Thesis

This chapter provided the motivation and justification for this research with some background information about the field in general as well as the specific area of interest. Chapter 2 provides a discussion of related work. Chapter 3 discusses Petri nets and two implementation approaches explored in modeling the dynamic behavior. Chapter 4 explains the rube™ research project and the implementation of a Petri net model to illustrate proof of concept, explore ideas, and aid in the development of the research directions. Finally, Chapter 5 presents the accomplishments and the conclusions of this research and then discusses the future work as a result of this research.

# CHAPTER 2
# RELATED WORK

This chapter covers the related work on various aspects of this thesis. It includes modeling and simulation within the rube™ project, the dynamic model type Petri nets, model reuse and repositories, and web based modeling and simulation.

## rube™ Project

The rube™ project is a research effort under the direction of Dr. Paul Fishwick at the University of Florida. The methodology used in constructing models is the utilization of metaphors in the creation of a model with the incorporation of aesthetics. The impetus for this methodology is to provide an aid in the understanding of the functionality of the system being modeled and to obtain higher memory retention of the model. The rube™ research encompasses web based modeling and simulation, 3D models with dynamic behaviors, and model reuse and repositories. An overview of related work within the rube™ project follows [11].

### Newell's Dynamic Teapot

Paul Fishwick, rube™ Project Director, authored the Newell's Dynamic Teapot [12] dynamic model. Newell's Dynamic Teapot is a multimodel consisting of three dynamic models. The dynamics of the teapot are modeled with a functional block model representing the action of 1) pressing the knob to heat water in the teapot, 2) the heating of the water, and 3) the sensing of the water temperature with a thermometer. The action of heating of the water is modeled with a finite state machine for the change in water temperature from cold to heating to cooling. The third dynamic model is ordinary

differential equation inside each state of the finite state machine to define the dynamics of the state.

Machine pipeline metaphor is used to model the functional block model. Processing Plant metaphors is used to model the finite state machine. Blackboard metaphor is used to model the equation. Dynamic metaphors used are changing transparency, changing color and audio of states.

The model can be found at: http://www.cise.ufl.edu/~fishwick/rube/worlds/tea/tea2.wrl . A screen shot of the model is shown in Figure 2-1.

Figure 2-1. Newell's Dynamic Teapot

**Cassini Probe**

Andrew Reddish, rube™ Research Assistant, created the Cassini Probe [13] dynamic model. The probe's dynamics are modeled as a three-phase transition from 1) probe separation, 2) probe descent to Titan's surface, and 3) impact. Architectural metaphor based on the Greek Parthenon was used to model these three phases. A human avatar moves from one room to the next, modeling the probe's phase transitions. The

underlying dynamic model type is a finite state machine used to model the probe's states and transitions.

The model can be found at: http://www.cise.ufl.edu/~fishwick/rube/worlds/cassini-small/jpl.wrl. A screen shot of the model is shown in Figure 2-2.

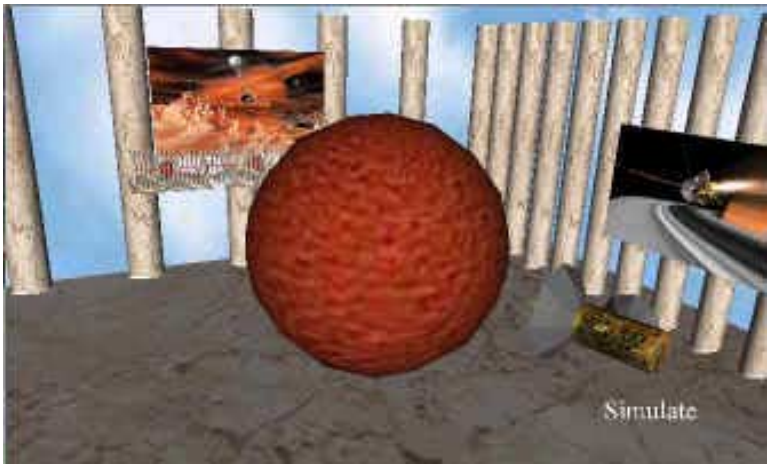



Figure 2-2. Cassini Probe.

**Operating System Kernel**

John Hopkins, rube™ Research Assistant, created the Operating System Kernel [14] model. The dynamics of the operating system kernel are modeled with a queuing network representing the queuing of tasks to schedule with the resources of a computer.

Floor Path metaphors map to paths taken to the resources. Office worker metaphors map to controllers of the resources. Dynamic metaphors used are avatars that move along the paths to model programs or tasks to be executed and serviced by the O/S resources. Audio metaphors are utilized to indicate the O/S resource.

The model can be found at: http://www.cise.ufl.edu/~fishwick/rube/worlds/os/facilityworld.wrl. A screen shot of the model is shown in Figure 2-3.

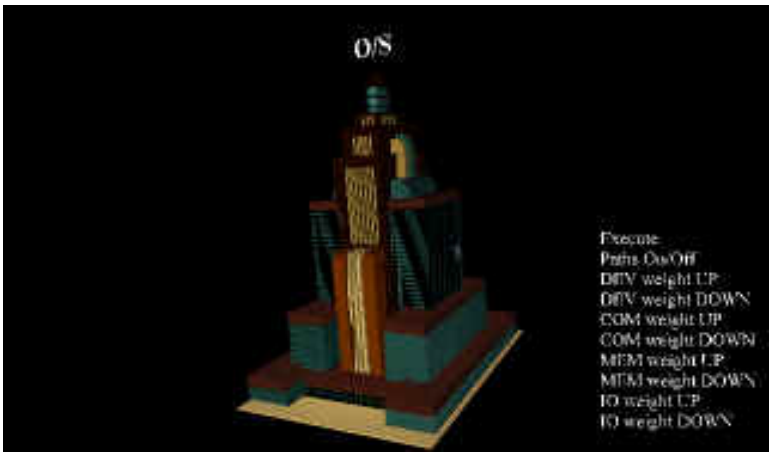



Figure 2-3. Operating System Kernel.

**rube™ Key Characteristics**

The rube™ related work of the project in general, including the above models, is briefly summarized by key characteristics.

The strengths of the rube models are:

- Artistic and immersive model
- 3D objects in 3D space to add realism
- Metaphors utilized to bring meaning into the model
- Effective visualization of dynamic behaviors
- Audio metaphors utilized for the dynamic behaviors to enhance understanding
- Memory retention of model, its objects and the relationship both static and behavioral
- Model reuse with the dynamic model templates
- Web based for efficient distribution

### Java Web Based 2D Dining Philosophers Petri Net

Esser's [15] 2D Petri net browser is a web based Petri Net Applet that includes the dining philosophers problem as one of his standard examples. His tool has the ability to create your own configuration Petri net and execute the model by stepping or run mode. It uses drag and drop objects to build a Petri net.

The strengths of this system are:

- Web based for efficient distribution
- Model reuse with the underlying model type built in
- Step and run mode for execution
- Visualization of dynamic behaviors

The weaknesses of this system are:

- Locked to the traditional 2D symbolism for a graphical representation
- 2D space for layout of system
- Visualization of dynamic behaviors not effective

Specific weaknesses in the dining philosophers example are:

- Layout in 2D space is hard to follow intuitively
- Hard to see static structure of resource contention and 2 states of eat and rest making it hard to follow the execution. (This will become apparent in my implementation in Chapter 4 with the use of metaphors in the layout of space.)
- Concurrency of event occurrence not illustrated (only one transition fires at a time)
- Ineffective communication of the real system through the model

The dining philosophers example can be viewed and executed at: http://www.cs.adelaide.edu.au/users/esser/philos.html. A screen shot of the browser and the layout of the dining philosophers example is shown in Figure 2-4.

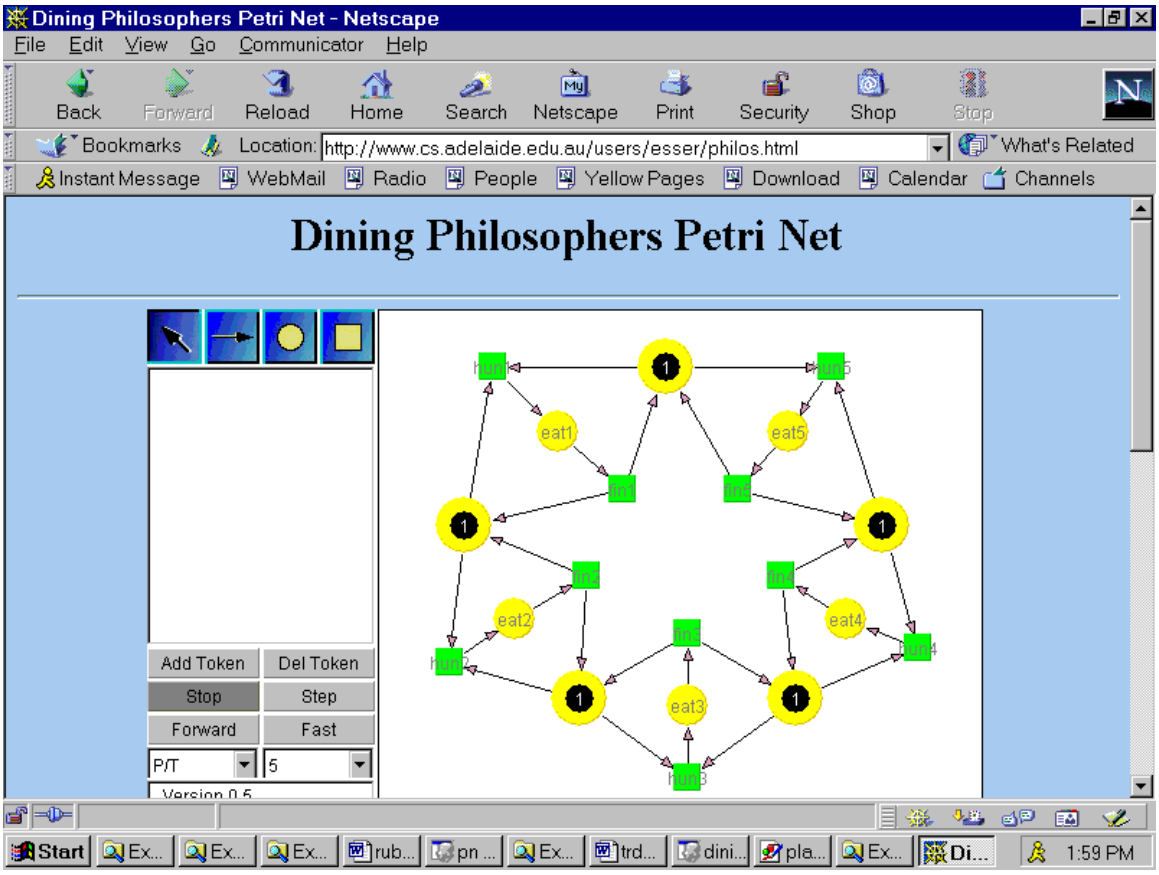



Figure 2-4. 2D dining philosophers model.

## Summary

The significance of the related work is how it helped me with my work for this thesis. The rube related work showed proof of concept of the rube methodology even though all of the models implemented are of different dynamic model types. It was useful to see working models and study the use of metaphors in the design of the model and see the importance to bringing understanding to the model. The dynamic model

templates for those model types also helped me in the implementation of the Petri net dynamic model template by providing a guideline for the starting point.

The Java web based 2D dining philosophers Petri net discussed last taught me most importantly how important the layout in 2D or 3D space is to the communication of the dynamic behaviors. If the layout is not logical to the system that is being modeled, it is much harder to make the connections of the relationships within the model, both the static relationships and the dynamic behavioral relationships.

# CHAPTER 3
# PETRI NETS

This chapter provides the background on the Petri net dynamic model type. This discussion includes the history, the formalism, the graphical representation, and the static structure and the dynamic behavior. The chapter also covers a high-level view of two implementation approaches explored and developed in this thesis.

## History

Petri net theory started with Carl Adam Petri's doctoral dissertation "Kommunikation mit Automaten" in 1962 at the University of Darmstadt, Germany [10]. His work, translated as communication with automata, formulated the basis for a theory of communication between asynchronous components of a computer system. Petri used a network to describe the causal relationships between events in a computer system. Petri is now a professor at the University of Hamburg.

Petri nets continued to develop in the later 60's with the work of Holt and others on the Information System Theory Project of Applied Data Research, Inc. Petri net theory, formalism, and representation of Petri nets developed from this effort. A report of the project "Events and Conditions" resulted which showed how Petri nets could be applied to model systems of concurrent components [10].

Petri net research continued with the Project MAC at Massachusetts Institute of Technology in the Computation Structures Group under the direction of Dennis [10]. Conferences and workshops have been held and there is a special interest group that was

formed in Germany. Interest in Petri nets continues to develop and grow. There is a web site, Petri Nets World, which can be found at http://www.daimi.au.dk/PetriNets. The web site is maintained by the CPN group at University of Aarhus, Denmark and provides online services to the international Petri Nets community including information on the International Conferences on Application and Theory of Petri Nets, mailing lists, bibliographies, newsletters, and research group.

## Formalism

A Petri network structure is a 4-tuple defined as PN = ( P, T, I, O ) [1, 10, 16] where:

$P = \{ p_1, p_2, \ldots , p_n \}$ defined as a finite set of places of size n

$T = \{ t_1, t_2, \ldots , t_m \}$ defined as a finite set of transitions of size m

$I : T \rightarrow P^{\infty}$ defined as the input function

$O : T \rightarrow P^{\infty}$ defined as the output function

I and O, the input and output functions, are the relationship of transitions to places. The function is the mapping from a transition to a bag of places. It is a bag instead of a set to allow for duplication of mapping to a place.

The input function I is the mapping from a transition $t_j$ to a bag of places $I(t_j)$. These are referred to as the input places of transition $t_j$. A place $p_i$ is an input place of transition $t_j$ if $p_i \in I(t_j)$.

The output function O is the mapping from a transition $t_j$ to a bag of places $O(t_j)$. These are referred to as the output places of transition $t_j$. A place $p_i$ is an output place of transition $t_j$ if $p_i \in O(t_j)$.

The initial marking of the Petri net is:

$\mu(p_i)$ = a nonnegative integer ($p_i$ is an element of the set of places)

Every place in the set of places of the Petri net will have its own marking to indicate the state or condition of that place.

## Petri Net Graphs

A Petri net graph is a bipartite directed multigraph. The two major components of the Petri net, places and transitions, are represented as nodes in the graph. Place nodes are represented with a circle O and transition nodes are represented with a bar $|$. Therefore the graph is bipartitie since it has two different types of nodes. It is directed because the direction of flow is imposed by the input and output functions. This direction of flow is represented with an arc $\rightarrow$. The graph is a multigraph because multiple arcs are allowed from a transition node to place nodes and from a place node to transition nodes.

Places are represented with circles and are the passive components of the system. Places store tokens or the condition. The places give the information about the state of the Petri net at any given point in time.

Transitions are represented with bars and are the active components of the system. The active component is an event occurrence such as machines or processors doing an operation.

Arcs are represented with arrows and are the coupling of the two components of the Petri net, places and transitions. The direction of flow between components and throughout the network can be visualized with the arc.

Tokens are represented as filled circles inside a place. Tokens are the moving objects that flow within the system to represent the condition of a place, or, to represent an object that an action is imposed on, or, for the purpose of a transport mechanism to move information through a system [17]. The distribution of tokens in the places of the net at any state is referred to as a marking of the Petri net at that state.

**The Static Structure**

The static structure shows the relationships of places and transitions with the arcs indicating the direction of flow. It also implies a time relationship from the flow through the structure. The static structure alone gives information about precedence relationships, concurrency, and resource contention.

Along with the Petri net physical static structure, the initial state of the model must be defined and it is represented with the initial marking. This initial marking can be represented in several ways, as a condition that is true or false in each place, or, with tokens to represent the number of true conditions in the place.

**The Dynamic Behavior**

The dynamic behavior results from the two transition rules, the enabling rule and the event occurrence rule. The enabling rule is dependent on the marking of the Petri net and the static structure, specifically the transition's input places. A transition is *enabled* at a marking if *all* the places that are in the input function to the transition contain one or more tokens, or a condition of true. Only when the transition is enabled can that transition fire, i.e., the event occurrence, creating a new marking for the next state. The new marking is represented by removing one token from every place that is in the input function of the transition and adding one token to every place that is in the output function of the transition. The event occurrence rule is the control of the *when* of the

event occurrence taking place, i.e., the transition firing. It must be true that the transition has been enabled or the event occurrence cannot take place and then if other controls are imposed (such as time), those conditions must be satisfied. The time control will be discussed as part of the model implementation approaches later in this chapter. The dynamic behavior is illustrated in the following examples.

**The Basic Building Blocks**

The basic building blocks of any Petri net provide illustrations of simple, easy to learn cases. The names given to the structure are from the transition relationship to the places. The marking of the Petri net defines the state of the Petri net at that time. The markings of two states are represented for the state transition. The examples illustrate the initial marking state and the resultant marking after the event occurrence.

One-in--one-out is one input place to the transition and one output place of the transition. The static structure is represent in Figure 3-1.

The formalism is:

PNET = ( P, T, I, O )

P = { $p_1$, $p_2$ }

T = { $t_1$}

$I(t_1)$ = { $p_1$ } $O(t_1)$ = { $p_2$ }

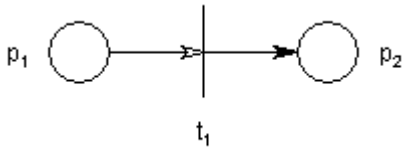



Figure 3-1. Static structure one-in--one-out.

With an initial marking of a single token in the input place, the transition becomes enabled as represented in Figure 3-2. The initial marking of the Petri net is:

$\mu(p_1) = 1 \qquad \mu(p_2) = 0$

In this configuration, the Petri net starts with a single token and the final result after a transition fires is a single token in the output place of the transition. The flow is visualized as a single token that moves out of the input place and then moves into the output place. This is represented in Figure 3-3. The marking of the Petri net after transition $t_1$ fires is:

$\mu(p_1) = 0 \qquad \mu(p_2) = 1$

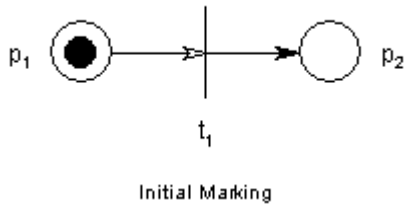



Figure 3-2. Initial marking one-in--one-out.

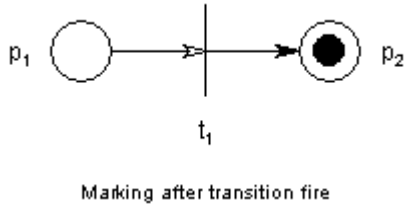



Figure 3-3. Result of transition fire one-in--one-out.

The state trajectories for $p_1$ and $p_2$ are illustrated in Figure 3-4. The unit of time on the x axis is for time representing the advancement of time after the transition fired. The state transition for this example is $(1, 0) \rightarrow (0, 1)$.

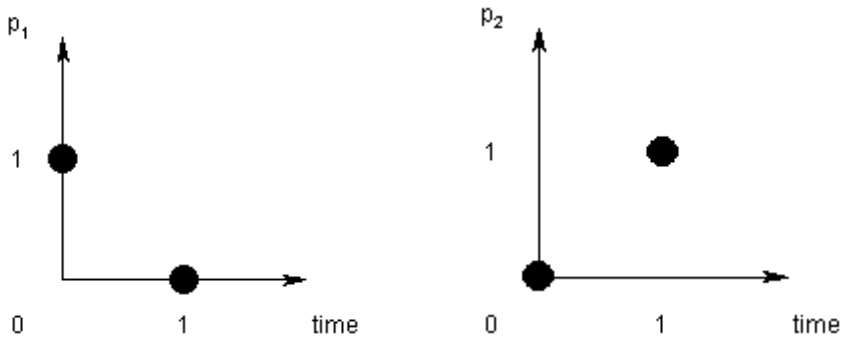



Figure 3-4. State trajectories for one-in--one-out.

One-in--two-out is one input place to the transition and two output places of the transition. This is the concept of a fork. The static structure in represent in Figure 3-5.

The formalism is:

PNET = ( P, T, I, O )

P = { $p_1$, $p_2$, $p_3$ }

T = { $t_1$}

I : T -> $P^{\infty}$

O : T -> $P^{\infty}$

$I(t_1)$ = { $p_1$ } $O(t_1)$ = { $p_2$, $p_3$ }

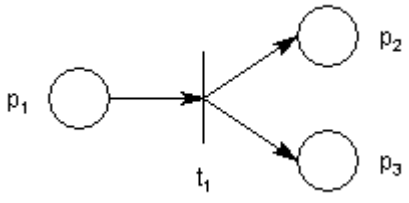



Figure 3-5. Static structure one-in--two-out.

With an initial marking of a single token in the input place, the transition becomes enabled as represented in Figure 3-6. The initial marking of the Petri net is:

$\mu(p_1) = 1 \qquad \mu(p_2) = 0 \qquad \mu(p_3) = 0$

In this configuration, the Petri net starts with a single token in the input place $p_1$ and the final result after a transition fires is two tokens, one in each of the output places, $p_2$ and $p_3$, of the transition. The flow is visualized as the single token moves out of the input place and becomes two tokens; a single token moving to each of the output places. This is represented in Figure 3-7. The marking of the Petri net after transition $t_1$ fires is:

$\mu(p_1) = 0 \qquad \mu(p_2) = 1 \qquad \mu(p_3) = 1$

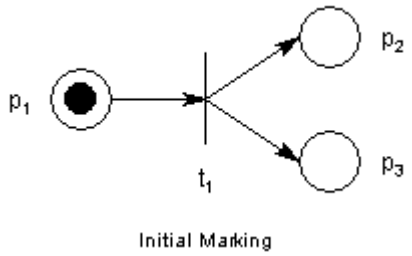



Figure 3-6. Initial marking one-in--two-out.

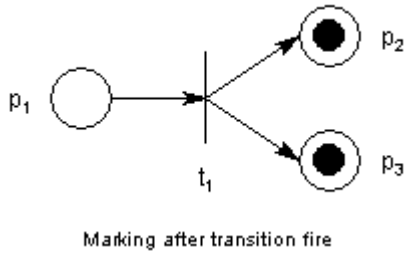



Figure 3-7. Result of transition fire one-in--two-out.

The state trajectories for $p_1$, $p_2$, and $p_3$ are illustrated in Figure 3-8. The state transition for this example is $(1, 0, 0) \rightarrow (0, 1, 1)$.

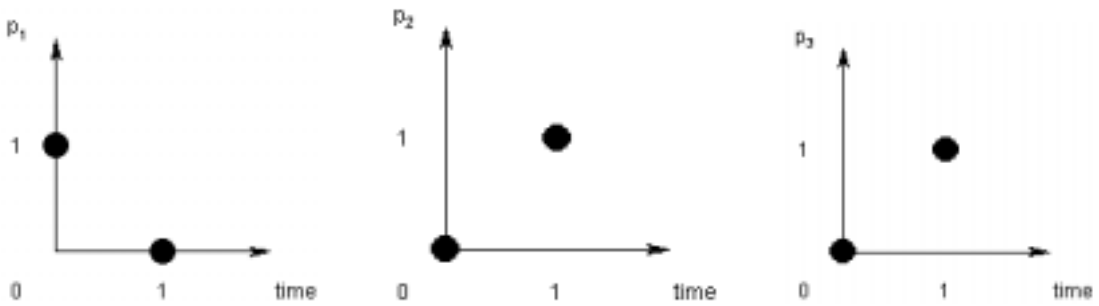



Figure 3-8. State trajectories for one-in--two-out.

Two-in--one-out is two input places to the transition and one output place of the transition. This is the concept of the join. The static structure in represent in Figure 3-9.

The formalism is:

PNET = ( P, T, I, O )

P = { $p_1$, $p_2$, $p_3$ }

T = { $t_1$}

I : T -> $P^{\infty}$

O : T -> $P^{\infty}$

$I(t_1)$ = { $p_1$, $p_2$ } $O(t_1)$ = { $p_3$ }

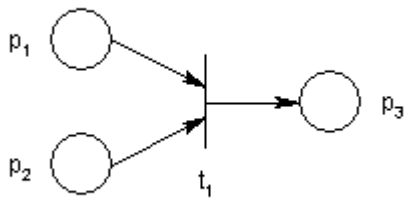



Figure 3-9. Static structure two-in--one-out.

With an initial marking of a single token in each of the input places, the transition becomes enabled as represented in Figure 3-10. The initial marking of the Petri net is:

$\mu(p_1) = 1 \qquad \mu(p_2) = 1 \qquad \mu(p_3) = 0$

In this configuration, the Petri net starts with two tokens; a single token in each of the input places, and the final result after transition $t_1$ fires is a single token, in the single output place of the transition. The flow is visualized as the single tokens move out of both of the input places and become a single token that moves to the output place. This is represented in Figure 3-11. The marking of the Petri net after transition $t_1$ fires is:

$\mu(p_1) = 0 \qquad \mu(p_2) = 0 \qquad \mu(p_3) = 1$

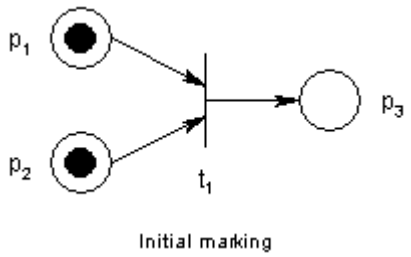



Figure 3-10. Initial marking two-in--one-out.

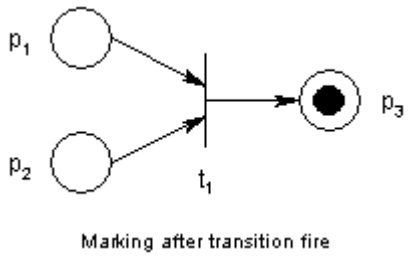



Figure 3-11. Result of transition fire two-in--one-out.

The state trajectories for p1, p2, and p3 are illustrated in Figure 3-12. The state transition for this example is $(1, 1, 0) \rightarrow (0, 0, 1)$.

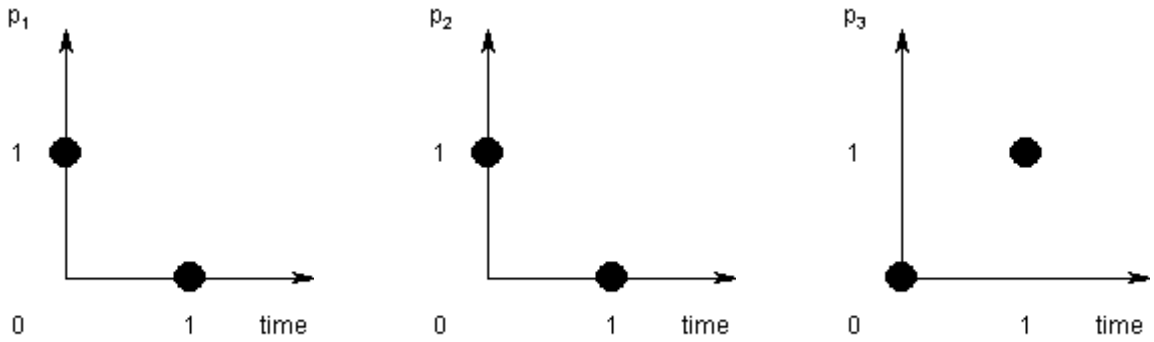



Figure 3-12. State trajectories for two-in--one-out.

Dead net at the initial marking is explained by example. With an initial marking of a single token in only one of the input places to transition $t_1$ is not enabled because the other input place $p_2$ lacks a token. This is a dead net initially because transition $t_1$ can never fire. This is represented in Figure 3-13. The initial marking of the Petri net is:

$\mu(p_1) = 1 \qquad \mu(p_2) = 0 \qquad \mu(p_3) = 0$

The transition $t_1$cannot be enabled and fire until a token arrives to the place $p_2$ that does not contain a token. In this case, place $p_2$ is not an output place, no arc leading into place $p_2$, of any transition so it will never receive a token, therefore the transition can never fire.

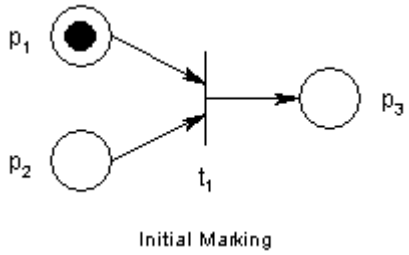



Figure 3-13. Initial marking two-in--one-out not enabled.

The “two-in” of the configuration of Figure 3-9 can be replaced with “n-in” input places. The “two-out” of the configuration of Figure 3-5 can be replaced with “n-out” output places. Any combination of in and out can be used to create the needed structure to represent a part of a system. Multiple arcs can also be used to require multiple tokens from a single input place or to create multiple tokens for a single output place of a transition.

With using combinations of these different building blocks, a system under study can be represented with a Petri net by first creating the static structure. This static structure represents the relationships between the components of the system. The understanding of the dynamic behavior allows a model builder to successfully design and build Petri net models. The understanding of the real system (the target object) and the understanding of the model (the source object) are necessary to represent the correct relationships and interactions in the model design. The mapping of the target and source objects will be discussed in Chapter 4.

## Classification of Petri Nets

Tokens can be either unstructured or structured and Petri nets can be classified by its representation of the tokens. An unstructured token does not contain any information other than it represents a mark or a way to count. A structured token contains information in addition to being a marker.

A Boolean token indicates the condition true or false. Since the token of a place can only be the single condition, the Petri net would have to satisfy the condition that a place never contained more than one token. A Petri net with at most a single token in a

place at one time is representative of the condition-event system where the conditions change between true and false, the Boolean value of the token.

Integer numbers of tokens in a place are any Petri net containing marker tokens without the restriction of the number of tokens in a place at a time. A Petri net with any integer number of tokens used as counters in a place indicate the quantity of tokens contained in a given place at a given time. In most applications, a single token is needed for the transition to fire. If a place to a transition has multiple arcs, the place would need that number of tokens in order for the transition to fire.

Structured tokens are high-level tokens that contain information that allows different types of tokens within a system. These are also referred to as coloured tokens.

Within the levels of token classification, many extensions have been developed. The details of the various extensions can be found throughout the literature.

## Application for Problem Solving

Application of Petri nets can be used in many areas to study and solve problems for complex systems. Any system that has components with complex interactions with other components of the system can be modeled with a Petri net. Some examples include economic systems, legal systems, traffic control systems, chemical systems, communication protocols, manufacturing systems, real-time software, information systems, transport systems, data base management systems, and multimedia. All of these systems involve individual components with interactions among other components of the system. Systems analysis using Petri nets allows analysis on how system components relate to each other before the system is built or to find a problem in an existing system. A good application is the analysis of ladder logic used for controls of very complex

manufacturing equipment so the controls can be tested and verified for a quicker start up and avoidance of costly equipment interference damage from incorrect control logic.

Liveness is important in an application where the number and distribution of moving objects, the flow of tokens of the Petri net, is an important metric. This would include data in a computer, material in a warehouse, documents in an administrative office, or the parts in process in manufacturing production. Failures would be detected if tokens in the Petri net can no longer move. Analysis includes places in the Petri net that will never obtain a token which is starvation, and places in the Petri net that will never lose all its tokens or tokens build up which is a bottleneck.

## Model Implementation Approach

Time, real or simulated, is an assumed variable in any dynamic system or in the execution of a dynamic model. Each new state of the execution implies the advancement of time. The classification given to the two model implementation approaches are based on time as a passive unit of measure used to distinguish states in the model, or, time as an active controlling input in the execution of the model.

If the rules of interaction of the model explicitly depend on time, the model is a time varying model, otherwise the model is time invariant [18]. This is applied to the Petri net model type and specifically to the modeling approaches of this thesis. As described above, the rules of interaction in a Petri net are the two transition rules, the enabling rule and the occurrence rule. The two approaches are based on what is the primary driver of the model. For the model that explicitly depends on time, i.e., the time varying model, time is the primary driving condition of the model and the classification given is a synchronous model approach. For the model that does not depend on time, i.e.,

the time invariant model, event occurrence is the explicit driving condition of the model and the classification given is an asynchronous model approach. A Petri net model can be designed with an asynchronous or a synchronous approach depending on the desired driving force of the model; explicitly time or strictly events.

An event driven model is one where the occurrence of an event initiates the occurrence of the next event due to the change in conditions from a previous event occurrence. The model does not wait on time to advance for the event occurrence, rather, the event occurrence advances the simulated time. The analogy drawn from the asynchronous send receive in message passing in distributed operating systems is the send is nonblocking, i.e., sends message but continues execution, and receive is blocking, waits for the send in order to continue execution [19]. The transition rules process (the send) sends the message to the time process (the receive) to advance the clock. The execution continues with the transition rules process since the time does not affect the outcome of the transition rules process.

A time driven model is one where time is the primary initiator for an event occurrence and that results in a change in conditions of the model. Time must advance to the desired time for the event occurrence to be allowed to take place. The analogy drawn from the synchronous send receive in message passing in distributed operating systems is both send and receive are blocking, waiting for the other to exchange information at the synchronization point and then continue their separate executions independently [19]. The rendezvous of the time process (send) and the transition rules process (receive) is the synchronization point. Both the transition rules process and the time process can proceed only when synchronization between the two processes is achieved.

A Petri Net represents a system with changing conditions resulting from an event that ultimately controls when events are allowed to take place resulting in another new state of the system. The choice of which model implementation approach to use for a particular model is based on the time variable and the time relationship desired for event occurrence. The two approaches will be discussed in more detail in the following sections.

**Asynchronous**

The asynchronous approach is strictly an event driven implementation. The occurrence of an event is the trigger or signal for the next event occurrence opportunity. It is called an opportunity because the event occurrence cannot take place unless the transition was enabled as a result of the previous event that resulted in this trigger. If the resultant change in conditions from the first event occurrence satisfy the rule for the transition enabling, the event occurrence takes place; otherwise, it waits until another event results in the necessary conditions to satisfy the rule for the transition enabling before the event occurrence takes place. The event occurrence takes place as soon as the transition is enabled and that enabling was a result of the previous event, thus the name event driven. The transition fires as soon as a transition is enabled without being time controlled or having any other global control imposed. The order of transitions firing is only a result of when a transition is enabled as a result of the previous event occurrence. This allows concurrency of asynchronous events without imposing restrictions on the timing of concurrency and the actual event occurrence. Each event is totally independent of other independent events in the system.

Event occurrence advances the time to indicate a change in state. The advancement of time shows the “happens before” relationships of events during the

execution. The “happens before” is the viewpoint of a single token that is flowing through the net. Once an event occurs, any other event that occurs that has a precedence relationship where the post conditions of the first event have an affect directly or indirectly on the preconditions of another event are in relationship to a single token. Different events can occur concurrently involving different tokens that are all independently moving through the network.

Because the system is event driven, there must be something to start the Petri net. This initial driver is the initial tokens of the Petri net, each one generating an event and each one seen as an individual event at incremental intervals of the execution. This jump start is a necessary control in the asynchronous approach because it is an event that is the only trigger for the next event to occur resulting in a change of state. The initial tokens must each create a pseudo event so the initial tokens each have a token arrival event.

This approach works for Petri net systems that are stand-alone models, i.e., not part of a multimodel. The system must also be composed only of truly asynchronous events that have no global control for coordination of time, or, a need to check the system as a global system for conflicts to impose a control. The asynchronous approach does not implement a conflict resolution scheme in the case of resource contention because there is no global control to resolve the conflict. Each transition control is independent from all other transitions in the model.

**Synchronous**

The synchronous approach is a time driven controlled implementation. Time controls when an event occurrence takes place if the transition has been enabled from a previous event occurrence. The synchronization of the time control and the transition enabling rule controls the timing of the event occurrence. The time control creates the

next event opportunity, just as the event control created the next opportunity in the asynchronous approach. It is also an opportunity in the synchronous approach because the event occurrence cannot take place unless the transition's enabling rule is satisfied. It this case, it does not necessarily have to be the previous event to satisfy the rule. It only has to be the result of an event within the time interval. If the enabling rule is satisfied, the event occurrence takes place; otherwise, it waits until the next time interval to again check the enabling rule. The transition fires at that time, the same time as all transitions in the Petri net with the time rule satisfied and the enabling rule satisfied therefore, achieving global synchronization of events.

Within the synchronous approach, the clock advances the simulation time and the events happen at the predetermined intervals of time. With the time control, delays can be imposed between events. The reasons for stalling event occurrences can be (1) to achieve concurrency of selected events; (2) to build up a buffer; or (3) to achieve rate differences between different events in the system.

Because time is the control, the initial marking of the Petri net does not need special consideration. Whenever the time condition is met, the transition enabling rule determines the event occurrence. This is another reason that this approach is superior to the asynchronous approach. The initial conditions and the changing conditions of the places are executed in the same way.

The synchronous approach provides flexibility in implementation with the ability to provide ordering of transition firing for conflict resolution thus imposing fairness. The ordering of transitions can affect the results of enabled transitions in terms of being allowed to fire at that time or no longer being enabled because of resource sharing with

another transition that obtained the resource first. This ordering can be the same order every time or a random order with a different random order each time. The result of ordering in the case of resource contention is fairness or starvation. A model implemented with random ordering is a nondeterministic or a probabilistic model. A deterministic model does not make use of a randomizing function. A nondeterministic model uses a randomizing function and for the Petri net model this is utilized because of the need to randomize the transition ordering each time for a model design that incorporates fairness when there is resource sharing and contention. The transition ordering is done at the global level of the Petri net and then the signal is sent to the component level, i.e., the transitions, via the clock.

The synchronous approach can be applied to model all systems. If an asynchronous model (strictly event driven) is desired, it can also be implemented with the synchronous architecture by using a very small time interval which essentially allows the result of the transition that previously fired to be the only affect on the Petri net, therefore attaining a simulated event driven system. The transition that has the place of the previously fired transition in its input function will be the only one enabled and therefore the only transition that fires at that time so the result is the same, albeit a different implementation approach.

One important reason that a synchronous approach should be used is that it allows the Petri net model to be incorporated as a submodel of a multimodel as well as allowing other model types to be submodels of the Petri net. With the Petri net as a submodel of a multimodel, the control of the start of the execution of the Petri net model and continuation of each step of the execution would have to be synchronized to the overall

system. Time as the common variable would be the easiest synchronization technique to accomplish integrating all types of models. A time approach would be using time to synchronize the causally related events necessary for correctness in the multimodel. For example if a Petri net were a submodel of a finite state machine, time in the finite state machine would control the starting of the Petri net or perhaps single transition firings of the Petri net.

Another important reason to model with the synchronized approach is to allow for controlling time between the changes of state so the visualization of the dynamic behaviors during the execution of a model can be captured in the "picture" for understanding. This visualization of the results showing event occurrences where concurrency, conflicts, deadlock, or liveness are important measures of the analysis can be observed in the model execution if sufficient time is allowed to capture and process the information.

**Architecture**

**Asynchronous approach**

For the asynchronous approach, control is at the component level of each individual transition and not globally at the level of the Petri net system as a whole. The only other system components that are affected by the enabled and fired transition are the transition(s) with places in its input function that are the same place as in the output function of the fired transition. This is illustrated in Figure 3-14. Place $p_1$ is the output place of transition $t_1$ and $p_1$ is also the input place to both transitions $t_2$ and $t_3$. Firing of a transition is independent of all other unrelated components of the Petri net. With each transition as its own control for firing, concurrency of unrelated components can happen asynchronously.

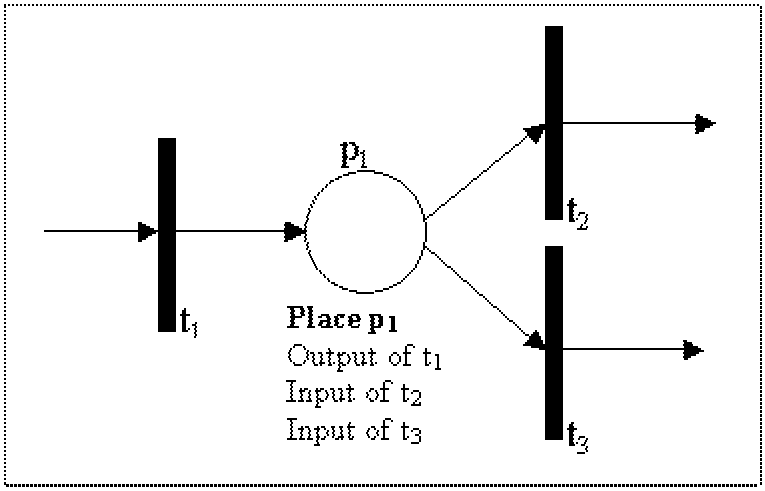



Figure 3-14. Place relationship to transitions.

The control is local to the triggered transition and the place in the transitions input function that was also in the output function of the previous event. The architecture is illustrated in Figure 3-15 with an example. The subnet in the circle, Group 1, grouping all the components with a relationship to transition $t_1$ are marked with the step numbers outlined below.

The following steps outline the sequence of local level control:

1. A token arrives to a place of the transition's input function.
2. The place sends a signal to the transition indicating a token has arrived.
3. The transition checks all the places in its input function.

   *If* at least one token in each place

   *Then* enable transition
4. *If* transition is enabled

   *Then* fire transition

   a. One token departs from each place in the input function

   b. One token arrives to each place in the output function

The token arrival to $p_2$, the place in the output function of transition $t_1$ in Group 1 is now the input place in Group 2 and Group 3 that is step 1 outlined above in the next local level of control. It is the change in state from an event occurrence in the Petri net that starts the chain of the local control in the following transition group(s). In this example the next local control is given to two transition groups, transition $t_2$ and transition $t_3$ concurrently.

If all the places in the input function of a transition do not contain at least one token during step 3, the transition waits. If only a single place in the input function of a transition does not contain a token, it is the arrival of a token to that place that starts the local control for the event occurrence because now the transition can be enabled.

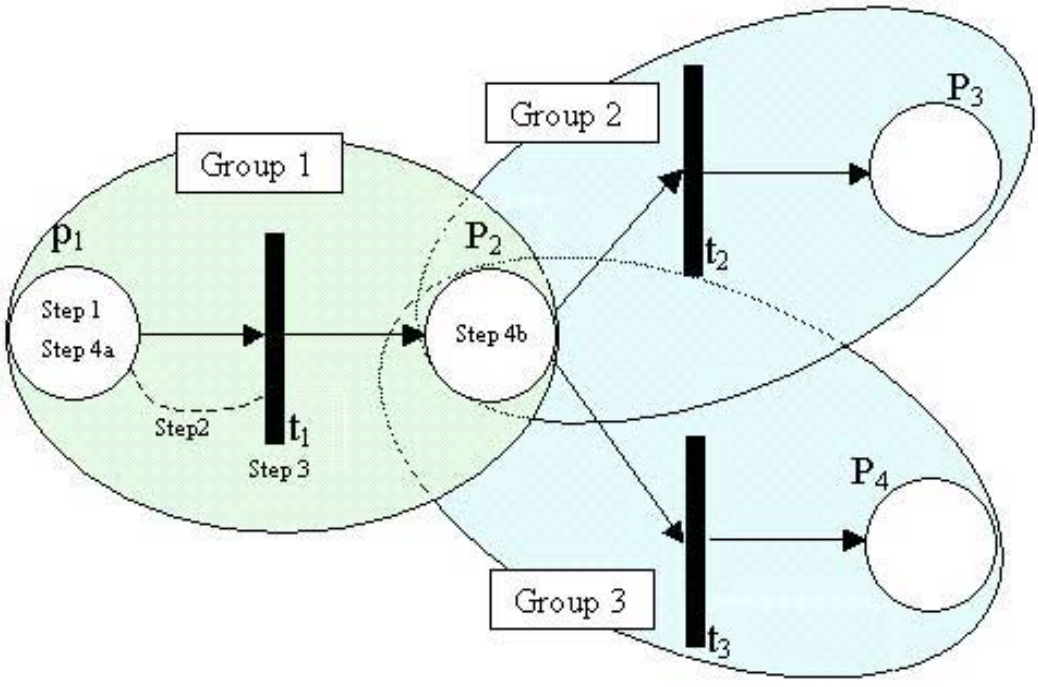



Figure 3-15. Architecture for asynchronous approach.

**Synchronous approach**

For the synchronous approach, the control is global and comes from the clock. The control from the clock is at the Petri net level of the model. This clock control is then transferred down to the component level, i.e., the transitions. The clock is connected to the Petri net and to all transitions of the Petri net as the input. This is illustrated in Figure 3-16. This approach uses the control at the model level for global synchronization between events within the model. This type of control is simpler than the local control described above for the asynchronous approach from the point of view of the local components of each transition. An arriving token to a place does not have to transfer the token arrival event to the transition(s) that it is in their input function. The global control initiates the event of checking all places in the input function of all transitions, thus the token arrival information is not necessary.

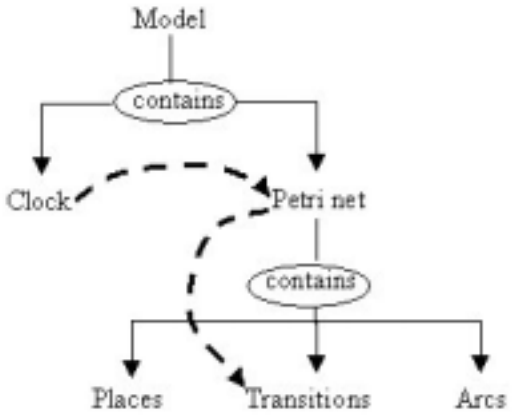



Figure 3-16. Clock input to model.

The architecture is illustrated in Figure 3-17 with the same static structure of the asynchronous approach above.

The following steps outline the sequence of global level control:

1. Global clock input to Petri net and clock input sent to transitions
2. If preset clock interval = time elapsed

   The transition checks all the places in its input function.

   *If* at least one token in each place

   *Then* enable transition
3. *If* transition is enabled

   *Then* fire transition

   a. One token departs from each place in the input function

   b. One token arrives to each place in the output function

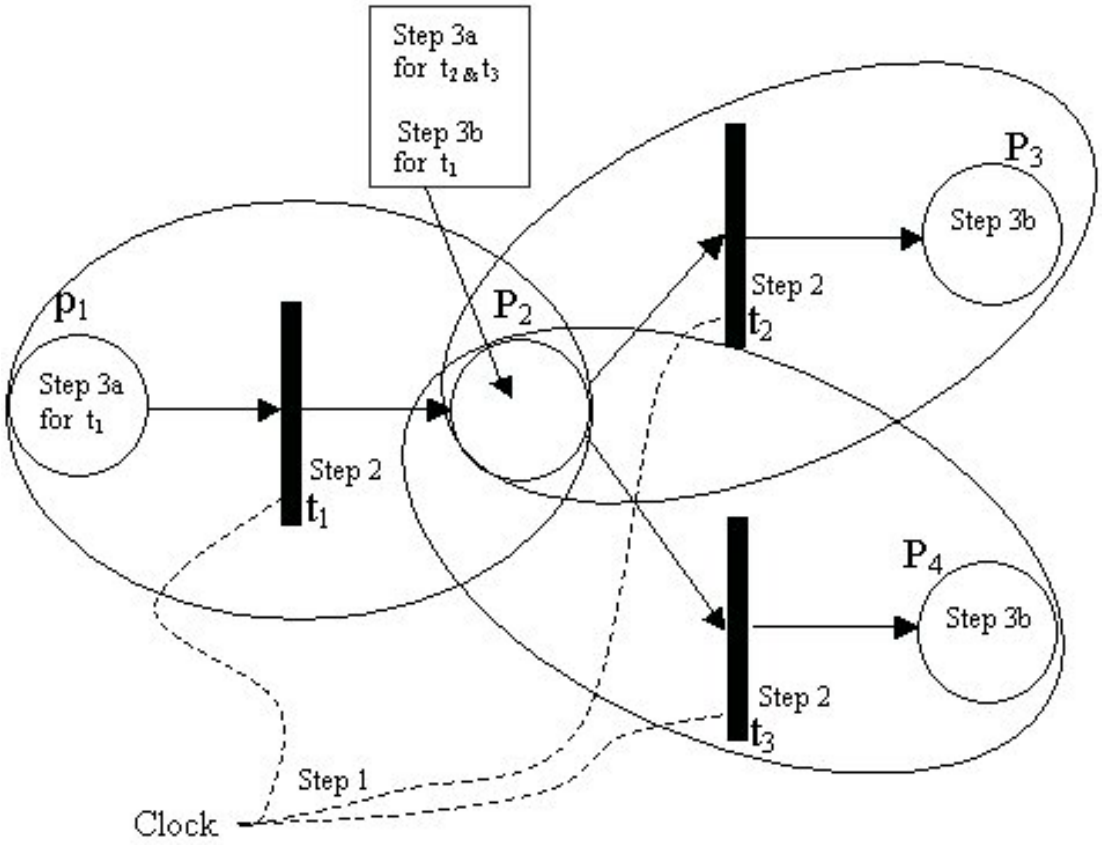



Figure 3-17. Architecture for synchronous approach.

**Summary**

Petri nets are a powerful tool for modeling and simulation. This chapter provided the fundamentals and knowledge of Petri nets to understand the tool and how it can be applied to model systems. The study of a system with the application of a Petri net can result in knowledge and understanding of both the static structure and the dynamic behavior of the system.

A Petri net is an event driven system in terms of what is the cause of change of state with either the asynchronous or synchronous approach. In both approaches, a

change of state of the system will only result if an event has occurred. The time element in the synchronous approach controls *when* the event occurs and not *if* the event occurs in the outcome of the execution.

The synchronous approach has numerous advantages over the asynchronous approach. It has the ability to simulate an asynchronous system within a synchronous architecture, it has time flexibility, the initial marking of the Petri net does not require special functionality, and it provides for the ability of integration into a multimodel.

# CHAPTER 4
# rube™

This chapter covers this work's contribution to the rube™ research project. It starts out with general information about the rube™ research project as well as the rube™ methodology and the concept of the use of metaphor within rube™. The rube™ development for the Petri net, which includes the dynamic model template design for Petri net, is covered next. The model implementation of a system utilizing the Petri net dynamic model template is then discussed. A description of the dining philosophers (the system modeled) is provided.

## The rube™ Connection to Modeling and Simulation

The research project name rube™ came from Rube Goldberg, the cartoonist of the 1940's, who drew fantastically complicated machines, which performed very simple tasks. The rube™ connection to modeling and simulation in this research suggests the design of models that will be remembered, not only the physical properties but also the behavioral properties and the relationships within the model. rube™ is a trademark of the University of Florida, and Rube Goldberg™ is the trademark of Rube Goldberg, Inc.

There is a Rube Goldberg™ competition where the most important goal is to challenge students to step back from reality in order to gain a new perspective on "how things work." This can be related to the rube™ research effort in that models are designed outside of the box of conventional 2D modeling and into a new 3D world using

metaphors instead of the conventional model symbols. The result is a new perspective on how the model works.

A Rube Goldberg™ machine incorporates the everyday machines or objects that people are used to seeing and connects them in ways that are ingenious. The likeness in the rube™ research effort is the use of metaphors, both physical and behavioral, for objects of the model. The other item of interest within a Rube Goldberg™ machine or cartoon is the memory retention encompassing all the objects and their relationship. The similarity to the rube™ research is when you study a rube™ model; you will understand and retain the model. Included in this understanding and memory retention is the static structure, the dynamic behaviors, and all the relationships within the model. This memory retention involves the use of pegging in mnemonics where we tend to easily remember and work with things that have real world significance to us, and we tend to forget nonreal world phenomena [20]. This is where metaphors fit in to rube™ methodology and aid in the memory retention goal of the model.

## rube™ Methodology

The Introduction chapter, Chapter 1, outlined the motivation and justification for the direction of this research. The rube™ methodology allows us to achieve the same results with the use of 3D and aesthetics as well as other goals of the research in the field of modeling and simulation.

The rube™ methodology is a model design technique used to create 3D web based virtual worlds that incorporate aesthetics in the model design and the visual representation of the model execution. It is this visual representation of the behaviors of the model that make the model come to life within its aesthetic static structure. The

result is a model that is easier to understand with high memory retention for the model and the relationships within the model. The key concept to accomplish these objectives is the use of metaphors in the design of the model.

**Metaphors**

The definition of a metaphor is to make an identification or fusion of two objects, i.e., to make one new entity partaking of the characteristics of both [21]. The task of the metaphor is to bring together the two different domains. Modeling is mapping target object and source object and the use of metaphors in the mapping creates a representation of the two objects from two different domains. Metaphors are used to make the connection between the target object and the source object with the underlying dynamic model type formalism.

The ultimate goal in the use of metaphors is to bridge the gap in understanding between the system being modeled and the model itself. This is also applied to the model designer and the model user. The model user may have little or no understanding of the model formalism used to create the model but needs an understanding of a system. The model designer on the other hand may have limited understanding of the system to model and needs something to bring the system into the model formalism. The two, the system and the dynamic model formalism, must connect through a mechanism. The mechanism we will use is metaphor by creating a “language” understood by both.

The use of metaphor in the model design actually is the answer to model design from purely an economic point of view. This is made clear if you consider the following scenario. A model is designed and built for a user, the consumer, to give an understanding of his real system. The model is designed using traditional symbols for the underlying dynamic model type. The model is tested and verified by the model designer

who completely understands the symbolism. The model is presented to the user and because the two domains were not bridged together to a common ground, the model is useless in the eye of the intended user. The economics of time and effort spent to attain the purpose of the model is obvious. The use of metaphors gives a user the understanding of a system and the underlying model formalism because the two domains have been connected to a common language. The economic choice should always be to build the useful product, which is a model that is understood by the user and the user can relate to the model product.

A metaphor is a tool that can be used to convey complex information about an object in a way that on the surface appears simple. This is a powerful concept when applied to modeling within the rube™ paradigm. The metaphor is used to give meaning that is inherent to the user without the need of creating a model that is more complex than it needs to be, addressing the concept of minimalism discussed in the Introduction chapter, Chapter 1.

The rube™ methodology utilizes static and dynamic metaphors in the creation of a model. The metaphors make the connection between the target system and the formal model type used to create the model. The metaphor concept utilized for the dynamic behavioral model is potentially the most powerful tool to create the connection between the target object and source object. Metaphors used for the behavioral aspects of the model are visualized through changes in the 3D objects, animation of objects, and with the use of sound. The purpose of the model design is to build in the mechanism to achieve an understanding of the dynamic behavior of the model. It is much harder to represent the behavior of the model than the static structure, i.e., the geometry. The

behavior of an object is abstract and harder to explain in either of the domains separately. The metaphor used to connect the real system behavior and the model behavior is the most important communication tool of the model. If the behavioral metaphors are effectively implemented with visual representations, then there is less analytical processing that needs to take place to understand the model. The uses of visual metaphors for behavioral relationships provide automatic information processing because of the graphical information processing discussed in Chapter 1. Maximizing the automatic information processing with the implementation of effective behavioral metaphors aids in the reasoning process of the model.

**Metaphors and the Petri Net Mapping**

rube™ methodology is used to turn the formal mathematical model of Petri nets into a computer program that provides not only the graphical static physical structure in 3D but also a visualization of the behaviors characterized by a Petri net during the execution of the model. This is successfully accomplished through the use of metaphors in the mapping to the Petri net model.

From the high-level, i.e., the model level, the system is mapped onto a Petri net model. This is illustrated in Figure 4-1. From the component level, the components of the system are mapped onto components of a Petri net model. First the static structure of the Petri net is represented with 3D object metaphors. The metaphors are then used to replace the traditional 2D Petri net graph symbols for the purpose of mapping the real system to the 3D model. This is illustrated in Figure 4-2. The metaphor mapping to the 3D objects will be covered in more detail in the model implementation later in this chapter.

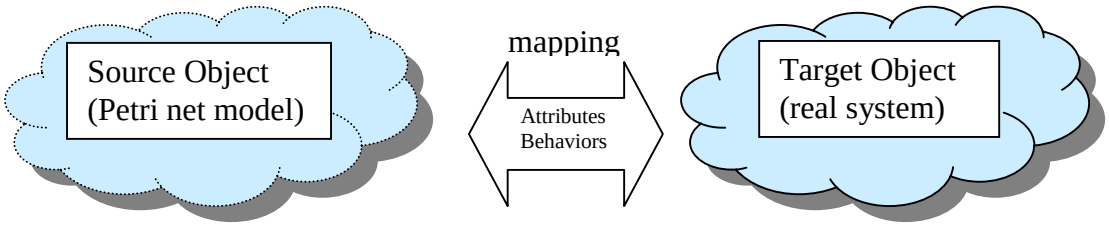



Figure 4-1. Petri net model and real system relationship.

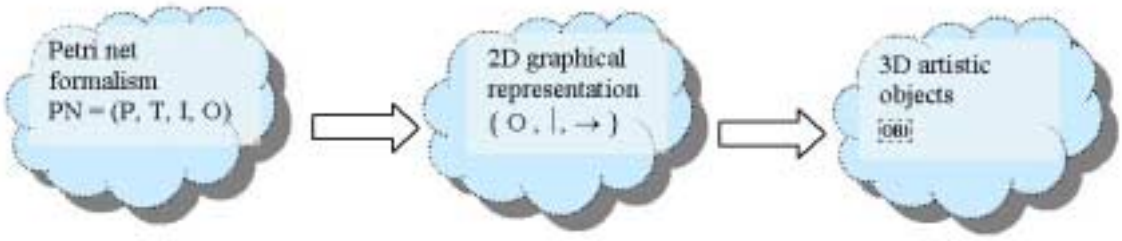



Figure 4-2. Petri net model mapping.

The dynamic behavior of the Petri net is the key component in making the model easy to understand in an effective and efficient way by observing what is actually taking place. This allows an understanding of the relationships and an opportunity to analyze the behaviors. The dynamics of the real system are mapped to the dynamic behaviors of the Petri net through metaphors. It is the visual and auditory metaphors of the dynamic behavior that allow the model to be studied while it is executing verses going through pages of textual output of token state information, or, multiple diagrams representing each new state of the model after each transition firing. This will be discussed in more detail in the model implementation section of this chapter.

**VRML**

VRML, Virtual Reality Modeling Language, is the implementation language used for the rube™ research project. It provides the capabilities to achieve 3D, multi-platform, web based models with dynamic visual and audio execution results. VRML allows the incorporation of aesthetic appeal and also the design approach for model proto reuse. VRML files are text files that specify the 3D shapes and the layout for the VRML browser to represent.

VRML uses a hierarchical structure of nodes. A node is given its structure in its PROTO. It is composed of fields, exposedFields, eventIn, and eventOut. The fields are the variables of the object and the events are the input and output used to make the connections to other objects of the world or model. This is accomplished with the ROUTE statements from node x eventOut TO node y eventIn.

The use of the scripting language, Java Script, makes it possible to implement the dynamic behavior of the model as it is executing. The Java Script is used in a Script node. The control is given to a function with the eventIn of a node with the ROUTE statements. The result of the execution of a script can be sent to other nodes with the use of eventOut and ROUTE statement or by set_exposedfieldname, which changes the value of an exposed field of another node explicitly.

EXTERNPROTOs allow multiple files of PROTOs to be used in a world. The Dynamic Model Template is a file of its own. Other objects are contained in separate files or grouped together in a logical manner. Separate files facilitate the objective of reuse. Reuse would be very difficult to achieve productively if everything for a model were contained in a single file.

The world VRML file includes the EXTERNPROTO definitions, the definition of the world, and the ROUTE statements. The choice of VRML for the rube™ project was a good choice for the startup phase because it allowed successfully use of the structure of the language to accomplish the objectives aside from some of the difficulties with the language specification and VRML browsers.

## Dynamic Model Template for Petri Net

The first objective of this thesis work was to create the Petri net dynamic model template. The Petri net dynamic model template was developed and then it was used to implement various simple models, which were then tested and verified.

A dynamic model proto template implements the specifications of a type of dynamic model that is used for all models built of that type. The Petri net dynamic proto template is a key component of the work in this thesis and is the reusable product for all Petri net model designs of interest. A template can be viewed as the blueprint followed to produce a product and also, as the design that product takes in actuality [22]. That is the specific purpose of each of the dynamic model templates in the rube™ project, i.e., to provide the pattern upon which models can be designed and built. It functions as both the foundation of the model type and the component for reuse for all models of that model type.

A model is a dynamic model because it models the dynamics of a system in the execution of the model. The dynamic behaviors of the model type that cause the model state to change during execution are incorporated in the model template along with the static physical structures.

To design the template, we have to define the Petri net model itself as well as all the components of a Petri net model and their relationships and the behaviors of the model to communicate during execution of the model. In terms of objects, the Petri net is an object and all of the components of the Petri net are objects. Both the static properties and dynamic properties of each object need to be defined. This is done in the PROTO of each object.

PN PROTO is the proto to define the model at the model level or global level. The Petri net model is composed of the components places, transitions, and traces. These Petri net components are all defined as a field type of the PN PROTO. Traces are classified by type, trace_places – the traces from places to transitions, and trace_transitions – the traces from transitions to places.

MFNode field type, an array of SFNode, is used for each of component types since the Petri net can contain more than one of any of the components.

```
PROTO PN [
        field     MFNode    places []
        field     MFNode    transitions []
        field     MFNode    trace_places []
        field     MFNode    trace_transitions []
        .
        .
        .
]
```

Each of the components has a PROTO to define its geometry and behavior. The PROTO's are PN_PLACE, PN_TRANSITION, PN_TRACE_PLACE, and PN_TRACE_TRANSITION.

The purpose of the template is for reuse for any Petri net model designed for a system thus the behavior of the model implemented in the template must reliably produce the correct model behavior for any static structure and initial marking. This behavior is the functionality of the Petri net by following the rules of interaction of the model. This verification was done by first checking each individual configuration of a transition as described in chapter 3 and then putting a system together using different configurations including cyclic conditions to verify correctness of the system as a whole.

The development of the Petri net template started with the implementation of the asynchronous approach and continued with the implementation of the synchronous approach. The asynchronous approach was an appropriate place to start in the development of the template to verify the correctness of the dynamic model template functions and ensure correctness of all the different possible combinations of the static structure and the effect of functionality. The asynchronous approach does not have the flexibility of the synchronous approach because time is not an explicit part of the model's behavioral design. Therefore, after successfully implementing simple models with the asynchronous template, the effort was switched to the development of the synchronous template. An overview of the asynchronous implementation will be given but the main focus is on the synchronous model template implementation.

**The Asynchronous Approach**

The logic implemented for the event driven system follows the steps outlined in Chapter 3. The arrival of a token to a place sends an input through a ROUTE to the

transition(s) where the place is in the transition's input function. The PROTO of the transition contains the input places and the output places so all information needed by the transition is defined for each transition. The transition then checks the conditions of all places in its input function. If it is enabled, it can then fire and send an event to all the places in its input function for a token to depart and another event to all the places in its output function for a token to arrive.

**The Synchronous Approach**

The synchronous approach was implemented with the addition of time and the corresponding control structure. Input to the model is needed at the model level to start the simulation clock. A clock is incorporated in the world and the ROUTE statement goes to the PN eventIn set_clock. The function set_clock of the PN PROTO does the transition ordering and that order is routed to each of the transition's set_clock function. Each of the transitions then checks if it is enabled. If it is enabled and fires, the routing takes care of updating the model status.

Transition ordering must be imposed in the synchronous approach to achieve mutual exclusion while checking the enabling rule of a transition that may have a common place in its input function with another transition that could be accessing the same token status information. The option was added to randomly order the transitions. This was done for resource contention so a different order could be achieved to avoid starvation of a transition. If two transitions that have a common place in both of their input functions, it is necessary to have a control function to impose an ordering. This is condition coordination of a shared variable, i.e., the token(s) contained in the place. Both transitions cannot concurrently take the same token, only one of the transitions can have the event occurrence and take the token. If it is done in the same transition order every

time, the same transition will fire every time resulting in starvation for the other transition. If the random order is chosen, the elements of the array of transition are randomized and the new random order is the order imposed of transition checking its enabling rule and if enabled, change the value of the token in the input places affected before checking the next transition in the randomized order. Either ordering will work depending on the behavior desired for the model as long as there is an ordering imposed. This solves the condition coordination problem of the shared resource of two transitions from a general perspective.

## rube™ Modeling Steps

The rube™ modeling steps used to construct the model are given initially as an overview of direction [23]. The details of each step will be explained by the example model implementation to follow.

Step 1: Define the system of interest to model.

Step 2: Select the dynamic model type.

Step 3: Choose a style.

Step 4: Define the mapping.

Step 5: Create the model.

The rube™ Style Guide [24] aids the modeler in defining a model using VRML. Most importantly is to create a world with human understanding, aesthetics, memory retention and communication in mind. The objects in the VRML world, through the use of metaphors, are the virtual equivalents of real objects of the system. Metaphors are also used to make abstract objects such as behaviors become realizable. Incorporation of sound in the world is important because it adds to the understanding of the model.

Using the rube™ modeling steps and the rube™ style guide, an example rube™ model implementation will be explained. As you will see, the use of the metaphor mapping and the design layout will result in an aesthetically pleasing model.

## rube™ Model Implementation

The real system is implemented as a three-dimensional Petri net model using the designed Petri net dynamic model template previously discussed. The rube™ modeling steps outline the process to implement a model achieving the objectives of rube™.

### Step 1: Define the System of Interest to Model

The classical dining philosophers problem is the system of interest to model. The dining philosophers problem illustrates concurrency and resource contention so it is a very interesting problem to study.

### The dining philosophers problem

The dining philosophers problem is one of the best known classical interprocess communication problems studied in operating systems. Dijkstra proposed the dining philosophers problem in 1965 as a synchronization problem [25]. Since that time there have been many solutions developed to solve the synchronization problem. The Petri net solution is one such solution. Although this thesis is not about how to solve the synchronization of the dining philosophers problem, it will be used to show how a system can be modeled using the Petri net formalism. The importance of the implementation is in the presentation of the model and the most effective mode for the communication of the information in the static structure and dynamic behavior of the model. The mapping between the problem and the model using metaphor is the important part of the model design and part of the focus of this research.

**Problem description**

Five philosophers are sitting around a table. Each philosopher has a plate in front of him. There are only five forks on the table, one in between each pair of philosophers. The philosopher must obtain both the fork on his left and the fork on his right to be able to eat. With this constraint, two neighboring philosophers will not be allowed to eat at the same time because they share one of the forks they both need. Because two neighboring philosophers cannot eat at the same time, only two of the five philosophers can eat concurrently. The philosopher is resting if he is not eating.

In terms of objects, the object of the group of dining philosophers is the system itself to model. The components of the system are the five philosopher objects and the five fork objects. The dynamic behavior of the system modeled is the synchronization of forks and the concurrent eating of philosophers.

**Step 2: Select the Dynamic Model Type**

The dynamic model type selected is Petri net. The Petri net dynamic model template that was implemented and discussed above using the synchronous approach will be used for the implementation.

The Petri net dynamic model type is part of the work of this thesis, so in this case the selection is already done. The reasons why the Petri net is a good choice to model the system of interest will be given.

Concurrency can be achieved inherently in a Petri net because independent events can occur in parallel. This means that the conditions of philosophers can change in parallel and two of the philosophers can eat simultaneously with the proper implementation of the fork constraints.

Each fork is a resource that is shared. There are five philosophers and five forks and the requirement of obtaining two forks to eat so all the forks must be shared and all the philosophers cannot eat at the same time. Each fork is shared with the neighboring philosopher on both sides. This is modeled with the input place to two of the philosopher's transitions that must be satisfied for the event occurrence of eating. Figure 4-3. illustrates this concept.

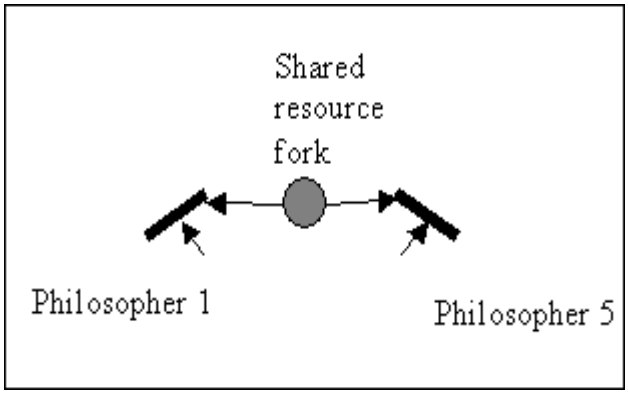



Figure 4-3. Resource sharing between two philosophers.

The model implementation achieves deadlock free by the physical structure of the Petri net and the Petri net model rules. The requirement that a philosopher must have the condition of both forks being available before he can pick up either fork enforces that he cannot pick up one fork and wait for the other to be free, resulting in a deadlock. If the condition of both forks being available was not required and every philosopher picked up his left fork and waited for the right fork, a deadlock would occur. Forcing the resource to be free if it cannot be used immediately eliminates the possibility of a deadlock with every one cyclically waiting on the other resource. The model implementation achieves starvation free by the random ordering of the transitions so every philosopher has an

equal opportunity to eat. It is the random ordering that provides the fairness of providing every philosopher the equal opportunity to eat.

In this Petri net implementation, both shared forks must be available for use and it must also be the philosopher's turn to check if both forks are available to actually obtain both forks and eat. The dining philosophers Petri net model is representative of a condition-event system because there is at most one token in a place at any time.

**Step 3: Choose a Style**

Minimalism using geometric 3D objects in 3D space. The style also incorporates the utilization of meaningful colors and sound.

**Step 4: Define the Mapping**

The mapping of the dining philosophers problem to the Petri net formalism is first defined. The dining philosopher problem as the system at the high-level is mapped to a Petri net model at the model level. The components of the dining philosophers problem are broken down to map to the components of the Petri net; places, transitions, and traces.

**Formalism**

The dining philosophers is defined:

$PNET = (P, T, I, O)$

$P = \{ p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14} \}$

$T = \{ t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9 \}$

$I : T \rightarrow P^{\infty}$

$O : T \rightarrow P^{\infty}$

$I(t_0) = \{ p_0, p_1, p_2 \}$ $\quad$ $O(t_0) = \{ p_{10} \}$

$I(t_1) = \{ p_2, p_3, p_4 \}$ $\quad$ $O(t_1) = \{ p_{11} \}$

$I(t_2) = \{ p_4, p_5, p_6 \}$ $\quad$ $O(t_2) = \{ p_{12} \}$

$I(t_3) = \{ p_6, p_7, p_8 \}$ $O(t_3) = \{ p_{13} \}$

$I(t_4) = \{ p_8, p_9, p_{10} \}$ $O(t_4) = \{ p_{14} \}$

$I(t_5) = \{ p_{10} \}$ $O(t_5) = \{ p_0, p_1, p_2 \}$

$I(t_6) = \{ p_{11} \}$ $O(t_6) = \{ p_2, p_3, p_4 \}$

$I(t_7) = \{ p_{12} \}$ $O(t_7) = \{ p_4, p_5, p_6 \}$

$I(t_8) = \{ p_{13} \}$ $O(t_8) = \{ p_6, p_7, p_8 \}$

$I(t_9) = \{ p_{14} \}$ $O(t_9) = \{ p_8, p_9, p_{10} \}$

**2D**

The formalism is mapped to the 2D graphical representation. Refer to Figure 4-4 for the mapping to the traditional 2D graphical symbols. The 2D graphical representation is illustrated in Figure 4-5.

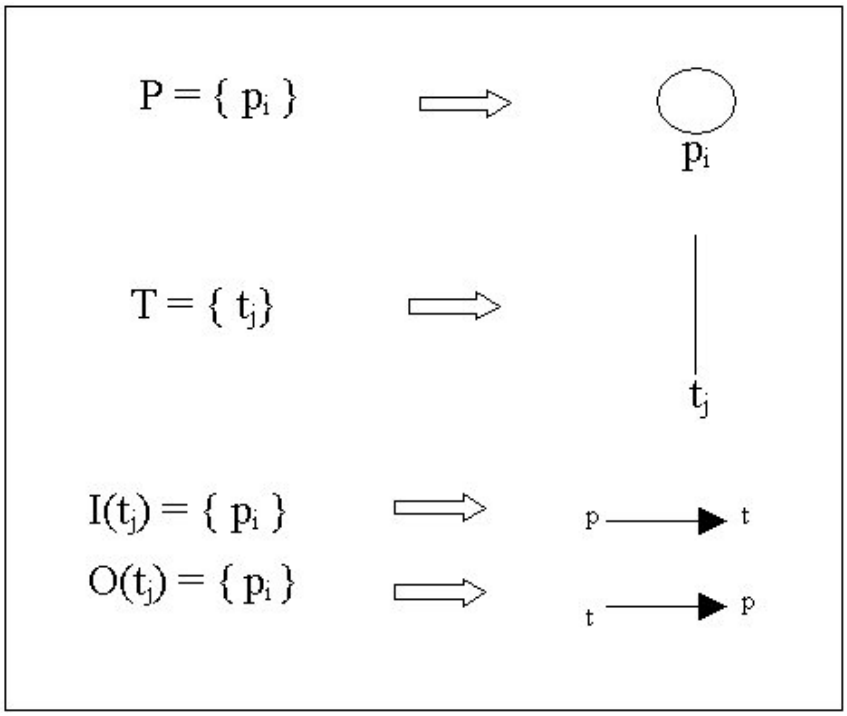



Figure 4-4. Formalism mapping to 2D graphical.

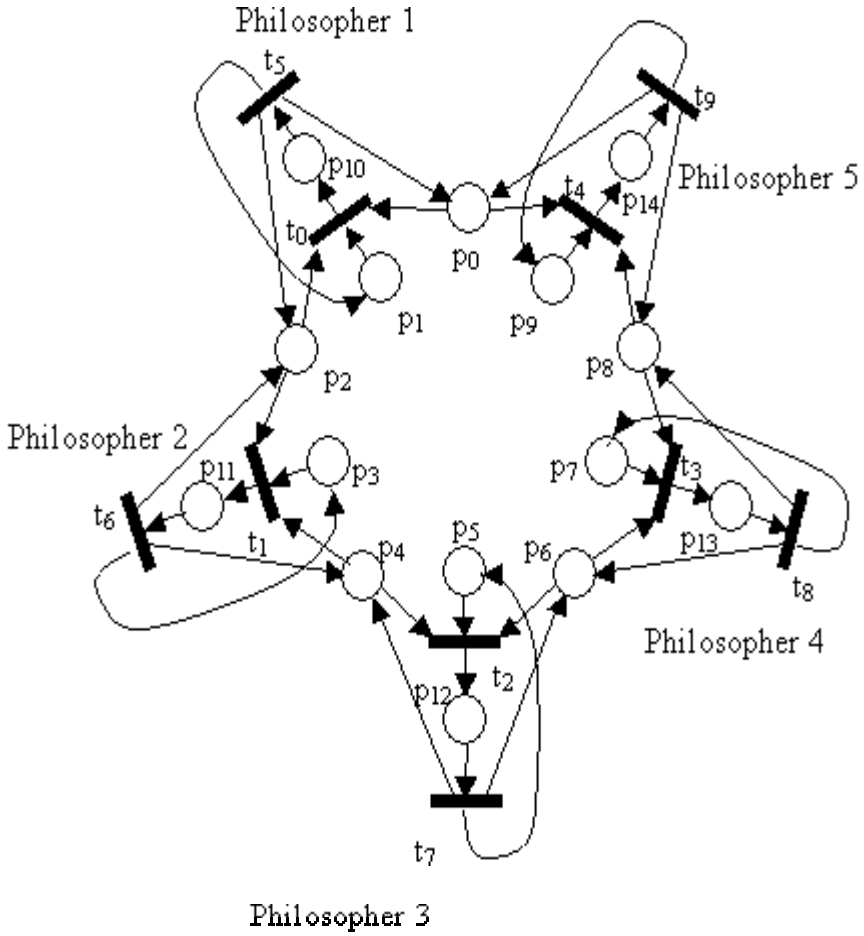



Figure 4-5. 2D Graphical representation of dining philosophers.

The initial marking, μ, of the dining philosopher model is with all forks on the tablecloth and none of the philosophers are eating, they are all resting.

$\mu(p_i) = 1$ for $i \in \{ 0, 1, 2, \ldots, 9 \}$

$\mu(p_i) = 0$ for $i \in \{ 10, 11, 12, 13, 14 \}$

The initial state is represented as ( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 ). The initial marking of the 2D graphical representation is illustrated in Figure 4-6.

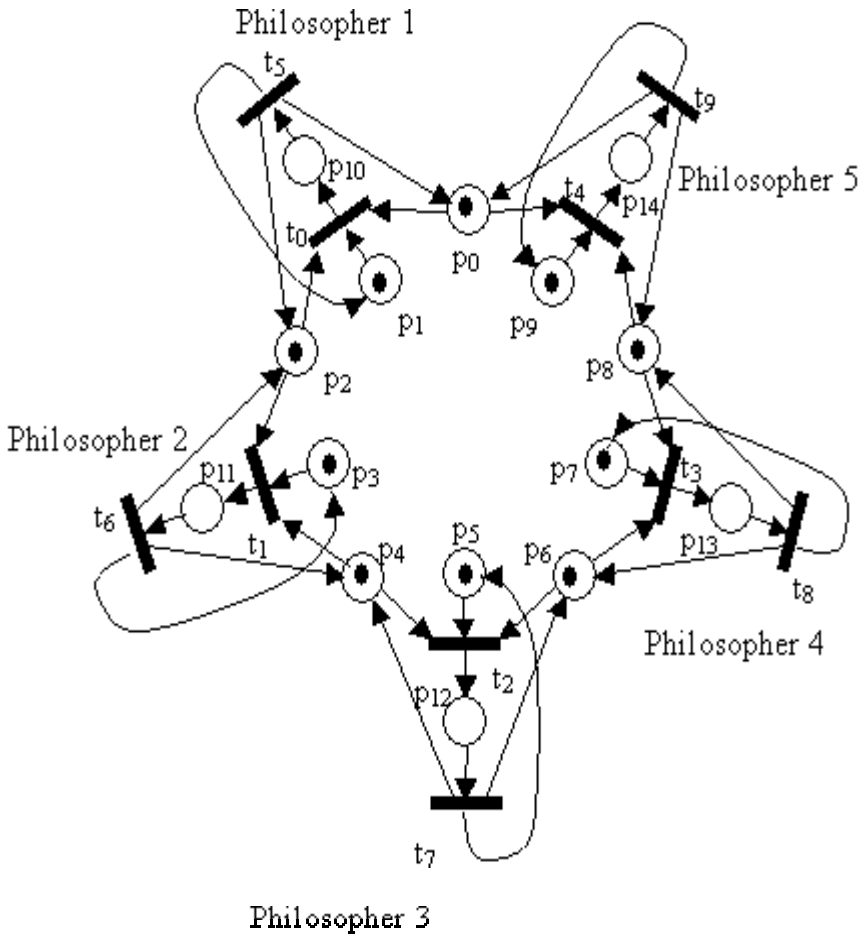



Figure 4-6. Initial marking of dining philosophers.

**Metaphors and mapping**

The 2D graphical representation is then mapped to a 3D representation using metaphors. There is a one-to-one mapping of the 2D components to the 3D objects in the static structure. The ideas behind the choice of the metaphors are given as they relate to the static structure and to the dynamic behaviors.

The static structure of the petri net is composed of the petri net components. The placement of these components in 3D space will define the static structure of the petri net model. The metaphor mapping of the components is illustrated in Figure 4-7.

Places are mapped to spheres. The color of the sphere indicates the token status of that place. The color choice will be explained in the behavioral mapping.

Transitions are mapped to red cubes. The cube carries the meaning of a black box of an operation on a high-level without showing the details of operation. The color red indicates an action to occur mapping to the transition event occurrence.

Arcs, the coupling components, are given the name trace in the dynamic model template for Petri nets. Traditional 2D representation of a trace is an arrow to indicate the direction of flow. Traces are mapped to ellipsoids to show the coupling of places to transitions and transitions to places in the 3D representation. Color is used as a metaphor in the 3D representation to map to the type of trace. The color of the ellipsoid indicates the type of trace representing the direction of flow and the color maps to the relationship of the components it connects. The metaphors used for the traces utilize the concept of *ready then fire* for direction of flow of the token and the functionality of the components. First the input places must be *ready*, i.e., enabled, in order for the transition to *fire,* i.e., transition fires the token(s) to the output place(s) of the transition. The metaphor of the color choice of the ellipsoid maps to the functionality associated with the components that define the direction of flow in the traces. The green color maps to ready and is used for traces with the direction of flow from a place, the place allowing a transition to be ready, to a transition. The place to transition relationship is ready for tokens to flow when the transition is enabled from the conditions in this place so the color green maps to the place to transition direction of flow. The color choice for the transition to place direction of flow is red which maps to a transition fire, i.e., the execution of an event.

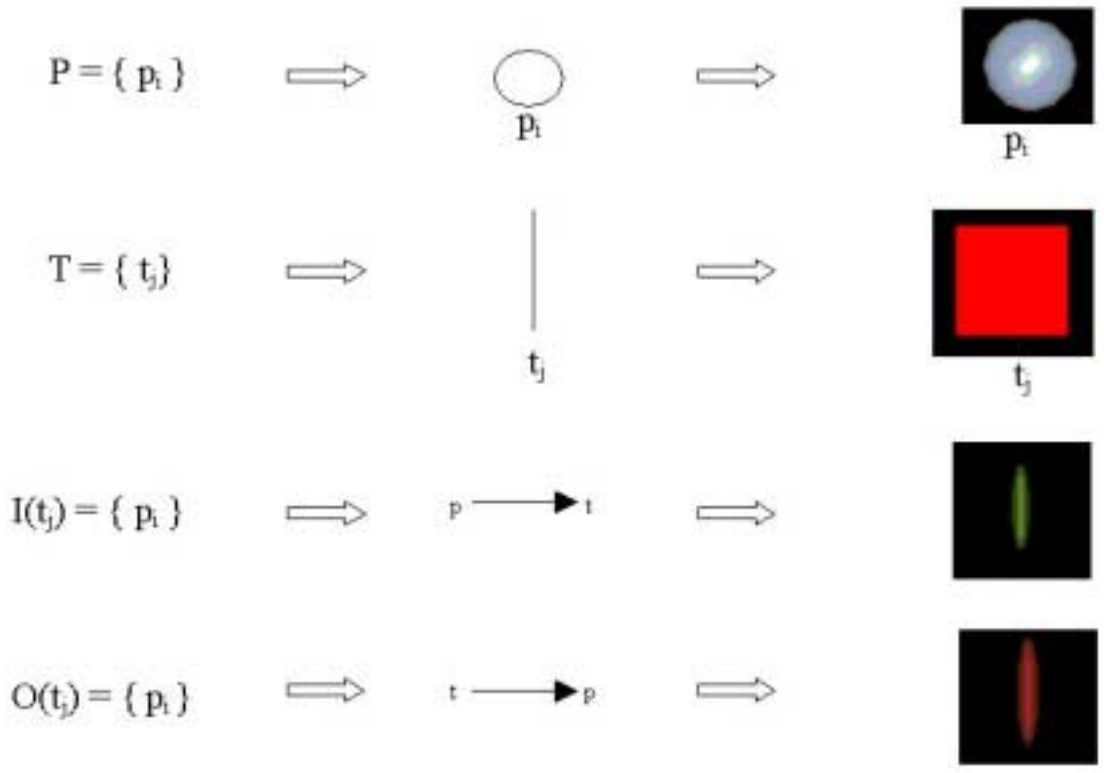



Figure 4-7. Static component metaphor mapping to 3D objects.

From the model view looking down, the symmetrical placement of the five philosophers sitting in a perfect circle maps to the logical and controlled setting of the dining philosophers and the control of synchronization. This logical and controlled mapping also defines the functionality of a Petri net dynamic model type, i.e., logical event occurrence controlled by conditions. From a side view, the placement of places and transitions belonging to a single philosopher form layering, mapping to the elevation of a person sitting at a table eating. The tablecloth where the forks reside when not in use are at the lowest level and the place representing the philosopher eating is at the highest level, i.e., mapping to where his head would be. This 3D space layout is very important in the efficient communication of the dynamic behavior. With the layering structure

metaphor, it is easy to automatically connect to the eating and resting. The placement of forks on the table also aids in the efficient visualization communication of the dynamic behavior. The placement of forks and the eating and resting in the model of Chapter 2 made the model extremely difficult to follow and analyze the dynamic behaviors.

The dynamic behavior of a Petri net is strictly the results of the component behaviors within the Petri net model itself. The component behaviors are the transitions that fire and the places in that transitions input and output function. The combined effect of all the components behaviors can then be collected and defined as the model or global behavior of the Petri net. This can be described in several ways and the implementation of the metaphors for the mapping in this model will be described in the model behavior section to follow. First the component behavior mapping will be described.

The component behavior within the model is mapped from each place marking in the formalism $\mu(p_i) = 0$ or1 for $i \in \{ 0, 1, 2, \ldots , 14 \}$. Each place has a marking and at the time that this changes from 0 to 1 or 1 to 0, the dynamic behavior of the model is the transition fire. The mapping in the 2D graphical representation is a single token in a place for a 1 or no token in a place for a 0. The mapping in the 3D model of this dynamic behavior is visualized through the metaphor of changing color of the place to correspond to the marking. A medium shade of gray spheres when the condition is false and the spheres change to white when the condition is true. This mapping of color change maps to a light bulb on and off for true and false. The color change utilizing the visual senses make it easier to see and understand how the conditions are changing as the Petri net executes.

The behavioral metaphors implemented at the component level of the model give insight into the Petri net model behaviors. A component has behaviors that result from a change in state as a result of the tokens moving from place(s) to place(s). The visual behaviors observed bring meaning to the part of the Petri net that has relationships to the component and can also be viewed at the model level for an understanding of the total system. It is the component level behaviors that give the understanding of how the use of a Petri net successfully models the dining philosophers problem with the given constraints. Because of concurrency in a Petri net, concurrent behaviors can be observed at the component level of the Petri net. In the dining philosophers model, the changing of the color of the place is a local behavior of the place component.

The state of Petri net model formalism at any time is the global information of the marking of all places in the Petri net or the model behavior. The state information, i.e., the global information is constructed from the places marking information, i.e., component information. The states of components of the system make up the global state. The state would be ( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 ) when none of the philosophers are eating. The information used is the last five numbers in this state representation since the information we are extracting out is just who is eating. A 1 in any of the last five locations of the state representation indicates the philosopher corresponding to that location is eating.

An *eat* status board is implemented to display the information of the current philosophers eating with a continuous visual update. Updating of the *eat* status board is a global behavior because it gives information about the model. It updates the status of the Petri net as a system from the global state information.

Audio used as a metaphor for behavior at the Petri net model level is also incorporated to enhance the visual execution of the model. The Petri net status of which philosophers are currently eating is vocalized with the of saying each of the eating philosopher's number one at a time during the time that the eating is taking place. Because of the audio in the design of the model, a blind person could observe what is happening and understand the behavior of the model without the visual effects. Even a person that was not blind would have a better and faster understanding of the behavior of the model because of the addition of the audio to both the visual behavioral changes of the status board and changing color of the place to indicate the change of condition, which translates to change of state of the philosopher. Involving more senses increases communication efficiency.

**Step 5: Create the Model**

By use of the metaphor mapping to each component and the design layout, an aesthetically pleasing Petri net model of the dining philosophers is created using VRML. During execution, the model achieves and illustrates concurrency, resource contention, mutual exclusion, fairness, deadlock free, and starvation free.

**Artistic mapping**

Artistic enhancements to the environment of the model add to the realism and immersion into the model. The background setting for the philosophers is traditionally sitting around a table to eat. However, for this implementation, an outdoors setting was chosen. The philosophers are on a picnic in a beautiful rolling meadow with a sky in the background. This background gives the feeling of an enjoyable experience when viewing and understanding the behaviors of the model itself. They are sitting on a tablecloth in a circle. In the middle is a candle, a yellow ball, which starts the simulation and maps to

lighting the candle to start eating which is the execution of the model. Background sound starts when the world in loaded. The sound node is set to loop TRUE so the audio file continuously loops. The background sound node definition is:

```
DEF Background Sound {
        source AudioClip {
           url    "28mix.wav"
           loop   TRUE
        }
        intensity       1
        priority1
        minBack         100
        minFront        100
        maxBack         100
        maxFront        100
}
```

The 3D artistic metaphor mapped representation is illustrated in Figure 4-8, top view and Figure 4-9, side view.

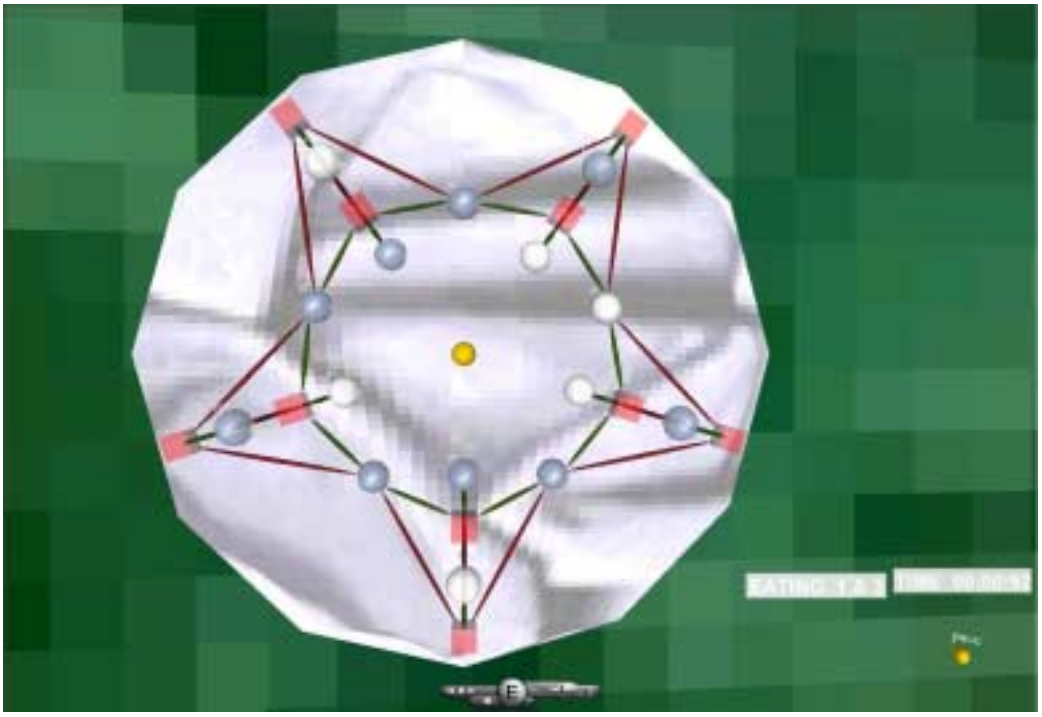


Figure 4-8. Top View in 3D.

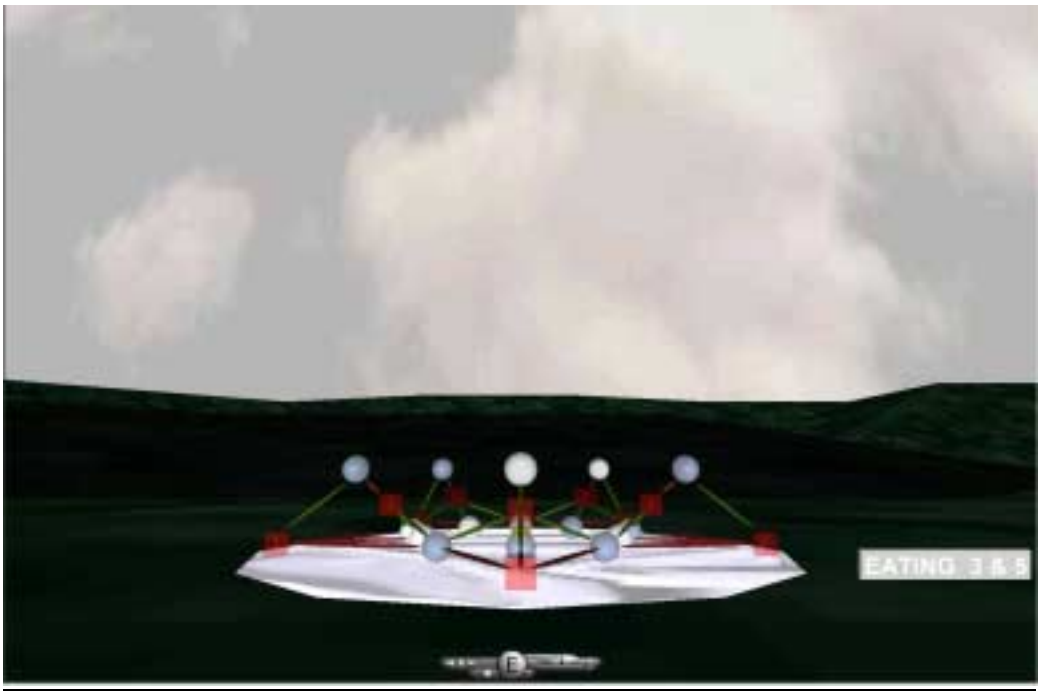


Figure 4-9. Side View in 3D.

The 3D artistic representation of the initial marking of the Petri net is shown in Figure 4-10. The five philosophers are resting and all the forks are in their places on the tablecloth. This is represented with all the places at the table level as white spheres indicating the marking of the place as "1" or the condition "true" for resting and forks in place on the table. The places above the table level representing the eating states of the philosophers are initially the color gray indicating the marking in those places as "0" or the condition "false" for not eating.

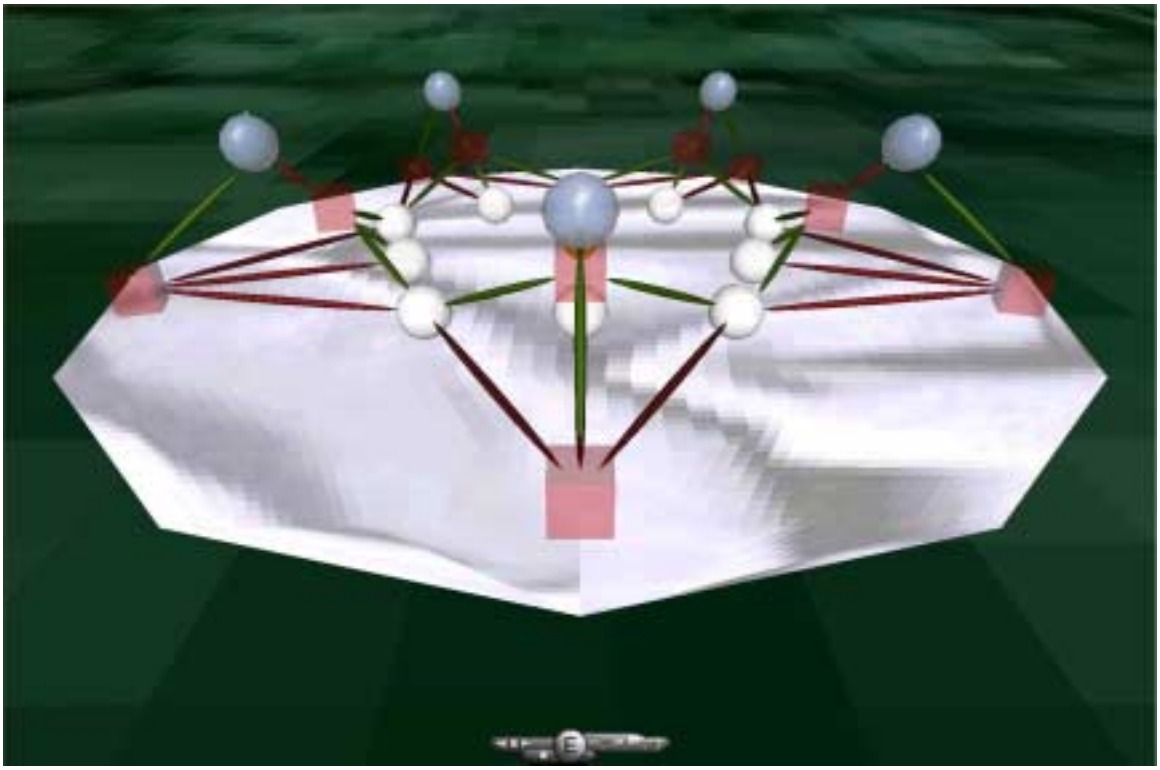


Figure 4-10. Initial marking in 3D artistic representation.

**Scene graph and routing**

The physical relationships of all objects in the Petri net model are defined in 3D space, thus creating the VRML scene tree. The routing structure is defined for the input

and output between objects. The scene tree of the top level is illustrated with a screen shot of the VrmlPad Scene Tree in Figure 4-11.

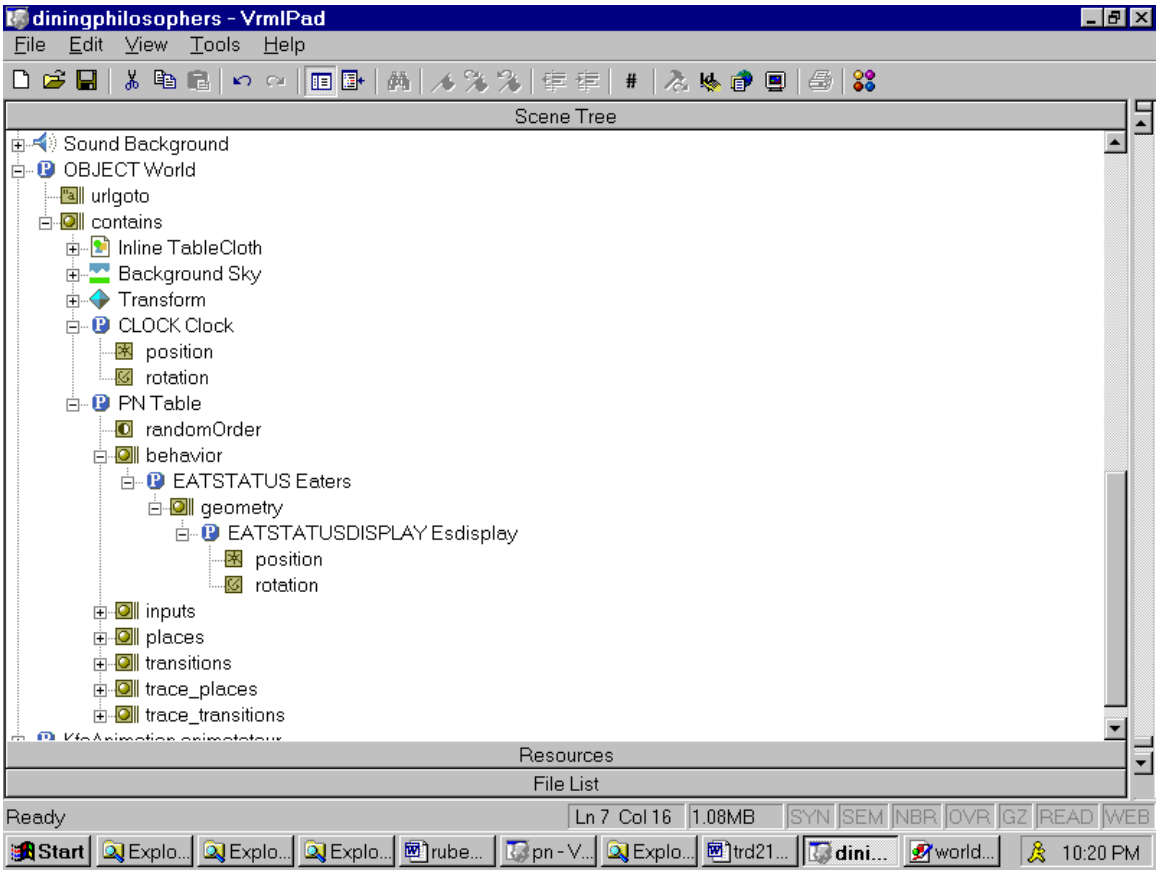



Figure 4-11. Model level scene tree.

The scene trees of the place and transition components of the Petri net are illustrated by expanding the nodes of the model level scene tree. The place node is in Figure 4-12 and the transition node is in Figure 4-13.

The VRML code examples to define the place and transition illustrated in the respective sceene trees are:

```
DEF P0 PN_PLACE {
      number_start_tokens  1
      place_id 0
      placeno 0
      tokencount 1
```

```
      geometry
            DEF P0Fork FORKPLACE {
                  position 0 18.54 0
                  text_position -2 2 0
                  name "p0"
                  contains [
                        DEF CI0 ColorInterpolator {
                              key [0.0 1.0]
                              keyValue [.3 .4 .6   1 1 1 ]
                        } #DEF CI0
                  ]
            }
      behavior
            DEF Board0 BOARD {
                  position 0 18.54 0
                  inside_state USE P0
                  text ""
                  behavior "javascript:
                  function set_clock(value,ts) {
                        node = inside_state.geometry[0];
                        if (inside_state.tokencount == 0) {
                           state = 0.0;
                        }
                        else {
                           state = 1.0;
                        }
                  }
                  "
            }
} #DEF P0



DEF T0 PN_TRANSITION {
      petrinet USE Table
      t_id 0
      placefrom [USE P0, USE P1, USE P2]
      placeto [USE P10]
      fireSpeed .0006
      speedFactor 1
      geometry DEF T0control CONTROL {
            text_position 0 0 0
            name "EAT"
            position  -11.76 16.18 5
            rotation 0 0 1 -.942
      }
 } #DEF T0
```

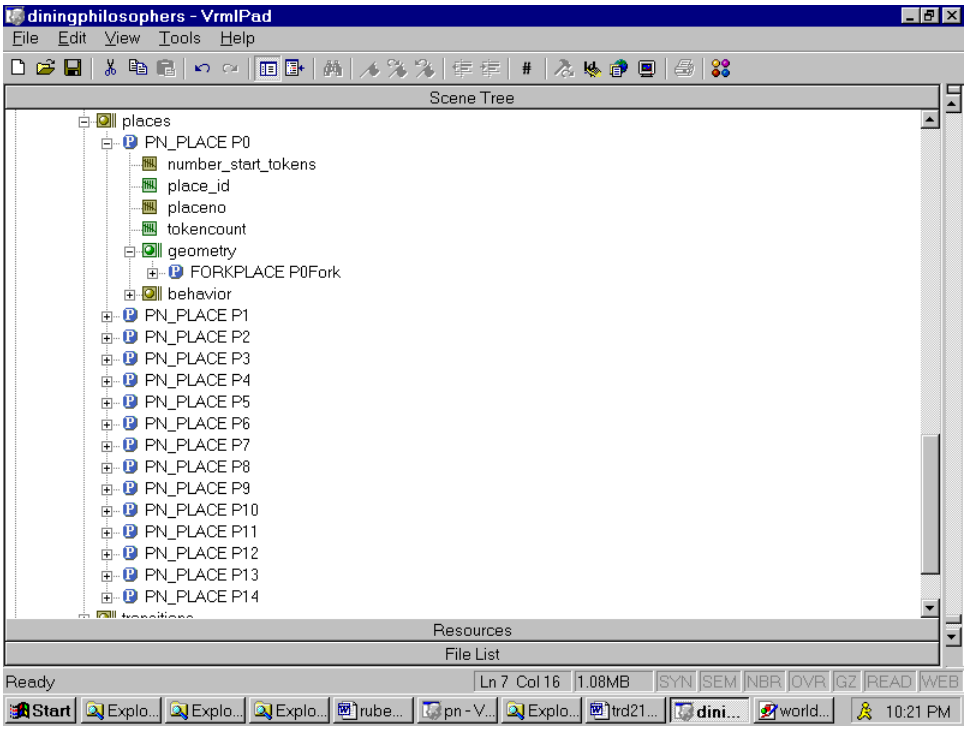



Figure 4-12. Place node scene tree example.

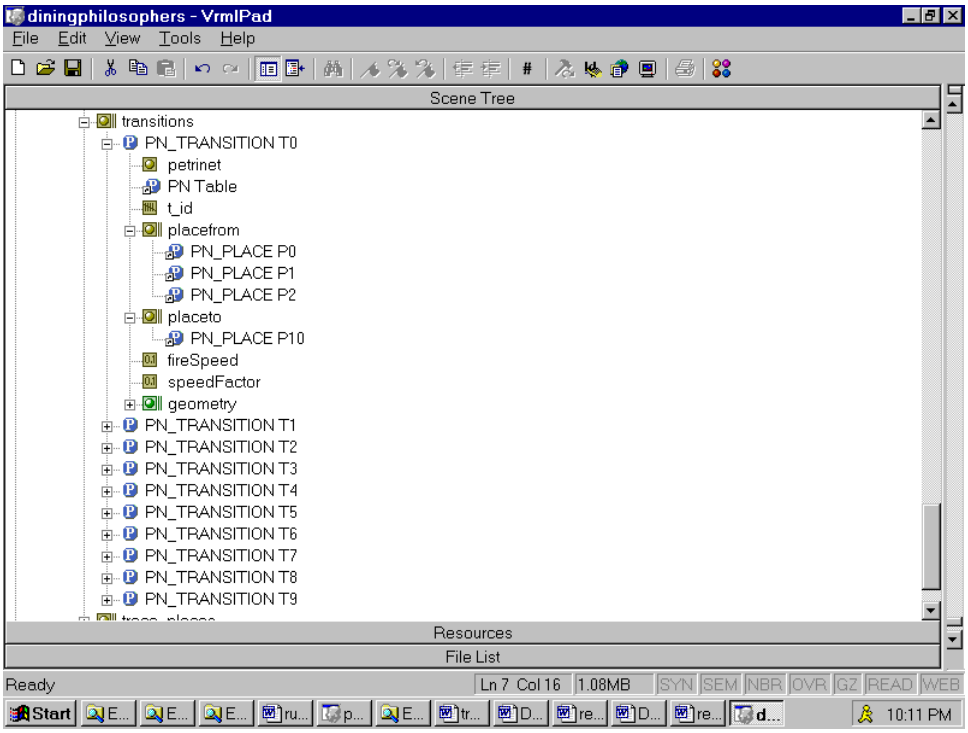



Figure 4-13. Transition node scene tree example.

Routing is shown for philosopher 1.

The touchsensor Inputstart starts the simulation clock, which sends the signal to the Petri net clock function to start the simulation.

ROUTE Inputstart.startsim TO Clock.start

ROUTE Clock.fraction_changed TO Table.set_clock

The following events initiate the global status update.

ROUTE P10.token_available TO Table.set_status

ROUTE T5.move_token TO Table.set_status

The following events control the dynamic behaviors.

ROUTE Eaters.set_sound1 TO Sound1.startTime

ROUTE Eaters.st_sound1 TO Sound1.stopTime

ROUTE Sound1.isActive TO Eaters.set_audcont

ROUTE Sound1.isActive TO P10.soundstatus

ROUTE Board0.state TO CI0.set_fraction

ROUTE CI0.value_changed TO P0Fork.set_diffuseColor

ROUTE Eaters.neweaters TO Esdisplay.update

## Petri Net Reachability Tree

The reachability tree is an infinite tree because the model can execute to infinity. The subtree illustrated in Figure 4-14 is the child of every leaf node. This same subtree of Figure 4-14 is again the child of every leaf node of the new tree and this process of representing the reachability tree continues infinitely as illustrated in Figure 4-15, “Reachability subtree” circles are used to denote the subtree of Figure 4-14.

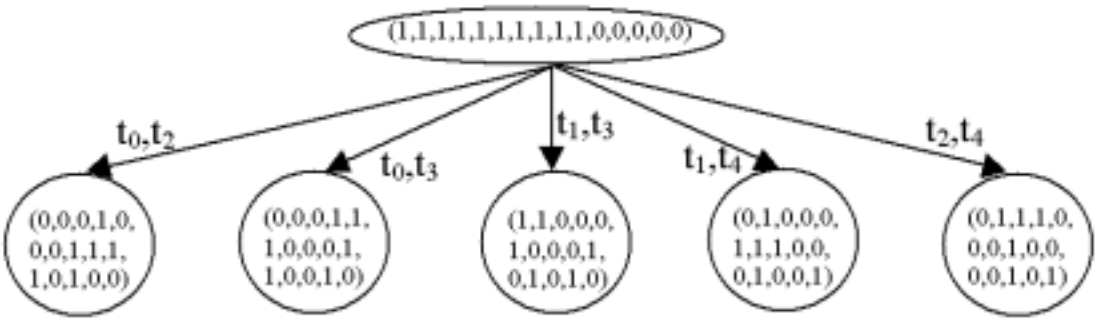



Figure 4-14. Reachability subtree.

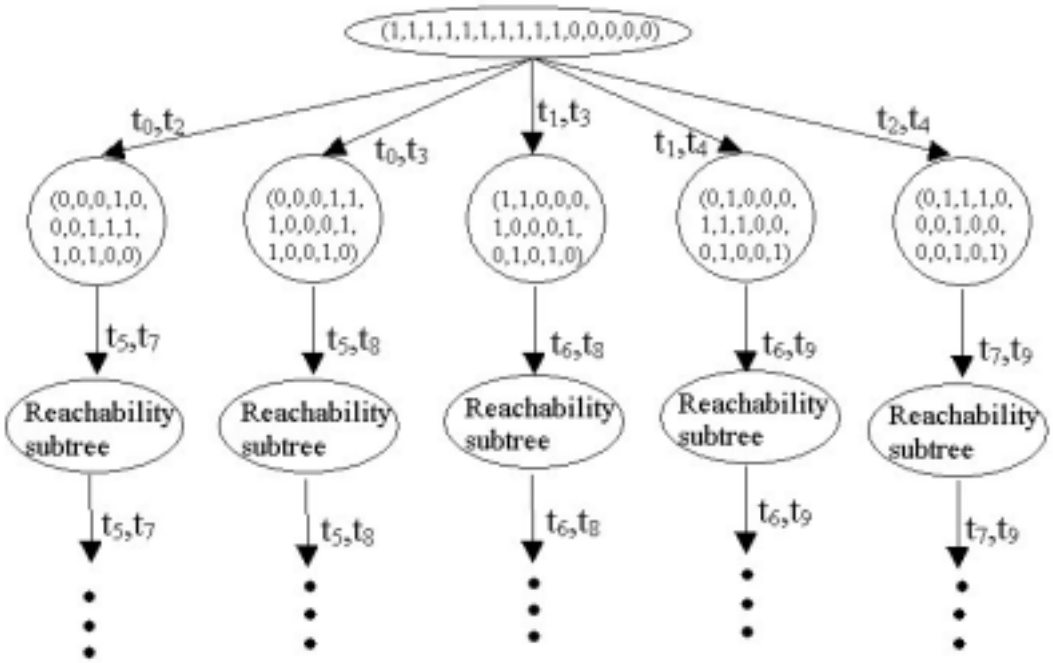



Figure 4-15. Reachability tree.

A simple way to view the path of the tree is to illustrate the tree as continuously looping back to the root node of the tree. Illustrating in this fashion only requires two levels of a tree. The root node represents the initial marking and that marking occurs

over and over again during the execution when all philosophers are resting. The second level of the tree contains a node for each of the possible combinations of the philosophers that are allowed to eat concurrently. This is illustrated as a graph in Figure 4-16.

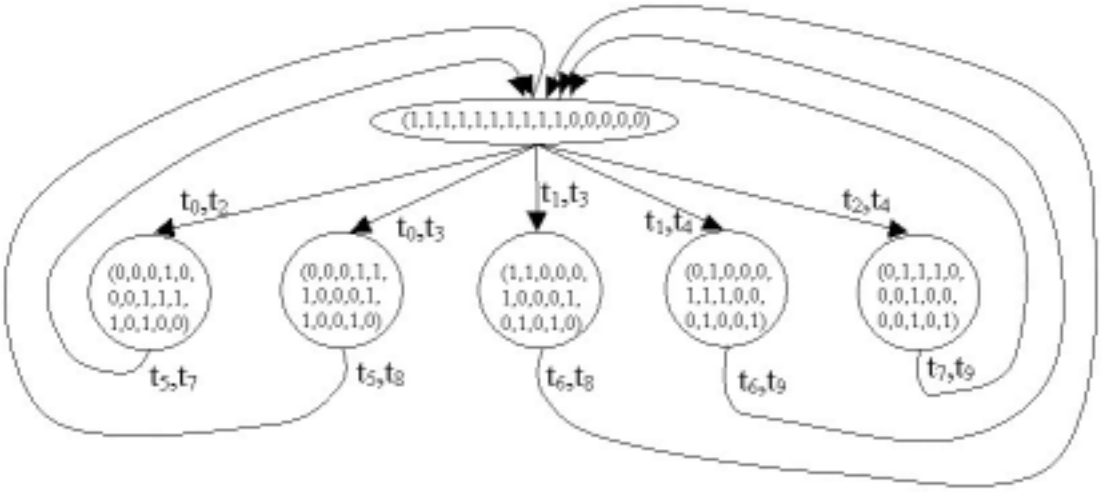



Figure 4-16. Reachability graph.

**Petri Nets Relationship to Other Dynamic Model Types**

Any Petri net that has a finite number of states can be represented as a finite state machine. A finite state machine that is nondeterministic can be represented as a Markov Model. The dining philosophers problem with the implementation in this thesis can be represented as a finite state machine and a Markov model.

The dining philosophers model has a finite number of states that the model can be in at any point in time even though the states can change an infinite number of times. The finite state machine representation of the implementation of the dining philosophers as a Petri net in this project is illustrated in Figure 4-17. The arcs denote the transitions fire to move to the state indicated.

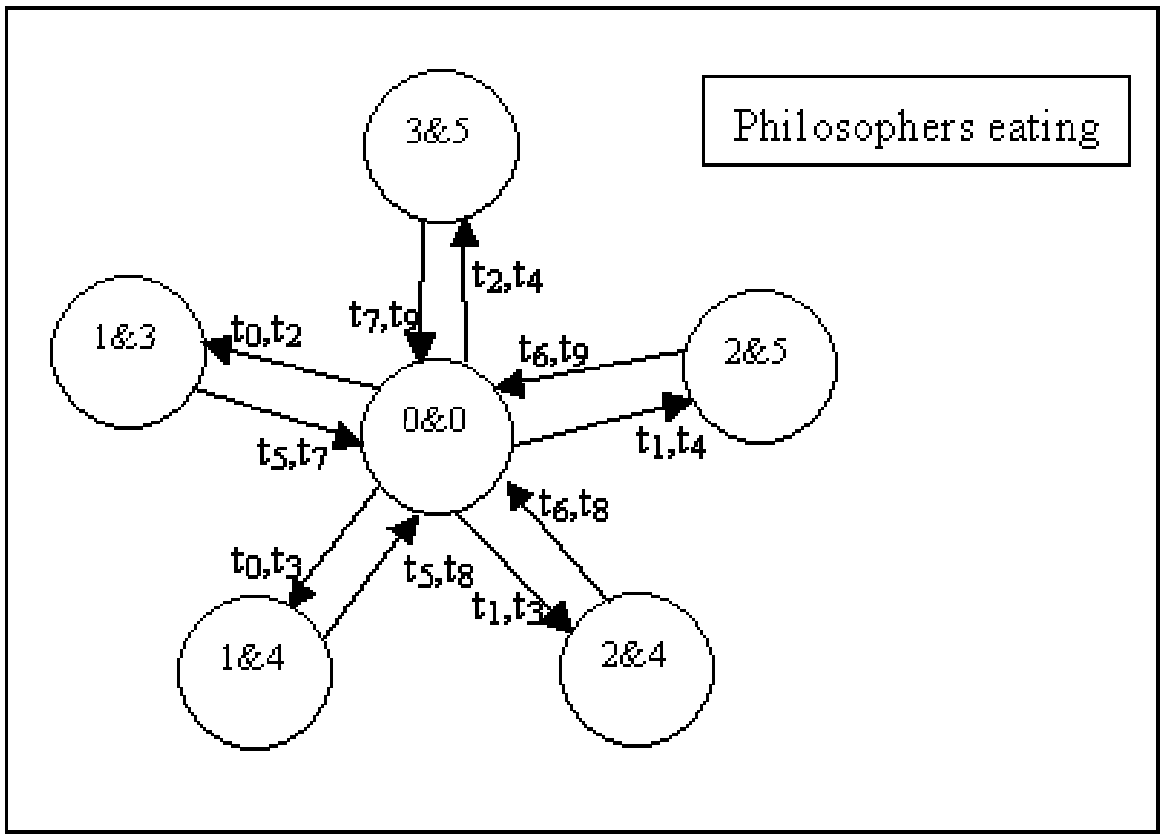



Figure 4-17. Finite State Machine for dining philosophers.

Since this implementation was done using a nondeterministic approach in determining the next state chosen, a Markov Model could be used as a representation. The difference between the finite state machine is the arc in the Markov model carries the label of the probability that the arc will be taken to transition to that state. The Markov Model representation of the implementation of the dining philosophers as a Petri net in this project is illustrated in Figure 4-18.

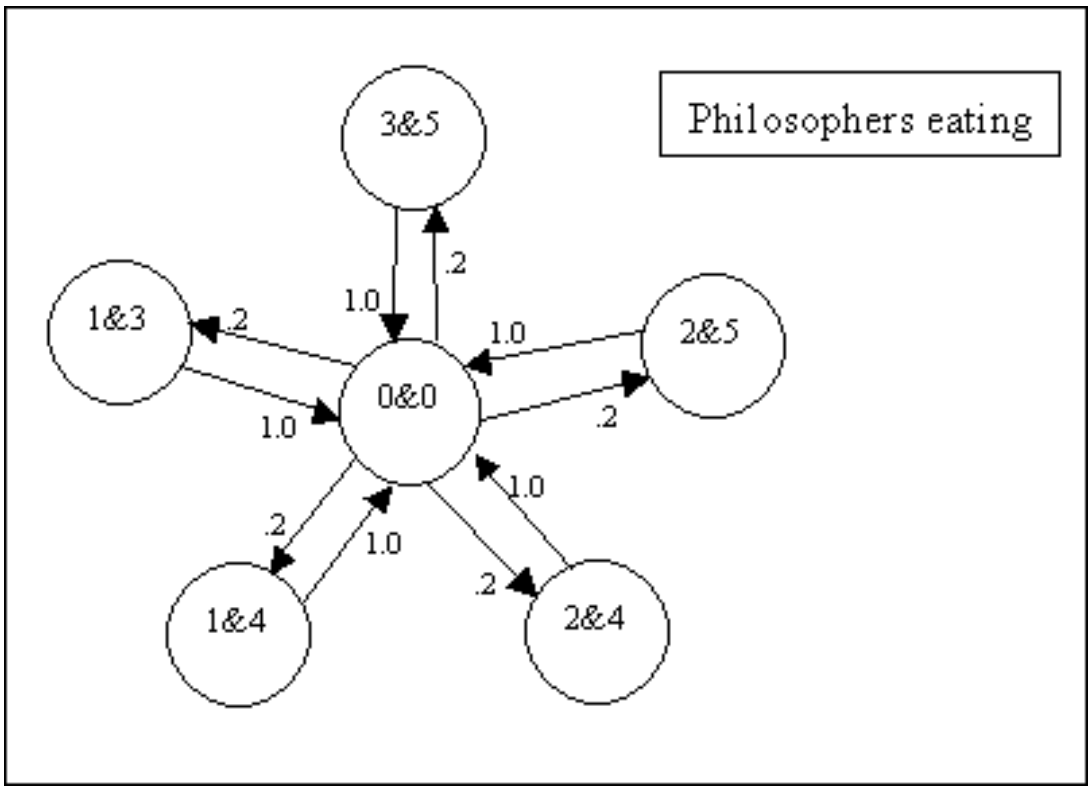



Figure 4-18. Markov Model for dining philosophers.

The Petri net model of the same system contains more information about both the system and the controls within the model, than the finite state machine model. Because of the level of information in the dining philosophers Petri net model, a clear understanding of the problem can be achieved. On the other hand because the details of synchronization and resource sharing have been abstracted away in the finite state machine model of the dining philosophers, an understanding of the synchronization problem cannot be seen or understood from the finite state machine model showing only the state change.

## Development Tools

An overview will be presented of the tools used in the development, implementation, debug, and presentation of the rube™ models. These include the environment to develop a model within, both textual and graphical, and the viewing of the model.

### Editing Tools

WordPad was used as the text editor during implementation. WordPad is a basic text editor that the VRML code can be implemented in but it provides no assistance in VRML coding specifically.

VrmlPad, a Parallel Graphics product, is a professional editor specifically for VRML programming. Unfortunately VrmlPad was not available while I was implementing the Petri net dynamic model template and the dining philosophers model. It incorporates valuable timesaving features. Smart AutoComplete of words saves typing after the start of a word is entered. Dynamic errors detection, syntax highlighting, automation and scripting, color coding variable names by variable type for easy identification, variable name type checking, automatic indentation and matching brackets are all invaluable aids for development. VrmlPad offers visual support for the scene tree, which hierarchically shows the world structure. It also provides a listing of the resources used in the world, as well as all the PROTOs with their file name, and the audio files and visual files used. It provides a preview of your work so you can easily go back and forth between the editor and the VRML browser.

### Geometry Creation Tools

Cosmo Worlds is a world development tool. It can be used for layout of objects in the world and for the creation of geometry objects, shape, size, color, and textures. It

was used it extensively for the experimentation of shades of color. It creates the VRML code from the selections made from the toolbars that are applied to the objects within the world.

**VRML Browsers**

The model created is viewed and executed in a VRML browser. Not all browsers result is identical behavior. They also are in development stages as they continue to resolve issues with each released version and work towards compliance with the VRML specification.

There is no single best tool to use in the debug phase. I found that the combination of concurrently using several tools during the development saved time in the long run. The VRML browsers are all tools used in this development phase because of errors given for not being able to render the world and the use of the console for print statements during execution. The different browsers gave different errors so that is the reason the concurrent combination works best. If development is done strictly with the use of a single browser, complete redesign might be necessary to be compliant and executable in another browser. Because of differences in VRML browsers, both Blaxxun and Cosmo Player were used to find problems in implementation as the development progressed. The advantage of having different browsers available is to verify behavior in the browsers to aid in finding errors because of the discrepancy between browsers. From there you can determine if it is a code problem or a browser problem.

Blaxxun works best with Internet Explorer browser. Parallel Graphics Cortona is a browser that became available after the dining philosopher model was designed. During testing of Cortona, as a possible browser for the rube™ project, a problem with

the sound implementation and logic was uncovered that was never detected with the Blaxxun browser. Cortona works with both Internet Explorer and Netscape.

Cosmo Player does not handle statements in the script as the code is written which results in behavior that does not represent the coded behavior. In many cases it does not show in a world. In the case of the dining philosophers, the world is not executed as the code is written, and therefore the behavior is not correct. Because a random ordering is imposed for each execution of transition order checking for fire, Cosmo takes the random order that is generated on the first execution of that code and uses the same random order for all executions of that code instead of using the new random order generated at each execution. A work around was tested in order to obtain the correct behavior by forcing the code ordering, but it imposed time delays in the model that affected the visual representation of the behaviors. Cosmo is still a useful tool for testing because the error messages are helpful in finding problems. Cosmo Player works best with Netscape. For viewing a model, I suggest using either Cortona or Blaxxun, and using Cosmo Player only for development purposes.

Blaxxun is the most lenient in terms of code and timing issues but it logically gives the correct behavior as expected from the code written as does Cortona. Both do statements in the script in the correct order so the behavior is as expected. Cortona is stricter in conformance to VRML specification.

Continued development of the VRML browsers has been done with input from users so needs surfaced in a development project can be addressed in future versions of the browser. The close working relationship with the browser vendors serves both

parties, i.e., the browser becomes a better product and the needs of the user are met. With this type of collaboration, progress can be made in parallel.

## Presentation of the Model Product

Once the model has been created, the presentation of the model product is made to the user. In the VRML browser this is done with the user navigating through the world or changing to the different viewpoints defined in the world.

### Viewpoints

The viewpoint shown when the world is loaded is the first viewpoint that is defined in the VRML file of the world. The starting viewpoint in the dining philosophers is a "movie" or animated tour of the execution of the model. It is a tour of the model from the outside looking in and then it moves to the inside of the model and looks outward. The movie gives the feeling of actually being there. It is an immersive model where you get inside the model and feel a part of it. Creating an animated viewpoint is creating an animation of the camera so you can walk through the world in a seamless fashion. Cosmo Worlds animation tool was used to create the animated viewpoint. The file of the world can be opened in cosmo world and then use the animation tool to record the camera movement as you navigate through the model. A presentation of this animated viewpoint of the model in execution was given at the start of the opening address at the 2000 Winter Simulation Conference in Orlando, Florida.

The viewpointrecorder PROTO was discovered after the creation was done in Cosmo Worlds with the animator. The viewpointrecorder is done inside a VRML browser making it easy to navigate. This navigation is recorded and the code is generated

to copy and paste so the navigation viewpoint is quickly incorporated as part of the model.

**AVI Files**

An avi file was made of this initial “movie” viewpoint of the dining philosophers model so that this animated viewpoint could be viewed without a VRML plug-in. The avi file was created with TechSmith Camtasia screen recorder. For best viewing, download Tech Smith’s TSCC Codec from http://www.techsmith.com. To view the animated viewpoint, click on the movie link here.

An avi file was also recorded with a static camera viewpoint during the execution of the model so an extended observation of the execution could be observed. The execution of the model can be viewed as it could in a VRML browser without the navigational abilities. The dynamic behaviors of the model in execution can be observed and studied with this method of communication. To view the static viewpoint, click on the movie link here.

Other viewpoints in the world are defined as static camera positions with the dynamics of the execution visible. An image gallery was created of static screen shots from the different viewpoints defined within the world while the model was in execution. Although an instantaneous screen shot does not give the viewer the picture of the behavior, it does show the static physical and aesthetic structure of the model and all of its objects and relationships. The Image Gallery, found in the Appendix, shows the static viewpoints defined in the dining philosophers model.

## Summary

The implementation of the Petri net dynamic model template and the implementation of a system using the Petri net dynamic model template within the rube™ methodology using metaphor mapping was successfully accomplished. The behavior of the Petri net resulting from the Petri net rules and the marking of the Petri net at each state was successfully implemented into a 3D behavioral visual representation with audio enhancement using metaphors. The dining philosophers problem is itself a very interesting system to model making it even more interesting to implement and demonstrate with rube™. The dining philosophers problem was the system of interest to model for this research implementation but the dynamic model template that was designed for Petri net as part of this research can be used for any system of interest to model with a Petri net using the rube™ methodology.

# CHAPTER 5
# CONCLUSIONS

This chapter summarizes the accomplishments in this research and the resulting conclusions. An opinion on communication is then presented. A discussion on the future of the field of modeling and simulation is also provided. Finally, comments on the future research within the rube™ research project are presented.

## Accomplishments

The following summarizes the accomplishments and goals attained through the implementation of the Petri net component of the rube™ research project. Successful implementation of the dynamic model template for Petri nets demonstrates proof of concept of the rube™ methodology through yet another dynamic model type. This demonstration is shown from the dining philosophers system, which is mapped to the formal mathematical model of Petri nets, mapped to the 2D graphical representation, and then mapped to the 3D artistic objects with the use of metaphors to accomplish an aesthetically pleasing static structure of the model. Metaphors representing the dynamic behaviors were then added to the model. The use of the metaphors resulted in a model that is quickly understood and remembered. The most important impact was the visual representation of the dynamic behaviors of the model during execution. The model is interesting and aesthetically pleasing, which creates an enjoyable experience while studying the model.

The rube™ team under the leadership and vision of Dr. Fishwick has been successful in demonstrating through implementations of models that the methodology of designing models is beneficial and that development should continue into the next phase.

## Conclusions

It is more effective and easier to convey the structure of a physical object in 3D space using 3D objects than it is for a 2D symbolic representation, a text description, or, a mathematical representation. This conclusion is easy to make from the dining philosophers formalism compared to its visual representation in 3D space. In this 3D artistic representation, it is easy to understand all the static relationships within the model.

When analyzing the model output or behaviors in text form, it is much more difficult to construct a "picture" of the results. If the output is displayed in a graphical visual form the understanding is much clearer and much more rapid. With the addition of involving more visual representation through 3D and metaphor mapping, the concept of the model becomes clearer. Understanding of the model comes faster and requires less effort. Use of audio adds an additional feature involving yet another intuitive and basic sense to achieve an even more meaningful model with more communication, thus further enhancing the model's understanding.

Metaphors make important connections between the real system and the model, and it is this common language that allows the user to connect the two domains and understand the model. A model that is interesting and aesthetically pleasing as a result of the use of metaphors and art is going to be a useful and beneficial model. Because of the model's acceptance to the user and the user's ability to feel a part of the model, it

ultimately results in an increased understanding. The user's acceptance to the model is even greater because the model is easier to relate to because of the use of metaphors.

The communication of the static structure is automatic with good visualization and layout. With the combined visual representation of both the static structure and the dynamic behaviors, it is easier to concentrate on understanding, reasoning, and analyzing the model's interactions and behaviors. The importance of the layout of the physical objects is critical to efficient understanding. This importance becomes clear from the comparison of the Petri net model in Chapter 2, Related Work. Because of the placement of components in the 2D space, it is difficult to follow the execution of the model. However, the placement of components in my layout, whether the 2D space or the 3D space with top view, aids in the understanding and visualization of the changes during execution of the model.

Model implementation time is reduced because of the visualization of behaviors. The visualization makes it easier to see and debug problems during the implementation stage. The analysis and verification of the model's physical and behavior correctness is more efficient. In addition, it is also more efficient for the model designer to create the model using metaphors because the metaphors bring the target object into a common language with the model. Therefore, the relationship between the target and source is automatic and intuitive.

The benefits of reuse of the dynamic model template are obvious. The work and effort in the design foundation of the underlying model type has already been created, tested, and validated. With this reuse product, the model designer can map the target object with the use of metaphors to create a working model of the system.

## Communication

This section provides information on the personal observations and opinions of the researcher on communication that was formulated from the research and preparation of this thesis. Communication comes in many forms, styles, and flavors. The goal of any type of communication is to express the ideas in a way that is meaningful and will be readily understood. The presentation of the product for communication is important in acceptance by the receiver. Communication is a powerful tool if it is used effectively.

As difficult as it is to express and explain many complex concepts in writing, the same is true of the process of interpretation of those same words. The goal is to have both the reader and the author see the same mental picture created by the words. Writing is a way to communicate, but it can also be accomplished in an aesthetic way that is easier to understand.

It is much easier to sketch a diagram to explain many concepts than it is to put the concepts into words. In a diagram, the interpretation is easier and faster, and can lead to a better understanding that is closer to the intent of the author. The communication between the author and receiver is still not perfect. The addition of another dimension and the involvement of other senses assist with clarifying the picture as it becomes closer to reality. Within a model, there is still another communication that is most important and that is the dynamic behavior. The communication of the dynamic behaviors is the essential part in the understanding of a model. The use of metaphors is the way to communicate a model aesthetically and effectively.

Communication of a model is the intended purpose in the design. The communication channel used should be the best available within the limits of cost and time. Good communication results in understanding. The concept of creativity applied

to model presentation using an artistic touch makes the model more alive, enjoyable, and easier to understand. The use of art in communication results in better presentation of the model. The rube™ methodology and VRML allow the communication of the model and the model behaviors to be effectively implemented in a 3D aesthetic environment.

## Future of Modeling and Simulation

Modeling and simulation is in its infancy with enormous opportunity remaining for contribution to the advancement of this field. As with any area that is not mature, many things are not well defined or not defined at all. This has advantages and disadvantages.

One advantage includes the opportunity to be a part of the definitions that later are viewed as the paradigms within the field. This research has proven that stepping away from tradition by integrating art into the model design along with presenting new methodologies receives acceptance from people in the field with an understanding of the benefits and direction within the field. The honor of Dr. Fishwick's rube™ web site being selected as the site of the month for February 2001 for "The Society for Simulation and Modeling International European Council" web page [26] shows the acceptance of the rube™ methodology to the field of computer simulation. Thinking outside the traditional box of textbook formalism and methodology can change "the way things are done," resulting in the benefits and satisfaction from a product that is more meaningful and useful.

A disadvantage is that all the technological tools may not yet be available to accomplish immediate objectives. As with any development process, progress is made

for the short-term. A concept may be proven or illustrated initially and then further work continues to accomplish the long-term objectives.

Some areas are mature and well-structured with no room for major changes; only room for small tweaks or fine-tuning is needed like in a well-developed piece of machinery. It is much more exciting and rewarding to be involved with an effort that can have a big impact on the field.

Cost is no longer an excuse for not moving in the direction of more powerful 3D animated models. The benefits of a model that makes use of more of the users senses will continue to grow with more innovation in model design because technology will continue to advance and be readily available.

The 3D will be web based because it only makes sense to utilize the infrastructure already in place for distribution. The web provides a mechanism for efficient worldwide distribution of models, thus helping to fuel the growth of the field. The web encourages more collaborative efforts. Collaboration in any sector is more productive when more people are involved. This collaborative effort is beneficial for the business, government, and educational arenas.

A paradigm shift will evolve toward the use of metaphors instead of the traditional symbols of a specific model type or even a symbol given to a component of a real system. Metaphors will be the link to bridge the two different domains, the real system and the model.

The intersection of art and science will be utilized to achieve aesthetic models created with artists working together with the model designer or dual talented artists creating the models themselves. This new breed of model designers will be produced

from university programs, like the newly created Digital Arts and Science Program at the University of Florida, Gainesville, Florida.

## Future Work

### Mapping Automation

Automating the process of mapping the model formalism to 2D graphical representation to 3D artistic representation would be included in the future development of this research. The first phase would involve the static structure mapping of the 2D to 3D. Then, research would proceed with the more difficult mapping of the formalism to the 2D graphical representation.

The mapping in this initial phase of the research effort was completed manually. The next step would be to develop a methodology that would allow automation of this mapping process. The 2D mapping to 3D is a much easier task to automate because of the one-to-one relationship of the components in each structure. The investigation into the tools and mechanism to accomplish automatic mapping would be the continuation of this initial phase into the next phase of the rube™ project.

### Creation of Electro Acoustic Music

A future application of the designed model and use of dynamic model template could be in the field of music. The combination of Petri net models and sound files could be a way of generating electro acoustic music. This might be accomplished with the designed Petri net model or creation of different static structures. The use of multimodeling to achieve levels of music could also be investigated.

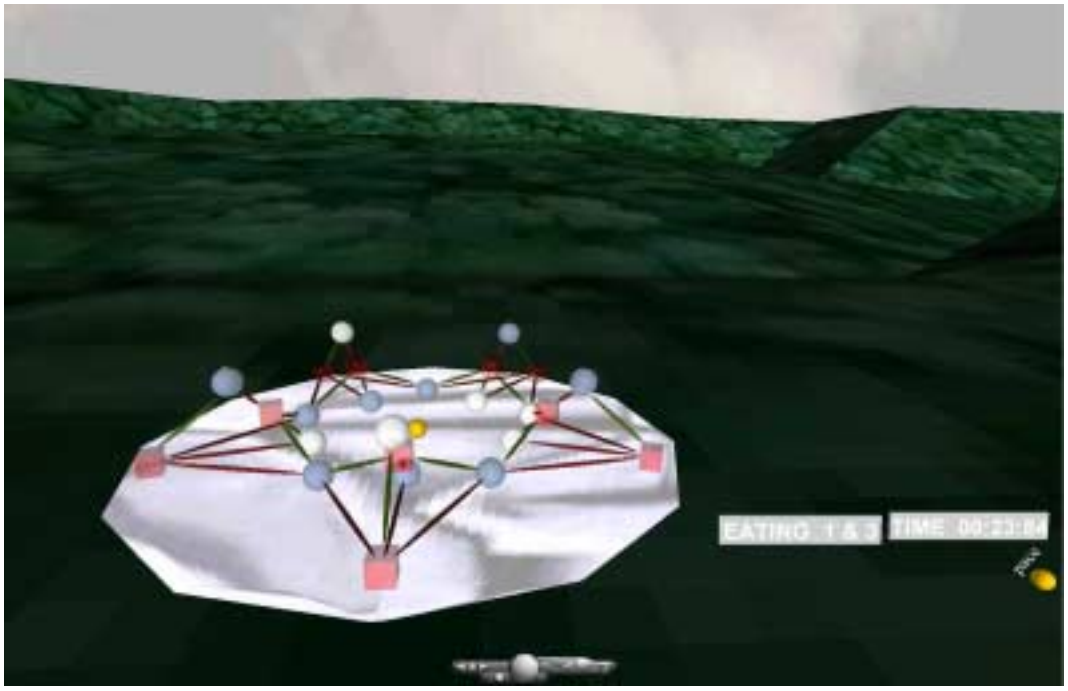
## Summary

Model design is an exciting field with growing room for creativity from the artistic side, innovative approaches for communication of behaviors, and advancements from technology for improvements in delivery. It provides the challenge of creating models that can go beyond the expectations of providing model output for analysis by providing insights into the system being modeled that might otherwise be overlooked or even unappreciated until they are revealed. These new model design methodologies will then be viewed as a necessity in future model design, thus allowing the field to expand.

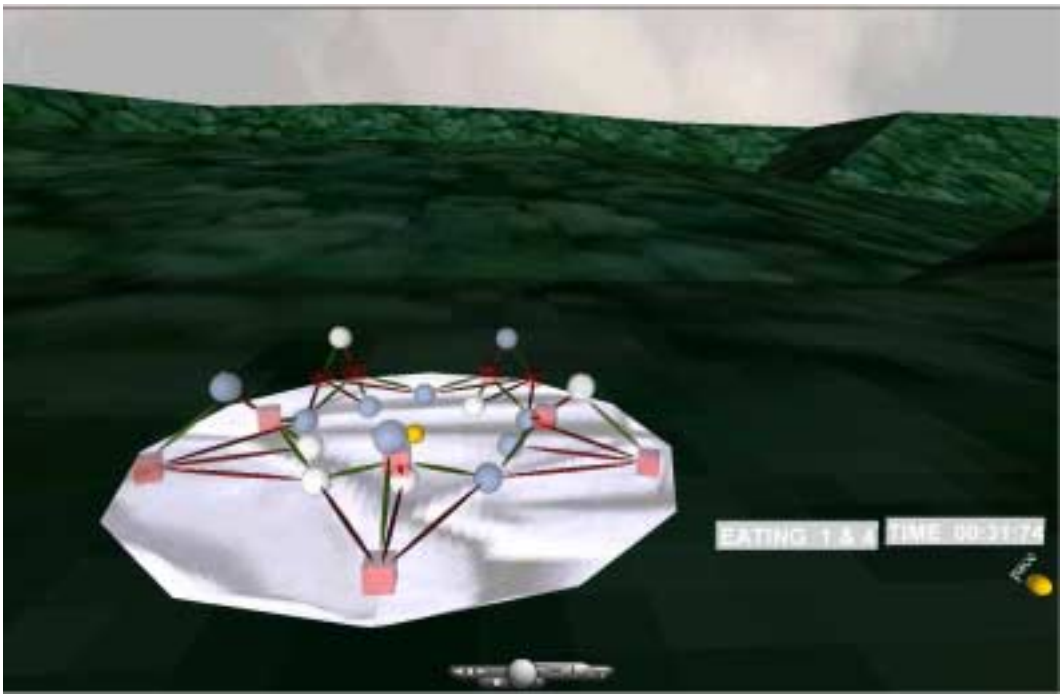
I feel from my personal experience in working on the rube™ research project, that the goal of designing a model that has magnitudes greater understanding and benefit in the 3D framework using metaphors has been achieved. My addition to the statement of Chapter 1, Introduction, if a picture is worth a thousand words, and the 3D Web is worth a billion words, "then 3D with behavioral metaphors is worth a trillion words for the field of modeling and simulation." This visual representation of the dynamic behaviors of the model during execution brings enormous value to the model. The rube™ modeling philosophy will continue to be developed because of the benefits realized from the model designs.

APPENDIX
IMAGE GALLERY

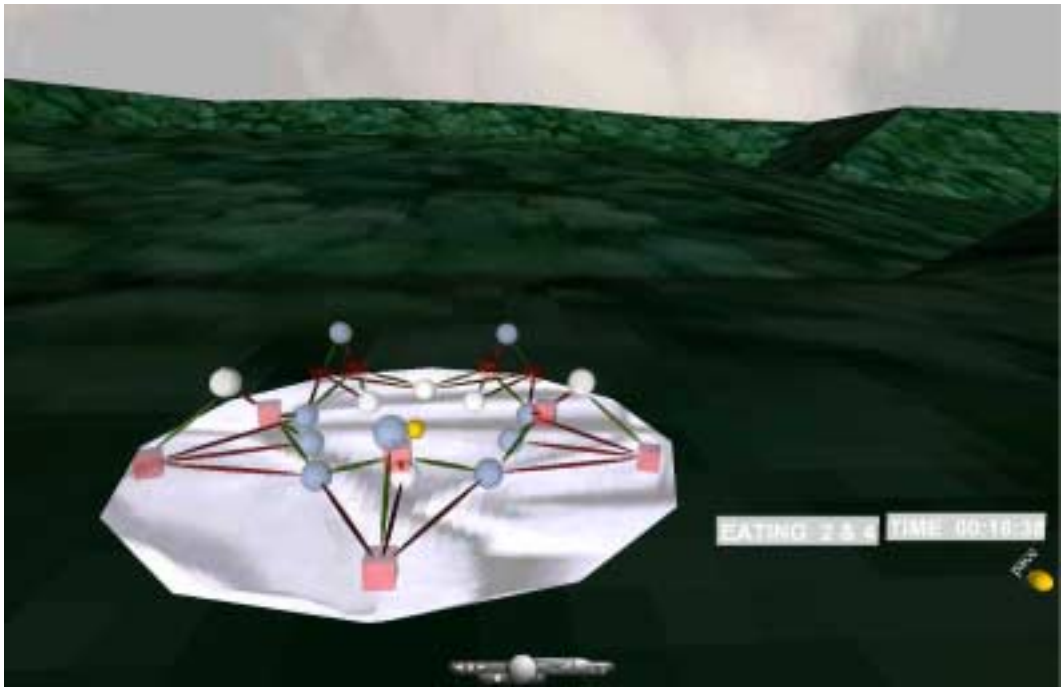
The names correspond to the viewpoint name in the dining philosophers world.



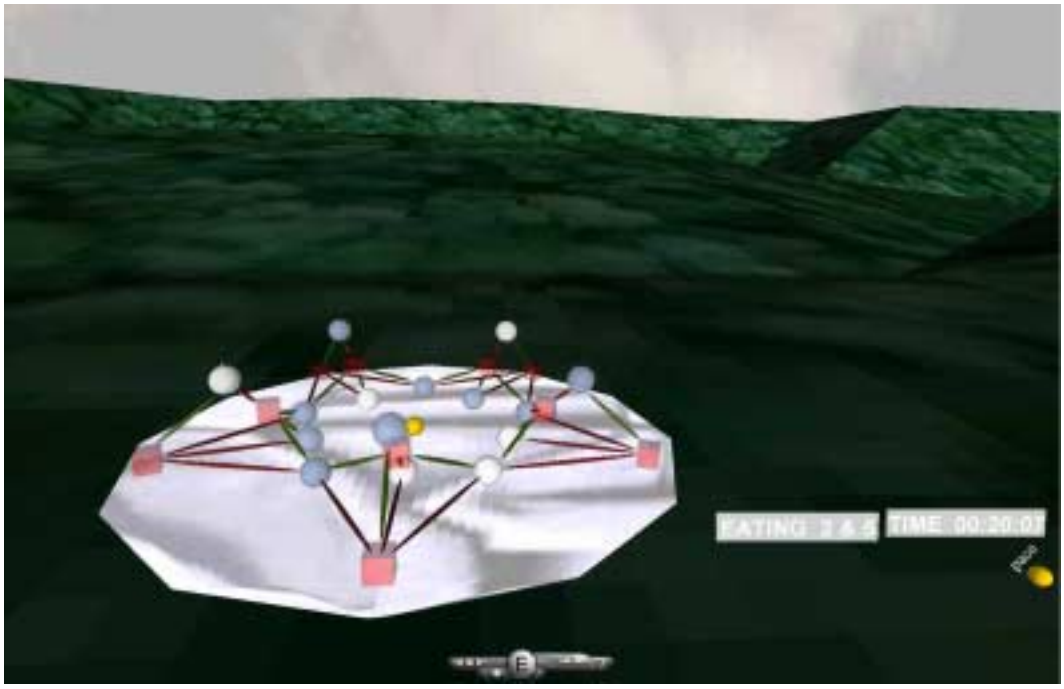

1 & 3 eating



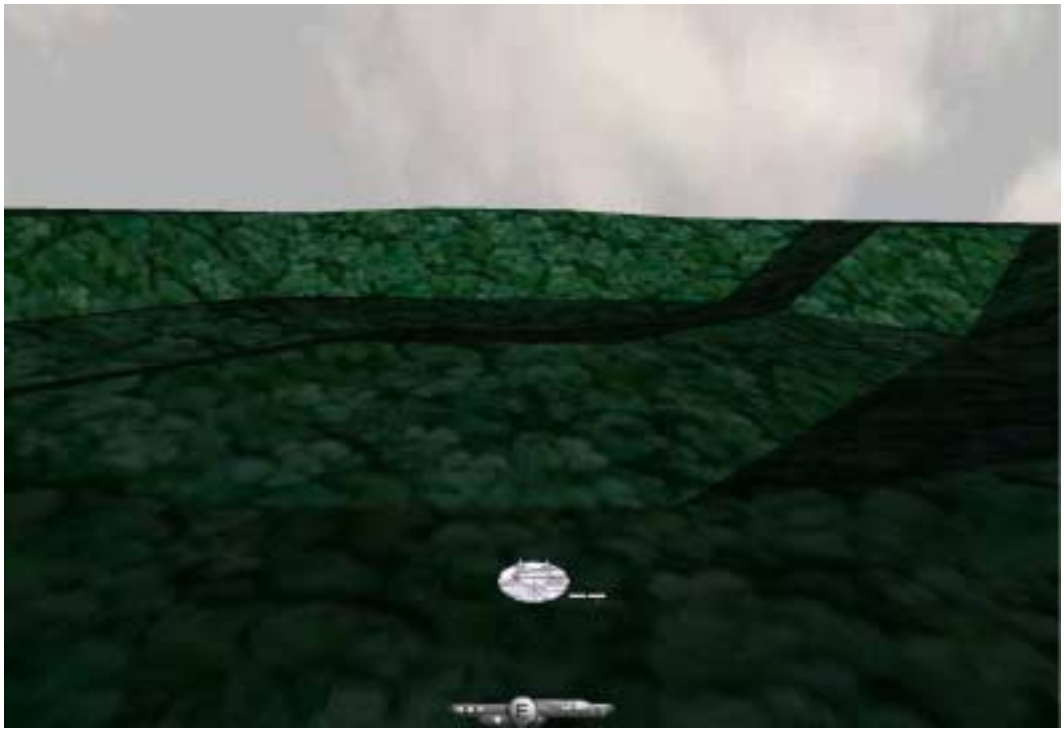

1 & 4 eating



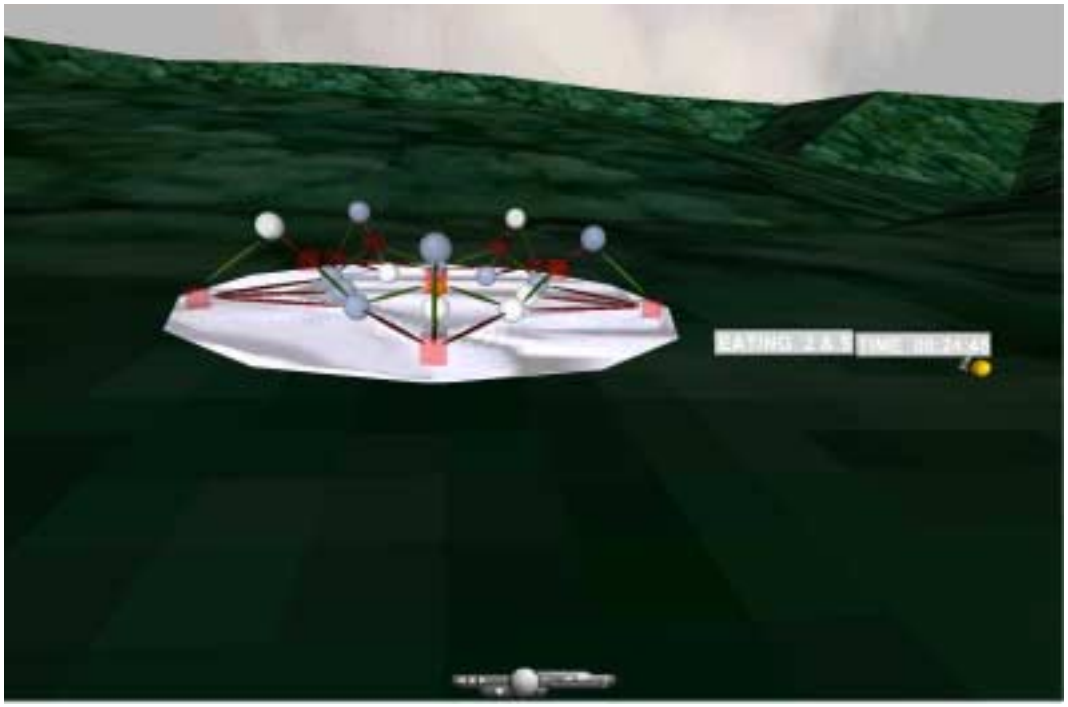

2 & 4 eating



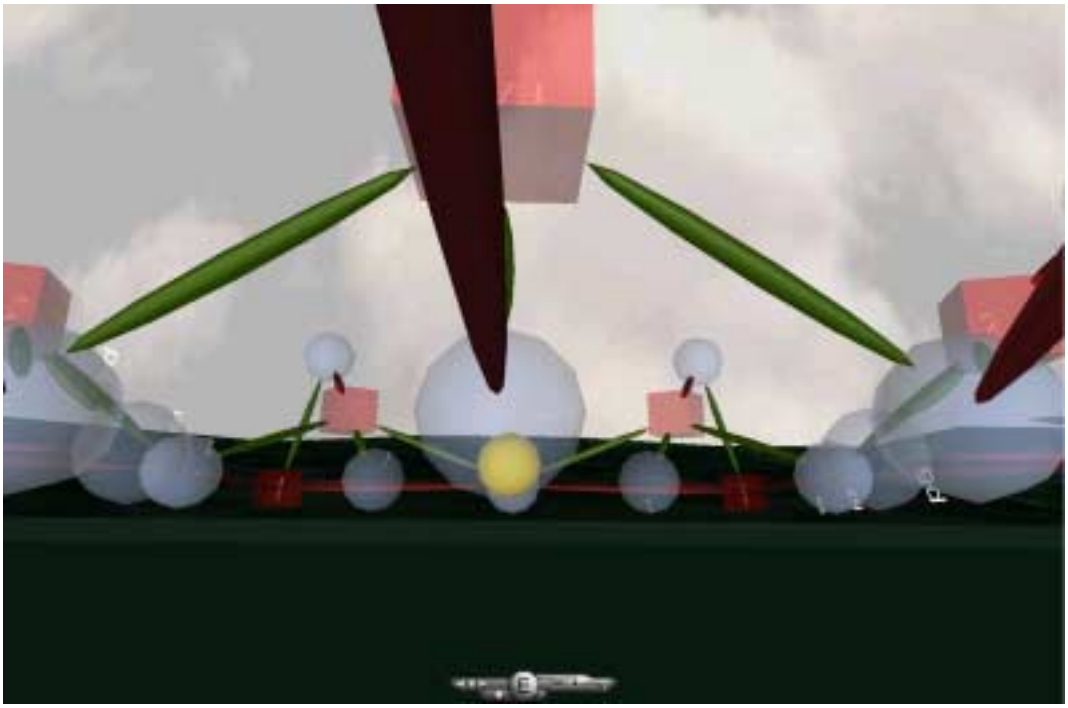

2 & 5 eating



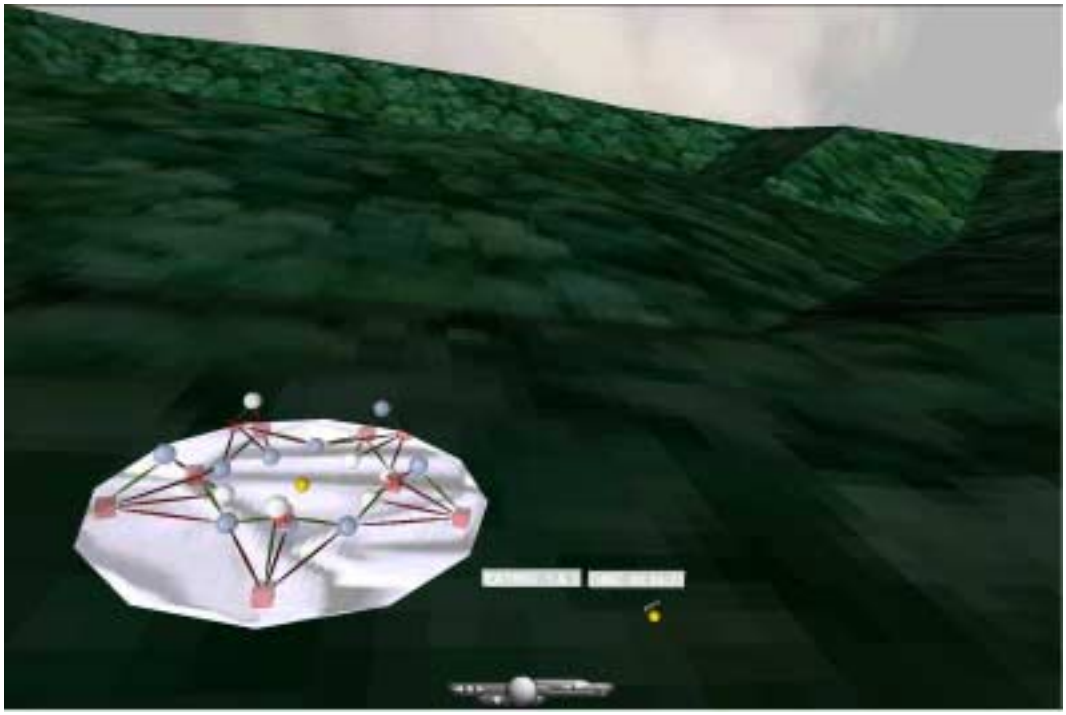
Distance



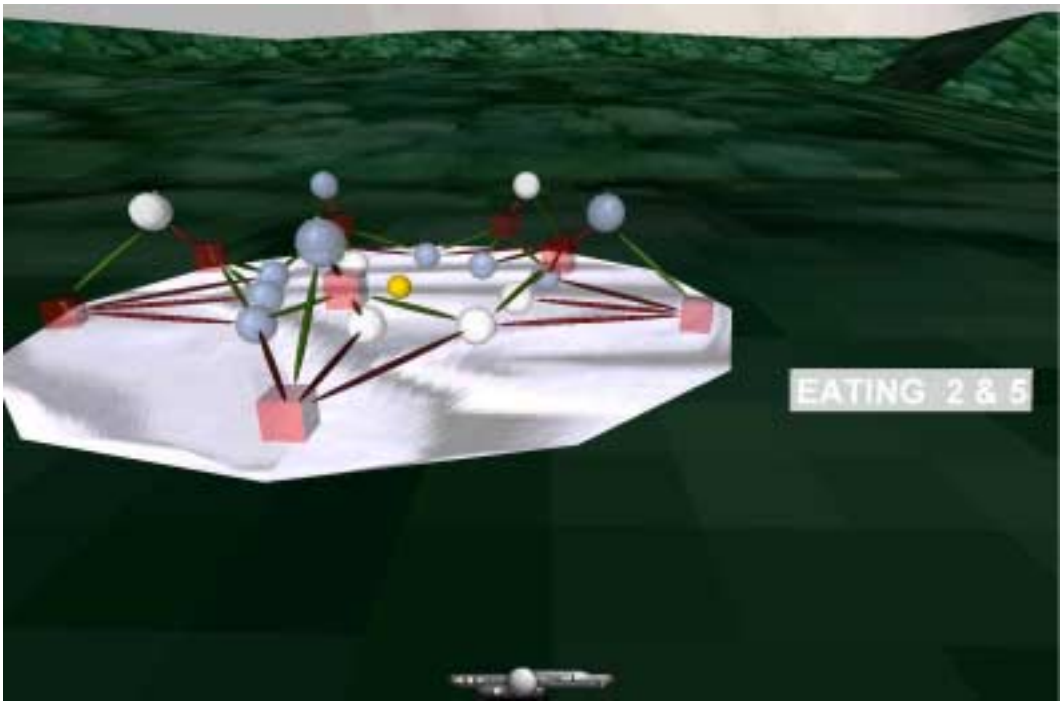
Eating



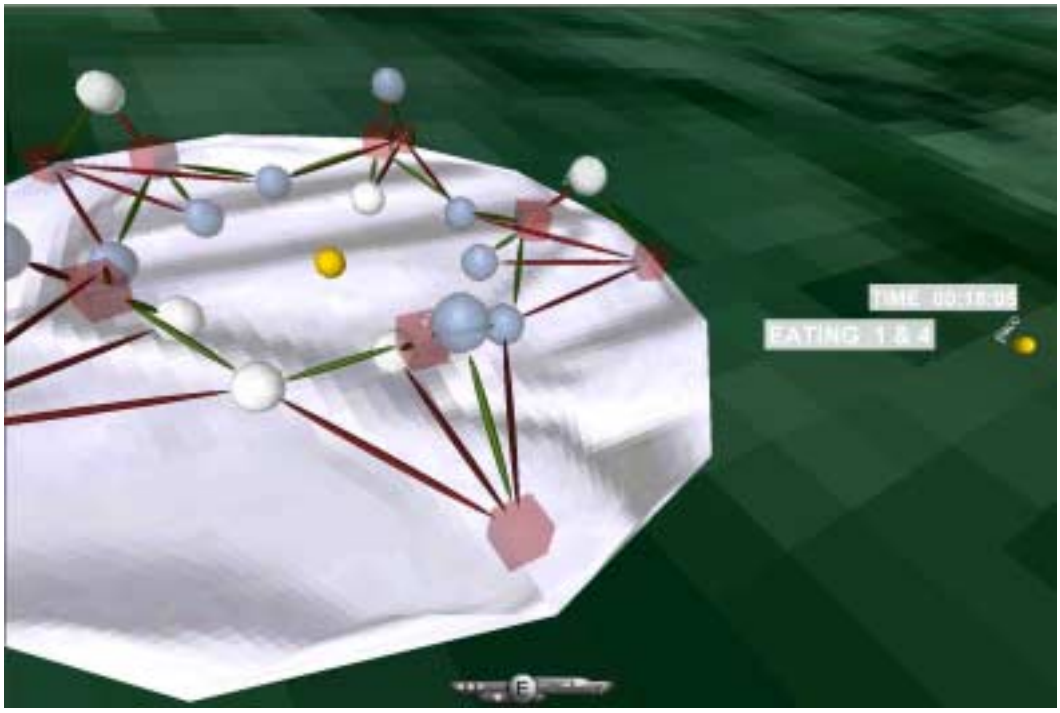
Inside



Observe



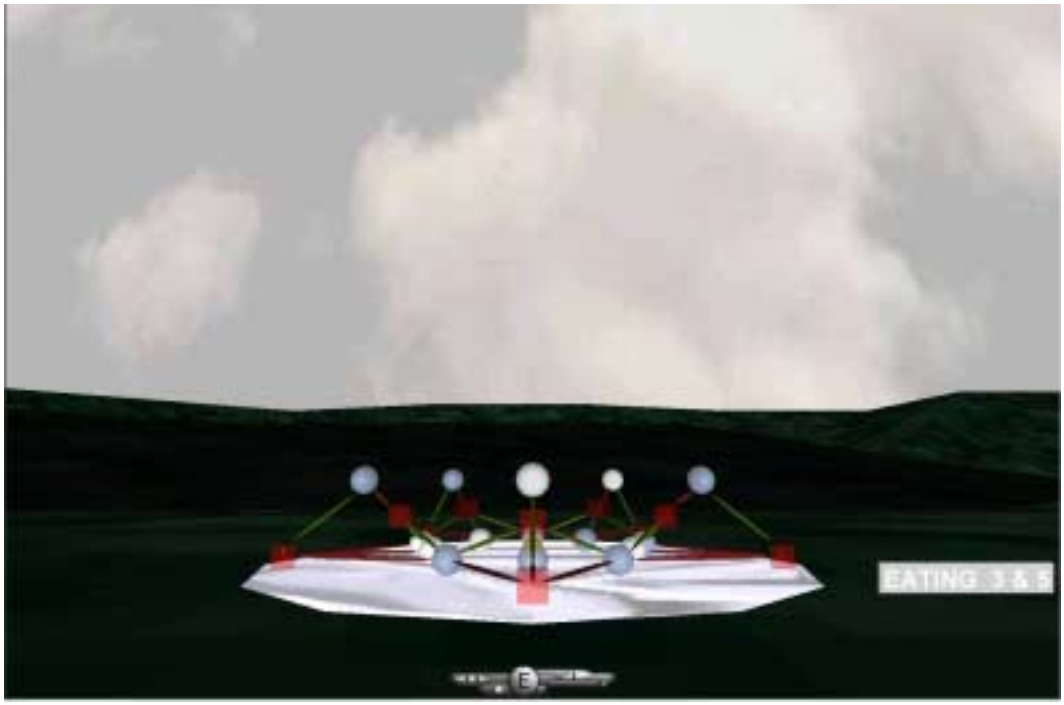

Observe2



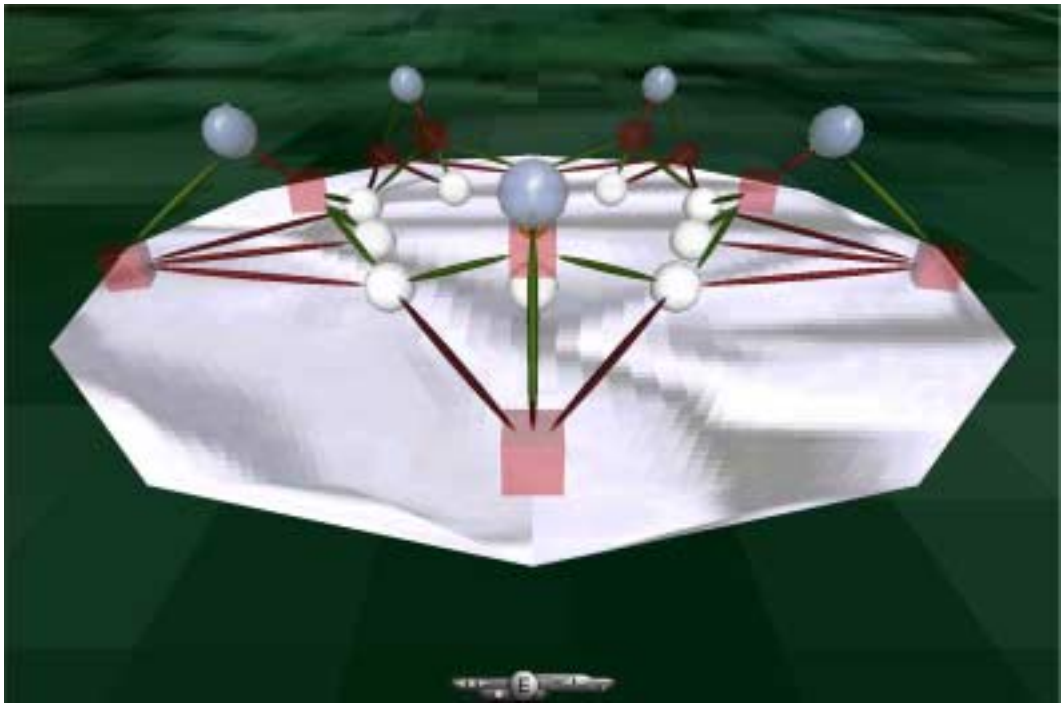

Observe3

Side distance

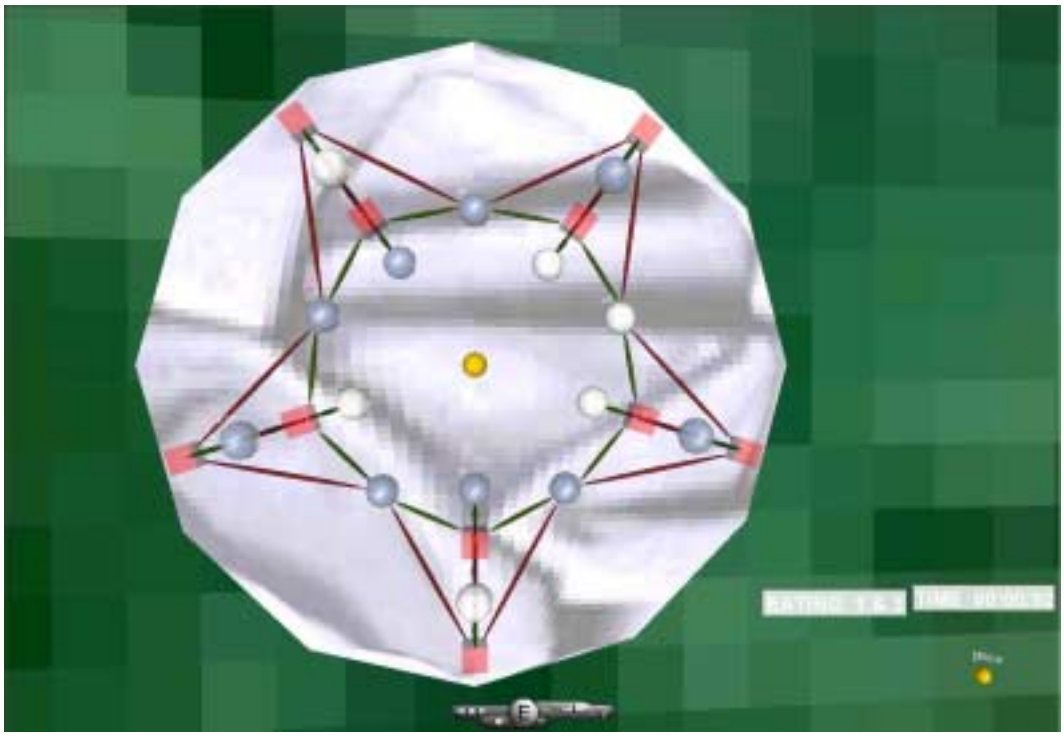
Side distance 2



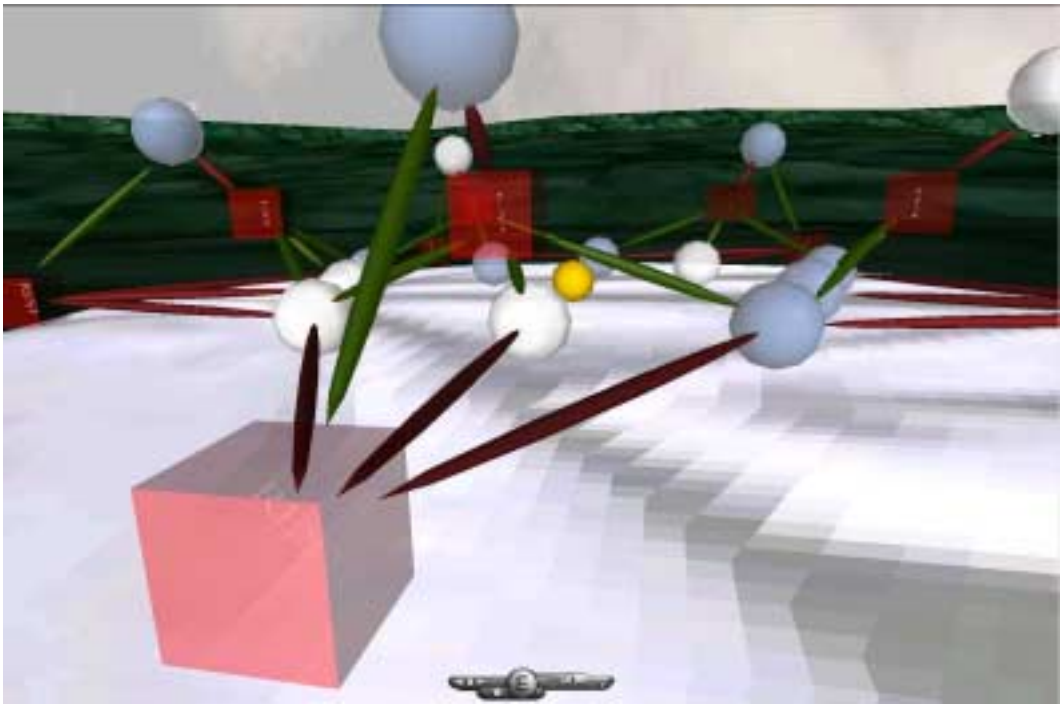

Side view



Start tokens



Top view



Upclose

LIST OF REFERENCES

[1] Paul A. Fishwick, *Simulation, Model Design and Execution, Building Digital Worlds*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1995.

[2] Paul A. Fishwick, CAP 5805 class notes, Computer Simulation Concepts. University of Florida, Gainesville, Florida, 1998.

[3] Paul A. Fishwick, "3D Behavioral Model Design for Simulation and Software Engineering," in *Web3D/VRML Conference*. Monterey, CA, February 2000, pp. 7-16. http://www.cise.ufl.edu/~fishwick/tr/tr99-017.html, December 2000.

[4] Anne Morgan Spalter, *The Computer in the Visual Arts*. Addison Wesley Longman, Inc., Reading, Massachusetts, 1999.

[5] Naim A. Kheir, *Systems Modeling and Computer Simulation*. Marcel Dekker, Inc., New York, New York, 1996.

[6] Matthew Mirapaul, "3-D Space as New Frontier," in *The New York Times on the Web*. http://www.nytimes.com/2000/10/05/technology/05SPAC.html, October 2000.

[7] Computer Science and Telecommunications Board, National Research Council, *Modeling and Simulation Linking Entertainment and Defense*. National Academy Press, Washington, D.C., 1997.

[8] Matthew W. Rohrer, "Seeing Is Believing: The Importance of Visualization in Manufacturing Simulation," in *Proceedings of the 2000 Winter Simulation Conference*. Orlando, Florida, December 2000.

[9] Alyn Rockwood, "A Techie picks up the Paintbrush," in *Accenture Digital Arts and Sciences Lecture Series*. University of Florida, Gainesville, Florida, February 2001.

[10] James L. Peterson, *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1981.

[11] Paul A. Fishwick, *rube™ Worlds*. http://www.cise.ufl.edu/~fishwick/rube/worlds.html, November 2000.

[12] Paul A. Fishwick, *Newell's Dynamic Teapot*. http://www.cise.ufl.edu/~fishwick/rube/worlds/tea.html, January 2001.

[13] Andrew Reddish, *Cassini Probe*.
http://www.cise.ufl.edu/~fishwick/rube/worlds/cassini.html, January 2001.

[14] John Hopkins, *Operating System Kernel*.
http://www.cise.ufl.edu/~fishwick/rube/worlds/os.html, January 2001.

[15] Robert Esser, *Petri Net Browser*.
http://www.cs.adelaide.edu.au/users/esser/index.html, October 2000.

[16] MengChu Zhou and Frank DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Academic Publishers, Boston, Massachusetts, 1993.

[17] Wolfgang Reisig, *Petri Nets*. Springer-Verlag, Berlin, 1985.

[18] Bernard P. Zeigler, *Theory of Modelling and Simulation*. Robert E. Krieger Publishing Company, Inc., Malabar, Florida, 1976.

[19] Randy Chow and Theodore Johnson, *Distributed Operating Systems and Algorithms*. Addison Wesley Longman, Inc., Reading, Massachusetts, 1998.

[20] Paul A. Fishwick, *What is rube?*
http://www.cise.ufl.edu/~fishwick/rube/intro/index.html, September 2000.

[21] "Metaphor," in *Encyclopædia Britannica*. http://www.britannica.com, December 2000.

[22] Peter Coad, *Object Models: Strategies, Patterns, and Applications*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1995.

[23] Paul A. Fishwick, "Aesthetic Programming," in *Leonardo*, submitted for review September 2000. http://www.cise.ufl.edu/~fishwick/tr/00/leo.pdf, December 2000.

[24] Paul A. Fishwick, *Rube Style Guide*.
http://www.cise.ufl.edu/~fishwick/rube/intro/style.html, September 2000.

[25] Andrew S. Tanenbaum and Albert S. Woodhull, *Operating Systems Design and Implementation.* Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1997.

[26] The Society for Simulation and Modeling International European Council.
http://hobbes.rug.ac.be/~scs/, February 2001.

# BIOGRAPHICAL SKETCH

Linda Kaye Dance was born January 21, 1956, in Albuquerque, New Mexico, to Gilbert and Ruth Dance. She has one sister, Faye, and one brother, Thom. Linda graduated from Avon Park High School, Avon Park, Florida. She graduated with honors with the Bachelor of Science in Engineering degree from the Department of Mechanical Engineering in the College of Engineering at the University of Central Florida in 1979. She worked in industry as a co-op student while working on her undergraduate degree. She continued her career in manufacturing with General Electric and worked on projects implementing state of the art manufacturing equipment and processes. She was a recipient of the General Electric Managerial Award in 1982 and again in 1985. Linda is a registered professional engineer in the state of Florida. She returned to academia in 1998 to pursue the Master of Engineering degree from the Department of Computer and Information Science and Engineering in the College of Engineering at the University of Florida. Her degree is expected in 2001.