DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Relatório Técnico
Technical Report
DCA-005/97

Using Petri Nets for Animation Modeling and Analysis

Alberto B. Raposo[1], Léo P. Magalhães[1,2]

[1]UNICAMP (State University of Campinas)
FEEC (School of Electrical and Computer Engineering)
DCA (Dept. of Computer Engineering and Industrial Automation)
{alberto,leopini}@dca.fee.unicamp.br

[2]University of Waterloo — Computer Science Dept. — CGL
lpini@cgl.uwaterloo.ca

April 1997

Using Petri Nets for Animation Modeling and Analysis

Abstract: This paper presents the use of Petri Nets for animation modeling and analysis. At first some traditional approaches for animation modeling are discussed. Then the use of Petri Nets in an animation environment is presented. For the sake of simplicity the paper does not explore the theorethical side of the methodology, believing examples are the most appropriate way to introduce this new tool for animation modeling. Two examples show some of the possibilities offered by the methodology.

Keywords: Computer Modeled Animation, Animation Control, Petri Nets

1 Introduction

This paper is an effort in the direction of supporting animation modeling using a formal framework based on Petri Nets theory.

At one side the utility of Petri Nets for modeling animation environments will be discussed; at other side the powerful contribution of Petri Nets for animation analysis purposes will be highlighted.

The following section will introduce animation and Petri Nets. Section 3 will address the use of Petri Nets for animation modeling and analysis. The last sections will present the conclusions and related bibliography.

2 Animation: synthesis and analysis

This paper addresses **computer modeled animation** from the point of view of its control, i.e. the control of characters and characters interactions acting in an animation.

Movements of characters in an animation can be defined using parametrical interpolations, kinematic and dynamic equations (direct and inverse), genetic models, etc. [1], [2], [11], [12], [15]

Characters interacting with the environment can be controlled from an antecipative point of view (as for example interpolation) or detected on the fly using for example collision detection techniques. [4], [13]

At a more abstract level one can consider characters subject to emotions, having a reactive behavior, or being task or goal oriented. Here control strategies are based on

logical description of behaviors, system of dynamic (or kinematic) equations, etc. [5], [9]

Section 3.1 will present some examples related to the above topics.

Petri Nets (from here on, PN) are a modeling tool applicable to a variety of fields and systems, specially suited to systems with concurrent events. They were introduced in 1962 [10]. Murata in [8] presents a very good introduction to the theme.

Formally, a PN can be defined as a 5-tuple PN={P, T, F, W, M_0 }, where:

- P= { p_1, ..., p_m } is a finite set of places;
- T= { t_1, ..., t_n } is a finite set of transitions;
- F $\subseteq$ (P x T) $\cup$ (T x P) is a set of arcs;
- W : F $\rightarrow$ {1,2,...} is a weight function;
- M_0 : P $\rightarrow$ {0,1,2,...} is the initial marking;
- (P $\cap$ T) = $\emptyset$ and (P $\cup$ T) $\neq \emptyset$

A very useful notation for PN is the graphic notation (see Figure 1) which will be used in the examples throughout this paper. In this notation, circles represent places, rectangles represent transitions, dots the marks (also called tokens), and arrows the arcs, with weights above (by definition, when no weight is shown above the arc, it is 1).

A transition *t* is said to be enabled if each input place p_i of *t* is marked with at least *w(p_i,t)* tokens, where *w(p_i,t)* is the weight of the arc between p_i and *t*. A transition will fire (once enabled) when its associated event occurs. Firing the transition *t*, *w(p_i,t)* tokens will be removed from each input place p_i of *t*, and *w(t, p_o)* tokens will be added to each output place p_o of *t*.

For example, in the PN of Figure 1, only transition t_2 is enabled; t_1 is not enabled because it would require two marks in P_1 to fire ($w(P_1, t_1)$ = 2). When t_2 is fired, the marks in P_2 and P_3 are removed and P_4 receives one mark (note that the number of marks in a PN is not conserved).

Briefly stated [8], the behavior of a system using PN is described in terms of its states and their changes. States are modeled by *places* (the *marks* define the current state of the system). *Transitions* (firing rules) model the dynamic behavior of the system. *Arcs*

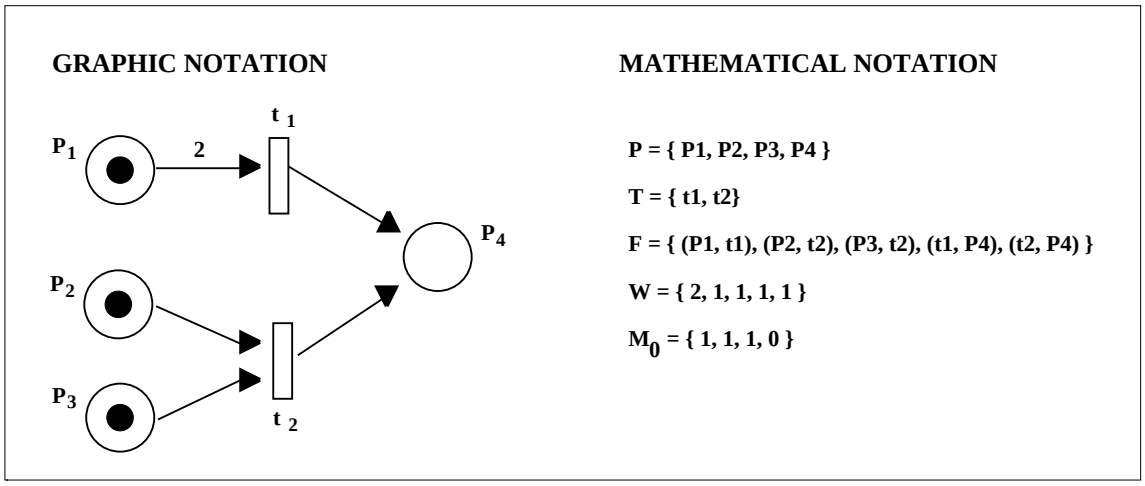

Figure 1: Petri Nets graphic and mathematical notation.

indicate the sequence of possible transitions between states and they can be *weighted* meaning the quantity of *marks* necessary to fire a transition.

Besides the modeling capabilities of PN, their support for analysis is very important and useful. This analysis is based on the properties of the mathematical model of PN. Some of these properties are:

- **Reachability**: is there a sequence of firing that reaches a desired state ?
- **Boundness**: will a place be overloaded ?
- **Liveness**: is there any state or sequence of states which will not be reached anymore (possible deadlock) ?
- **Reversibility**: is it possible to return to a defined initial state M_0 ?
- **Persistence**: is the firing of any pair of enabled transitions interdependent, i.e. the firing of one will disable the other ?
- **Synchronic Distance**: defines a metric related to the degree of mutual dependence between two events in a condition/event system.

The correct modeling and use of PN properties for analysis will allow the animator to preview the behavior of an animation even before starting any implementation. For example:

- one can detect modeling problems related to defined animation states which will never be **reached**;
- one can define some **boundness** related e.g. to a number of actors wished in a state;
- one can find states which will never happen if an specified state is reached (**liveness**);
- **reversibility or home state** allows to test if an initial state can be reached from another state;
- interdependence between animation (or actors) states can be tested through **persistence** and **synchronic distance** characteristics of the PN.

3 Exploring PN to support animation

Section 3.1 introduces the powerful characteristics of PN applied to animation problems by means of a simple but clear example. The benefits that can be achieved in a more complex environment will be discussed in Section 3.2.

3.1 Presenting animation paradigms

The example of Figure 2 shows two buttons and a sphere which can follow two different trajectories depending on which button was chosen. One button is associated to an internal trajectory of the sphere and the other with an external one.

In order to highlight some of the animation modeling methods, it will be shown how to model this example using different animation techniques.

The script of Figure 3 can represent the definition of an animation of the above environment.

Interpolation

First of all the so called *keyframe* technique will be presented. Using this technique the animator predefines the initial and final frames of an animation sequence (the key frames) and an interpolation function or a set of interpolated values responsible for the transformation between those frames. After that the animation will run. Time and number of frames play in this case similar roles.

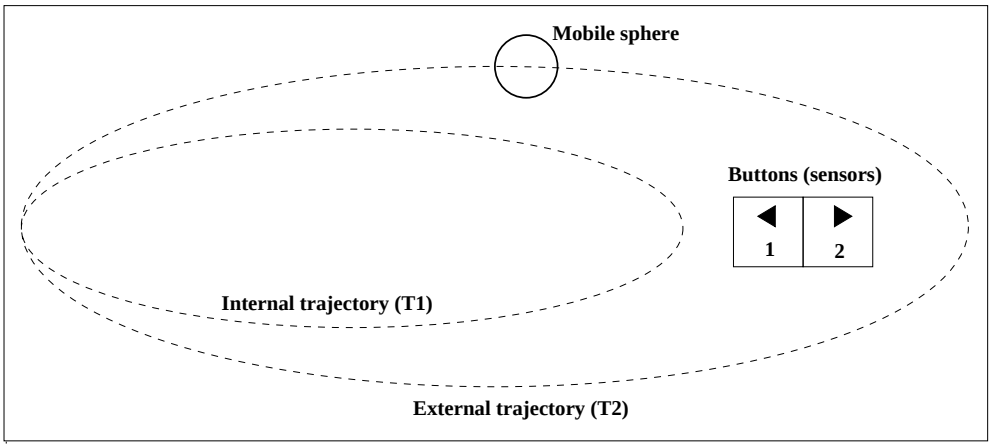

Figure 2: Basic example.

```
position objects (button-1, button-2, sphere)
define the trajectory files/functions
repeat until end-condition
    if button-2 is chosen
        move sphere using trajectory-T2
    else move sphere using trajectory-T1
end
```

Figure 3: Basic animation script.

Normally, on the fly modifications of the animation are not performed since the animation is entirely predefined.

Kinematics

Using this technique the animator predefines the movement using a time function or an equation system with kinematic variables (e.g., objects position, velocity, acceleration). For instance, for a pendulum it can be stated

$$\theta = k.\cos(wt + \beta),$$

where θ is the angular distance from the normal, *w* is the pendulum angular frequency (constant), β is the movement phase (constant), and *k* is the initial movement amplitude.

To implement the example of Figure 2 using kinematics, one has to define an equation

to control the sphere movement relating space and time. This equation will be similar to the pendulum equation relating distance (x,y) and time. After that the animation will run. If the model is appropriately defined, movements more realistic than those using interpolation can be obtained.

Dynamics

This technique defines the movement using a time function or an equation system with dynamic variables (e.g., torque, inertia, force, mass). Tipically a dynamic equation states (here for a car shock absorber)

$$x_i = \frac{\Delta t^2.F_{ext} + (b.\Delta t + 2m).x_{i-1} - m.x_{i-2}}{m + k.\Delta t^2 + b.\Delta t},$$

where F_{ext} is the external force acting on the system, *m* is the mass, *b* is the viscosity friction, *k* is the elasticity constant, and *x* is the distance.

In the example of Figure 2, one has to state an equation relating external forces, sphere mass, air friction, etc to define the sphere trajectory equations. After that the animation will run. Again, if the model is well designed, more realistic movements can be achieved than by the last two techniques.

Behavioral

This is a higher abstraction level technique, where interactions between actors and the environment define characters behavior. Tipically, characters movements are defined by one of the above techniques and the interaction control is performed based on some predefined event.

For the above example the event of pressing one button could be associated to a warning signal (e.g. button 1 danger, button 2 no problems), signalizing the trajectory to be followed.

The four techniques presented above are very similar considering the available tools to analyse a defined animation. The usual procedure is to run simulations (normally using degraded quality in terms of rendering and frames/sec) and then correct possible undesired effects.

The point addressed in this paper is related to the offer of a set of tools to provide the animator with some (the most possible) knowledge about the behavior of his/her animation, even before the first frame be shot.

3.2 Modeling animation by PN

In this section a PN is used to model the example of Figure 2. After that some of its analysis properties will be commented.

The animation is modeled as follows:

- place 1 (P_1) is associated to trajectory T1;
- place 2 (P_2) is associated to trajectory T2;
- transition 1 (t_1) is associated to the press of button 1;
- transition 2 (t_2) is associated to the press of button 2.

Figure 4 shows the representation using PN.

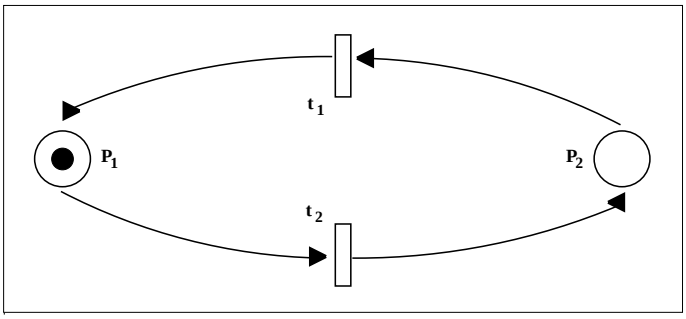

Figure 4: PN representation of the basic example.

First of all it is interesting to note that the main character of the animation (the sphere) is represented by the mark (dot token) in the graph. Since the interest is to represent the behavior of the environment, more attention is dedicated to the trajectories and sensor values (respectively places and transitions).

The graph gives the reader the following direct information :

- there are two stationary states in the environment (sphere follows trajectory T1; sphere follows trajectory T2);
- The effect of pressing a button is dependent on the current trajectory (e.g., if the sphere is following T1 — token in P_1 —, the press of button 1 has no effect, since t_1 is not enabled);

- the movement will remain forever.

Information can also be obtained by means of an informal analysis of the graph based on the properties of PN:

- all possible states can be reached;
- there are no deadlocks (the graph is alive);
- whichever the initial state is, it is possible to return to it;
- the transitions are mutually dependent, i.e. it is not possible to fire one of them two times sequentially, without firing the other (note that it is possible to press the same button two times sequentially, but the second time has no effect in the animation).

Actually, this example is very simple. In the following section the complexity of the presented example will be increased, in order to stress the use of PN for the analysis of an animation behavior.

3.3 Analysis

This section will detail some modeling and analysis aspects of PN taking a more complex environment.

The basic example, Figure 2, will be modified as shown in Figure 5. Now there are two spheres, each one travelling in one of the two possible trajectories (internal or external) controlled by a button. The animation has a behavioral restriction defined by the rule that only one of the spheres can be at its external trajectory at a time (otherwise they could collide).

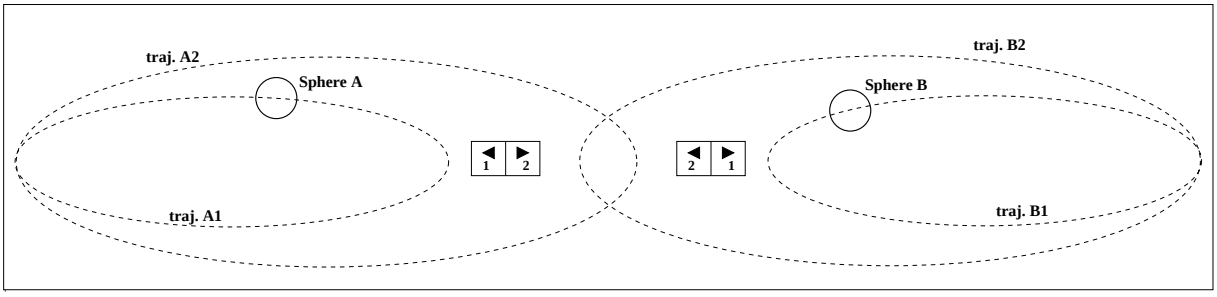

Figure 5: Two spheres example.

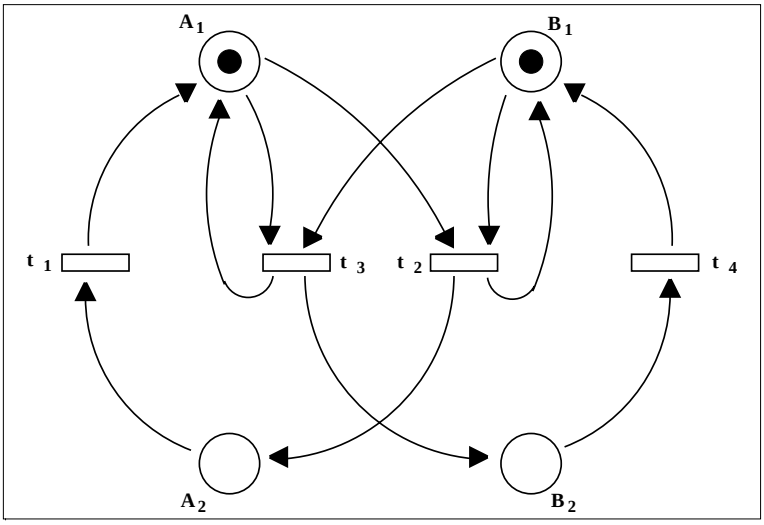

Figure 6: PN for the example with 2 spheres.

Figure 6 introduces the PN model for this example.
The following characteristics can be obtained from the modeled environment:

- there are 4 places (defining the 2 possible trajectories for each sphere);
- there are 4 transitions defining button press events;

  -transitions t_1 and t_4 are associated to the press of the buttons that put the spheres A and B, respectively, in their internal trajectories;

  -transitions t_2 and t_3 are associated to the press of the buttons that put the spheres A and B, respectively, in their external trajectories;
- M is the set of possible markings (A_1,A_2,B_1,B_2): M = {(1,0,1,0); (1,0,0,1); (0,1,1,0)}.

A powerful tool for the analysis of PN is the coverability graph which offers a vision of the complete sequence of transitions and states in a PN. Sometimes a state like *(0,w,1)* could appear representing a repetitive addition *(w)* of a mark in a place (but the presented example does not exhibit this behavior). Figure 7 illustrates the coverability graph for this example, considering the initial state M_0 = (1, 0, 1, 0).

Based on the coverability graph and taking into account the described PN properties (see Section 2) the following PN characteristics can be stated:

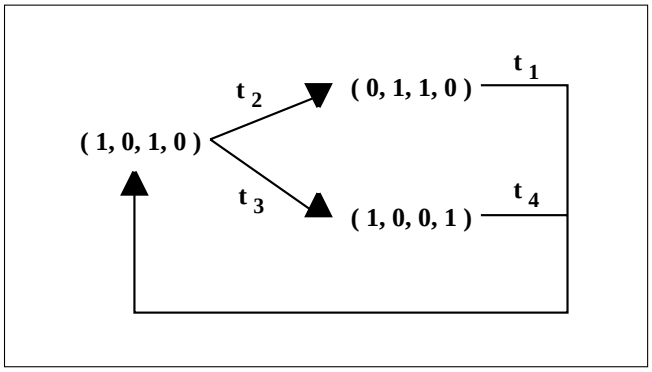

Figure 7: Coverability graph for the example.

- **Reachability**: starting at one of the states of M the system never goes to the forbidden state (0,1,0,1) and to the impossible states like (1, 1, 0, 0) in which one sphere follows two or no trajectories. All other states can be obtained by a defined firing sequence;

- **Boundness**: the PN is 1-bounded, i.e. no place will have more than one mark at a time. A PN with this property is called *safe*;

- **Liveness**: each valid state has a following valid state for a firing sequence (there are no deadlocks). In addition, all transitions appear in the coverability graph, meaning there are no *dead* transitions;

- **Home State**: it is always possible to return to an initial state by a firing sequence;

- **Persistence**: this PN is not persistent. Firing t_2, t_3 will be inhibited and vice versa. This expresses the mutual exclusion relationship between both events;

- **Synchronic distance**: this characteristic expresses the level of mutual dependence of two transitions: $d_{i,j}$ = max $\mid \sigma(t_i)$ - $\sigma(t_j) \mid$, where σ is a firing sequence at any marking in M and $\sigma(t_i)$ is the number of times t_i fires in σ.

  For instance: $d_{1,2} = d_{3,4} = 1$, defining that these pairs of transitions are interdependent (in fact, the transitions of each of these pairs are associated to events of the same sphere); $d_{1,4} = d_{2,3} = \infty$ defining that these pairs of transitions are associated to independent events (they are associated to events of different spheres).

It can be concluded from the above points:

- the developed model has no undesired states. It conforms with the initial animation description;
- there are no deadlocks (the control statements will always put the animation in a valid state);
- the animation will run forever (there is no final state);
- there are two mutually excludent transitions, confirming the behavioral restriction;
- the only mutually dependent transitions ($d_{i,j} = 1$) are associated to events of the same object (sphere 1 or sphere 2).

4 Conclusion and future works

This paper introduced the use of PN for the process of animation modeling and analysis. [3] is other of the few attempts to use PN in Computer Graphics (in that case for an interface design environment).

Although many aspects of PN were left out of this paper, e.g. colored PN [7], state equations and priority net [8] among the most important ones (they are going to be considered in an ongoing work), the authors believe the essential aspects of the methodology have been presented and its most useful characteristics have been exposed.

The example of Section 3.1 was implemented in a VRML 2.0 (Virtual Reality Modeling Language) environment [6], [14]. The techniques used in this implementation were the keyframe (the trajectories were predefined by key values) and the behavioral (the value of the sensors defines which of the predefined trajectories is followed). Figure 8 shows two frames of this animation (in the left frame the sphere is following the internal trajectory and in the right frame it is following the external one). Each half of the two-colored cube is one sensor and the other cubes are only references to distiguish between the trajectories.

The authors believe that the use of PN, a well established methodology, will help to accelerate the process of animation testing, moving the current methods based on an attempt/error approach to a more accurate one. Furthermore, the use of PN allows a clear separation between the animation actions related to the movement control and those related to the animation appearance (e.g. rendering, camera and ligths control), contributing

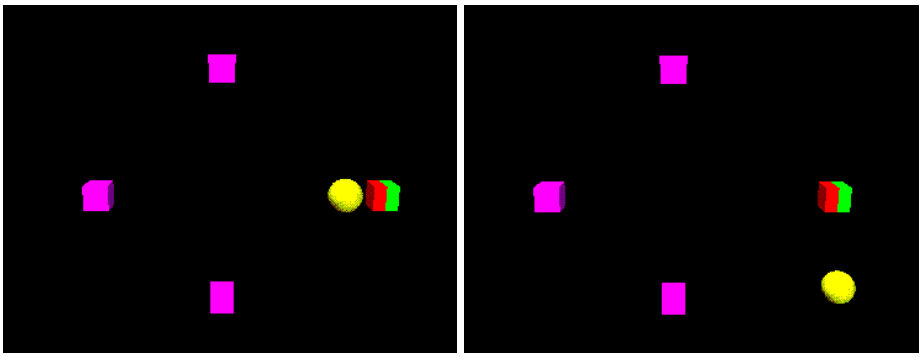

Figure 8: Animation implemented with VRML 2.0.

to a “cleaner” modeling environment.

Acknowledgements

The authors would like to thank the following institutions for the expressive support granted to this research: UNICAMP (State University of Campinas), FAPESP (Foundation for Research Support of the State of São Paulo) and the University of Waterloo. Thanks also to Prof. Dr. I. L. M. Ricarte and Dr. J. T. F. de Camargo for the clarifying discussions related to PN and the examples of this paper.

References

[1] N.I. Badler. Computer animation techniques. In *Introduction to Computer Graphics*, Course Notes for SIGGRAPH 95, Vol. 21. ACM–SIGGRAPH, 1995.

[2] R. Barzel. *Physically-Based Modeling for Computer Graphics - A Structured Approach*. Academic Press, Inc., 1992. ISBN 0-12-079880-8.

[3] R. Bastide and P. Palanque. *A Petri-Net based environment for the design of event-driven interfaces*. Lecture Notes in Computer Science (935). Springer–Verlag, 1995. pages 66-83.

[4] J.T.F. Camargo, L.P. Magalhães, and A.B. Raposo. Modeling motion simulation with DEDS. In *Proc. of IFIP 94 – $13^{th.}$ Congress of the International Federation of Information Processing*, pages 162–167, 1994. (http://www.dca.fee.unicamp.br/projects/prosim/publiPS.html).

[5] M. Costa and B. Feijó. Agents with emotions in behavioral animation. *Computer & Graphics, Pergamon Press*, 20(3):377–386, June 1996.

[6] Dimension X. *Liquid Reality*, 1996. (http://www.dnx.com/products/lr/).

[7] K. Jensen. Coloured petri nets: A high level language for system design and analysis. In *High-level Petri-Nets : Theory and Application*, pages 44–119. Springer–Verlag, 1991. ISBN 3-540-54125X.

[8] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[9] K. Perlin and A. Goldberg. Improv: A System for Scripting Interactive Actors in Virtual Worlds. In *Proc. of SIGGRAPH 96*, pages 205–216, 1996.

[10] C.A. Petri. *Kommunikation mit Automaten*. Schriften des IIM Nr.3, Bonn: Institut fuer Instrumentelle Mathematik, 1962.

[11] M. Preston and W.T. Hewitt. Animation using NURBS. *Computer Graphics Forum*, 4(13):229–241, October 1994.

[12] K. Sims. Evolving Virtual Criatures. In *Proc. of SIGGRAPH 94*, pages 15–22, 1994.

[13] N.M. Thalmann and D. Thalmann. Complex Models for Animating Synthetic Actors. *IEEE Computer Graphics and Applications*, 11(5):32–44, 1991.

[14] *The Virtual Reality Modeling Language Specification – Version 2.0, ISO/IEC CD 14772*, August 1996. (http://vrml.sgi.com/moving-worlds).

[15] B. Wyvill. A computer animation tutorial. In *Computer Graphics Techniques: Theory and Practice*, Editors - Rogers, D.F. and Earnshaw, R.A., pages 235–282. Springer–Verlag, 1990.