

# An Approach to Petri Net Based Formal Modeling of User Interactions from X3D Content

Jianghui Ying\*  
Virginia Tech

## Abstract

X3D standard provides a well defined and well controlled runtime model for user interaction and scripting. Information captured from X3D content, combined with the characteristics of X3D viewers, can be used to formally describe and model user interface and user interactions. Virtual Environments (VEs) and entities they contain are accessed using interaction techniques supported by available input and output devices. The interaction process can be modeled using Petri nets. In this paper, we give a formal definition of interaction Petri net model (IPN). The basic structure of IPN can be automatically generated from information extracted from X3D content. The generated Petri net model is analyzed to provide information about the 3D interface presented to a user while viewing X3D content. The formal description provides a sound mathematics for the analysis of user interface characteristics.

**CR Categories:** H.5.2 [INFORMATION INTERFACES AND PRESENTATION (e.g., HCI)]: User Interfaces—Theory and methods; D.2.2 [SOFTWARE ENGINEERING]: Design Tools and Techniques—Petri nets

**Keywords:** X3D, Petri net, Formal Methods

## 1 Introduction

Virtual environments (VE) based user interfaces have different characteristics compared to traditional user interfaces. Traditional user interfaces follow an existing design standard, WIMP (Windows, Icon, Menu, Pointer) while VE user interfaces lack such standards. VE user interfaces have a higher degree of interactivity and exhibit continuous and concurrent input/output “exchanges”. A large number of different tools and file formats [Walsh and Borges-Svenier 2001] makes this situation even more challenging. New abstractions and approaches for describing and implementing these interfaces are needed.

X3D [Web3D 2003] is an Extensible Markup Language (XML) based standard that is emerging as the de facto open standard for cross-platform, inter-application 3D content delivery. X3D is divided into profiles such as interchange, interactive, immersive, and full profile. The interactive profile provides for basic interactions in a 3D environment. Various sensor nodes provide support for user navigation and interactions, enhanced timing, and additional lighting.

---

\*e-mail: [jying@vt.edu](mailto:jying@vt.edu)

Copyright © 2006 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM Inc., fax +1 (212) 869-0481 or e-mail [permissions@acm.org](mailto:permissions@acm.org).  
Web3D 2006, Columbia, Maryland, 18–21 April 2006.  
© 2006 ACM 1-59593-336-0/06/0004 \$5.00

X3D can be used to construct interactive, 3D applications, either stand-alone or web-based. A component based approach, used in the CONTIGRA architecture [Dachselt et al. 2002] is based on declarative XML documents describing the component implementation, interface, configuration and composition of 3D user interfaces and VEs. This approach reuses both the implementation level and the component level. In [Dachselt and Rukzio 2003] an XML-based language, Behavior3DNode, is used to define new behavior nodes that can be used together with built-in nodes as first class scene graph elements. XSLT (eXtensible Stylesheet Language Transformations) can be used for separation of the presentation from the content, thus providing multiple views and “personalized” interaction schemes with the same data set [Polys 2003].

The X3D format contains information describing how a user can interact with the presented 3D content. That information can be extracted to identify available user interactions and to analyze the user interface characteristics.

Formal methods are used for user interfaces design and analysis to abstract from the details of users, software, and their interaction. Formal methods provide a basis for reasoning and analysis, and can ensure correct implementation of the required software [Rouff 1996]. Formal specification of a user interface provides a number of advantages, such as early use in the design process, precise prediction of usability aspects, precision for design and communication, and enabling automatic generation of user interfaces. A well defined formal method must have a sound mathematical basis, typically given by a specification language, which provides precise notations for constructing mathematical models of a target system. Using formal specifications, the desired properties are described in a way that can be reasoned about, either formally or informally but rigorously.

## 2 Petri net based User Interface Modeling

Petri net is a behavior-based formal method used for modeling concurrent systems [Peterson 1981]. A Petri net is a directed graph consisting of two types of nodes: places and transitions. A place and a transition are connected by an arc either from a place to a transition or vice versa. An arc is labeled with an integer weight  $k$ . In a graphical representation, places are drawn as circles and transitions as boxes. Each place has an assigned marking (state) with  $k$  tokens, where  $k$  is a nonnegative integer. The graph has an initial state called the initial marking,  $M_0$ .

Petri net based approaches to user interface design include PUIST [Li and Mugridge 1994], PENGUI [Shirogane and Fukazawa 1998], and OSU [Keh and Lewis 1991]. However, those approaches focused on a 2D user interface design.

Petri net graph can be described in Petri Net Modeling Language (PNML) [PNML 2000], an XML-based interchange format [PNML 2000]. Since both X3D and PNML are XML-based, a sequence of XSLT steps can be applied to X3D content to generate the equivalent Petri net using the PNML as the target.

Most of tasks performed within a 3D environment (or VE) are

application-specific so it is difficult to model all of them. However, there are some basic interactions that most 3D interactions are composed of. The majority of these interactions fall into three task categories: navigation, selection / manipulation, and system control [Bowman et al. 2001].

Navigation is the most prevalent user action in most VEs. Manipulation involves changing properties of virtual objects such as position, orientation, scale, shape, etc. Manipulation always implies selection, but selection may be a stand-alone task. System control covers other commands that the user gives to accomplish work within the application, such as save the current scene, etc. If Petri net can be used to model these basic interactions, other complex tasks could also be modeled.

The space in a 3D environment is constructed using a number of virtual objects. The state of the 3D environment is determined by the state of each object. A Petri net model of that environment captures the state changes of each object and describes the complete system. Within the Petri net model, the places represent the state of the objects and the transitions refer to how the user interacts with the objects. The marking  $M_0$ , a collection of place's markings, describes the state of the system.

In addition to 3D content description, 3D file formats typically include an inherent event model. An event has a life cycle that begins with the action or condition that initiates the event and ends with the final response. The life cycle of a typical event consists of the following steps: user action or condition occurs, event instance is created, event fires, event propagates, and event related behavior is triggered.

There are two types of event providers. The first type includes external sources, like mouse, glove, keyboard, etc. Normally, a user fires an event by manipulating external input devices. The second type of events include internal sources, like timers or a user defined functions or routines, which can generate any type of events. Normally, user interactions are based on external sources. However, the internal sources may be also involved when an event needs to propagate to other nodes in the scene graph.

The event model provides the necessary information about user interactions. Although different file formats use different ways to propagate events, it is sufficient to know the result of each action. Therefore, different parsers are needed to analyze different file formats, but the extracted information should be similar. That is, how user navigates in a 3D environment, how user selects/manipulates objects, and how user performs system control.

### 3 Formalism of Interaction Petri net

The Petri nets model describing interaction process with VE is called Interaction Petri Net (IPN). In this section, we will give a formal definition of IPN based on the Petri net formalism.

There are three types of places in IPNs. The first type is the device resource place ( $dp$ ) representing the resource of an input device like a mouse or a data glove. The second type is the sensor place ( $sp$ ), which represents the sensors that are responsible of receiving inputs from the input device. The transitions that connect  $dp$  and  $sp$  change the token in the third type of place, the entity place ( $ep$ ).

Tokens associated with different places have different meanings. There are three types of tokens:  $d$ -token,  $s$ -token, and  $e$ -token.  $D$ -tokens represent the available device resources. Initially, a  $d$ -token exists in a  $dp$ . Only when the  $d$ -token is available, the  $dp$  can lease

it to an  $sp$  (the interaction transition associated with the  $dp$  is triggered). This transition is called *lease transition* ( $lt$ ). The leased  $d$ -token will be returned by another interaction transition, called *return transition* ( $rt$ ). The user accomplishes a sequence of actions using input devices.  $S$ -tokens represent the interaction status as they move from one  $sp$  to another  $sp$ .  $E$ -tokens exist in  $eps$  and a  $ep$  is connected with an *action transition* ( $at$ ). An  $E$ -token represents the visual status of the associated  $ep$ . Normally, the value of the  $e$ -token is an attribute value of a visual property, such as color ( $r, g, b$ ), position ( $x, y, z$ ), light or sounds on/off ( $true, false$ ), etc. When an  $at$  is triggered, the  $e$ -token flows in or out of  $EPs$  and changes its value. Transitions that are not  $lt$ ,  $rt$ , or  $at$  are called *neutral transitions*  $nts$ .

**Definition 1** An Interaction Petri net,  $IPN = \{P, T, F, W, M_0\}$ , is defined as follows:

- $P = \{p_1, \dots, p_m\}$  is a finite set of places,
- $T = \{t_1, \dots, t_n\}$  is a finite set of transitions,
- $F \in (P \times T) \cup (T \times P)$  is a set of arcs (flow relation),
- $W : F \rightarrow \{1, 2, \dots\}$  is a weight function,
- $M_0 : P \rightarrow \{0, 1, \dots\}$  is the initial marking,
- $P \cap T = \emptyset$ ,
- $DP = \{dp_1, \dots, dp_i\}$  is the set of device places,  
 $SP = \{sp_1, \dots, sp_j\}$  is the set of sensor places,  
 $EP = \{ep_1, \dots, ep_k\}$  is the set of entity places,  
 $1 \leq i, j, k < m, i + j + k = m, DP \cap SP = \emptyset, DP \cap EP = \emptyset,$   
 $SP \cap EP = \emptyset$ ,
- $LT = \{lt_1, \dots, lt_u\}$  is the set of lease transition,  $LT \subseteq T$ ,  
 $\forall lt_r \in LT, \exists dp \in DP \wedge \exists sp \in SP$  such that  $dp$  and  $sp$  are  
input places of  $lt_r$ ,
- $RT = \{rt_1, \dots, rt_v\}$  is the set of return transition,  $RT \subseteq T$ ,  
 $\forall rt_s \in RT, \exists dp \in DP \wedge \exists sp \in SP$  such that  $dp$  and  $sp$  are  
output places of  $rt_s$ ,
- $AT = \{at_1, \dots, at_w\}$  is the set of action transition,  $AT \subseteq T$ ,  
 $\forall at_t \in AT, \exists (ep_{g_1}, ep_{g_2})$  such that  $ep_{g_1}$  is an input place and  
 $ep_{g_2}$  is an output place of  $at_t, ep_{g_1} \in EP, ep_{g_2} \in EP$ ,
- $NT = T - LT - RT - AT, LT \cap RT = \emptyset$ ,
- $\forall dp_l$  where  $dp_l$  is an input place of an  $lt_r, lt_r \in LT, \exists rt_s$  such  
that  $dp_l$  is an output place of a  $rt_s, rt_s \in RT, w(dp_l \times lt_r) =$   
 $w(dp_l \times rt_s)$ ,
- $\forall dp_l$  where  $dp_l$  is an output place of an  $rt_s, rt_s \in RT, \exists lt_r$  such  
that  $dp_l$  is an input place of a  $lt_r, lt_r \in LT, w(dp_l \times lt_r) =$   
 $w(dp_l \times rt_s)$ ,
- $M_0(DP) = \{1, \dots, 1\}, \forall dp_l \in DP, dp_l$  has an initial  $d$ -token  
representing the availability of the input device,  $1 \leq l \leq i$ ,
- $M_0(SP) = \{1/0, \dots, 1/0\}, \forall sp_f \in SP, sp_f$  may or may not  
have an  $s$ -token,  $1 \leq f \leq j$ ,
- $M_0(EP) = \{1, \dots, 1\}, \forall ep_g \in EP, ep_g$  has one initial  $e$ -token  
representing one or more attributes value of an visual entity,  
 $1 \leq g \leq k$ .

In an IPN, all  $dps$ ,  $sps$  and their associated transitions represent **how** user interactions are performed, while  $eps$  represent **what** is the result of an interaction.

An example IPN is given in Figure 2, for a Desk Lamp example [Web3D 2001] in Figure 1. We simplified the example for the purpose of generating smaller Petri net model. We removed all sensors except two *TouchSensors*: *touch1* and *touch2*. Therefore, only two

user interactions are allowed: turn on the lamp using the left (red) button and turn off it using the right (blue) button.

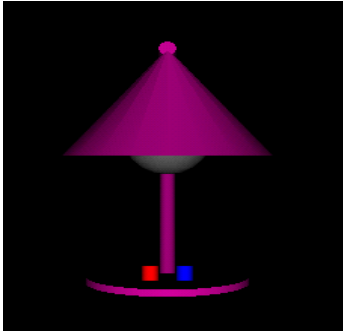


Figure 1: Desk Lamp example

This IPN has one *dp, mouse*, with one token inside, which means only one pointing device (mouse by default) is initially available. There exist two groups of *sps: touch1.Px* and *touch2.Px*, representing the corresponding status of the left and right buttons when accessed by mouse. There are also two *eps: color2*, and *light*, representing the color and light of the bulb. The token associated with *color2* and *light* tells the lamp status, on or off. They have two possible value combinations,  $([0.4, 0.4, 0.4], [true])$  and  $([1, 1, 0], [false])$ . The former pair is on and the latter off.

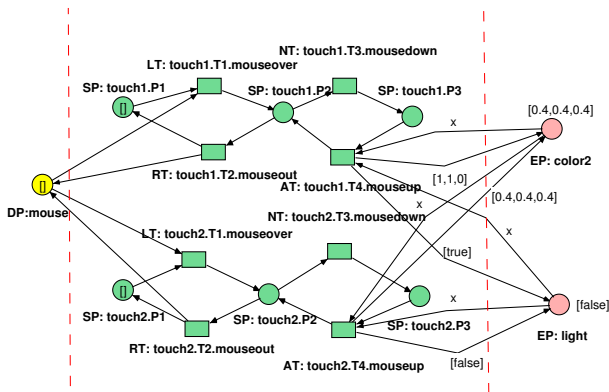


Figure 2: Interaction Petri net model for the Desk Lamp example

Each time the mouse can only access the left button or the right button, so the *dp mouse* is the input place of both *lts: touch1.T1.mouseover* and *touch2.T1.mouseover*. Only one of them can lease the mouse resource at one time. The resource can be returned by *rts: touch1.T1.mouseout* and *touch2.T1.mouseout*. The *eps* are connected to *ats: touch1.T1.mouseup* and *touch2.T1.mouseup*, because the lamp status may only be changed when the mouse is clicked above one of the buttons. No *eps* are associated with *nts, touch1.T1.mousedown* and *touch2.T1.mousedown*.

## 4 Creating Interaction Petri net from X3D Content

An X3D file describes two types of graphs: a transformation hierarchy, which describes the spatial relationship of rendered objects,

and a behavior graph, which describes the connections between fields and the flow of events through the system. The goal is to extract the behavior graph from the original X3D file and model it using the corresponding Petri net model.

The X3D behavior graph can have separate event trees. When a user interacts with the 3D world, the root node of one event tree may trigger an event. The event is sent out through the tree branches to other nodes. The nodes receiving this event will result in some changes to the state of nodes. Each node may also generate events to report state changes. That way, the event generated by the user interaction is propagated to the whole or part of the 3D world and changes the state of the system.

In X3D, sensor nodes are responsible to detect the user actions and initiates events for the system. In other words, in the event graph of a X3D world, the root node of an event tree would be a sensor node. There are five types of sensor nodes: point device sensors, environment sensors, key device sensors, load sensors and time sensors. The focus of current research is on the events that are originated from the pointing device related sensors. That includes *TouchSensor*, *CylinderSensor*, *PlaneSensor*, and *SphereSensor*. The Interaction Petri net model is generated from X3D content in four steps and the detail description is provided in [Gračanin and Ying 2005]:

- Identification of the event-triggering sensors: find all sensor nodes in the X3D file and the nodes whose state are changed by events originated from those sensors.
- Object nodes reorganization: produces a list of all sensor nodes and the affected visual nodes.
- Mapping from events to Petri Net models: the results from previous step are converted to a PNML format by mapping the event type and Petri net places, transitions and arcs according to the Event-Petri net mapping.
- Determining the layout of Petri net: calculate positions of all places, transitions and arcs.

The algorithm has some limitations. It can analyze how the event flows in the scene graph of X3D, which node will be impacted by that event and what property that node will be updated. However it needs more detailed analysis on the X3D source code to capture what value the updated property will be for the impacted nodes. Currently, the basic Petri net structure can be automatically generated while the arc inscriptions that describe the token values flowing into/out of places are not provided.

## 5 Analysis of Interaction Petri net Model

Petri nets provide rigorous analysis capability and have shown their value for assuring the reliability and correctness of concurrent systems. Behavior properties are properties that dependent on the initial marking, such as reachability, boundedness, liveness, coverability, reversibility and home state, etc. Analysis of the behavior properties can help to understand the characteristics of VE interface.

The Petri net behavior properties have implications on related interactions with the VE interface. A deadlock occurs in a VE when two interactive entities both cannot be accessed because each of them requires other to be interacted with first. A VE deadlock example detected by Petri net is described in [Gračanin and Ying 2005]. The requirement of predictability of a command in VE interface can be understood as follows: no matter what user interaction is, the resulting state of the VE is predictable. Since user interactions are

represented with transitions and the result of each transition is always explicitly expressed by token's movement, VE interface modeled by Petri net satisfies this requirement. Reinitialability for VE interface means the user can go back to the initial state of the VE. This requirement can be met if the VE Petri net is reversible or one can go back to some home state. The availability of a command in VE can be guaranteed by the reachability of Petri net. A succession of commands (a sequence of user interactions with a VE interface) and exclusion of commands (conflicts among user interactions with a VE interface) can also be modeled using Petri net.

We will apply Petri net reachability graph (formal analysis) on the described example. Suppose two users with different roles use the lamp. A "privileged" user can access both buttons to turn on and off the lamp, while a restricted user can only turn on the lamp.

From Figure 2, we can see that places *color2* and *light* always have one token in all possible markings. In other words, the statuses are reflected by token existence in other places: *mouse*, *touch1.P1*, *touch1.P2*, *touch1.P3*, *touch2.P1*, and *touch2.P3*. For the places that represent the virtual object's attributes, e.g. *color2* and *light*, the specific token value is more meaningful. The two possible values:  $([1,1,0], [\text{true}])$  or  $([0,4,0,4,0,4], [\text{false}])$ , represent that the lamp is on or off. To make it easier to analyze the IPN by applying PN reachability graph technique, we convert the IPN to an equivalent basic Petri net. We replace the places *color2* and *light* with places *lamp.on* and *lamp.off*. To connect *lamp.on* and *lamp.off* with the other parts, some temporary *NT* (*T0.help* and *T1.help*) and *SP* (*touch.ready-to-change* and *touch.flag*) are added to maintain the consistency of the model.

The final Petri net model for a privileged user is shown in Figure 3. A reachability graph for this model is shown in Figure 4.

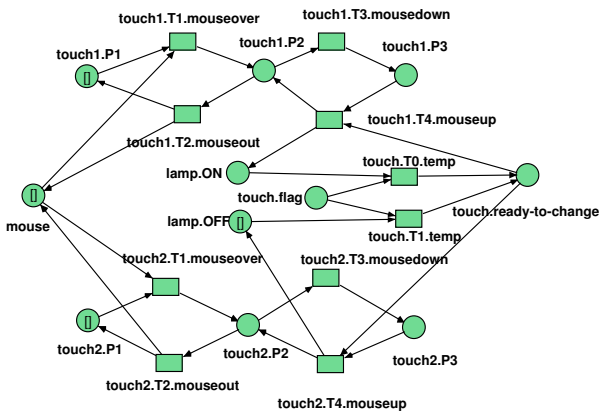
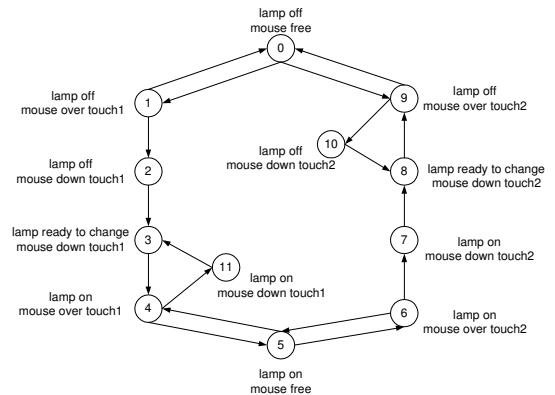


Figure 3: Petri net model of lamp example for privileged user

Figure 5 shows the petri net model for a restricted user. Figure 6 is the corresponding reachability graph.

Each node in the graph structure represents a Petri net state. Each arc leaving a node is labeled with the number of the transition that must fire to create the marking at the end of the arc. From the reachability graph, we can gain information about the properties of the lamp example.

The privileged user has 12 states and the state 0 is initial one: the lamp is turned off and the mouse is free. The user can either move the mouse over the left button (*touch1*) to reach state 1, or the right button (*touch2*) to state 9. Pressing the mouse button down, the user can reach state 2 (from state 1) or state 10 (from state 9), and further to state 3 or state 8 so the lamp is ready to change states.



REACHABILITY GRAPH :	MARKINGS:
0 -> {touch1.T1.mouseover}/1, {touch2.T1.mouseover}/9	0 : {lamp.P2.OFF} {mouse.P1} {touch1.P1} {touch2.P1}
1 -> {touch1.T2.mouseout}/0, {touch1.T3.mousedown}/2	1 : {lamp.P2.OFF} {touch1.P2} {touch2.P1}
2 -> {T1.help}/3	2 : {lamp.P2.OFF} {touch.P4.flag} {touch1.P3} {touch2.P1}
3 -> {touch1.T4.mouseup}/4	3 : {lamp.P3.ready_to_change} {touch1.P3} {touch2.P1}
4 -> {touch1.T2.mouseout}/5, {touch1.T3.mousedown}/11	4 : {lamp.P1.ON} {touch1.P2} {touch2.P1}
5 -> {touch1.T1.mouseover}/4, {touch2.T1.mouseover}/6	5 : {lamp.P1.ON} {mouse.P1} {touch1.P1} {touch2.P1}
6 -> {touch2.T2.mouseout}/5, {touch2.T3.mousedown}/7	6 : {lamp.P1.ON} {touch1.P1} {touch2.P2}
7 -> {T0.help}/8	7 : {lamp.P1.ON} {touch.P4.flag} {touch1.P1} {touch2.P3}
8 -> {touch2.T4.mouseup}/9	8 : {lamp.P3.ready_to_change} {touch1.P1} {touch2.P3}
9 -> {touch2.T2.mouseout}/0, {touch2.T3.mousedown}/10	9 : {lamp.P2.OFF} {touch1.P1} {touch2.P2}
10 -> {T1.help}/8	10 : {lamp.P2.OFF} {touch.P4.flag} {touch1.P1} {touch2.P3}
11 -> {T0.help}/3	11 : {lamp.P1.ON} {touch.P4.flag} {touch1.P3} {touch2.P1}

Figure 4: Reachability graph of lamp example for privileged user

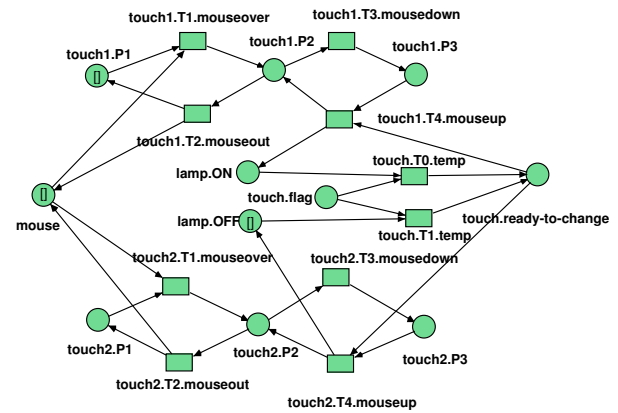


Figure 5: Petri net model of lamp example for restricted user

From state 3, state 4 will be reached, i.e. the lamp is turned on. State 4 can change to state 5, which means the mouse is moved out from the left button. State 4 can also change to state 11, which means the user presses down the button again and can go back to state 3. Similarly, from state 5, the user can move mouse over right button (*touch2*) to turn off the lamp through states 6, 7, 8, and 9 to reach the initial state 0.

For the restricted user, the Petri net model is different. The token in *touch2.P1* is removed so the restricted user can not access the right button. Although the marking distribution for each state for the restricted user is different, from the system perspective, some states can be considered the same as the Table 1 shows. Therefore, the sequence of actions for the restricted user to turn on the lamp is the same as the one for the privileged user.

We can infer some other characteristics like reachability, reinitialability, boundedness, and liveness, from the Petri net models:

**Reachability:** We can verify that all modes can be reached via

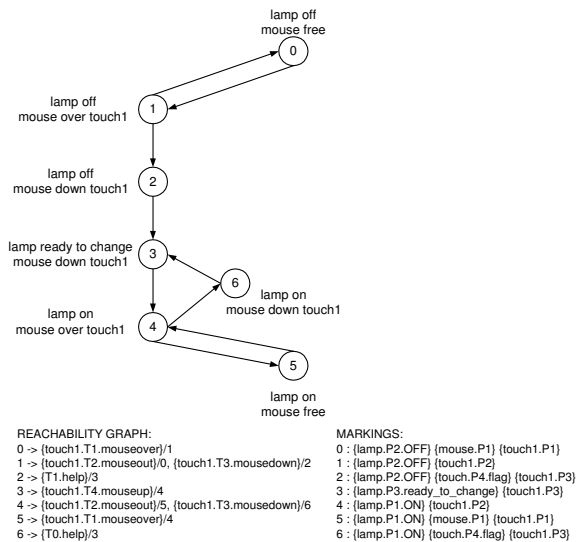


Figure 6: Reachability of lamp example for restricted user

Table 1: State comparison for privileged and restricted user

State		Privileged User	Restricted User
lamp	mouse		
off	free	0	0
off	over touch1	0	0
off	down touch1	2	2
ready to change	down touch1	3	3
on	over touch1	4	4
on	free	5	5
on	down touch1	11	6

some path and/or certain states can not be reached from a particular marking. The privileged user can reach more nodes than the restricted user because the token of *touch2.P1* is taken off for restricted user.

**Reinitialability:** From the reachability tree, we know that the privileged user can return to the initial state but the restricted user cannot (after it reaches state 2).

**Boundedness:** Maximum number of tokens for the privileged user is 4 and for the restricted user is 3. Both trees are bounded.

**Liveness:** The privileged user's Petri net is alive, i.e. no dead markings and transitions. In case of the restricted user, the corresponding Petri net is not alive since it has 4 dead transitions: *touch2.T4.mouseup*, *touch2.T3.mousedown*, *touch2.T2.mouseout* and *touch2.T1.mouseover*.

## 6 Conclusion

In this paper, we discuss the use of Petri net formalism to model and analyze user interactions extracted from X3D content. A formal definition of Interaction Petri Net is provided. A four steps procedure is used to automatically generate the basic structure of IPN in PNML from X3D content. A simple example illustrates the discovery of some characteristics of 3D interfaces, such as reachability, reinitialability, boundedness, and liveness. Currently, a subset of the X3D interactive profile is supported. Ongoing research efforts focus on providing a full support for the interactive profile and analysis of a representative set of X3D examples. The goal is

to improve the algorithms to automatically generate complete IPN based on X3D content.

## References

- BOWMAN, D., KRUIJFF, E., LAVIOLA, J., AND POUPLYREV, I. 2001. An introduction to 3D user interface design. *Presence: Teleoperators and Virtual Environments 10*, 1, 96–108.
- DACHSELT, R., AND RUKZIO, E. 2003. Behavior3d: an XML-based framework for 3D graphics behavior. In *Web3D '03: Proceeding of the eighth international conference on 3D Web technology*, ACM Press, New York, NY, USA, 101–112.
- DACHSELT, R., HINZ, M., AND MEINER, K. 2002. Contigra: an XML-based architecture for component-oriented 3D applications. In *Web3D '02: Proceeding of the seventh international conference on 3D Web technology*, ACM Press, New York, NY, USA, 155–163.
- GRAČANIN, D., AND YING, J. 2005. An approach to formal description of user interfaces based on X3D content. In *Proceedings of the Virtual Reality International Conference*.
- KEH, H. C., AND LEWIS, T. 1991. Direct-manipulation user interface modelling with high-level Petri net. In *Proceedings of the 19th annual conference on Computer Scienc*, ACM Press, 487–495.
- LI, X., AND MUGRIDGE, R. 1994. Petri net based graphical user interface specification tool. In *Software Education Conference, 1994. Proceedings.*, 50–57.
- PETERSON, J. L. 1981. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, New Jersey.
- PNML, 2000. Specification of PNML. <http://www.informatik.hu-berlin.de/top/pnml/pnml.html>. Last accessed May, 2005.
- POLYS, N. F. 2003. Stylesheet transformations for interactive visualization: towards a Web3D chemistry curricula. In *Web3D '03: Proceeding of the eighth international conference on 3D Web technology*, ACM Press, New York, NY, USA, 85–90.
- ROEHL, B., 1995. Some thoughts on behavior in VR systems. <http://ece.uwaterloo.ca/~broehl/behav.html>. Last accessed October, 2003.
- ROUFF, C. 1996. Formal specification of user interfaces. In *ACM SIGCHI Bulletin*, ACM Press, 27–33.
- SHIROGANE, J., AND FUKAZAWA, Y. 1998. Method of user-customizable GUI generation and its evaluation. In *Software Engineering Conference, 1998. Proceedings.*, 377–384.
- WALSH, A. E., AND BORGES-SVENIER, M. 2001. *Core Web3D*. Prentice Hall PTR, Upper Saddle River, NJ.
- WEB3D, 2001. Student project: Desklamp. <http://www.web3d.org/x3d/content/examples/StudentProjects/DeskLamp.x3d>. Last accessed May, 2005.
- WEB3D, 2003. Extensible 3D (X3D). ISO/IEC FDIS 19775:200x. Last accessed May, 2005.

