Fuzzy-Timing Petri Net Modeling and Simulation of a Networked Virtual Environment – NICE

Y. Zhou, T. Murata, T. DeFanti, and H. Zhang
Department of Electrical Engineering and Computer Science
University of Illinois at Chicago
Chicago, Illinois 60607-7053 USA
{yzhou1, murata, tom, hzhang1}@eecs.uic.edu

SUMMARY Despite their attractive properties, networked virtual environments (net-VEs) are notoriously difficult to design, implement and test due to the *concurrency*, *real-time* and *networking* features in these systems. The current practice for net-VE design is basically trial and error, empirical, and totally lacks formal methods. This paper proposes to apply a Petri net formal modeling technique to a net-VE - NICE (Narrative Immersive Constructionist / Collaborative Environment), predict the net-VE performance based on simulation, and improve the net-VE performance. The NICE is essentially a network of collaborative virtual reality systems called the CAVE - (Cave Automatic Virtual Environment). First, we present extended fuzzy-timing Petri net models of both CAVE and NICE. Then, by using these models and Design/CPN as the simulation tool, we have conducted various simulations to study real-time behavior, network effects and performance (latencies and jitters) of NICE. Our simulation results are consistent with experimental data.

Key words: *Petri nets, networked-virtual environments, fuzzy timing, formal modeling*

1. Introduction

A networked virtual environment (net-VE) is a network of collaborative virtual reality systems which provides multiple users with the ability to interact with each other in real time, share information, and manipulate objects in the shared environment through immersive computer graphics [1]. The first modern networked game / virtual environments can be dated as early as to *Amaze* [2] in 1984, and experimental net-VE systems have been around for decades [1]. Due to the recent rapid development of computing and networking techniques, there are diverse applications and increasing use of net-VEs, such as battlefield simulation [3], education [4], entertainment (*Doom, Atari, Nintendo*) [1], virtual conference [5], collaborative design [6], etc.

Accompanying the fast-growing net-VEs are many challenging research problems including those associated with *multimedia synchronization*, keeping *fairness* to every user while providing *heterogeneous access ports*, *bandwidth allocation* with the limited *network capacity*, keeping *consistent real-time view* to all users despite *network latency*, *user authentication*, *failure management*, *scalability*, etc. [1]. Most of these problems also exist in other kinds of distributed real-time systems such as distributed database systems. But problems in net-VEs are much more complex and difficult to solve, because they must: 1) Rely on network and contend with all the challenges of managing network resources, data loss, network failure, and concurrency; 2) Maintain smooth, real-time display; and 3) Process real-time data input from users. Users may be distributed at multiple remote hosts. Yet, users should see the virtual environments as if they existed locally. Natural and real-time interaction among users should be maintained.

Despite their attractive properties, networked virtual environments (net-VEs) are notoriously difficult to design, implement and test due to the *concurrency*, *real-time* and *networking* features in these systems, as mentioned in the above. The current practice for net-VE design is basically trial and error, empirical, and totally lacks formal methods. There are several low-level simulation programming packages and libraries have been developed ([7], [8]). However, they aim to help programming, instead of modeling and performance analysis. Although many techniques are developed for concurrent real-time systems, few approaches have been applied to the field of virtual environments. ASADAL / PROTO [9] is a tool using Visual Object Specification (VOS), Data Flow Diagrams and Statecharts, to support the development of virtual environments. However, it focuses on the modeling visual objects and interaction between visual objects. Network issues and network effects on CVEs are not the subjects there. A timed Petri net model for a stand-alone virtual environment is presented in [18]. However, no network issue exists for the stand-alone VE and the work only shows simple simulation results.

This paper proposes to apply a Petri net formal modeling technique to a net-VE - NICE (Narrative Immersive Constructionist / Collaborative Environment) ([13], [14]). NICE is essentially a network of collaborative virtual reality systems called CAVE™[1] - (CAVE Automatic Virtual Environment) ([10], [11], [12]). Applying formal modeling techniques such as the Petri net, we hope to open an avenue to give clear semantics to the configurations of net-VEs, and to apply formal validation and verification techniques. In this paper, we first present fuzzy-timing Petri net models ([15], [16]) of both CAVE and NICE. Then, using these models and Design/CPN as the simulation tool [21], we have conducted various simulations to study real-time behavior, network effects and

[1] CAVE™ is a registered trademark of the Board of Trustees of the University of Illinois

performance (latencies and jitters) of NICE. Our simulation results are consistent with experimental data [22], and shows that a formal method can provide great insight into net-VEs design and dynamic behavior, estimate system performance, evaluate network effects on net-VEs, and thus help to improve design of net-VEs.

This paper is organized as follows: Section 2 presents an introduction of CAVE and NICE. Section 3 reviews briefly fuzzy-timing Petri nets and proposes an extension of fuzzy-timing Petri nets (EFTN). EFTN models of CAVE and NICE are given in Sections 4 and 5. Section 6 discusses simulation results of EFTN models for TCP (Transmission Control Protocol) and NICE. Our conclusion and future research plan are given in Section 7.

2. About CAVE & NICE

The CAVE (CAVE Automatic Virtual Environment) ([10], [11], [12]) is a virtual reality environment designed and implemented at the Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago (UIC). The CAVE is a surround-screen, projection-based virtual reality system. The actual environment is a 10x10x10 foot cube, where images are rear-projected in stereo on 3 walls (front wall, left wall, and right wall), and down-projected onto the floor. (The floor can be considered as floor wall. So there are 4 walls total.) The 4 walls each display computer generated stereo images of the virtual world in real-time based on the position and orientation of the user's head and hand in the CAVE. The viewer wears LCD shutter glasses to separate the stereo images. The viewer's head and hand position and orientation are tracked through sensors on the shutter glasses and on the 'wand' (the CAVE input device). The viewer can grab and move objects in the virtual world with the wand.

The Narrative Immersive Constructionist / Collaborative Environments (NICE) project at EVL of UIC is a collaborative learning environment: a virtual garden, where children can do gardening and learn cooperatively. In NICE, children located in distributed virtual environments (e.g., CAVEs), can take care of a virtual garden together in the center of a virtual island. The children, represented by avatars, collaboratively plant, grow, and pick vegetables and flowers. They make sure that the plants have sufficient water, sunlight, and space to grow, and they keep hungry animals away from sneaking into the garden and eating the plants.

We will give a more detailed description of the configurations of the CAVE and NICE in Sections 4 and 5.

3. Fuzzy-Timing Petri Nets and Extended Fuzzy-Timing Petri Nets

To deal with temporal uncertainties in real-time systems, Murata [15] has proposed Fuzzy-Timing High-Level Petri Nets (FTHNs), which employ fuzzy set theory to express uncertain or subjective timing information. The main features of the FTHN are the following four fuzzy set theoretic functions of time: *fuzzy timestamp, fuzzy enabling time, fuzzy occurrence time* and *fuzzy delay*. A *fuzzy timestamp* $\pi(\tau)$ is a *fuzzy time function* or *possibility distribution* giving the numerical estimate of the possibility that a particular token arrives at time τ in a particular place. In a FTHN, each arc (t, p) from transition t to place p is associated with fuzzy delay $d_{tp}(\tau)$. For simplicity, trapezoidal or triangular *possibility distributions* specified by the 4-tuple $(\pi_1, \pi_2, \pi_3, \pi_4)$ as shown in Fig. 1, are used to represent fuzzy time functions. The formal definition of FTHNs and the method to compute and update *fuzzy enabling time* and *fuzzy occurrence time* when a transition firing occurs, are given in [15]. FTHNs provide additional information on partial ordered events in terms of their degrees of possibilities, instead of transforming them into a total ordering. The computations involved in FTHNs are basically repeated additions and comparisons of real numbers and are necessary only for certain finite firing sequences, and need not generate the entire state space. Thus these computations can be done very fast and thus FTHNs are suited for estimating the performance of time-critical systems.

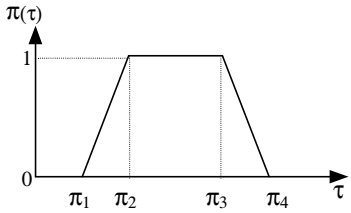


Fig. 1 Trapezoidal *possibility distribution*

A Fuzzy-timing Petri Net (FTN) [16] model is an unfolded version of the fuzzy-timing high-level Petri net (FTHN). We extend FTN by integrating FTN with Merlin's time Petri net [17]. An Extended Fuzzy-Timing Petri Net (EFTN) model is a FTN with the default value of $d_{tp}(\tau)$ being (0,0,0,0), and with an additional function CT: $T \rightarrow Q^+ \times Q^+ \times (Q^+ \cup \infty)$ which is a mapping from the transition set T to firing intervals with *possibility p*: i.e., each transition is associated with a firing interval $p[\alpha, \beta]$, where the default interval is 1[0, 0] (a transition definitely fires as soon as it is enabled) (CT is taken from Merlin's time Petri net [17]). If a transition t is enabled at time instant τ, t may not fire before time instant $\tau + \alpha$, and t must fire before or at time instant $\tau + \beta$. *Possibility* $p \in [0,1]$. p is 1 if transition t is not in conflict with any other transition. p can be less than 1 when we want to assign different chances to transitions in structural conflict. For example, if transition t1 and transition t2 are in structural conflict, t1 fires with 99% chance and t2 fires with 1% chance, we assign $p_1 = 0.99$ and $p_2 = 0.01$. A transition firing itself is an atomic event and takes zero time. Our purpose of attaching firing intervals (with possibilities) to transitions is to give a firing possibility and priority among transitions in conflict. This is very useful for modeling real-time systems.

In EFTN, assume that transition t is enabled by n

tokens, the *fuzzy enabling time* $e_t(\tau)$ of transition t is computed by $e_t(\tau) = latest\{\pi_i(\tau),\ i = 1, 2, ..., n\}$, where *latest* is the operator that constructs the "latest-arrival/lowest-possibility distribution" from n distributions [15], and $\pi_i(\tau)$ is the *fuzzy time distribution* that the enabling token i arrives in the input place P_i of transition t. When there are m transitions in conflict enabled with their *fuzzy enabling times*, $e_i(\tau)$, i = 1, 2, ..., t, ..., m, and $CT(t_i) = p_i[\alpha_i, \beta_i]$, we compute the *fuzzy occurrence time* $o_t(\tau)$ of transition t whose *fuzzy enabling time* $e_t(\tau)$, as follows: $o_t(\tau) = min\{e_t(\tau) \oplus p_t(\alpha_t, \alpha_t, \beta_t, \beta_t), earliest\{e_i(\tau) \oplus p_i\ (\alpha_i, \alpha_i, \beta_i, \beta_i),\ i = 1, 2, ..., t, ..., m\}\}$, where *earliest* is the operator that constructs the "earliest-arrival/highest-possibility distribution" from m distributions, *min* is the intersection of distributions, and $\oplus$ is the extended addition defined as in the computation of the *fuzzy time distribution* that a token arrives at transition t's output place p: $\pi_{tp}(\tau) = o_t(\tau) \oplus d_{tp}(\tau) = h_1(o_1,o_2,o_3,o_4) \oplus h_2(d_1,d_2,d_3,d_4) = min\ (h_1, h_2)\ (o_1 + d_1,\ o_2 + d_2,\ o_3 + d_3,\ o_4 + d_4)$ ([15]) ($d_{tp}(\tau)$ is the *fuzzy delay* associated with the arc (t,p)).

4. EFTN models for CAVE

The CAVE has the following three main subsystems [10]:

- *Tracker subsystem*: which obtains 6-dimensional data about the position of the viewer's head and hand.
- *Main subsystem*: which creates images to be displayed on the walls of the CAVE. There are four graphic pipelines working concurrently. Each is used to render the image on one wall. The CAVE implementation uses double buffering between the main subsystem and the display subsystem. While the main subsystem is writing into one buffer, the display subsystem reads from the other buffer. The buffer swapping is synchronized by a monitor signal at 46 HZ frequency. Once images for all 4 walls have been rendered, a buffer swapping takes place at the leading edge of the next coming monitor signal if the display subsystem is also ready to swap buffers.
- *Image display subsystem*: which draws the images on the four walls. When the drawings of 4 images are all finished, the display subsystem is ready to swap buffers.

Fig. 2 shows our EFTN model for the CAVE. In order to analyze EFTN models, we illustrate two reduction rules ([19], [20]) for EFTNs. Our reduction rules can reduce the size of EFTN models, while preserving safeness, deadlock and timing properties of EFTN models. Applying the two reduction rules shown in Fig. 3 to our EFTN model for the CAVE in Fig. 2, results in a reduced EFTN model, as is shown in Fig. 4. Fuzzy delays in the EFTN models in Figs. 2 and 4 are specified as follows. In Fig. 2, the delay for Head and Wand data arriving at the Tracker, $D_{headwand\_to\_tracker}(\tau)$ = (50,50,50,50) ms. The delay for converting and transferring tracker data to the Unix workstation, $D_{convert}(\tau)$ = (10,10,10,10) ms. $D_{render\_front}(\tau)$, $D_{render\_left}(\tau)$, $D_{render\_right}(\tau)$, and $D_{render\_floor}(\tau)$ are the fuzzy delays of rendering images for front wall, left wall, right wall, and the floor. We assume $D_{render\_front}(\tau)$ = (25.0,37.4,50.0,62.4) ms, and $D_{render\_left}(\tau)$ = $D_{render\_right}(\tau)$ = $D_{render\_floor}(\tau)$ = (10,20,30,35) ms, since the image on the front wall is usually more complicated than the ones on other walls. The delay for drawing images on each wall is $D_{draw\_front}(\tau) = D_{draw\_left}(\tau) = D_{draw\_right}(\tau) = D_{draw\_floor}(\tau)$ = (2,2,2,2) ms, and ε3 = ε4 = 0.2 ms (a short time period that a monitor signal lasts). In Fig. 4, we have *latest* $\{D_{render\_front}(\tau), D_{render\_left}(\tau), D_{render\_right}(\tau), D_{render\_floor}(\tau)\}$ = *latest* {(25.0,37.4,50.0,62.4), (10,20,30,35), (10,20,30,35), (10,20,30,35)} = (25.0,37.4,50.0,62.4) ms, and *latest*($D_{draw\_front}(\tau)$, $D_{draw\_left}(\tau)$, $D_{draw\_right}(\tau)$, $D_{draw\_floor}(\tau)$) = (2,2,2,2) ms.

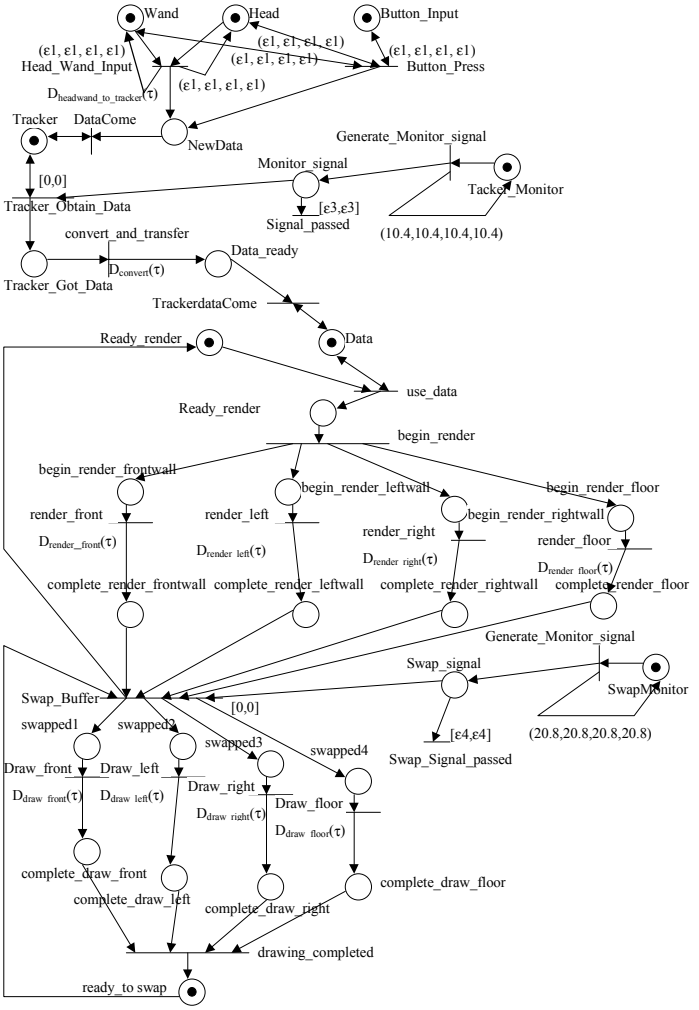


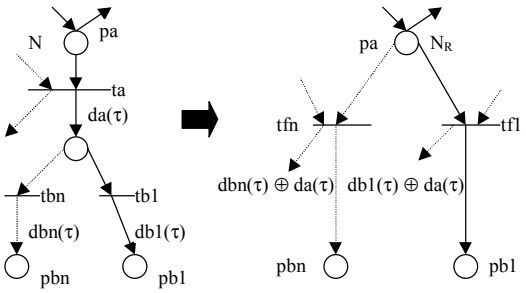
Fig. 2 An EFTN model for the CAVE

5. EFTN models for NICE

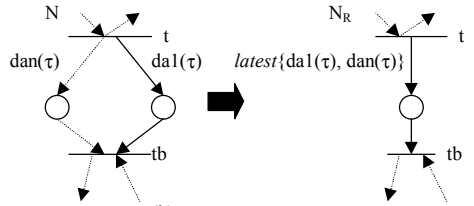
The main distributed components of NICE consist of a garden simulation server, an avatar repeater for avatar state information, and NICE clients ([13], [14]). A NICE client uses an unreliable (but faster) protocol UDP (User Datagram Protocol) to send avatar information (local tracker data) to the avatar repeater and a reliable (but slower) socket connection (TCP protocol) to send the local avatar's world-changing messages to the server. The avatar repeater broadcasts avatar state information by using

UDP. The NICE server supports the garden simulation, updates the world (garden) once receiving an avatar's world-changing message, and broadcasts the new world state information to all clients by using TCP.

The Information Request Broker (IRB) is the core of all client and server applications in the NICE. An IRB is an autonomous repository of persistent data that is accessible by a variety of networking interfaces. A key is a handle to a storage location in an IRB's database. Keys are uniquely identified across all IRBs. A local key can initiate and accept multiple linkages to and from other remote IRBs. Any modifications that are made to one key will automatically be propagated to all the other linked keys ([13], [14]).



(a) Post-fusion (post-fuse transition ta with tb1, …, tbn)



(b) Parallel fusion of places

Fig. 3 Illustration of two reduction rules: Post-fusion of transitions, and Parallel fusion of places.

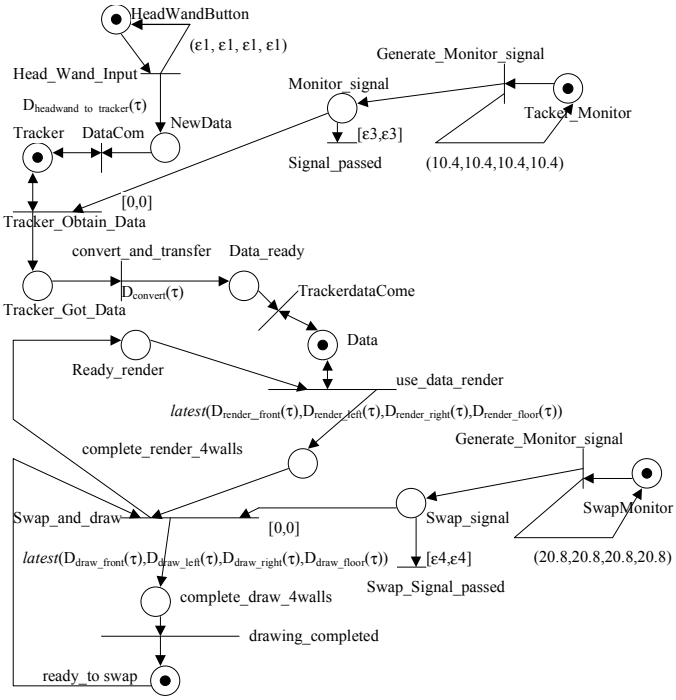


Fig. 4 Reduced EFTN model of the CAVE

The garden server is an IRB with two main keys: an incoming message key and an outgoing message key. If the local avatar has any action changing the garden (e.g., "plant a tree"), the local VE will send a message from the local OUT Key to the server's IN key by using TCP. Then the garden server updates the world state and sends the new world state information to each client's IN key via the server's OUT keys by using TCP. The garden world evolves itself as the plant grows, the weather changes, and animals appear. So the server sends each client the new world state information by using TCP once it updates.

The avatar repeater has a key for each client to hold its avatar-state information. When client1 updates the local screen (swapping-buffer happens), avatar1's state information will be sent from client1 to the avatar1-state key on the repeater by using UDP. Another client (e.g. avatar2) will get the state of avatar1 by subscribing to the avatar1-state key on the repeater.

In Fig. 5, we give the EFTN model for the garden server, avatar repeater and communication interface of two existing NICE clients communicating with each other, the repeater and the garden server. Each client sends local avatar's tracker information to the repeater by using UDP and the repeater broadcasts it to all other clients also by using UDP. Using UDP, the sender just sends out the Protocol Data Unit (PDU) and never retransmits. So we use a transition UDP with fuzzy delay $D_{UDP}(\tau)$ = (50,100,150,200) ms (which is based on statistics data collected from network monitoring [22]) to represent UDP channel in Fig. 5, and we assume the data loss rate of UDP is 1%.

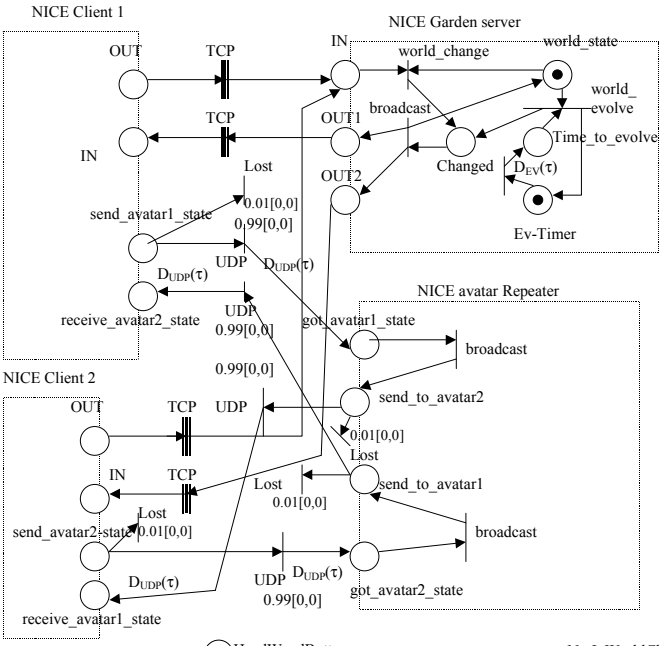


Fig. 5 An EFTN model for 2 existing NICE clients communicating with each other, the repeater and the server, where the delay of UDP channel is $D_{UDP}(\tau)$ = (50,100,150,200) ms, the world evolves in the interval $D_{EV}(\tau)$ = (10,60,180,300) sec.

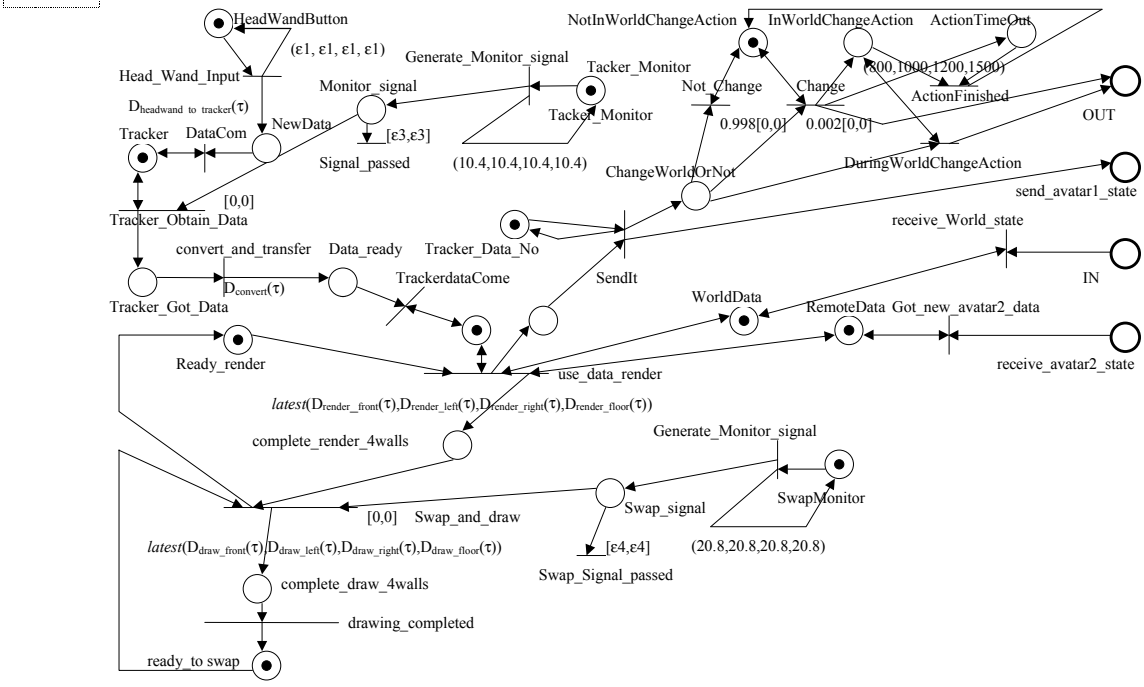


Fig. 6 The EFTN model for a CAVE in NICE as a distributed component

Each TCP transition in Fig. 5 is an abstract of a subnet for TCP protocol. TCP is a reliable and ordered transport layer protocol. One PDU's loss will delay all subsequent PDUs. No subsequent PDU can be delivered to the application layer until that PDU is successfully received. We have built our EFTN model for TCP and tested the behavior of TCP model by simulation. Our simulation results are consistent with network monitoring results [22]. Because the space is limited, we do not include our EFTN model for TCP in this paper. We only show our simulation results and discuss the behavior of TCP protocol and its effects on NICE in Section 6.2 and 6.3.

In NICE, a local VE (e.g., CAVE) will need local avatar-state information, remote avatars state information, and world-state information to render images. The EFTN model for a CAVE in NICE as a distributed component is shown in Fig. 6. When an avatar wants to change the world, it usually takes about 1 second (*possibility distribution* (800,1000,1200,1500) ms) to complete the action. During an avatar's world-changing action, all of the tracker data used for updating the local screen will be sent to the server by using TCP. An avatar may change the world 2~3 times per minute and the local screen may be updated 16~24 times per second (16 times/sec if the image rendering delay ≤ 41.6 ms each time, 24 times/sec if the image

rendering delay is in the interval (41.6, 62.4) ms each time). So, we assume 0.2% of the tracker data used for updating local screen may indicate that local avatar wants to change the garden. (In Fig. 6, when the avatar is not already in an world-changing action, place ChangeWorldOrNot has two output transitions, the *possibility* of firing transition Change is 0.002, and the *possibility* of firing Not_Change is 0.998.) After we put the communication interface and internal structure of distributed CAVEs together, we have studied the network effects on NICE and the dynamic performance of NICE by using a simulation tool within Design/CPN [21]. These simulation results are given next in Section 6.

6 Simulation Results

We have simulated fuzzy delays of our EFTN models by defining a function to generate single values in trapezoidal possibility distributions within the Design/CPN tool [21]. Given a fuzzy delay $D(\tau) = (a,b,c,d)$, we define a function *FUZZY*(a,b,c,d) to generate a single delay value in trapezoidal possibility distribution (a,b,c,d) as the representation of the *fuzzy delay,* whenever the fuzzy delay $D(\tau)$ is encountered in simulation. In function *FUZZY*(a,b,c,d),

1) A random value *atime* in the interval [a,d] is generated.
2) If *atime* is in the interval [b,c], *atime* will be picked up as the delay value since the *possibility* is 1 in the interval [b,c].
3) If *atime* is in the interval [a,b) or (c,d], we generate a random value *V* in (0,1) and compute the possibility D(*atime*) in the trapezoidal distribution (a,b,c,d). `atime` will be picked up as the delay value only if D(`atime`) ≥ *V*.
4) Repeat the above procedure until a delay value can be picked up.

The following are the results of the simulations. Section 6.1 is about strategies for utilizing remote avatar's state information received from UDP channel and the simulation results for remote avatar's corresponding display behavior on local screen. Section 6.2 discusses simulation results of our EFTN model for TCP protocol. And Section 6.3 shows the TCP protocol's effects on message response time between the NICE garden server and clients.

6.1 Remote Avatar's Display Behavior on Local Screen

Unreliable protocols (e.g. UDPs) are used for the (faster) transmission of avatar state information (remote tracker data). That is because: 1) The loss of one tracker data is usually followed shortly afterwards by newer ones; and 2) Unreliable protocols have a lower latency and utilize lower bandwidth than reliable protocols. However, UDP protocol is unordered. Using unordered remote tracker data will make the remote avatar jump back and forth on the local screen. Fig. 7 shows the time of avatar2's original movement and the time that movement is displayed on the NICE client1's screen in the original design. We see the remote avatar's display may jump back and forth. To avoid the jumping back behavior, the NICE currently uses a filter to accept remote avatar data in increasing order. Fig. 8 shows the display behavior using the filter. Now the jumping back phenomenon is eliminated. However, one early arriving remote avatar tracker data will make the filter discard all remote tracker data that are sent before, but received later than that early arriving data. From Fig. 8, we see that the display of the remote avatar is not smooth yet.

To display the remote avatar's movement more smoothly, we suggest using a buffer to store the incoming remote avatar state information sent after the last one used for local display, and using the remote avatar's state information in smooth gap. Fig. 9 shows the improved strategy using a buffer for remote avatar tracker data. When a new remote tracker data comes and its sequence number is greater than the last one's used for rendering images, it is inserted into the buffer and the list of sequence numbers of the remote tracker data is kept in ascending order in the buffer. When it is time to render new images, pick up the remote tracker data in the middle of the list from the buffer. Fig. 10 shows that the display of the remote avatar is much smoother using the improved strategy than using the filter.

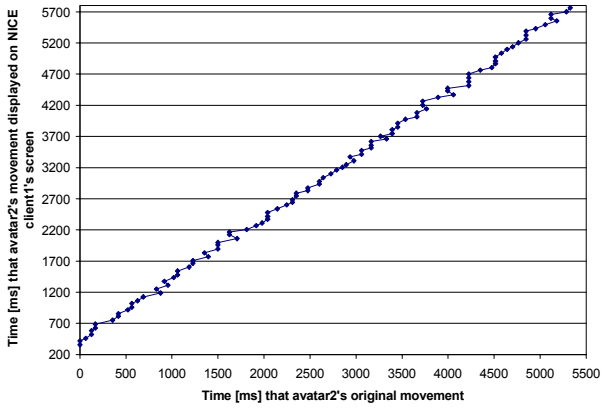


Fig. 7 The simulated display behavior of a remote avatar on the local screen without using the filter

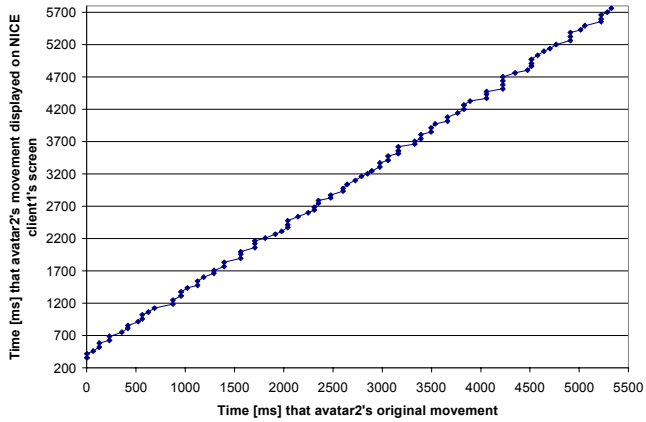


Fig. 8 The simulated display behavior of a remote avatar on the local screen with using the filter

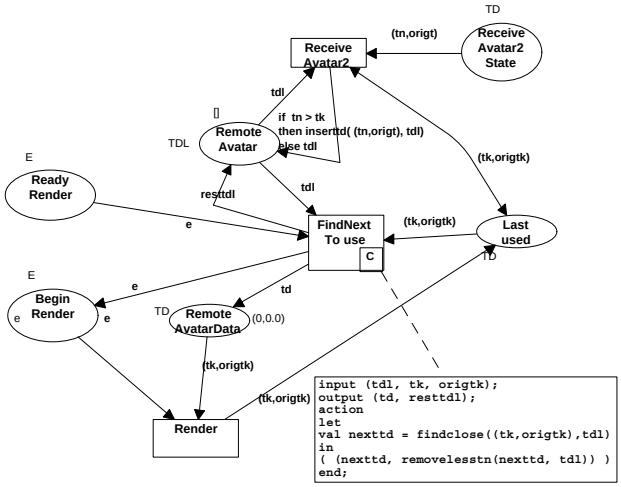


Fig. 9 Improved strategy: use a buffer for remote avatar state information

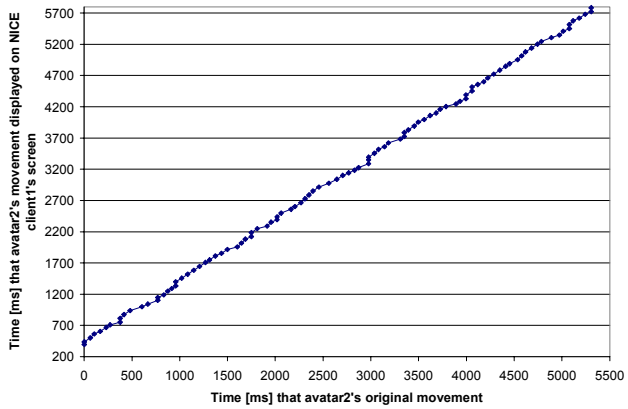


Fig. 10 The simulated display behavior of a remote avatar on the local screen with using buffer

6.2 Test of TCP protocol

We tested our model for TCP protocol by giving one data unit to TCP sender for transmission every 50 ms and 100 ms respectively, and recording the delay from the time the data unit is passed to the TCP sender to the time the TCP receiver delivers it to the application. Our simulation result is consistent with the experimental results in [22], which is obtained by monitoring the network delay on the Internet.

- *Case 1*: give the TCP sender a data unit every 50 ms:

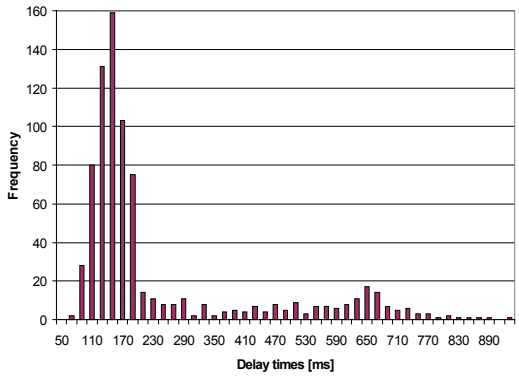


Fig.11 (a) Delay distribution for data units coming every 50 ms

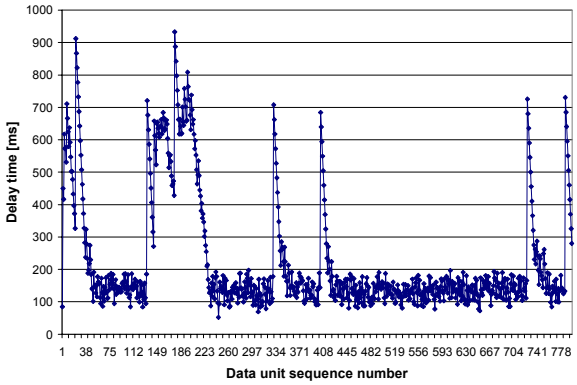


Fig.11(b) History chart for each data unit coming every 50 ms

As shown in Fig.11, it is obvious that the TCP has a slow start and one data unit's long delay will delay all subsequent data units after it starts. TCP's reliable and ordered behavior greatly increases the average network latency and jitter.

- *Case 2*: give the TCP sender a data unit every 100 ms:

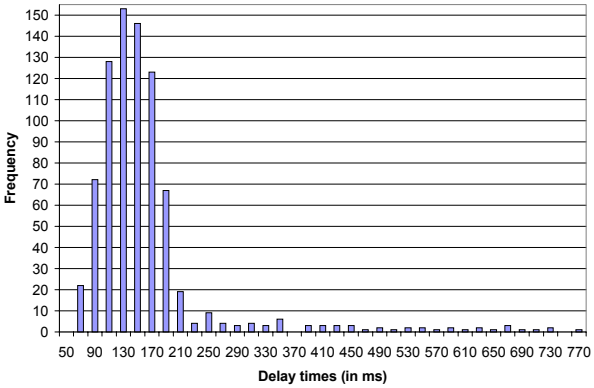


Fig.12(a) Delay distribution for data units coming every 100 ms

Fig. 12(a) shows that decreasing the traffic to one data unit every 100 ms makes the delay distribution very similar to UDP's delay distribution (50,100,150,200)

ms. However, Fig. 12(b) shows that one data unit's long delay (because of loss and retransmit), still greatly delays the subsequent data units. Also the slow start still exists.

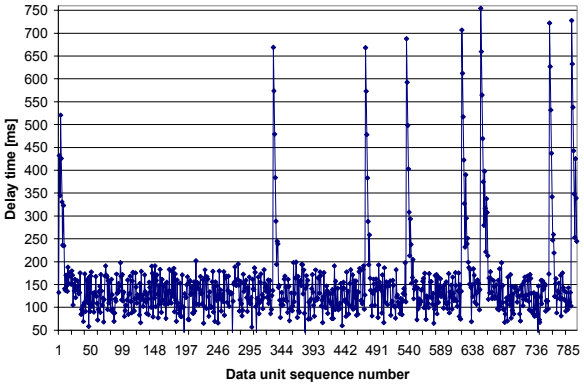


Fig.12(b) History chart for each data unit coming every 100 ms

6.3 Response Time via TCP channel

Once an avatar has a world-changing action, a world-changing message is sent from the NICE client to the garden server via TCP channel. Once the garden server updates the world state according to the client's world_changing message, the changed world state is sent from garden server to all clients via TCP channels. We refer the response time as the time duration from the time that a world-changing message is sent by a NICE client to the time that the updated world state is received by that client.

Here we show the simulation results of the response time via TCP channels for a NICE project with 5 clients. Fig. 13 shows the history chart recorded at client1's site for the NICE with 5 clients. We can see the effects of TCP's slow start. And it is obvious that the response times for all avatars' world-changing activities have the same trend at client1's site, if it happens that all avatars want to change the world at the same moment (e.g., around 78000 ms in Fig. 13.) The loss of one data on the TCP channel from the server's OUT key to a client's IN key will not only cause a long delay for client1 receiving response for one avatar's world-changing activity, but also postpone client1 receiving response for all avatar's world-changing activities. The TCP channel from the server's OUT key to a client's IN key may get congested when the number of clients is increased and all avatars happen to try changing the world at the same time. This situation becomes a bottleneck.

7. Conclusion and Future Work

As mentioned in the introduction, networked virtual environments (net-VEs) are not novel, but at present, there is no formal method available for designing, implementing and testing net-VEs. This paper proposes to apply Petri net formal modeling and simulation of NICE - a net-VE developed at EVL of UIC. We have shown that EFTNs are powerful enough to model complex systems such as CAVE and NICE, and simulate the uncertain temporal behaviors of a net-VE system. Using our EFTN models, we have simulated the remote avatar's display behavior on local screen and proposed an improved strategy to make the display smoother. The characteristics and bottleneck of the response time via the TCP channel was easily captured in the simulation results Our simulation shows that the TCP is reliable but greatly increases the average network latency and jitters. Thus, it is desirable to design a new transport layer protocol which can transmit shared state information with less latency and jitter than the TCP. We plan to propose new protocols, and model, simulate and analyze them in our future papers.

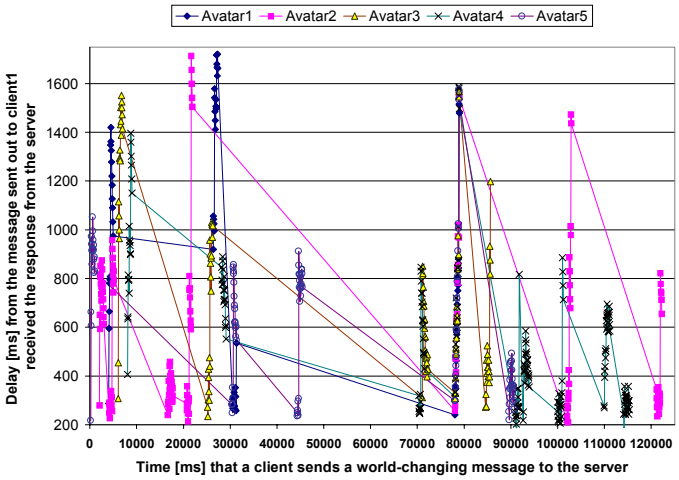


Fig. 13 History chart of the response time for the NICE with 5 clients

ACKNOWLEDGEMENTS

We sincerely thank Robert Kenyon for explaining net-VEs, Jason Leigh and David Pape for clarifying net-VE implementation details, Andrew Johnson and Dan Sandin for discussing problems that exist in net-VEs, and all members of the EVL at UIC for providing valuable comments on our work.

References

[1] S. Singhal, M. Zyda, Networked Virtual Environments: Design and Implementation, Addison-Wesley, New York, July 1999

[2] E. J. Berglund, and D.R. Cheriton, "Amaze: A multiplayer computer game," IEEE Software vol.2, no.1, pp.30-39, May 1985.

[3] D. Miller, J.A. Thorpe, "SIMNET: The advent of simulator networking," Proceedings of the IEEE vol.83, no.8, pp.1114-1123, August 1995.

[4] A. Johnson, T. Moher, J. Leigh, Y. Lin, "QuickWorlds: Teacher driven VR worlds in an Elementary School Curriculum," Proceedings of SIGGRAPH 2000 - Educators Program, New Orleans LA, July 23-28, 2000.

[5] C. Carlsson, and O. Hagsand, "DIVE- A multi-user virtual reality system," Proceedings of the IEEE Virtual Reality Annual International Symposium, pp. 394-400, Seattle, September 1993.

[6] G. Singh, L. Serra, W. Prg, et al, "BrickNet: A software toolkit for network-based virtual environment," Presence: Teleoperators and Virtual Environments, vol.3, no.1, pp.19-34, Winter 1993.

[7] C. Shaw, J. Liang, M. Green, and Y. Sun, "The Decoupled

Simulation Model for Virtual Reality Systems," Human Factors in Computing Systems CHI'92 Conference proceedings, pp. 321-328, Monterey, California, May 1992.
[8] P. Strauss, and R. Carey, "An Object-Oriented 3D Graphic Toolkit," Proceedings of the ACM Computer Graphics Conference (SIGGRAPH'92), pp. 341-349, 1992.
[9] G. J. Kim, K. C. Kang, H. Kim and J. Lee, "Software engineering of virtual worlds," Proceedings of the ACM Symposium on Virtual reality software and technology 1998 , pp. 131 – 138, 1998.
[10] D. Pape, "CAVE user's guide," Electronic Visualization Laboratory, University of Illinois at Chicago, Dec. 1996.
[11] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti, "Virtual Reality: The Design and Implementation of the CAVE," Proceedings of SIGGRAPH '93 Computer Graphics Conference, ACM SIGGRAPH, pp. 135-142, August 1993.
[12] T.A.DeFanti, D.J.Sandin, and C. Cruz-Neira, "A `Room' with a `View'," IEEE Spectrum, pp.30-33, October 1993.
[13] M. Roussos, A. Johnson, T. Moher, J. Leigh, C. Vasilakis, C. Barnes, "Learning and Building Together in an Immersive Virtual World," Presence, vol. 8, no. 3, June, 1999, pp.247-263.
[14] A. Johnson, M. Roussos, J. Leigh, C. Barnes, C. Vasilakis, T. Moher, "The NICE Project: Learning Together in a Virtual World," Proceedings of VRAIS '98, Atlanta, Georgia, pp 176-183, Mar 14-18,1998.
[15] T. Murata, "Temporal Uncertainty and Fuzzy-Timing High-Level Petri Nets," Application and Theory of Petri Nets, Lecture Notes in Computer Science, Vol. 1091, pp. 11-28, Springer-Verlag, New York, June 1996.
[16] Y. Zhou and T. Murata, "Petri Net Model with Fuzzy-Timing and Fuzzy-Metric," Special Issue on Fuzzy Petri Nets, International Journal of Intelligent Systems, Vol.14, No.8, August 1999, pp. 719-746.
[17] P. Merlin, A study of the Recoverability of Computer Systems, Ph.D. thesis, Computer Science Dept., University of California, Irvine, 1974.
[18] R. Mascarenhas, D. Karumuri, U. Buy, and R. Kenyon, Modeling and analysis of a virtual reality system with time Petri nets, Procs. 19th Int. Conf. on Software Engineering, pp. 33-42, April 1998, Kyoto, Japan.
[19] E. Juan, J. P. Tsai, T. Murata, and Y. Zhou, "Reduction Methods for Real-Time Systems Using Delay Time Petri Nets," to appear in the IEEE transactions on Software Engineering.
[20] Y. Zhou, T. Murata, and J. Tsai, "Reduction Methods for Real-Time Systems Using Fuzzy Timing Petri Nets," Technical report, EECS Dept., University of Ilinois, Chicago, 2000.
[21] K.Jensen, and Design/CPN group, Design/CPN Online, Department of Computer Science, University of Aarhus, Denmark. Online: http://www.daimi.au.dk/designCPN/.
[22] K. Park, and R. Kenyon, "Effects of Network Characteristics on Human Performance in a Collaborative Virtual Environment," Proceedings of IEEE VR `99 , Houston TX, March 13-17, 1999.