Design, Analysis, and Implementation of a Telemedicine Remote Consultation and Diagnosis Session Playback Using Discrete Event System Specification

Pinkesh J. Shah, *Student Member, IEEE,* Ralph Martinez, *Member, IEEE,* and Bernard P. Zeigler, *Fellow, IEEE*

***Abstract*— Telemedicine remote consultation and diagnosis (RCD) software is a complex and distributed system. RCD allows physicians to collaborate on radiology or pathology cases from distributed geographic locations. It is very important to simplify design, construction, and maintenance of such a system. Currently, object-oriented design methodology is used to design and develop a software system in a modular fashion. Object-oriented software is made of various objects that work together. From the design of the software system, we get information about object methods and inheritance. We also get information about which objects are contained in a particular object and which objects are used by another object. One important element that the traditional object-oriented design misses is time. We propose the use of discrete event system specification (DEVS) in the design and analysis of a software system, such as RCD. With DEVS, coupling between objects can be specified explicitly and an object behavior can be shown in time. We introduce DEVS, show the time-line analysis of Remote Consultation and Diagnosis session playback using DEVS, and then describe its implementation.**

***Index Terms*— Discrete event systems, distributed information systems, medical information systems, multimedia systems, object-oriented methods, picture archiving and communication systems, software design/development, telemedicine**

I. Introduction

ADVANCES IN computer and communication technology have made it possible to create a distributed telemedicine system for remote consultation and diagnosis (RCD) [2], [6]. This system lets physicians from various geographic locations collaborate on a particular radiology or pathology case. In a consultation session, multiple media data, such as image, text, annotation, and audio are exchanged. A consultation session can be digitally recorded and played back at a later time for education. The RCD system has been implemented in the JAVA language [1]. The contemporary approach to developing such a system is to use object-oriented analysis and design methodology. Telemedicine RCD was designed and developed using an object-oriented approach to system design and programming. An object from an object-oriented design is usually implemented as a class in software, so we will use the word object and class interchangeably. In the documentation for the RCD, various class diagrams are provided. From these class diagrams, we can get information about which class is inherited from which class, what classes are used by a particular class and what classes are contained in a particular class. There is no time base involved in the description of the objects. By looking at the design in the object-oriented paradigm we have the following problems.

- One object is used by another, but in what way it is used is not clear.
- By looking at the diagram, one cannot get any information about the flow of events (method invocations) in time.
- No explicit coupling in terms of recognized input and output is specified between objects.
- Hierarchy of classes is specified, but this hierarchy involves only information about specialization. It does not have elements of decomposition and coupling in it.

An object model in DEVS formalism operates on a time basis [10]. Each object has a recognized set of input ports and output ports through which it communicates with the external environment. A system model can be constructed by coupling its component objects in a hierarchical fashion. Time-line analysis of an object model shows how an object behaves in a system with respect to time. By this we mean that it is possible to see the exact behavior of an object using a graph. With the time-line analysis information, we know exactly how an object will react to a discrete event.

Recorded RCD sessions can be played back using a JAVA-enabled Web browser or using stand-alone RCD software. In this paper, we describe the browser version of the RCD session playback. We use time base in the analysis of the objects that are developed for the RCD playback. Various events take place and multimedia data is transferred across the network during the playback, so a time-line analysis can give a lot more information than otherwise available. In this paper, first we describe the telemedicine RCD software and then we give a brief introduction to DEVS. In Section IV, we show time-line analysis of the objects involved in the RCD session playback. Section V is devoted to the description of the implementation of the playback.

Manuscript received August 11, 1997; revised October 15, 1997.

The authors are with Electrical and Computer Engineering Department, University of Arizona, Tucson, AZ 85721 USA (e-mail: pinkesh@ece.arizona.edu; martinez@ece.arizona.edu; zeigler@ece.arizona.edu).

Publisher Item Identifier S 1089-7771(97)09248-0.

II. Remote Consultation and Diagnosis Software

The remote consultation and diagnosis system [1] is a collaborative telemedicine system for remote radiology and pathology consultations. It allows physicians to access patient information from distributed databases. Patient information can be history, textual diagnosis, pathology images, radiology images, and voice diagnosis. Physicians can do diagnosis alone or can consult with other physicians by joining in a multiparty consultation session.

When multiple physicians are in a session, they all see the same patient information. All the images on the screen and its layout are the same. Each physician has the ability to annotate images using rectangle, circle, or free-hand annotations. When an RCD session is going on, multimedia data consisting of text, image, and annotation is exchanged to all the physicians' workstations in real time. RCD is designed using object-oriented design and analysis methodology and is implemented in JAVA language. Playback of a RCD session is designed using DEVS, which is the topic of this paper. RCD uses tool kits for GUI, threads, I/O, image processing, and networking, which are a part of the JAVA development kit [15].

RCD is a part of the Global Picture Archiving and Communication System (GPACS) [2], [6]. Physicians collaborating in a session can be located anywhere using heterogeneous machines. Communication among different places is achieved using the Internet. To provide this kind of features, RCD has six major components.

1) Client RCD used by physicians.
2) Communication server to support multiparty group RCD consultation session.
3) Web Server to provide access to RCD from a Web browser and to transport patient data.
4) Database Server to do management of patient data at each site of the global network.
5) Record a multimedia RCD session.
6) Playback of recorded RCD session using a Web browser or in stand-alone mode.

Playback of a previously recorded RCD session is an important functionality of the RCD system. This functionality allows digital recording of all the annotation commands and audio during a consultation session. Later on, these recordings can be retrieved and played in the RCD software or in a Web browser. Various objects collaborate to make the playback happen. Specially with real-time multimedia data, timing is extremely important. RCD is a dynamic system, where various discrete events occur at different times, and the RCD system responds to those events. Some events, such as playback of audio and annotation commands, occur at a fixed time interval; other event, such as user interaction can occur at any time.

III. Discrete Event System Specification (DEVS)

In DEVS, an object can be specified mathematically; it is called a system. A system has time base, inputs, states, outputs, and functions to determine output/state [9]. An object model that cannot be decomposed any further is called an atomic model. Input in a discrete event system can occur at an arbitrary time. An event is a change in a variable value that occurs instantaneously. The DEVS formalism defines how to generate new values for variables using the functions specified in the model and at what time those values will take effect. External events appear at the input port of an object model. The model description should be able to respond to the occurrence of an external event at its input port. Internal events arise within the model, they change the model's state and may create an output. If an output is generated, it will appear as an event at the output ports of the model which is then transmitted to the other model components through the coupling definition.

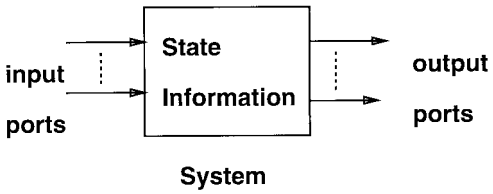

Fig. 1. Abstract atomic model.

An atomic model in DEVS formalism is defined by the following structure:

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, t_a \rangle.$$

Each of the above elements are defined as follows:

M atomic model;
X set of external input event types;
S set of sequential states for a model
Y external event types generated as output;
δ_{int} internal transition function dictating state transitions, due to internal events inside the model;
δ_{ext} external transition function dictating state transitions, due to external input events
λ output function generating external events as outputs;
t_a time advance function indicating the time the system is allowed to stay in a state if no external event occurs.

Fig. 1 shows an abstract atomic model, which has the following information associated with it.

- Set of input ports through which external events outside of the model are received.
- Set of output ports through which events generated by the model are sent as external events.
- Set of state variables and parameters. Variables are values that changes often, parameters are values that are set occasionally. Three variables are usually present:
  phase: The current state of the model; *phase* $\in S$.
  sigma: Specifies how much time the model stays in the current phase if no external events occur at the input ports of this model; *sigma* $= t_a$.
  elapsed time e: The time passed in the current *phase*; $e \in R^{+}_{0,\infty}, 0 \leq e \leq t_a$.
- The time advance function that controls the timing of internal transitions. The time advance function returns a value that tells how much time should pass from the last event for this model for the next internal transition to occur if there are no external events. Value returned by the time advance function can be anywhere from 0 to ∞

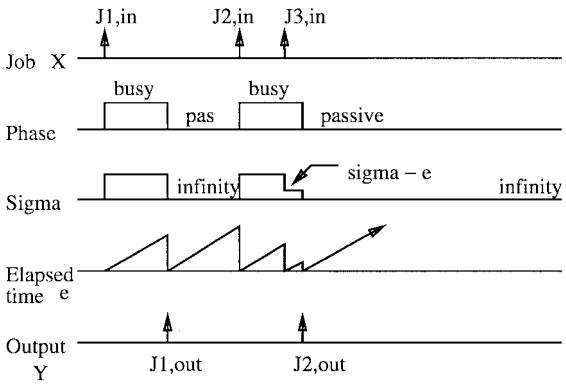

Fig. 2. Time-line analysis of simple processor.

on the real axis. Its value depends only on the current state. Mathematically it is described as: $t_a{:}S \rightarrow R^{+}_{0,\infty}$

- The internal transition function that causes a system to change its state $s \in S$ internally after $t_a(s)$ units of time has elapsed, provided no external event has been received during $t_a(s)$. When an internal transition occurs, elapsed time is set to zero because you just entered a new state and you haven't spent any time in that state yet. Mathematically it is described as: $\delta_{\text{int}}{:}S \rightarrow S$
- The external transition function that specifies how a system changes state when an input is received at its input port. This puts the system into a new *phase* and *sigma*, scheduling it for the next internal transition. The next state is computed by using the value of the current state of the system, time elapsed in the current state, input port, and the value of the external event. More formally, when an event $x \in X$ is received by the system which has been in the state $s \in S$ for an elapsed time $0 \leq e \leq t_a$, it will go into another state $s \in S$ instantaneously. Also, because the *phase* is changed, elapsed time is initialized to 0. Mathematically it is described as: $\delta_{\text{ext}}{:}S \times X \rightarrow S$.
- The output function that generates an external output at particular ports of the model. Output is generated just before an internal transition takes place. Mathematically it is described as: $\lambda{:}S \rightarrow y, y \in Y$.

A. Time-Line Analysis

Fig. 2 shows the behavior of a simple processor by using the time-line analysis. The Y axis has job (external input event), phase, sigma, output, and elapsed time. The X axis is a time line. Now we consider two scenarios.

- When job J1 comes in from the outside on port "in," the processor is passive, so it gets busy and starts processing. This can be seen by the change in phase to "busy" and value of sigma equal to processing_time. After sigma time units have elapsed, phase goes back to "passive," elapsed time to 0, sigma to "infinity," and an output is generated on port "out."
- When job J2 arrives, the processor is passive; it gets busy processing the job. Now job J3 arrives at port "in" before the processor is finished with job J2. So "continue" is executed. The effect of this is to reduce sigma by elapsed time, reset elapsed time, and keep phase unchanged. After the new sigma time units have elapsed, the output is generated from job J2 and the processor becomes "passive" again. J3 is ignored.

By doing the analysis with the time element involved in it, it is possible to see the exact behavior of an object in time.

TABLE I
WEBSERVER OBJECT EVENTS AND PHASES

Ext. Input	Phase	Output
0 Request_info	0 Passive	0 Data_sent
	1 Sending	

B. Coupled Models

A coupled model tells how several atomic models can be connected to create a new model. This new coupled model can later be used as a component of another coupled model, thus allowing hierarchical construction of models. A coupled model has a set of input ports through which external events are received and a set of output ports through which external events are sent out. An external input coupling specifies how input ports of the coupled model are connected to the input port of its components. An external output coupling specifies how the output ports of its components are connected to the coupled model output ports. An internal coupling specifies coupling between its components. It tells how the output ports of a component model are connected to the input ports of other component models. A multicomponent coupled model can be expressed as an equivalent basic model that can be used as a component in another coupled model. This shows that the formalism is closed under coupling, as required for hierarchical model construction.

IV. ANALYSIS AND DESIGN OF RCD SESSION PLAYBACK

A. Time-Line Analysis

During a playback, image, annotation, and audio of a consultation session is presented to a user synchronized in time. In this section, we show, using a time line, the behavior of the objects (which are atomic models) that are involved in the playback.

1) WebServer: This object receives requests for information. It waits for requests; when requests are received they are serviced. Table I shows the numbers for different external input events, phases, and outputs, which are used to describe the behavior of this object.

Fig. 3 shows that, initially, the WebServer is passive when waiting for a client request. When a request is received (event 0), it goes to the Sending phase. At the end of the Sending phase, it sends data to the output port (output 0). When it is in the Sending state (phase 1) and more events occur, they are buffered. So after the end of phase 1, it goes back to phase 1 to finish the pending requests. When no requests are left, it becomes passive.

2) RcdPlayback: This object is the main playback object. It tells the RCD and Control object to display themselves.

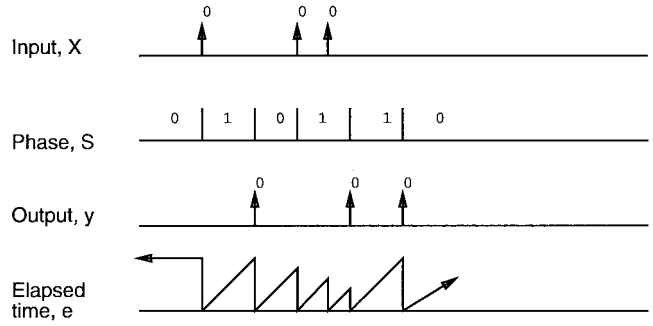

Fig. 3. Time-line analysis of WebServer.

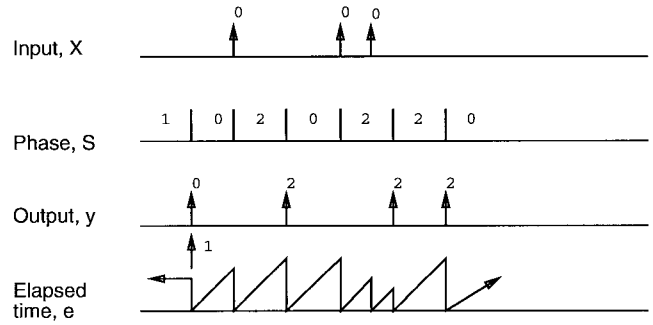

Fig. 4. Time-line analysis of RcdPlayback.

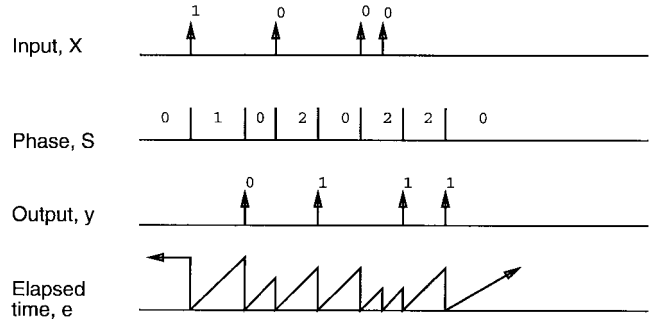

Fig. 5. Time-line analysis of Control.

When it receives user selection at its input, it passes it to the output port. Table II shows the numbers for different external input events, phases, and outputs, which are used to describe the behavior of this object.

At initialization, as shown in Fig. 4, this object is in the Display phase. At the end of sigma for the Display phase, it outputs event 0 and 1, which tells the Control and RCD object to display themselves; then it becomes passive. When user_selection (event 0) is received, it goes to phase 2; at the end of that phase, it outputs user_selection (output 2). While in phase 2, if more user_selections are received, they are buffered, and this object continues to stay in phase 2 and generates outputs (event 2). When no buffered events are left, it becomes passive.

3) Control: This object displays listboxes for a case selection. User clicks on the Graphical User Interface are sent out to the output port. Table III shows the numbers for different external input events, phases, and outputs, which are used to describe the behavior of this object. Fig. 5 shows the time-line analysis for Control.

TABLE II
RCDPLAYBACK OBJECT EVENTS AND PHASES

Ext. Input	Phase	Output
0 User_selection	0 Passive	0 Display_control
	1 Display	2 User_selection
	2 User_select	1 Display_rcd

TABLE III
CONTROL OBJECT EVENTS AND PHASES

Ext. Input	Phase	Output
1 Display	0 Passive	0 GUI_screen
0 User_clicked	1 Show_GUI	1 User_selection
	2 User_click	

TABLE IV
AUDIOCLIP OBJECT EVENTS AND PHASES

Ext. Input	Phase	Output
0 Start	0 Passive	0 Audio_packet
1 Stop	1 Load_file	1 Audio_request
2 Load_file	2 Playing	
	3 Loading_file	

Initially, Control is in the Passive phase. When it is told to display itself (event 1), it processes the event in the phase Show_GUI. The output is to show its GUI components on the screen (output 0). It becomes passive after that. When a user clicks (event 0) on a GUI component, it is processed in the phase User_click and the output generated is sent out. When multiple events 0 occur while in the User_click state (phase 2), they are buffered and processed sequentially until none are left. As can be seen in the analysis, when Control is done with phase 2, it goes back to phase 2 to process the saved requests. After that it becomes passive.

4) AudioClip: This object reads a particular audio file and continuously generates audio packets until either it is told to stop or the end of file is reached. Table IV shows the numbers for different external input events, phases, and outputs, which are used to describe the behavior of this object. Fig. 6 shows the time-line analysis for AudioClip.

AudioClip is initialized in the Passive phase. When the external event 2, Load_file is received, it goes to phase 1 (Load_file) and reads the file. After that it becomes passive again. When Start is received (event 0), it goes to phase Playing and outputs audio packets (output 0) at a continuous interval of time. It stays in phase Playing until Stop (event 1) is received. It becomes passive (phase 0) after receiving Stop (event 1). While it is in phase Playing (phase 2) and Stop (event 1) is received at the input, if elapsed_time is not equal

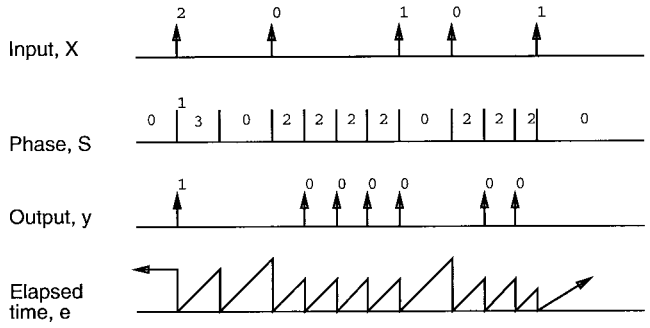

Fig. 6. Time-line analysis of AudioClip.

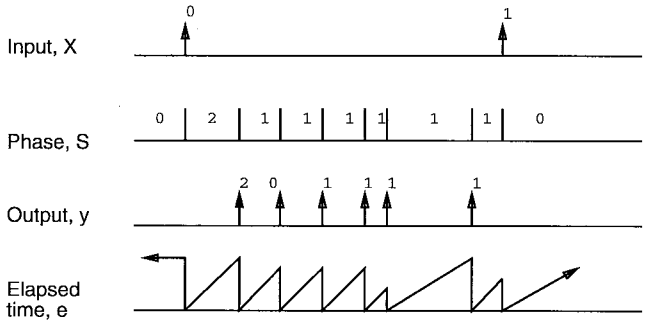

Fig. 7. Time-line analysis of RCD, External event stop received.

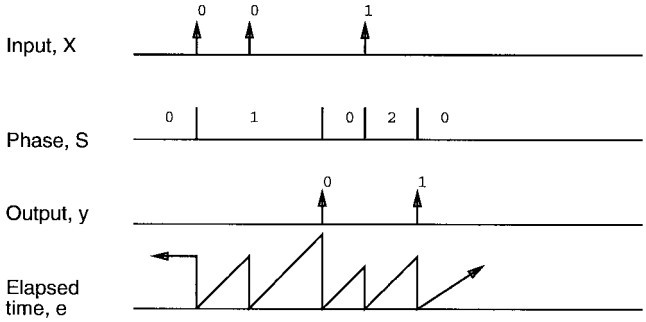

Fig. 8. Time-line analysis of ImageFrame.

to sigma, the output packet is not created and it goes to the Passive phase.

5) RCD: This object receives output from the RcdPlayback object as its input events and responds by sending output to AudioClip, WebServer, and Network object. It reads the saved playback file over a data network and tells other objects to display the recorded information (image, annotation, and audio) so that the presentation is synchronized for all the media. Table V shows the numbers for different external input events, phases, and outputs, which are used to describe the behavior of this object.

In Fig. 7, RCD starts in the Passive phase. A state variable is kept to indicate whether audio is playing or not. When the external event 0 asking to play a file is received, it goes into the phase Reading_file (phase 2). After the file is read, a request is sent to the WebServer for multimedia data, and it changes to phase Playing in the internal transition function. At the end of the first Playing phase, start audio (output 0) is created. If an audio is already playing, it is stopped automatically by

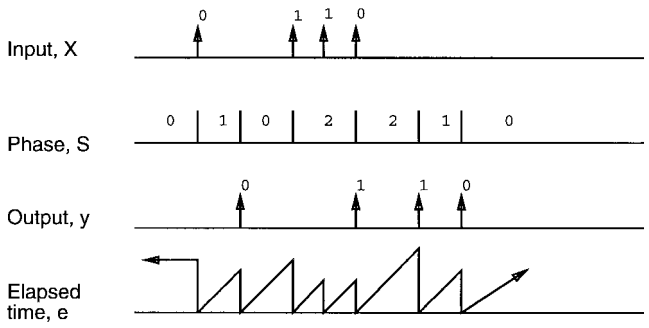

Fig. 9. Time-line analysis of Network.

TABLE V
RCD OBJECT EVENTS AND PHASES

Ext. Input	Phase	Output
0 Play_file	0 Passive	0 Start_audio
1 Stop	1 Playing	1 Annotation_commands
	2 Reading_file	2 Req_session_data

TABLE VI
IMAGEFRAME OBJECT EVENTS AND PHASES

Ext. Input	Phase	Output
0 Draw_image	0 Passive	0 Image_on_screen
1 Draw_annotat.	1 Get_image	1 Annotat._on_screen
	2 Draw_annotat.	

checking the state variable so that the new one can be played. Sigma for phase Playing is based on the time that is read from the recorded file. This Sigma represents gaps between different annotations. At different times, the duration for phase 1 is different; at the end of each duration, annotation (output 1) is sent out. When Stop (event 1) is received while in Playing phase, RCD simply goes to phase Passive.

6) ImageFrame: This object is responsible for displaying the diagnostic images on the screen. It also receives annotation commands and displays them appropriately on the image. Table VI shows the numbers for different external input events, phases, and outputs, which are used to describe the behavior of this object. Fig. 8 shows the time-line analysis for the ImageFrame.

When this object receives the Draw_image event (event 0) at its input, it changes phase to the Get_image and retrieves a big image and draws it on the screen (output 0). If it receives another event 0, while in phase 1, it starts again in phase 1 with a new request for the required time, and the old request is lost. After the image is drawn on the screen (output 0), it becomes passive. When the Draw_annotation command is received, it processes it in phase 2, and draws the annotation on the screen (output 1) and becomes passive.

7) Network: This object deals with sending and receiving information over a data network. In playback, it receives information from the RCD object and sends it to the ImageFrame object. The information transferred are control and annotation

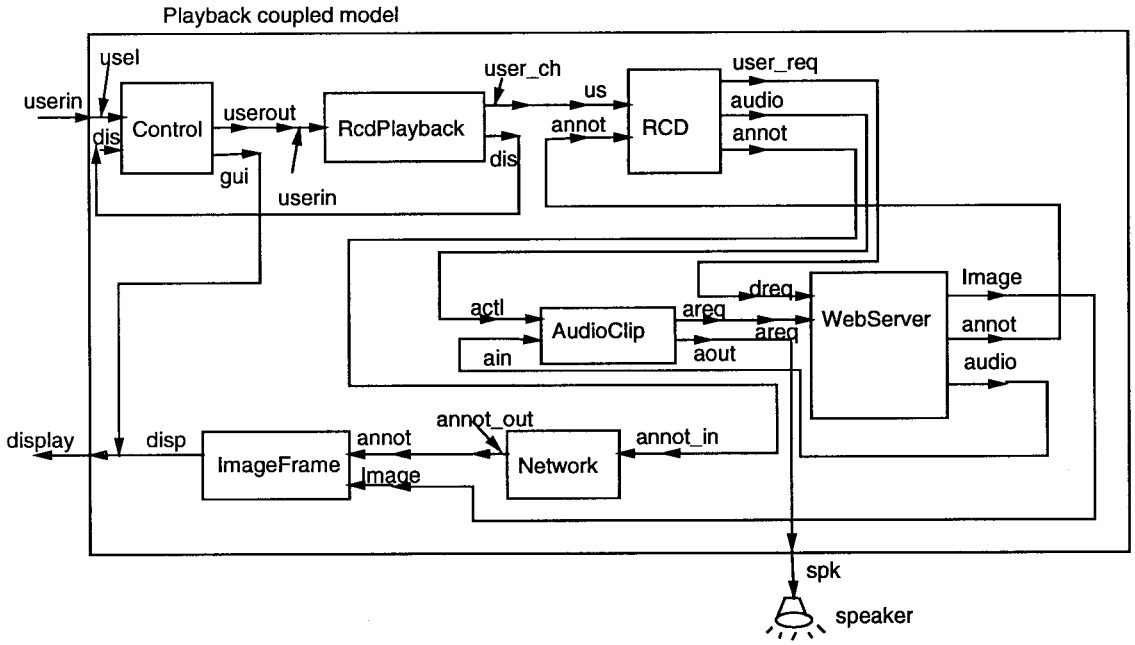

Fig. 10. Coupled model for Playback.

TABLE VII
NETWORK OBJECT EVENTS AND PHASES

Ext. Input	Phase	Output
0 Send_info.	0 Passive	0 Send_info_to_network
1 Recv_info	1 Sending	1 Recv_info_from_netw
	2 Receiving	

commands. Table VII shows the numbers for different external input events, phases, and outputs, which are used to describe the behavior of this object. Fig. 9 shows the time-line analysis for the Network.

Initially, this object is in the Passive state. When an event to send the information is received at its input port, it goes to phase Sending and outputs the information (output 0). When it is in either the Sending or Receiving state (phase 1 or 2) and receives more events to send or receive information (event 0 or 1), they are all buffered in a queue and it stays in the same phase. At the end of phase 2, it generates output 1 and stays in phase 2 to process more buffered requests. When done with all the requests, it goes to (phase 0) the Passive phase. Queued requests are checked in the internal transition function. If the queue to send or receive is not empty, it continues in phase 1 or 2, otherwise it becomes passive.

B. Coupled Model for RCD Playback

As explained in Section III-B, we can make a coupled model from atomic models to construct the total system. Fig. 10 shows the RCD playback coupled model. Notation "Playback,userin → Control,usel" means that events from the userin port of Playback model goes to the usel port of Control. In Fig. 10, an arrow going into a model and out of a model means input and output port, respectively.

All the external and internal couplings are as follows.

External input coupling:
Playback,userin → Control,usel
External output coupling:
AudioClip,aout → Playback,spk
ImageFrame,disp → Playback,display
Control,gui → Playback,display
Internal coupling:
Control,userout → RcdPlayback,userin
RcdPlayback,user_ch → RCD,us
RCD,user_req → WebServer,dreq
RCD,audio → AudioClip,actl
RCD,annot → Network,annot_in
WebServer,Image → ImageFrame,Image
WebServer,annot → RCD,annot
WebServer,audio → AudioClip,ain
Network,annot_out → ImageFrame,annot
AudioClip,areq → WebServer,areq
RcdPlayback,dis → Control,dis

V. IMPLEMENTATION

Each DEVS atomic and coupled model is implemented as a class. Incoming events can be thought of as a method invoked on a class; the return value of a method can be considered as an output from an object. State variables of an object are class variables. Atomic model classes are contained in a coupled model class to create the total system.

A. Overall System Architecture

Fig. 11 shows the overall architecture of the playback system. The overall architecture consists of a Web browser as a client and a HTTP server running across a local or wide area network. A request from a client is fulfilled by the server by sending the appropriate data. JAVA applet and classes are

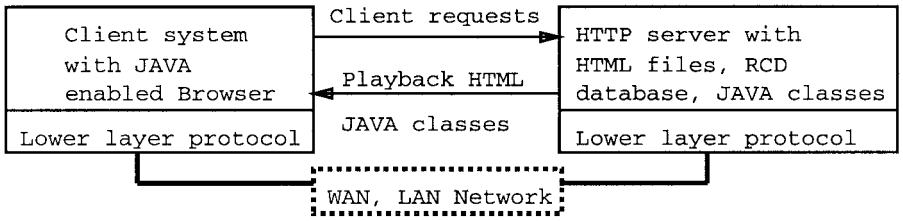

Fig. 11. Overall system architecture.

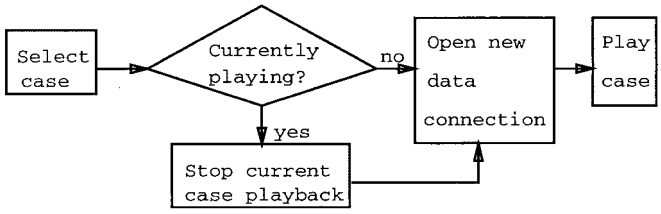

Fig. 12. Playback selection control flow.

downloaded from the server and then executed in the browser. Because all the processing is done locally in the user's machine after byte codes are downloaded, this architecture is scaleable. After the applet is displayed, the control flow for viewing the playback cases is shown in Fig. 12.

When a case is selected, information about that case is retrieved from the server. After information is retrieved, images are displayed and a playback session starts. The Stop button can be pressed to stop the currently playing case. If nothing is pressed, the case is played until it finishes. When a new case is selected before the current playing case is finished, the currently playing case is stopped. Then a request for information about the new case is sent to the server. When data is received about the new case, playback is started on the new case.

B. Data Flow for Playback

Fig. 13 shows how different media and data types flow into the whole system. Data types involved are control data, case selection data, annotation data, audio, images, and image movement data. When a case is selected, control information about the case selection is sent to the server. The server responds by looking up the data for that particular case and then by sending the data back. The following is a description of how the various data types are handled by different classes of the software.

- Audio: It goes from the server to AudioClip over the data network. AudioClip plays the audio. Audio is controlled from the RCD class.
- Images: Once a request is sent, the RCD delegates work to retrieve images to the ImageFrame class, which is responsible for local display of images on the client side. Image traffic goes from the server through the data network and directly to the ImageFrame class.
- Control and Annotation: These data travels from the server through the data network to the RCD and then to the Network class. From the Network class, it pipelines to the ImageFrame class as shown in Fig. 13.

The ImageFrame class responds to image movement, annotation, and other control commands that it receives from the Network class. By transferring image directly to the class that deals with it, a lot of unnecessary data movement inside the client is avoided. Due to the large memory requirements of the images, this design reduces overhead by avoiding unnecessary copying of data within the client and makes the client system more efficient.

C. Multimedia Synchronization in RCD Playback

RCD playback involves different types of data. During playback, synchronization among these data types is essential to provide real-life playback experience [4]. In playback, we are dealing with two types of data. The first one is control and annotation data and the second is voice. Voice is recorded in a μ-law format. Audio is isochronous data because it requires a constant data rate. Once you play one packet of audio, the other has to be played after a fixed amount of time determined by the sampling rate. All these data need is to be accessed from the WebServer, as shown in Fig. 13, through the data network. Since we are using TCP/IP Internet to transfer our data, there is no guaranteed quality of service (QoS) available for our data. Thus, we have to deal with the network delay and network jitter in the playback system.

In playback, network delay can be dealt with by delaying the start of the playback. This is the scheme used in our playback. Network jitter can be dealt with by buffering some packets so that if some packets are delayed more, there are still packets to play from the buffer before the next packet arrives. Annotation data and voice is saved in separate files, so they are retrieved separately. They also have a separate travel path, as shown in Fig. 13. Synchronization is done at the client site so that you don't have to worry about different delays and jitter in audio and annotation commands. Three different classes running in different threads interact to do synchronization. The RCD class receives image annotation commands over the network from server. A pipeline is created between the RCD class and the Network class. The Network class thread waits for annotation commands from the pipeline, and when it receives it, it calls appropriate methods of the ImageFrame class.

The annotation file that RCD receives contains the time-stamp information. This information is relative to the beginning of the session. Both voice and image annotation refers to the same starting point. When voice and annotation are started together in different threads, the RCD thread checks the time-stamp information for annotation from the starting point. It sleeps for whatever time is required to synchronize annotation command to voice; then it sends commands to the Network

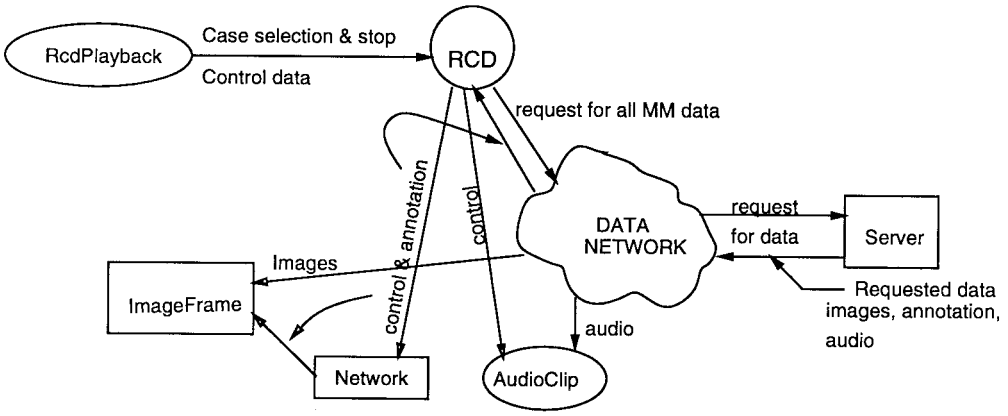

Fig. 13. Multimedia data flow between objects.

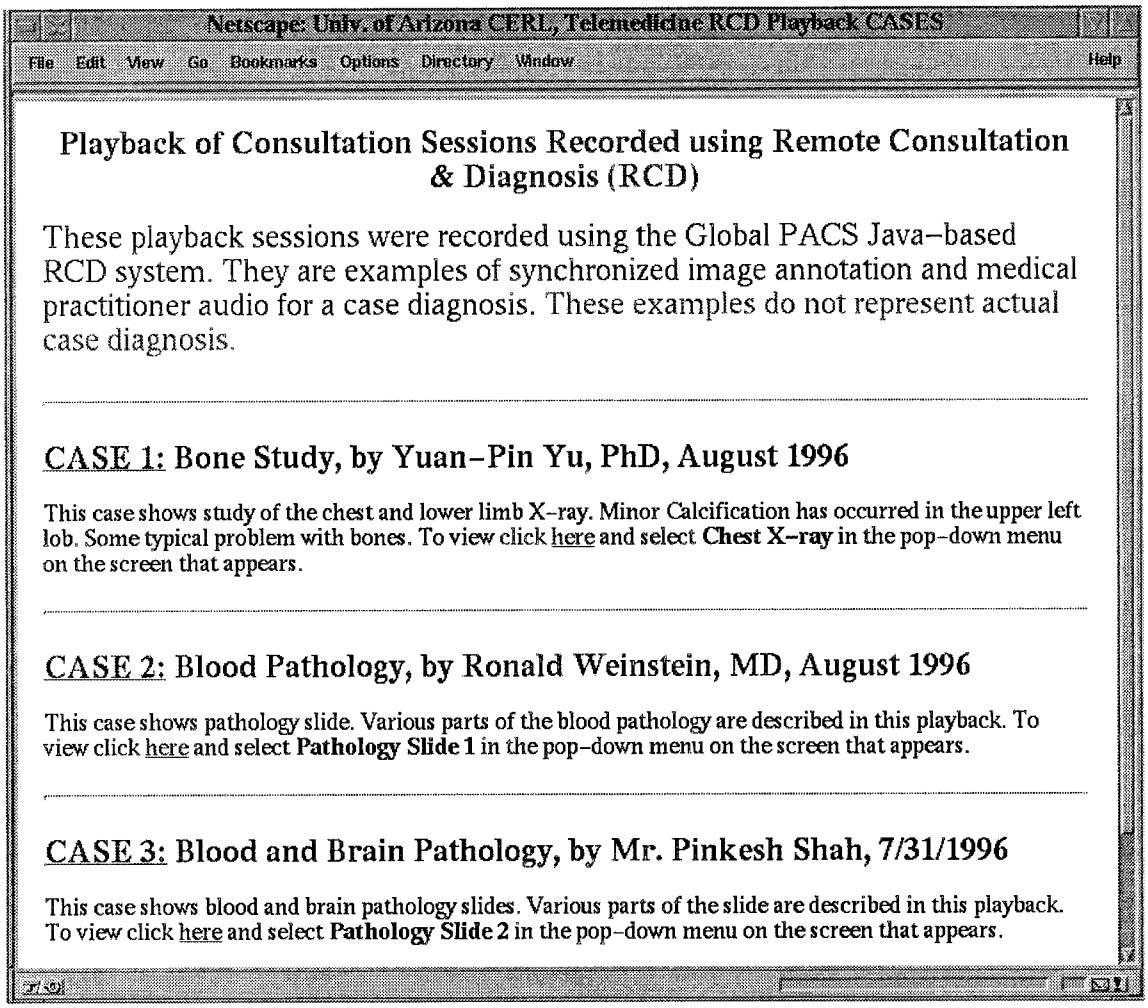

Fig. 14. RCD playback cases description page.

class through the pipeline. So, voice is played continuously, and image annotations are shown at discrete times.

D. RCD Playback GUI

In this section, an actual view of the RCD playback screens is shown. Fig. 14 shows the picture of the main web page as it looks in the browser. This page has descriptions about different cases. The link is provided from each description to the web page with the applet that does the playback. An actual display of a playback screen is shown in Fig. 15.

VI. Results

We applied DEVS formalism to identify RCD playback atomic models. In Section IV-A, DEVS time-line analysis is

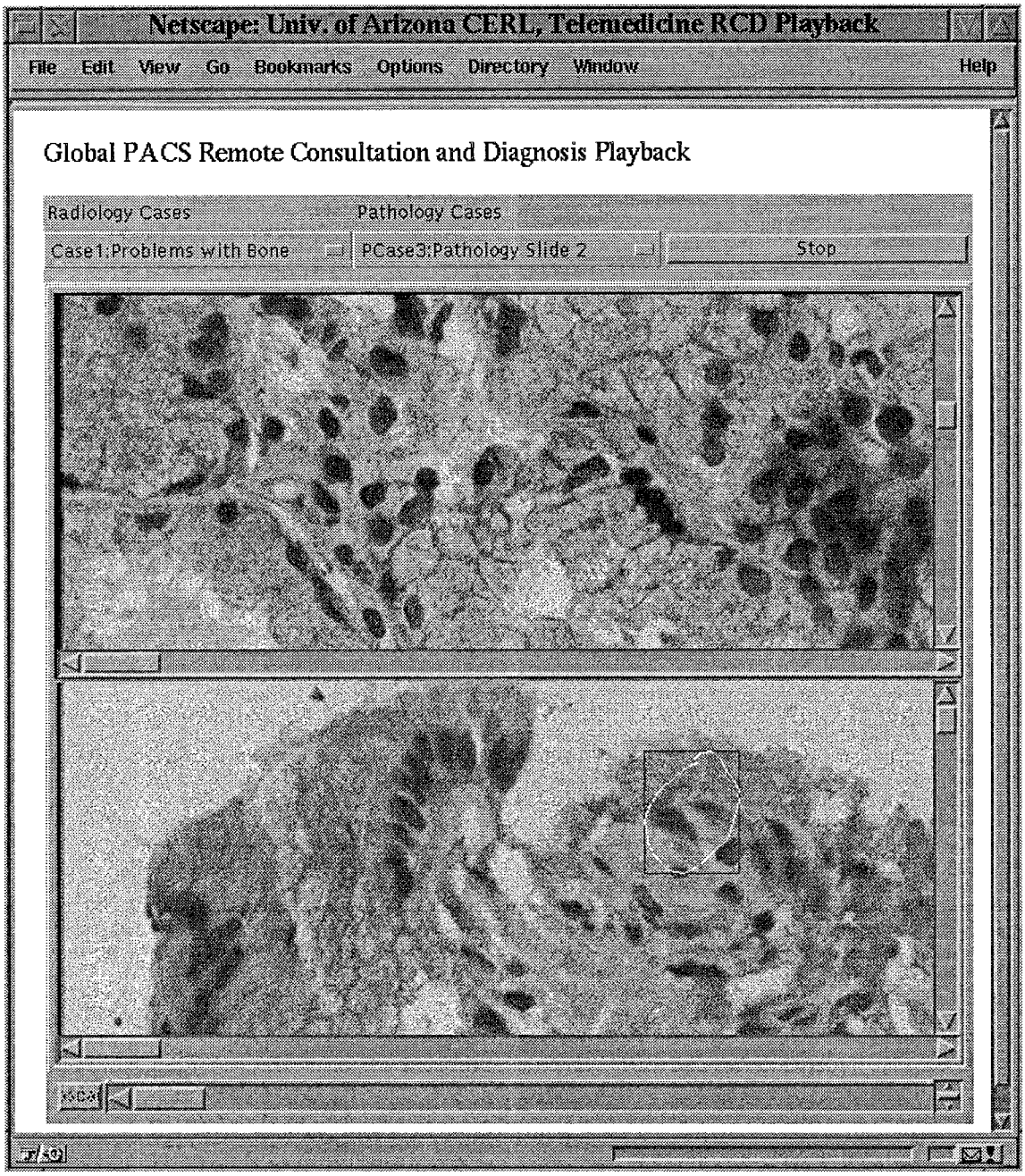

Fig. 15. Playback of a case in a browser.

used to design and analyze the behavior of the models. We used the DEVS coupled model specification to create a coupled model of the RCD playback in Section IV-B. The RCD playback classes were implemented using the time-line analysis and the coupled model information. The RCD coupled model defined method invocations or output messages based on user, recorded, or internal events. For example, when an annotation event is received by RCD class, it knows how much time to wait before sending the "Annotation_commands" output to the Network class, so that the annotation is synchronized with audio. As described in Section V, the RCD playback JAVA applet was implemented using DEVS-based design and analysis.

VII. Summary and Conclusion

RCD is an important part of a telemedicine system. You can use it for local as well as remote diagnosis. By integrating the multimedia playback part to the current RCD, we have provided an important learning tool. Separate tutorials or previous consultation sessions can be made available for learning at anytime and any place where a computer and an Internet connection is available. Using DEVS, we are able to incorporate time in the design and analysis of distributed multimedia system, such as RCD, which is not possible using traditional object-oriented methodology. Time-line analysis and a coupled model define how a system responds to an

event, and they also guide us in developing algorithms to implement the system. Behavior of the system is fully known before the implementation by using DEVS methodology. The DEVS approach reduces implementation, testing, and debug time in the software development cycle. The DEVS design and analysis should be an important part of the development process for distributed telemedicine systems like RCD.

REFERENCES

[1] Y. Yu, "Object oriented remote consultation and diagnosis in global PACS using multi-threaded JAVA," Ph.D. dissertation, University of Arizona, Tucson, 1996.

[2] R. Martinez, J. Kim, J. Nam, and B. Sutaria, "Remote consultation and diagnosis in a global PACS environment," *Proc. SPIE Medical Imaging IV Conf.*, vol. 1899, pp. 296–307, 1993.

[3] R. Martinez, B. Sutaria, and F. Pardede, "A distributed file management system for remote consultation and diagnosis in global PACS," *Proc. SPIE Medical Imaging Conf.*, vol. 2165, Feb. 1994.

[4] R. Martinez, W. J. Chimiak, J. Kim, and F. Pardede, "Synchronized voice and image annotation in remote consultation and diagnosis for the global PACS," *Proc. SPIE Medical Imaging Conf.*, vol. 2165, pp. 9–20, Feb. 1994.

[5] R. Martinez, W. J. Chimiak, J. Kim, and Y. Alsafadi, "The rural and global medical informatics consortium and network for radiology services," *J. Comput. Biol. Med.*, vol. 25, no. 2, pp. 85–106, Mar. 1995.

[6] R. Martinez *et al*., "Design of multimedia global PACS distributed computing environment," in *Proc. Twenty-Eighth Hawaii Int. Conf. System Sciences*, vol. 3, pp. 461–469, 1995.

[7] R. Martinez, Y. Alsafadi, and J. Kim, "OSF distributed computing environment for multimedia telemedicine services in global PACS," *Proc. SPIE Medical Imaging Conf.* vol. 2435, 1995.

[8] R. Martinez and S. L. Hsieh, "Design of multimedia global PACS CORBA environment," in *Proc. IFIP/IEEE Int. Conf. Distributed Platforms*, 1996, pp. 201–212.

[9] B. P. Zeigler, *Multifaceted Modeling and Discrete Event Simulation*. Orlando, FL: Academic, 1984.

[10] ———, *Theory of Modeling and Simulation*. Malabar, FL: Krieger, 1985.

[11] ———, *Objects and Systems: Principled Design with Implementations in C++/JAVA*. New York: Springer-Verlag, 1997.

[12] G. Booch, *Object-Oriented Analysis and Design with Applications*, 2nd ed. Reading, MA: Addison-Wesley, 1994.

[13] W. J. Chimiak, "The digital radiology environment," *IEEE J. Select. Areas. Commun.*, vol. 10, pp. 1133–44, Sept. 1992.

[14] B. Meyer and J. Nerson, *Object Oriented Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[15] See http://java.sun.com:80/products/jdk/1.0.2/api/.

Pinkesh J. Shah (S'97) received the B.S. degree (summa cum laude) in electrical engineering from the New Jersey Institute of Technology, Newark, in 1989 and the M.S. degree in electrical engineering from Columbia University, New York, NY, in 1992. He is a Ph.D. candidate in the Electrical and Computer Engineering Department at the University of Arizona, Tucson.

He worked on various software engineering projects from 1989 to 1994 at IBM. He worked on telecommunications anti-fraud software and mobile data health-care applications at Bellcore during summer 1995 and spring 1996. His current research interests are distributed multimedia systems, telemedicine systems, computer networking, and telecommunications.

Mr. Shah is a member of ACM, Tau Beta Pi, Eta Kappa Nu, and Omicron Delta Kappa.

Ralph Martinez (M'84) is an Associate Professor in the Electrical and Computer Engineering Department with joint appointments in the Radiology and Biomedical Engineering Departments, University of Arizona, Tucson. He has been at the University of Arizona since 1982. Before then, he spent 14 years in industry as a researcher in computer system design and applications, specializing in distributed processing architectures and Internet gateways for computer networks. At the Naval Ocean Systems Center (1974–1979), he was responsible for applications of new VLSI devices to naval systems. At General Dynamics Electronics Division (1979–1982), he was the System Architect for the design of the Global Positioning System, Phase II, and was branch head for an R&D group in local area network protocol development and applications to new business areas. Since joining the Electrical and Computer Engineering Department, he has been involved in research in interoperable global information systems, internetworking, picture archiving and communications systems, and multimedia telemedicine systems.

Bernard P. Zeigler (SM'87–F'94) received the B. Eng. Phys. degree from McGill University, Montreal, P.Q., Canada, in 1962, the M.S.E.E. degree from the Massachusetts Institute of Technology, Cambridge, in 1964, and the Ph. D. degree from the University of Michigan, Ann Arbor, in 1968.

He is Professor of Electrical and Computer Engineering at the University of Arizona, Tucson. He has published over 200 journal and conference articles in modeling and simulation, knowledge-based systems and high autonomy systems. He has also published several books. He is currently heading a multidisciplinary team to demonstrate an innovative approach to massively parallel simulation of large scale ecosystem models within NSF's HPCC Grand Challenge initiative. He is also sponsored by Rome Laboratory to research the use of such high-performance simulation technology in support of optimization and model abstraction. In April 1996, he started as a co-principal investigator in a major contract with the USAF Armstrong Lab Logistics Research Group to develop a simulation-based, group collaborative, business reengineering environment.

Dr. Zeigler served on a National Research Council committee to suggest directions for information technology in the 21st Century U.S. Army in 1995, and is currently a member of an NRC committee given a similar task by the U.S. Navy, focusing on modelling and simulation. He is currently editor-in-chief of the *Transactions of The Society for Computer Simulation*. He was elected as Fellow of the IEEE for his innovative work in discrete event modeling theory.