5

Designing Distributed Multimedia Systems using PARSE

*A.Y. Liu** *T.S. Chan*** *I. Gorton****

CaST Lab, School of Computer Science and Engineering,
University of New South Wales, Kensington, NSW 2052, Australia
tel: +61 -2 385 4019, fax: +61 -2 385 5995
contact email: annaliu@cse.unsw.edu.au

*Division of Radio Physics, CSIRO, Sydney, Australia***
contact email: tchan@rp.csiro.au,

*Division of Information Technology, CSIRO, Sydney, Australia****
contact email: ian.gorton@dit.csiro.au

Abstract

With recent vast improvements in computer hardware, in particular, the processing capacity of multimedia database servers, and high performance of networks, distributed multimedia applications are becoming a reality. This paper presents an object-based approach to the design of distributed multimedia software. In particular, the PARSE methodology for designing parallel and distributed systems is employed. Justification of the object-based approach is given, and an overview of the PARSE process graph notation is presented. A case-study of a video-on-demand application is then presented, and a mapping from the design to an implementation based on Windows NT is described.

Keywords

Distributed system design, multimedia systems, parallel software engineering, PARSE

1 INTRODUCTION

Advances in computer and media technology have enabled the development of high performance multimedia workstations and servers (Jadav.1995), (Taylor.1995). In addition to processing traditional computer data, these workstations are designed to integrate processing of other media types, such as video, image, voice and sound. On another level, the emergence of high-speed, broadband networks such as B-ISDN (Broadband Integrated Services Digital Network) (Minzer.1989) have accelerated the development of highly interactive distributed

I. Jelly et al. (eds.), *Software Engineering for Parallel and Distributed Systems*

multimedia systems (Furht.1994). These systems are designed to transport high bandwidth multimedia information across the network, while supporting real-time interactive interfaces.

Applications developed using multimedia technology can benefit significantly from the rich expressive graphical presentation, with the potential to incorporate distributed and collaborative processing. Examples of such applications are interactive video conferencing systems, video-on-demand systems, computer-aided collaborative design and multimedia electronic shopping systems. It is anticipated that the proliferation of multimedia applications development will continue to gain momentum and acceptance just as graphical user interface has replaced traditional command prompt interface. Consequently, there is an increasing need for the development of a software engineering approach to facilitate the modelling and design of a potentially complex multimedia system (Gibbs.1995).

This paper investigates the use of a concurrent, object-based modelling design technique for developing multimedia systems. In particular, the paper describes the use of the PARSE (Parallel Software Engineering) software engineering methodology (Gorton.1995). Issues relating to the modelling of distributed multimedia systems are described in section 2. Section 3 gives a brief overview of the PARSE notations, with an emphasis on tthe features of PARSE that supports distributed multimedia system designs. Section 4 and 5 presents a case study on the design of a video-on-demand (VOD) multimedia application using PARSE, and a mapping to the WIN32 Application Programming Interface (API) is given in section 6.

2 MODELLING DISTRIBUTED MULTIMEDIA SYSTEMS

The development of software for distributed multimedia applications presents several challenging requirements. These range from the need to transmit and synchronise multiple media types, through to support for distributed interactive processing with real-time performance parameters. The development process is further complicated by the need of applications to support heterogeneous hardware platforms, as well as different device drivers and operating systems. Specifically, a software engineering methodology for distributed multimedia systems should support the following key design aspects:

Synchronisation: In multimedia applications, the synchronisation between different types of medium is important. For example, it is crucial to control and synchronise the broadcast of video and audio components of a video segment. In addition, as different components may operate at different speeds, control messages are often sent to several processes in order to synchronise the total system activity. Any design methodology for distributed multimedia software must cater for the explicit design of synchronisation mechanisms.

Dynamic Process and Communication Path Creation: The client-server paradigm can often be found in distributed multimedia applications, where the server process objects spawn extra helper processes to carry out the work as requested by clients. The software design method should be able to capture features such as the dynamic creation and deletion of processes and communication paths.

Strong Modularity: Modularity is essential in distributed multimedia systems design. Encapsulated software objects representing multimedia system components reduce the development effort, shielding the programmer from the complex details of programming and operating media hardware. Essentially, the object model contains all the necessary functions for the operation of the media device, making direct programming unnecessary (Friesen.1995).

This also promotes portability of code, as objects can be reimplemented for different platforms while maintaining a constant functional interface (see Figure 1).

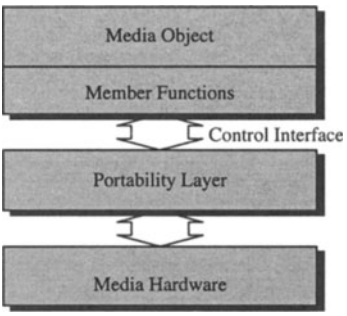

Figure 1 Media Object Interface.

3 OVERVIEW OF PARSE

PARSE (PARallel Software Engineering) is an object-based software engineering methodology that facilitates the design of reliable and reusable parallel and distributed systems. Software design in PARSE is centered around a process graph notation. The notation allows the partitioning and synchronisation of the software to be expressed in a graphical manner. Designs represented as process graphs are simple and concise, and can be progressively refined to capture all the structural and dynamic properties of a design. The three basic features of PARSE process graphs are: explicit classification of process objects using a small set of system supplied general classes; interaction between process objects is done via message passing on typed communication paths; and the unique feature of path constructors, used to specify relationships between process object communication paths. The basic design components of PARSE are shown in Figure 2, and for a full description of the individual components, please refer to (Gorton.1995).

The basic PARSE process graph notation cannot handle the design of distributed systems which incorporate the dynamic creation and deletion of processes and of communication paths. Hence, the dynamic reconfiguration of systems cannot easily be captured in the design. In a typical multimedia application, the provision of these facilities is vitally important. The Extended-PARSE (Ext-PARSE) (Liu.1996) process graph notation (Figure 2) is designed to supplement the basic PARSE process graph notation for this purpose.

Process objects may be created and deleted dynamically. New process objects may enter/exit the system at run time. This should not affect the execution of other processes. Function servers and control process objects may create and delete dynamic process objects by *invoking* create and or delete signals. They are shown via the *twisted arrow* notation. There are two rules of usage: the process object at the *invoked* end of a creation/deletion arrow must be of *dynamic* type; and the process object at the *invoking* end of a creation/deletion arrow must be an *active* process object (this means data server is excluded). There are also three ways that dynamic process objects may exit from the system: *assassination*, where an active process object kills a dynamic process object; *suicide*, where a thread terminates its own execution; and *aging*, which is the default termination mode. The created thread dies from aging when it

completes its work. This occurs naturally, hence the term *aging*. The notation caters for all three possible ways of termination, and leaves the design decision for the software engineer.

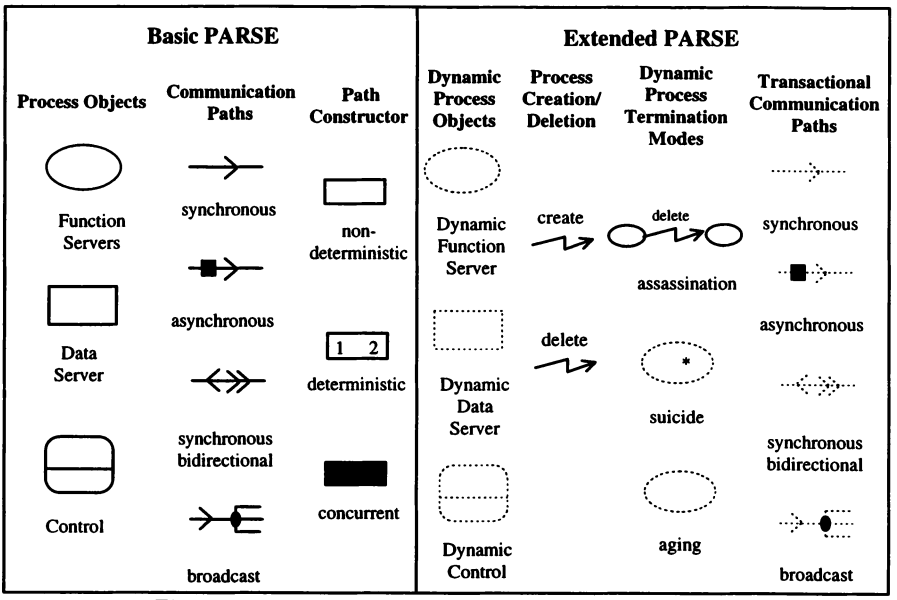

Figure 2 Summary of the PARSE Process Graph Notation.

Communication paths going into and or coming out from dynamic process objects are dynamic in nature. These communication paths are set up when the associated process objects are created, and are destroyed when process objects terminate.

Dynamic process objects are often replicated. Each replicated instance has the same internal behaviour. However, not all instances of the process are created simultaneously: different instances may be created and terminated at different times. The series of numbers enclosed by square brackets [0..n] denotes the range of the number of instances of the object that may be present in the system at any one time. The symbol 'n' may be replaced by a constant integer, or by default, is the maximum number of thread instances a process may have as defined by the underlying system.

In Figure 3a, there is (at any time) a maximum of one communication path between the two dynamic process objects. However, this is not always the case. By default, a communication path going into or coming out from a dynamic process object is replicated if there are multiple instances of the process.

By default, all associated processes are fully connected by communication paths. Specific path restriction notations can also be used to override default behaviours (see Figure 3b).

Transactional Communication Path: Software designers can explicitly show that the communication between two processes is of a transactional type by using dotted arcs (see Figure 2 Extended PARSE). In many software systems, such as database applications, *transactional communication paths* are often used. These are different to the ordinary

communication paths in the sense that they are set up only when they are needed to transfer messages. As soon as the transfer is complete, the path is no longer valid. Hence, the life span of a communication path is not dependent on the life spans of the processes using it, but is dictated by the activity of transferring the message.

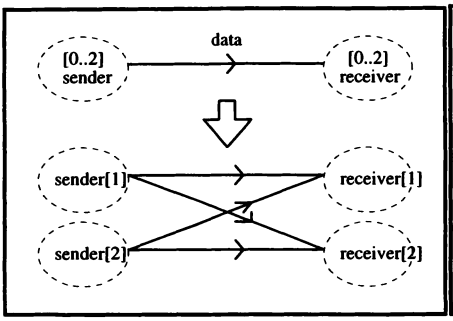

Figure 3a

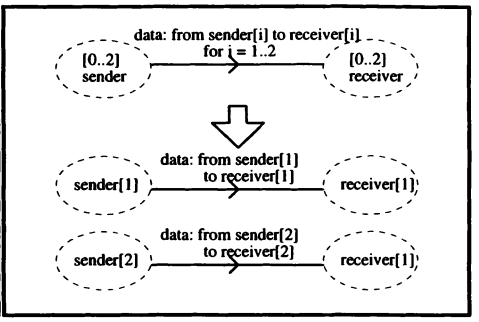

Figure 3b

Figure 3a Replication Of Dynamic Communication Path.
Figure 3b Path Restriction.

The period of validity of the communication path depends on the associated process internals. This can be explicitly defined using the behavioural specification language [Gorton95].

PARSE uses hierarchical decomposition to handle large designs. Further, in typical client-server systems, it is often desirable for processes to spawn new helper processes to service multiple clients' requests. *Multithreaded objects* provide abstraction for the low level process creation/deletion activities. The default structure (expressed within the roundangle) is as shown in Figure 4.

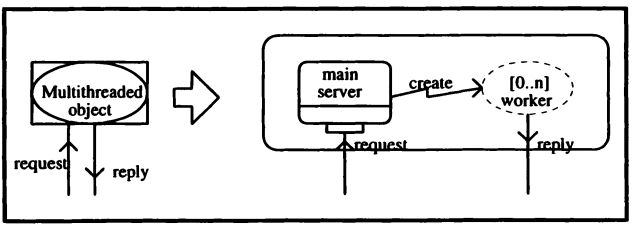

Figure 4 Default Behaviour Of A Multithreaded Object.

A designer may override this default structure and or behaviour by providing the decomposition of the multithreaded object, thus the multithreaded object simply provides an abstraction at the higher level of design, for the low level parallelism.

4 CASE STUDY: VIDEO-ON-DEMAND

Previous work with PARSE has focused on the design and development of closely-coupled parallel systems (Gorton.1994). In this paper, we wish to illustrate the use of PARSE to design loose-coupled distributed systems such as multimedia applications.

This section describes a case study on the use of PARSE for the design of a distributed video-on-demand (VOD) multimedia application[1]. In addition to providing VCR-like functions for controlled playback, the application is designed to support interactive non-linear access to video footage. Compressed videos are manually segmented, and a descriptor file is created for each significant segment of the file. Each descriptor consists of several control parameters and pointers to the start and end of the associated video segment. This control information is loaded into the main memory at start-up time, so as to facilitate high-speed retrieval of the requested video segments. The video server is designed to support multiple concurrent connections. For each new connection requested by a remote client, the parent server process creates a child process to handle the newly requested service.

4.1 PARSE Process Graph Design For VOD

The top-level PARSE design diagram shows the client-server structure of the video-on-demand application (see Figure 5).

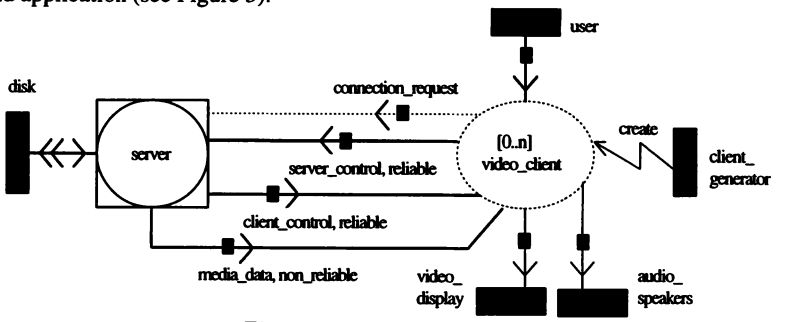

Figure 5 Top Level Diagram.

The video client is modelled as a dynamic process object. The notation *[0..n]* denotes that there may be *0* or more active clients at any one time. Here, *n* is unspecified, and hence only limited by system resources.

A *video_client* is created dynamically by the external *client_generator*[2], which represents the external interface of the system (here, a *video_client* is created in response to the user starting up the application). Upon creation, the client software sends the initial *connection_request* message via the 'transactional' communication path. This prompts the server to initiate a connection for the client (for more details, the internal structure of the multithreaded server is shown in Figure 6). For each connection service requested by a newly created client, a set of connection paths are established for the exchange of control and media information between the distributed client and server processes. The *server_control* connection path is used by the

[1] The precise application details are commercial in confidence.

[2] The solid bar represent an 'external entity' in the PARSE notation (Gorton.1995).

client process to transfer control messages to the server for controlled playback and process synchronisation. Similarly, the *client_control* connection path is used by the server process to provide feedback control information to the client. These control paths are specified as asynchronous in nature, and of type *reliable*, which specifies the protocol used on these paths. Logical protocol definitions are specified textually using a simple protocol definition language: they are omitted from here due to space restrictions. In contrast, the *media_data* path is declared as *unreliable*, as this can survive information loss, but requires low delay and jitter control to support continuous media stream playout at the client.

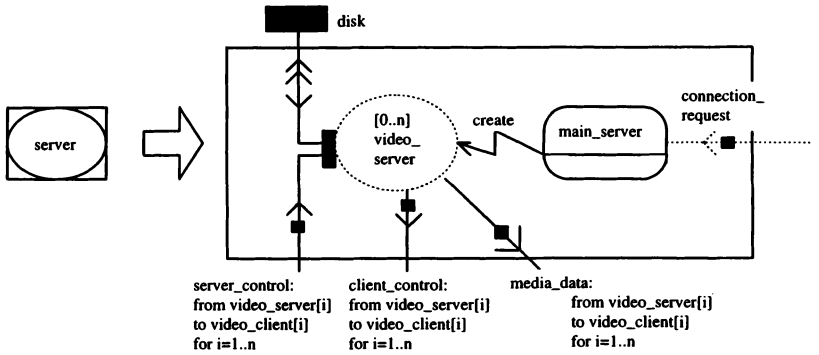

Figure 6 Behaviour of Multithreaded 'Server' Object.

The communications characteristics specified in PARSE can be mapped directly to the appropriate communication Application Programming Interface (API) supported by the underlying network. For example, in an ATM network, the use of reliable communication path specified in PARSE can be mapped to a communication API based on TCP/IP using AAL5 adaptation layer (Boudec.1992). Similarly, the use of an unreliable communication path with specified delay and jitter parameters for continuous media communication can be mapped directly to an API that supports real-time traffic such as AAL 1 and AAL2 (Boudec.1992).

The behaviour of the 'multithreaded server' process is decomposed as shown in Figure 6.

The *video_server* process object is created dynamically in order to service the video clients' requests. This activity is co-ordinated by the *main_server* control process object.

Hence, for each instance of the client process, there is a copy of video server to service the client's request. That is, for each *client[i]*, a *video_server[i]* is created, for $i = 0..n$. The *path restriction* feature (Gorton.1995) in PARSE has been used to specify that the communication paths: *server_control*, *client_control*, and *media_data* only exist between corresponding video server and client.

The video server process is further decomposed in Figure 7.

The server control unit receives control data from the client via the *server_control* communication path. The control information is parsed and the appropriate control actions are sent to *MJPEG_file_server* for processing. Based on the control actions received from the control unit, *MJPEG_file_server* retrieves the appropriate video segment from the disk via the bi-directional communication path, and sends the *media_data* to the client across the network. In addition, *MJPEG_file_server* is responsible for flow control synchronisation with the client to ensure that the client's media buffer is within the lower and upper buffer mark. This is

accomplished by having the client process periodically feedback the buffer information and the rate in which the media is being played (via *client_control*). This feedback information is processed by the file server and the appropriate control actions are invoked to ensure correct synchronisation. For example, *if server_control_unit* detects a progressive increase in the client's media buffer such that the buffer's upper threshold has been reached, a control message is sent immediately to the *MJPEG_file_server* to reduce the media transmission rate.

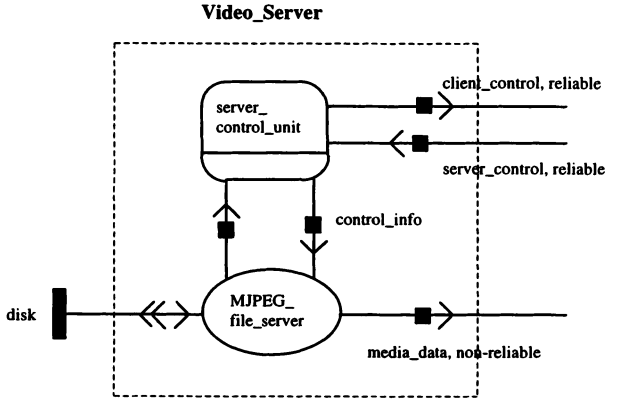

Figure 7 Video Server.

Figure 8 shows the decomposed design of the *video_client* process.

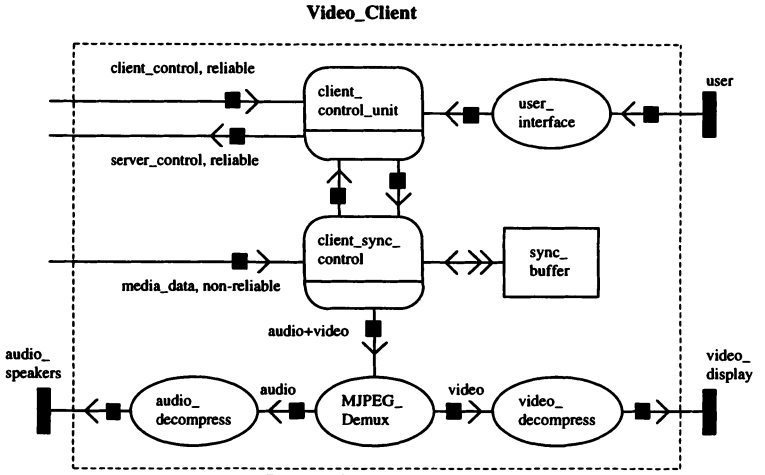

Figure 8 Video Client.

The user interface function is responsible for the processing of user's input. Based on the command received from the user interface function, *client_control_unit* is responsible for invoking the appropriate actions for the local and remote server processes. For example, upon

receiving a pause command from the user, the control unit issues a control message to the local client synchronisation unit to stop processing. Subsequently, a control message is also issued to inform the remote server to temporarily cease sending any further media data so as to prevent buffer overflow at the client. The client synchronisation control unit (*client_sync_control*) is responsible for synchronous playout of the continuous media data. For each block of interleaved audio and video frame, based on the frame's playout rate, the synchronisation control unit is responsible to ensure the synchronous playout of the continuous media. In addition, the unit is also responsible for the synchronisation between the client and server to ensure that proper flow control is enforced. The interleaved audio and video data received from the synchronisation control unit is demultiplexed by the *MJPEG_Demux* function unit. The demultiplexed video and audio data unit are sent to the respective decompression functions for media decompression and subsequent playout.

5 SPECIFYING PROCESS OBJECT BEHAVIOUR USING BSL

The dynamic behaviour of the primitive (lowest-level) process objects is specified using a behavioural specification language (BSL). This language contains constructs for describing sequential program structures and includes sequences, iterations, selections, and guarded selections. Primitive send and receive operations for various kinds of communication path types are also included. In addition, dynamic creation, deletion of processes and communication paths can be specified. For a full description of the syntax and usage of BSL, see (Gorton.1995).

The specification of all primitive process objects is beyond the scope of this paper. We will however demonstrate the use of BSL by providing partial descriptions of the behaviour of two primitive process objects taken from Figure 6 and Figure 8. BSL descriptions should fully describe the relative order of inputs and outputs for a given process object: internal processing of inputs can be left unspecified, at least initially. This gives a skeleton specification of the process object's interactions which is sufficient to simulate and verify the system's behaviour (Russo.1995).

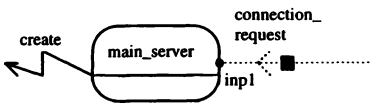

Figure 9 'Main_Server' From Multithreaded 'Server' Object.

```
PROCESS MAIN_SERVER
      SEQUENCE
         WHILE TRUE
            -- main_server sets up transactional communication path
            setup (connection_request)

            -- take input from asynchronous path Connection_request
            -- i is the instance of the client object, -1 denotes
      -- untimed comm.
            receive (inp1, i, -1, connection_request)

            --create helper process after receiving
--Connection_request
```

```
            create (video_server)
         ENDWHILE
      ENDSEQUENCE
END PROCESS
```

The main_server process object has a simple BSL description. It simply waits for a client process to connect via the *connection_request* dynamic path, and creates a *video_server* process object to service further requests. *MJPEG_Demux* specifies that for each input message received on the *audio+video* path, a corresponding output message is sent to both output paths. These completed behavioural descriptions can then be translated into programming languages using APIs supported by the underlying platform.

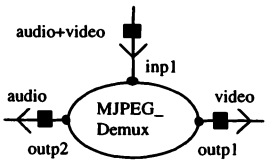

Figure 10 'MJPEG_Demux' From 'Video_Client'.

```
PROCESS MJPEG_DEMUX
      SEQUENCE
         WHILE TRUE
            receive (inp1, i, -1, audio+video)
            -- separates the two types of medium
            as-send (outp1, i, -1, video)
            as-send (outp2, i, -1, audio)
         ENDWHILE
      END SEQUENCE
END PROCESS
```

6 IMPLEMENTATION SUPPORT

Windows NT has been chosen to be the implementation platform. The Application Programming Interface (API) Win32 provides a rich set of function calls that enables the development of distributed multimedia systems. Table 1 gives a summary of Win32 API calls that supports the implementation of any Ext-PARSE designs.
Note: actual Win32 and socket function calls are in *italic*.

7 CONCLUSION AND FURTHER WORK

A number of software design techniques for distributed systems exist. For example: Booch (Booch.1991), CODARTS (Gomaa.1993), HOOD (Robinson.1992), MOOD (Lee.1994), and PROOF (Yau.1994). However, they do not exhibit the ability to easily capture the dynamic, distributed system structures and complex synchronisation requirement frequently occurring in multimedia applications.

This paper has demonstrated the suitability of PARSE for the design of distributed multimedia systems. The complete PARSE process graph notation enables the dynamic system

structure to be captured precisely, and succinctly. Multithreaded process objects, and the hierarchical process object structuring promotes a high level of design abstraction. In typical client-server systems, there are often multiple instances of the server object created for servicing multiple clients' requests. The multithreaded process object in PARSE allows the designer to specify this dynamic interaction between client and server easily. The information captured in the textual annotations, such as path restriction and dynamic process object replication ranges, further aids the eventual implementation.

Ext-PARSE process graph feature	**Win NT/Win32 API Equivalent**
Dynamic process objects • create dynamic process objects • delete dynamic process objects: - aging - assassination - suicide • data server objects	Processes, threads • *CreateThread, CreateProcess, CreateRemoteThread* • thread termination: - return from function - *TerminateThread* - *ExitThread* • thread: Thread-Local Storage, shared memory: synchronisation objects
Communication Paths • creation • synchronous • asynchronous • bi-directional synchronous • broadcast	NamedPipes • *CreateNamedPipe* (an instance of a named pipe is always deleted when the last handle to the instance of the named pipe is closed) • blocking send, blocking receive, overlapped mode not enabled, *pipe-specific mode = PIPE WAIT* • non-blocking send, blocking receive, *pipe open mode = overlapped, pipe-specific mode = PIPE WAIT* • blocking send, blocking receive, *pipe open mode = PIPE ACCESS DUPLEX* • multiple instances of named pipes, or mailslots
Network Communication Paths • creation • synchronous • asynchronous • bi-directional • broadcast	Socket facility provided in winsock.h • creating new socket, *socket, bind* • reliable path with blocking send, blocking receive, *socket_type = SOCK_STREAM* • non-blocking send, blocking receive, for reliable path *socket_type = SOCK_STREAM*, non-reliable path *socket_type = SOCK_DGRAM* • sockets are inherently bi-directional • multiple instances of sockets. In some networks, eg. ATM, efficient implementation can be achieved using network multicast facility.
Path Constructors • unspecified (not used in primitive processes) • concurrent • non-deterministic • deterministic	Input selection methods • no need to consider this in Win NT • independent threads carrying out work separately • NamedPipes non-deterministic by default • *WaitNamedPipe*

Table 1 Mapping between Ext-PARSE and Win32

The PARSE graphical design method would seem intuitive to use and easily comprehensible. In this project, the lead designer produced a PARSE design for the system in

a matter of days, with no previous PARSE exposure. We intend to quantitively explore this issue further through additional, controlled case studies.

The verification of designs is the issue that needs to be considered. Currently, static PARSE designs can be easily translated into Petri Nets (Gorton.1994), and subsequent design verification can be carried out automatically. However, this technique can not be used with PARSE designs with dynamic properties, since any reachability analysis would lead to an infinite state system. It is anticipated that an alternative verification technique would be employed. The works of (Birkinshaw.1995) and (Milner.1980) may offer possible solutions.

References

Birkinshaw, C.I. and Croll, P.R. (1995) Modelling the Client-Server Behaviour of Parallel Real-Time Systems Using Petri Nets, *Proc. 28th Ann. Hawaii Int'l Conf. System Sciences, Parallel Software Engineering Minitrack, Vol.2: Software Technology*, IEEE Computer Society Press, Calif., 339-48.

Booch, G. (1991) Object-oriented Design With Applications. Benjamin/Cummings.

Boudec, J.Y.L.(1992) The ATM: A Tutorial, *Computer Networks and ISDN Systems*, Vol.24, 279-309.

Friesen, J.A., Yang, C.L. and Cline, R.E. (1995) DAVE: A Plug-and-Play Model for Distributed Multimedia Application Development, *IEEE Parallel and Distributed Technology*, Vol.3, No.2, Summer, 22-8.

Furht, B. (1994) Multimedia Systems: An Overview, *IEEE Multimedia*, Vol.1, No.1, Spring, 37-50.

Gibbs, S.J. (1995) Multimedia Programming: objects, environments, and frameworks, ACM Press.

Gomaa, H.(1993) Software Design Methods for Concurrent and Real-Time Systems, Addison-Wesley.

Gorton, I., Chan, T.S. and Jelly, I.E. (1994) Engineering high quality parallel software using PARSE, in *Lecture Notes in Computer Science 854*, Proceedings of CONPAR-VAPP 94, Linz, Austria, September, 381-92, Springer-Verlag.

Gorton, I., Gray, J.P. and Jelly, I.E. (1995) Object-based Modelling of Parallel Programs, *IEEE Parallel and Distributed Technology*, Vol.3, No.2, Summer, 52-63.

Jadav, D. Choudhary, A. (1995) Designing and Implementing High-Performance Media-on-Demand Servers, *IEEE Parallel and Distributed Technology*, Vol.3, No.2, Summer, 29-39.

Lee, P.J., Chen, D.J. and Chung, C.G. (1994) An Object-oriented Modelling Approach To System Design, *Information and Software Technology*, Vol.36, No.11, 683-94.

Liu, A. and Gorton, I. (1996) Modelling Dynamic Distributed System Structures in PARSE, to appear in *4th Euromicro Workshop on Parallel and Distributed Processing*, Braga, Portugal, January.

Microsoft Corp. (1991) Microsoft Windows Multimedia Programmer's Workbook.

Milner, R. (1980) A Calculus of Communicating Systems, in *Lecture Notes in Computer Science Volume 92*, Springer-Verlag.

Minzer, S.E. (1989) Broadband ISDN and Asynchronous Transfer Mode (ATM), *IEEE Communications Magazine*, September, 17-24.

Robinson, P.J. (1992) HOOD, Prentice-Hall, 1992.

Russo, S., Savy, C., Jelly, I.E. and Collingwood, P.C. (1995) Petri Net Modelling of PARSE Designs, *Joint Technical Report*, Computing Research Centre, Sheffield Hallam University/ Departmento di Informatica e Sistemistica, University of Naples.

Taylor, H., Chin, D. and Knight, S. (1995) The Magic Video-on-Demand Server and Real-Time Simulation System, *IEEE Parallel and Distributed Technology*, Vol.3, No.2, Summer, 40-51.

Wallace, G.K. (1991) The JPEG Still Picture Compression Standard, *Communications of ACM*, Vol.34, No.4, April, 30-44.

Yau, S.S. and Bae, D-H. (1994) Object-oriented and Functional Software Design for Distributed Real-time Systems, *Computer Communications*, Vol.17, No.10, October, 691-8.