SISTEMA DE DISTRIBUCIÓN Y ADMINISTRACIÓN DE PROCESOS EN RED

JOSÉ DAVID PARRA LÓPEZ

FRANCISCO LUIS RODRÍGUEZ VILLABONA

UNIVERSIDAD NACIONAL DE COLOMBIA

Facultad de Ingeniería

Departamento de Sistemas

Santa Fe de Bogotá

1999

SISTEMA DE DISTRIBUCIÓN Y ADMINISTRACIÓN DE PROCESOS EN RED

JOSÉ DAVID PARRA LÓPEZ

FRANCISCO LUIS RODRÍGUEZ VILLABONA

Proyecto de Grado presentado como requisito parcial para optar al título de Ingeniero de Sistemas
Director: GABRIEL MAÑANA GÜICHÓN

UNIVERSIDAD NACIONAL DE COLOMBIA

Facultad de Ingeniería

Departamento de Sistemas

Santa Fe de Bogotá

1999

Nota de Aprobación

_____

_____

_____

_____

_____

Director del proyecto

GABRIEL MAÑANA GÜICHÓN.

_____

Jurado

_____

Jurado

Santa Fe de Bogotá, octubre de 1999

Comentarios

AGRADECIMIENTOS

A nuestros padres, que tuvieron el cuidado y el atrevimiento de educarnos. En verdad que les agradecemos.

También, a nuestros compañeros de estudio —en especial al compañero de tesis— con quienes sacamos adelante un montón de proyectos y esperamos sacar otro montón.

A los demás amigos, amigas y familiares que hicieron barra y pusieron su parte, así no fuera su intención ni se dieran cuenta.

Y a los elementos distractores que ayudaron a mantenernos despiertos y a hacer más llevadera la construcción de todo esto: Metallica, Futurama, White Zombie, los Simpsons, Korn, South Park, etcétera.

José David y Francisco Luis

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS

GLOSARIO

ACTIVEX: Es un conjunto de tecnologías para interoperabilidad independiente del lenguaje, que permite que los componentes de *software* se escriban en diferentes lenguajes para trabajar conjuntamente en ambientes de red. Los componentes centrales son COM y DCOM. Estas tecnologías están licenciadas por la organización de estándares *The Open Group* y están siendo implementadas sobre múltiples plataformas, principalmente Win32.

AUTOMATION: Tecnología de comunicación entre aplicaciones basada en interfaces de despacho. Estas interfaces tienen la propiedad de que pueden ser accedidas sin conocer los nombres o los parámetros de las funciones que componen las interfaces, pues esta información se obtiene en tiempo de ejecución. Los controles ActiveX y muchas aplicaciones utilizan Automation para exponer su funcionalidad.

API (*Application Programming Interface* – Interfaz para la programación de aplicaciones): Es un conjunto de funciones y definiciones que una aplicación usa para hacer requerimientos y cargar servicios ofrecidos por el sistema operativo o por una tecnología de programación determinada. Existen APIs para muchas clases de servicios, como computación gráfica, manejo de periféricos, acceso remoto a aplicaciones, etc. Algunas de las APIs empleadas en este proyecto son GDI, DirectX, OpenGL, etc.

ATL (*Active Template Library* – Librería activa de plantillas): Es un *framework* utilizado para construir componentes de *software* que utilicen COM y DCOM. El propósito de ATL es permitir la construcción de componentes pequeños, rápidos y eficientes.

AVI (*Audio – Video Interlaced* – Video y audio entrelazados): Es un formato de archivo que permite almacenar sonidos y videos. Es el formato estándar de video (pero no el único) en las aplicaciones Win32.

BUFFER: Área de memoria o de disco utilizada para almacenar datos temporalmente. Usualmente, los datos almacenados en un *buffer* se pueden acceder más rápidamente que en su sitio de almacenamiento original, y son accedidos al mismo tiempo por varias aplicaciones.

CLIENTE / SERVIDOR: Es un modelo de computación donde las aplicaciones que corren en un computador (clientes) hacen solicitudes de información o de operaciones a computadores remotos (servidores).

COM (*Component Object Model* – Modelo de componentes): Es un modelo de programación orientado a objetos, que permite la construcción de aplicaciones que interactúan entre sí a través de interfaces.

DCE (*Distributed Computing Environment* – Ambiente de computación distribuida): Es un estándar definido por un grupo de compañías de desarrollo de sistemas, que se usa para la construcción de tecnologías para sistemas distribuidos. RPC, DCOM y otras arquitecturas de construcción de sistemas distribuidos se basan en DCE.

DCOM (*Distributed Component Object Model* – Modelo de componentes distribuidos): Es una extensión de COM, que adiciona la facilidad de construir componentes y aplicaciones distribuidas sobre redes, de manera transparente y eficiente.

DIRECTX: API que provee servicios de computación gráfica y de multimedia a las aplicaciones Windows. Posibilita el manejo de gráficas, audio, video, periféricos especiales para juegos, etc., y permite añadir estas características a aplicaciones de productividad, juegos y páginas Web.

DLL (*Dynamic Link Library* – Librería de enlace dinámico): Es un archivo que expone un conjunto de funciones, que pueden ser llamadas por otras aplicaciones. No es una aplicación por sí misma, funciona únicamente como repositorio de funciones. Un componente COM puede ser construido como una aplicación o como una librería de este tipo.

FARM: Es un grupo de dos o mas máquinas intercomunicadas por una red, destinadas a la ejecución de un proceso común, utilizando los mismos datos o datos diferentes para cada equipo en el *farm.*

FRAME (marco): Es uno de los fotogramas o imágenes que componen una animación. La cantidad de *frames* por segundo en una animación determinan la suavidad del movimiento de los objetos dentro de la misma.

FRAMEWORK (marco de trabajo): Es un conjunto de definiciones de *software* que permiten la construcción de aplicaciones. MFC y ATL son ejemplos de *frameworks.*

GDI (*Graphical Device Interface* – Interfaz gráfica de dispositivo): API incluida en los sistemas operativos Win32 y que provee servicios gráficos básicos. Es usada por la gran mayoría de las aplicaciones Windows.

IDL (*Interface Definition Language* – Lenguaje de definición de interfaces): Es un lenguaje que permite definir las interfaces que un objeto construido bajo

RPC expone, sin importar el lenguaje o la tecnología de programación utilizadas en su implementación. Los objetos construidos a partir de COM y DCOM deben, obligatoriamente, definir sus interfaces utilizando este lenguaje.

INTERFAZ: En COM, es un conjunto de funciones relacionadas, que un objeto expone a través del sistema operativo para que puedan ser usadas por otros objetos.

INTERFAZ DE DISCO: En lo que respecta al *hardware*, la interfaz de disco es el mecanismo de comunicación entre el disco duro y la tarjeta principal del computador. Los sistemas más usados comúnmente para esta tarea, en su orden de velocidad, son PCI, EIDE e IDE.

LPC (*Local Procedure Call* – Llamada a un procedimiento local): Técnica que se usa para invocar funciones expuestas por un objeto que se encuentra fuera del espacio de proceso del objeto invocador, en el mismo computador.

MFC (*Microsoft Foundation Classes* – Clases fundamentales de Microsoft): *Framework* de propósito general que permite la construcción de aplicaciones Windows de diversos tipos, con o sin interfaz de usuario, con o sin modelo documento – vista, componentes COM, controles ActiveX, servicios y programas residentes, aplicaciones de consola (DOS), etc.

MODELO DOCUMENTO - VISTA: Es un modelo de diseño e implementación de *software*, en el cual los datos que son manejados y almacenados por una aplicación son agrupados en estructuras llamadas documentos. Éstos son independientes de la forma en que estos datos son presentados al usuario, lo que se hace a través de otras estructuras, conocidas como vistas.

MULTITHREADING: Es la habilidad de un programa de ejecutar simultáneamente múltiples fragmentos de código de una misma aplicación (*threads* o hebras de ejecución) simultáneamente.

OPENGL: API que provee servicios de computación gráfica para diversas plataformas, como Win32, Macintosh, UNIX, etc.

PDC (*Primary Domain Controller* – Controlador Primario de Dominio): En Windows NT, es el equipo encargado de manejar un dominio, que es como se denomina a un conjunto de computadores que comparten el mismo esquema de administración. Gobierna los permisos de acceso a aplicaciones y datos, los usuarios, la seguridad, la auditoría, el manejo de la red, el acceso a los recursos de red, etc.

REGISTRY: Es el nombre técnico con el que se denomina la base de datos central de configuración de un equipo con sistema operativo Windows. En esta base de datos se almacena la información de *hardware*, aplicaciones, usuarios, seguridad, recursos de red a los que tiene acceso, etc.

RENDERING: Es el proceso que consiste en generar las imágenes o *frames* que conforman una escena o una animación, a partir de los datos que la definen. Más precisamente, consiste en crear una proyección bidimensional de un modelo tridimensional basado en una estructura de datos, a la que además se han aplicado una serie de técnicas de eliminación de superficies ocultas, mapeo de texturas, iluminaciones, etc., que le añaden realismo.

RPC (*Remote Procedure Call* – Llamada a un procedimiento remoto): Técnica que se usa para invocar funciones expuestas por un objeto que se encuentra fuera del espacio de proceso del objeto invocador, y además en otro computador.

SMP (*Symmetric Multiprocessors* – Multiprocesamiento simétrico): Es un computador único compuesto por un conjunto de procesadores, donde cada uno de ellos tiene las mismas capacidades, está interconectado con lo demás y accede a un bloque de memoria global.

THREAD (hebra): Es un fragmento de código definido dentro de un programa, que se ejecuta independientemente de la secuencia principal de código del programa que lo contiene.

UML (*Unified Modeling Language,* Lenguaje unificado de modelamiento): Es un lenguaje para especificar, construir, visualizar y documentar los componentes de un sistema basado en *software*. Fue desarrollado por un consorcio integrado por empresas dedicadas al desarrollo de programas, como Rational Software, Microsoft, Hewlett-Packard, Oracle, IBM, etc., para establecer un estándar de modelamiento y documentación de *software.*

WIN32: Nombre común para los sistemas operativos Windows de 32 bits. Entre ellos se encuentran Windows 95, Windows 98, Windows NT y Windows 2000.

RESUMEN

Antares es un sistema que permite hacer uso del poder computacional conjunto que ofrece una red existente de microcomputadores con el sistema operativo Microsoft Windows (95/NT). Su arquitectura permite que estos computadores se comporten lógicamente como una sola maquina virtual paralela con el poder combinado, y de esta manera solucionar mucho mas rápidamente problemas de alta complejidad computacional.

Y aunque se eligió probar sus beneficios en la solución de un problema de Computación Grafica, el sistema puede ser fácilmente utilizado en resolver casi cualquier problema que permita ser dividido y repartido en unidades atómicas independientes.

INTRODUCCIÓN

A pesar de que el mercado de sistemas de información siempre está a la expectativa de obtener equipos de cómputo mas rápidos y poderosos —y estas expectativas son siempre satisfechas—, la velocidad de tales equipos no es siempre el asunto a considerar. Cualquier persona que haya estado involucrada con el mundo de la computación sabe que es increíblemente difícil mantener un computador permanentemente ocupado. Aunque la sensación usual es percibir que el usuario siempre está esperando al computador, lo que típicamente ocurre es exactamente lo contrario: el computador está esperando al usuario. Si se observa la proporción de uso de una red de computadores, se encontrará que el porcentaje de "ocupación" de los equipos que se encuentran en la red es menor al 10% del tiempo total.

Por otro lado cualquier inversión en *hardware* que se haga, al poco tiempo estará desactualizada, tal como lo dicta la ley de Moore, que se está cumpliendo puntualmente desde hace tres décadas. Para complicar las cosas aún más, fácilmente se observará que estamos utilizando nuestra inversión solo un 10% del tiempo, mientras que el resto del tiempo se pierde en entradas por teclado o activación de protectores de pantalla.

Para labores que involucran un cálculo intensivo (como por ejemplo: complejos modelos del clima, análisis de materiales, simulaciones de accidentes y de comportamiento de objetos y maquinaria, animaciones para el cine, etc.) no es

suficiente contar con los aspectos relacionados con el poder de los equipos de cómputo. Usualmente, un solo equipo, por más poderoso que sea, no puede realizar todo el trabajo por su propia cuenta, y lo pocos equipos que pueden realizar tal trabajo —supercomputadores— tienen un costo que definitivamente los aleja del alcance de la mayoría de las personas y las aplicaciones que los necesitan.

La solución más obvia consiste en reunir el poder de procesamiento de varios computadores pequeños, en un arreglo que es conocido en el medio como *farm*. En pocas palabras, un *farm* es un conjunto de computadores conectados a través de un sistema de comunicación, que cuenta con un esquema de compartición de recursos (sobre todo de almacenamiento) y con un método para ejecutar y administrar aplicaciones que corren sobre todos los equipos. Pero tampoco basta con conectar los equipos a través de una red, pues el trabajo que se desea hacer no se va a distribuir por su cuenta en los equipos que hacen parte del *farm*. También es muy importante contar con un poderoso sistema de administración y despacho de trabajos. Esto nos remite directamente al concepto de sistemas distribuidos, el cual es empleado constantemente a lo largo de este proyecto.

Y a pesar que se cuente con un número determinado de estaciones dedicadas al procesamiento, es obvia la necesidad de poder hacer uso de la gran cantidad de poder computacional que reside en las estaciones de trabajo de la red, que normalmente no es utilizado en su totalidad. La situación ideal consiste en manejar los recursos computacionales conectados en red como un gran sistema con el poder conjugado de todos estos. Cualquier usuario podría correr trabajos sobre el *farm* como si éste fuera una extensión transparente de una sola máquina de escritorio, es decir, como un sistema fácil de usar, confiable y eficiente. De esta forma, el usuario final no sabe (y si pudiera saber no le importaría) dónde o cómo se procesa el trabajo. Este acercamiento, además de las ventajas ya mencionadas, es mucho mas económico que un único sistema de procesamiento masivo con iguales características y desempeño.

Este esquema es muy utilizado comercialmente y por lo general incluye una estación maestra que administra los trabajos asignados al *farm* y distribuye la ejecución de cada uno de ellos sobre las estaciones, además de proveer el almacenamiento compartido necesario para alojar los resultados. Uno de los ejemplos mas notables y conocidos de la utilización de esta arquitectura lo constituye el *farm* de 117 estaciones de alto desempeño Sun, que construyó la firma Pixar para el desarrollo de la primera película generada por computador en su totalidad: "*Toy Story*".

Desafortunadamente, no es fácil contar con un arreglo de computadores tan grande y poderoso. Y en ocasiones, no se requiere llevar a cabo un trabajo de la calidad y tamaño del que hizo Pixar, sino en generar animaciones más sencillas empleando las tecnologías de computación gráfica más populares en el mercado. Es importante anotar que para producir "*Toy Story*" se empleó un sistema de animación conocido como RenderMan, el cual es propietario de Pixar y funciona sobre una plataforma de computación también propietaria, y por consiguiente tiene requerimientos de sistema, de programas y de entrenamiento de los usuarios que también lo alejan del alcance de la mayoría de las personas interesadas en las animaciones por computador. Por otro lado, existe un gran interés en el medio para construir y usar un sistema que permita emplear un *farm* para distribuir la ejecución de trabajos de diversas características y tamaños.

El objetivo principal de este proyecto es presentar un sistema que pueda satisfacer, aunque sea en parte, tales expectativas. Este sistema, bautizado como *Antares*, es una aplicación de *software* distribuido en una red de computadores que tiene como fin mostrar las bondades del procesamiento distribuido en la disminución de los tiempos de procesamiento de trabajos de alta carga computacional, sin importar la clase de trabajo.

Su arquitectura está diseñada para el procesamiento de tareas que se puedan beneficiar de ser ejecutadas en una ambiente distribuido, y la plataforma sobre la cual fue implementada *Antares* aprovecha la gran proliferación de estaciones de trabajo de mediano y alto desempeño que corren con sistemas operativos Windows. Se decidió entonces hacer uso del poder computacional que estas máquinas representan en su conjunto y ponerlo al servicio de tareas computacionalmente muy complejas que hubieran exigido un sistema dedicado de alto rendimiento y costo prohibitivo.

La descripción del proceso de construcción y prueba de *Antares* está contenida en este documento. En primer lugar, se presenta todo el conocimiento que fue considerado y utilizado para el proceso de análisis del problema que representa construir un sistema como el que se ha propuesto. Una vez hecho esto, se continúa con la explicación del diseño del sistema que pretende solucionar el problema en cuestión, y luego se muestran los detalles de la implementación del sistema en un conjunto de componentes de *software*. Finalmente, se evalúa el desempeño del sistema con varios trabajos de prueba —que son animaciones— y se presentan los resultados y las reflexiones producto de tales pruebas.

Se ha escogido realizar animaciones por computador debido al gran interés que generan y a la disponibilidad de sistemas avanzados de computación gráfica, que permiten, al mismo tiempo, construir animaciones con un realismo bastante aceptable en computadores personales y sistemas que permitan tratar la generación de tales animaciones como un proceso distribuido.

1. MARCO TEÓRICO

1.1 SISTEMAS DISTRIBUIDOS

1.1.1 Por qué un sistema distribuido

El desarrollo de los sistemas de computación se ha caracterizado, a lo largo de los años, por la forma en que los trabajos eran procesados y por la relación que tenían con sus usuarios. Estas han sido las dos características que han marcado y jalonado su evolución. A continuación se hará una breve retrospectiva para hacer énfasis en esta idea.

- En la década de 1950, los computadores eran procesadores seriales, es decir, sólo podían ejecutar una tarea al mismo tiempo. Un operador los manejaba desde una consola que recibía y presentaba datos desde y hacia una cinta magnética o una tarjeta perforada, y no existía el concepto de usuario como tal, pues estos equipos estaban muy alejados de la vida y las necesidades cotidianas.
- A continuación, en la década de 1960, los trabajos que tenían características de cálculo similares y empleaban los mismos recursos eran acumulados, para que todos fueran procesados por el computador, reduciendo así su tiempo de inactividad. Este es el concepto que más tarde sería denominado procesamiento por lotes.
- En los siguientes años (1970-1980) se llegó a la compartición de capacidad de procesamiento y de recursos, es decir, que varios usuarios, fueran éstos personas u otros computadores, pudieran acceder a la misma máquina, y distribuirse entre ellos el tiempo de procesamiento. De esta forma, era

posible aprovecharla en mayor cuantía, y el aumento de usuarios hacía que los computadores comenzaran a salirse de ese ambiente cerrado y exclusivo en el que se encontraban. Por otro lado, aunque este esquema de funcionamiento permitió el desarrollo de sistemas que podían ejecutar varias instrucciones al mismo tiempo, aún subsistían los problemas de rendimiento y confiabilidad inherentes al uso de un equipo como un objeto aislado.

- Durante la década de 1980 se presentó la aparición y la popularización de la computación personal. Los equipos se tornaron más baratos, y con recursos de almacenamiento, ingreso y presentación de datos propios. Ya era posible que cada usuario contara con su propia máquina, dedicada a resolver sus necesidades específicas.
- Finalmente, en la década actual se ha presentado el desarrollo y la utilización extensiva de los sistemas distribuidos, los cuales consisten en una serie de computadores conectados a través de una red, que trabajan en conjunto para llevar a cabo las tareas que normalmente eran asignadas a un solo computador. Esto fue posible gracias al abaratamiento de los recursos de procesamiento y almacenamiento de información, y a la excelente relación precio - desempeño que estos sistemas presentan, apoyados en los más recientes desarrollos en tecnologías de redes.

Los sistemas distribuidos pueden tener diversas configuraciones físicas: un grupo de computadores personales conectados a través de una red, un conjunto de estaciones que comparten sistemas de memoria y de archivos, o un grupo de procesadores a los cuales son asignadas las tareas provenientes de las terminales, las cuales no están asociadas directamente a un procesador en particular. Una característica común de todas estas configuraciones es que el usuario final utiliza el sistema completo como si fuera un solo computador.

La anterior es sólo una de las motivaciones que condujeron al desarrollo de sistemas distribuidos; entre otras pueden mencionarse las siguientes:

- Existen aplicaciones que son distribuidas por naturaleza, y además el almacenamiento de la información también es distribuido en la mayoría de los casos. Por ejemplo, la información de las distintas subsidiarias de una compañía se encuentra almacenada en los equipos de cada una de las sedes, y es necesario poder acceder a toda esta información para realizar operaciones y análisis globales.
- El paralelismo inherente a los sistemas distribuidos permite obtener un mejor desempeño en el procesamiento de datos, disminuyendo la posibilidad de que se presenten cuellos de botella y otros problemas que pueden dificultar o detener tal procesamiento.
- Un sistema distribuido puede soportar eficientemente la compartición de información y de recursos entre múltiples usuarios, sin importar dónde se encuentren. Además, cada usuario puede esperar un buen rendimiento del sistema como consecuencia de la asignación correcta de recursos.
- Es posible modificar y extender un sistema distribuido para que pueda acometer nuevas tareas, o atender las actuales con mayor eficiencia. El sistema puede adaptarse a los cambios de ambiente y de necesidades, sin necesidad de interrumpir su operación.
- Debido a la presencia de múltiples unidades de procesamiento y almacenamiento de datos, un sistema distribuido bien construido es resistente a fallas y está siempre disponible para sus usuarios. Es posible lograr que el sistema continúe trabajando si alguno de sus componentes deja de funcionar.

1.1.2 Organización física

La arquitectura tradicional de un computador, conocida como Von Neumann, se compone de la unidad central de procesamiento (CPU), la memoria, y los dispositivos de entrada y salida de información (I/O). Los programas, que son almacenados en la memoria, son ejecutados secuencialmente por el procesador. Los datos de entrada y salida también se guardan en la memoria, y por medio de los dispositivos I/O estos datos fluyen desde y hacia el usuario.

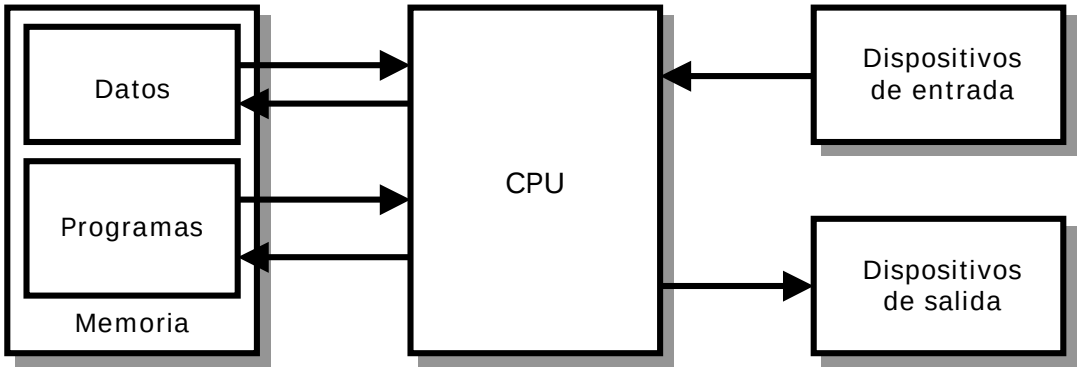


Figura 1. Organización Von Neumann de un computador.

Un sistema distribuido puede organizarse, con respecto a la ubicación de sus componentes, de dos formas. Una de ellas consiste en unir una serie de procesadores por medio de un sistema de comunicación, y asignar un bloque de memoria al cual todos pueden acceder (Figura 2), mientras que en la otra varios equipos, cada uno de ellos con su propio procesador y espacio de memoria, se comunican entre sí a través de una red de interconexión, por donde fluyen mensajes que los equipos se pasan entre sí para coordinar su funcionamiento (Figura 3).

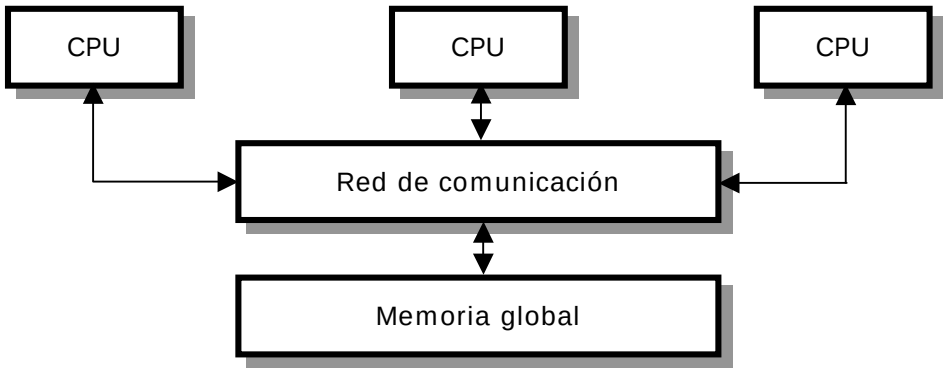


Figura 2. Sistema distribuido con un bloque de memoria global

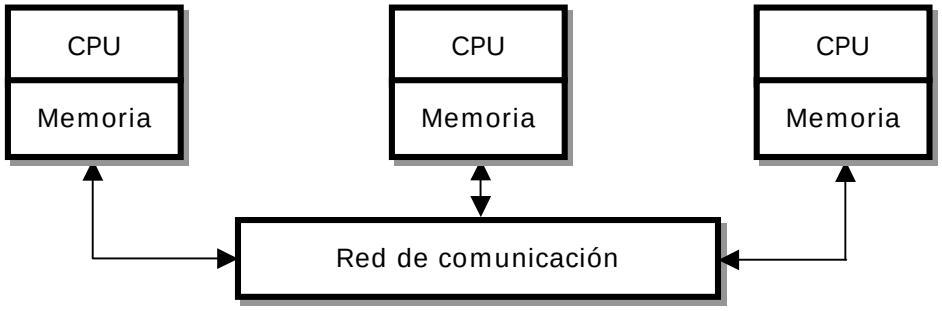


Figura 3. Sistema distribuido con un bloque de memoria para cada procesador

Los ejemplos típicos de cada una de estas organizaciones son un computador con múltiples procesadores, en el primer caso, y una red de PCs en el segundo. El acceso al bloque global de memoria, o la comunicación entre los equipos de la red son servicios que el usuario debe implementar o ya son proveídos por el sistema operativo del sistema distribuido, si hay alguno disponible. Si es necesario programar tales servicios, se cuenta con estructuras como semáforos y secciones críticas para proteger el acceso a memoria, y con estándares de definición e intercambio de mensajes para implementar la comunicación entre máquinas.

1.1.3 Definición

Al hablar de sistemas distribuidos es frecuente encontrar los siguientes términos: red de computadores, paralelismo, concurrencia, distribución, etc. El campo de los sistemas distribuidos es relativamente nuevo, y por ello no hay un acuerdo general para definir todos estos términos, los cuales son usados indistintamente con frecuencia. Como punto de partida, se pueden definir estos términos de la siguiente forma, donde cada uno de ellos engloba a los precedentes:

- Paralelo: Significa que se realiza el mismo conjunto de operaciones sobre una serie de datos, en un conjunto de computadores.
- Concurrente: Las operaciones que se hacen en los distintos equipos que conforman el sistema pueden realizarse independientemente las unas de las otras.
- Distribuido: El costo de realizar un cálculo o proceso está relacionado directamente con la transmisión de secuencias de datos y de control entre los componentes del sistema.

Teniendo en cuenta lo anterior, un sistema donde los equipos que lo componen no interactúan estrechamente entre sí puede ser denominado como un sistema paralelo. Un sistema concurrente es entonces aquel donde las operaciones, aunque pueden ser distintas para cada equipo en un momento dado, no están directamente relacionadas entre sí.

En el caso contrario, donde todos los equipos se comunican e intercambian información intensivamente mientras ejecutan las operaciones que les son asignadas, se habla de un sistema distribuido.

Precisando la definición anterior, se puede descomponer el grado de distribución por medio de las siguientes dimensiones, las cuales definen los componentes básicos de un sistema distribuido: *hardware* (procesamiento), datos, y control. Las escalas relacionadas con cada una de estas dimensiones se refieren a continuación.

***Hardware*:**

1. Un solo procesador, con memoria y sistema de entrada y salida exclusivas.
2. Varios procesadores, con un sistema global de memoria y otro de entrada y salida.
3. Varios procesadores, cada uno de ellos con sus propios sistemas de memoria y de entrada y salida.

Datos:

1. Almacenamiento único de los datos en un sitio determinado.
2. Replicación de los datos en cada uno de los equipos del sistema.
3. Distribución de los datos entre los equipos, con una copia de los datos almacenada en un sitio determinado.
4. Distribución de los datos entre los equipos, sin copia de los datos.

Control:

1. Sitio único y centralizado de control, desde donde se gobierna todo el sistema.
2. Sitio centralizado de control, que cambia de equipo con el tiempo.
3. Varios puntos de control similares ubicados en el sistema.
4. Varios puntos de control, diferentes entre sí, ubicados en el sistema.

A partir de estas dimensiones, se puede dar una definición estricta de un sistema distribuido. Este sólo puede considerarse como tal si su dimensión de *hardware* es la No. 3, la de datos es la número 4, y la de control es la número 3 o 4. Un sistema que solamente tenga distribuido el proceso, los datos o ambos se podría denominar parcialmente distribuido. No tiene sentido pensar en un sistema que tenga solamente distribuido el control. Todas estas ideas se encuentran representadas en la Figura 4.

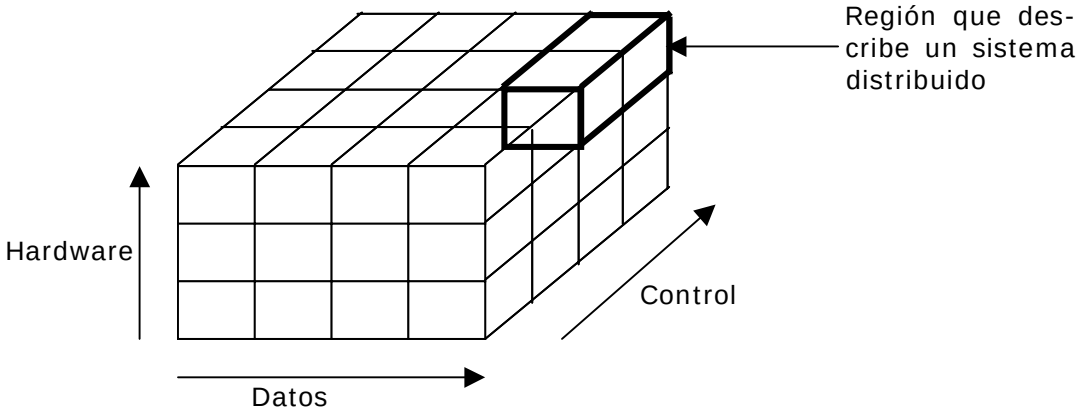


Figura 4. Cubo que representa las tres dimensiones que representan el grado de distribución de un sistema.

1.1.4 Aplicaciones y estándares

Los sistemas distribuidos son utilizados para ejecutar diversas clases de aplicaciones, las cuales son ejecutadas más favorablemente en sistemas de este tipo que en otros, como por ejemplo sistemas con un solo procesador, o un conjunto de procesadores con un bloque compartido de memoria. Algunas de estas aplicaciones son:

- Paralelas y de alto desempeño: En principio, las aplicaciones paralelas pueden funcionar bien en equipos con varios procesadores, pero no es posible escalarlos fácilmente para alojar nuevos procesadores. Por esto es más conveniente usar un sistema distribuido.
- Tolerantes a fallas: Los sistemas distribuidos son más confiables, pues las unidades de proceso (computadores) que hacen parte de él son autónomas, es decir, si una de ellas falla el sistema en conjunto no ve comprometido su funcionamiento.
- Aplicaciones distribuidas por naturaleza: Muchas aplicaciones, debido a la ubicación de los datos o la disposición de los equipos y los usuarios, son inherentemente distribuidas. Por ejemplo, un sistema de procesamiento de transacciones debe verificar, consultar y modificar información que se encuentra en múltiples sitios.

Otras aplicaciones de los sistemas distribuidos son los programas de trabajo en equipo, conocidos en el medio como *groupware,* que permiten coordinar y potenciar el trabajo de múltiples usuarios. También se pueden mencionar los sistemas de videoconferencia, para la comunicación integral entre personas, y una extensión de esta tecnología, la telepresencia, que consiste en la manipulación remota de elementos o maquinaria, mediante la combinación de sistemas de control, de comunicación y de procesamiento intensivo de información.

Estas aplicaciones, y muchas otras, necesitan de un ambiente de computación que permita a los desarrolladores construir sistemas que aprovechen todos los recursos disponibles de manera eficiente, encapsulando las múltiples configuraciones de equipos, sistemas operativos, redes de comunicación, etc. Además, este modelo debe ser transparente a los usuarios finales, que deben ver el sistema distribuido como si se tratara de un solo equipo.

Uno de esos estándares es el DCE (*Distributed Computing Environment*, ambiente de computación distribuida), el cual es un OSF (*Open System Foundation,* definición abierta de un sistema) definido por un grupo de compañías de desarrollo de sistemas, el cual opera como un estándar para la construcción de tecnologías para sistemas distribuidos. DCE provee servicios de seguridad para proteger la información, de denominación para poder identificar e invocar los equipos, recursos y programas distribuidos, y un modelo escalable para la organización de usuarios. Puede correr en muchas de las plataformas de computación líderes y está orientado a satisfacer las necesidades que implican múltiples sistemas operativos y muchos equipos de diferentes configuraciones, que son características de las redes actuales.

DCE ha sido implementado de varias formas. Algunas de ellas son:

- CORBA (*Common Object Request Broker Architecture*, arquitectura común para la distribución de solicitudes de objetos): estándar desarrollado por el Object Management Group (OMG), un consorcio sin ánimo de lucro compuesto por 3Com, Canon, Data General, Hewlett-Packard, Philips, Sun Microsystems y Unisys.
- DCOM (*Distributed Component Object Model*, modelo distribuido de componentes): Desarrollado por Microsoft y basado en RPC (*Remote Procedure Call* – Llamada a procedimiento remoto), y orientado a la construcción de programas que funcionan en sistemas distribuidos.

- Enterprise Java Beans: Especificaciones que deben cumplir los programas elaborados en Java para que puedan comportarse como un sistema distribuido. Fue desarrollado por Sun Microsystems.

Para demostrar brevemente las mejoras en tiempo de procesamiento de utilizar un modelo distribuido, se ha implementado una aplicación basada en RPC (*Remote Procedure Call*) que calculara los números primos dentro de un rango dado. Esta aplicación se compuso básicamente de un cliente, que se encargaba de almacenar los números primos encontrados, y de *n* servidores distribuidos, que se encargaban de calcular si el número asignado por el cliente es primo o no. La aplicación se probó con varias configuraciones de servidores (corriendo cada uno de estos en un computador diferente conectado en red con los demás) y de rangos de números, y se obtuvieron los siguientes resultados de tiempos:

Rango	1 Servidor (seg.)	4 Servidores (seg.)	Rata de ganancia
1-1000	35	40	0.88
1-100,000	40	42	0.95
1-1’000,000	100	61	1.64
1-10’000,000	581	170	3.42

Tabla 1. Resultados de uso de la aplicación distribuida de prueba (generación de números primos).

Aquí puede apreciar que cuando el numero de cálculos es relativamente pequeño (1 a 1000), la distribución de una aplicación puede causar que el rendimiento se caiga debido principalmente al costo de RPC (y de la comunicación en general). Aunque RPC hace que la programación sea más fácil, también hace más lenta la aplicación. Pero bajo algunas circunstancias, un mejor rendimiento puede ser alcanzado utilizando RPC. Si el numero de cálculos es grande (10’000,000 y más), el costo de RPC se vuelve insignificante. Y el rendimiento puede llegar a ser de 3.5 veces mas rápido con cuatro computadores que con solo uno.

1.2 DISTRIBUCIÓN DE PROCESAMIENTO

1.2.1 Definición

La distribución de procesamiento es un componente de administración de recursos de un sistema distribuido, que se encarga de repartir transparentemente el trabajo del sistema de tal manera que el desempeño general se maximice.

1.2.2 Clasificación

Generalmente los sistemas de distribución de carga se clasifican de la siguiente manera:

- Local vs. Global: La distribución de carga local se encarga de asignar procesos a las diferentes porciones de tiempo de un procesador, mientras que la global decide primero sobre la asignación de un proceso a un procesador y luego planea la ejecución de estos procesos en cada procesador.
- Estático vs. Dinámico (dentro de la categoría de Global): En el modelo estático, la asignación de procesos es hecha *a priori*, es decir, antes que éstos sean ejecutados. Mientras que en el modelo dinámico, esta asignación es hecha en tiempo de ejecución. El modelo estático también se le conoce como planeación determinística, mientras que el modelo dinámico es llamado balance de procesamiento o de carga.
- Óptimo vs. Sub - óptimo (dentro de las categorías Estático y Dinámico): Un modelo de distribución de carga es óptimo si una asignación puede ser hecha de tal forma que se minimice el tiempo de ejecución y se maximice el desempeño del sistema. Soluciones sub - óptimas, que no siempre son las mejores, son aceptables para algunos casos. Cuatro tipos de algoritmos (ambos óptimos y sub-óptimos) son utilizados para encontrar una solución: enumeración y búsqueda en el espacio de solución, modelos de grafos, programación matemática y modelos de colas.

- Aproximado vs. Heurístico (dentro de la categoría sub-óptima): en el modelo aproximado el algoritmo de distribución de carga busca solo en un subconjunto del espacio de solución y termina cuando encuentra una "buena" solución. En un modelo heurístico el algoritmo de planeación utiliza algunos parámetros especiales que le ayudan a aproximar un modelo del sistema real.
- Centralizado vs. Descentralizado (dentro de la categoría Dinámico): en el modelo descentralizado el trabajo de administración es distribuido entre los diferentes procesadores, mientras que tal responsabilidad es asignada a un procesador en el modelo centralizado.
- Cooperativo vs. No-cooperativo (dentro de la categoría Descentralizado): Los mecanismos de distribución de carga dinámicos pueden ser clasificados entre los que implican una cooperación entre los componentes distribuidos (cooperativo) y aquellos en los que los procesadores toman decisiones independientemente y no intercambian información entre sí (no – cooperativo).

La Figura 5 muestra la organización de todos estos algoritmos de distribución.

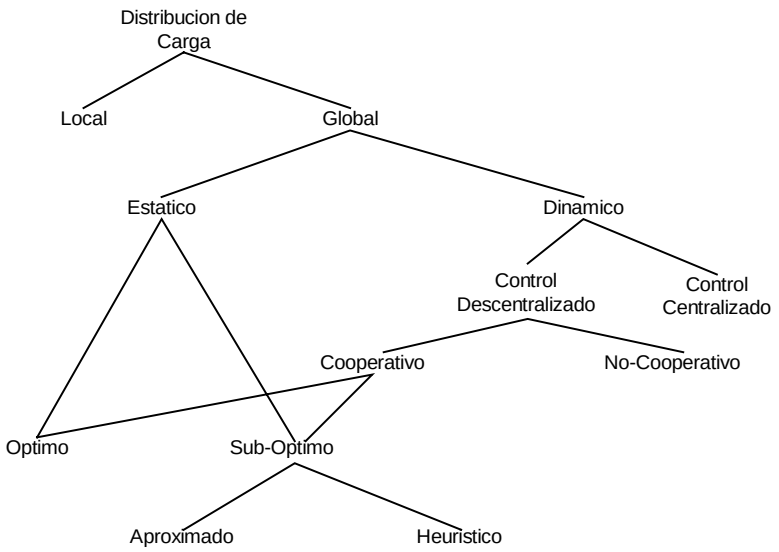


Figura 5. Clasificación de las arquitecturas de distribución de carga.

1.2.3 Otras Clasificaciones

Otra posible organización cubre algunos de los algoritmos que no corresponden directamente con la clasificación anterior:

- Única vs. Múltiples Aplicaciones: Muchos algoritmos de distribución de carga son enfocados a solo una aplicación. Además, en algunos casos, múltiples aplicaciones pueden ser tratadas como una sola aplicación, pero por lo general la distribución de procesos en múltiples aplicaciones es más compleja que con una sola.
- Con Tarea ininterrumpida vs. Con tarea interrumpible: En un algoritmo de distribución de carga donde las tareas son interrumpidas, éstas no pueden ser detenidas una vez esta comienza su ejecución. En un algoritmo de distribución de carga con tareas interrumpibles, una tarea puede ser interrumpida y removida de un procesador, para ser reasignada mas tarde.
- No adaptativo vs. adaptativo: Un algoritmo no adaptativo utiliza solo una política de distribución de carga y no modifica su comportamiento en tiempo de ejecución. Mientras que un algoritmo adaptativo ajusta su comportamiento con respecto a la retroalimentación que le da el sistema. Generalmente el modelo adaptativo consiste en una colección de algoritmos de distribución de carga. La selección de estos algoritmos depende de varios parámetros del sistema.

1.2.4 Sistemas estáticos de distribución de procesamiento

En los algoritmos estáticos, las decisiones de distribución de carga se hacen teniendo un conocimiento *a priori* del sistema. Las cargas no pueden ser redistribuidas en tiempo de ejecución. El objetivo es planear la ejecución de un conjunto de tareas que tengan un tiempo mínimo de ejecución en el PE (Elemento de Procesamiento) elegido. El modelo de distribución de carga estático es también llamado el problema de planeación. En general, la interconexión de los procesadores, partición de las tareas (decisión de la

granularidad) y la asignación de tareas, son las tres mayores decisiones en el diseño de una estrategia de planeación.

Normalmente, tanto las tareas como las estructuras de PEs son representadas por un modelo de grafos. Podemos utilizar un grafo de precedencia de tareas o un grafo de interacción de tareas para modelar un conjunto de tareas. En un grafo de precedencia de tareas también llamado un grafo acíclico directo (ver Figura 6 (a)), cada enlace define un orden de precedencia entre las tareas. Los números asociados con los nodos y los enlaces dirigidos, son los tiempos de ejecución de las tareas y los tiempos requeridos por estas para comenzar la ejecución de sus tareas hijas, respectivamente. En un grafo de interacción de tareas (ver Figura 6 (b)), cada enlace representa las interacciones entre las tareas, y está asociado con un par de números que representan los costos de comunicación cuando dos tareas son asignadas al mismo PE y cuando estas dos tareas son asignadas a diferentes PEs.

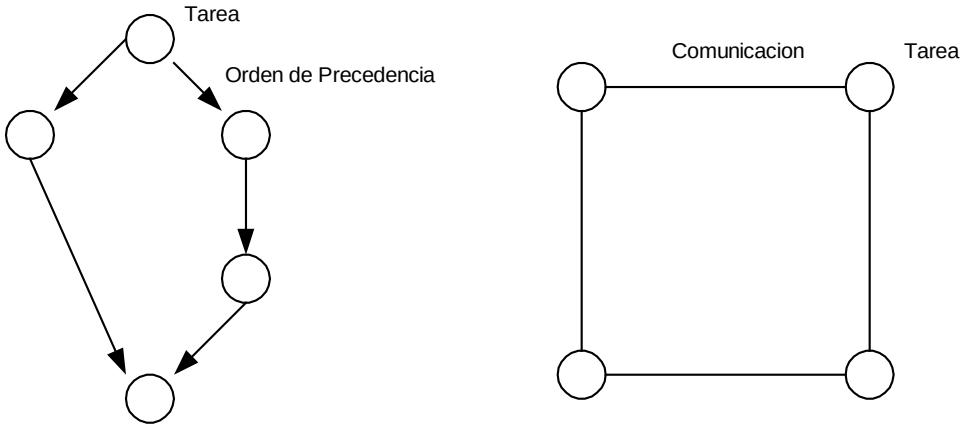


Figura 6. Grafos que representan las posibles relaciones entre tareas.

1.2.4.1 Interconexión de Procesadores

La topología de una red de interconexión puede ser estática o dinámica. Las redes estáticas están formadas por conexiones directas punto - a - punto que

no cambian en tiempo de ejecución. Las redes dinámicas son implementadas con canales conmutados que son configurados en tiempo de ejecución para suplir las demandas de comunicación de los programas.

1.2.4.2 Partición de Tareas

La granularidad de la descomposición de una tarea dada definida por el tamaño promedio de las piezas en la descomposición, puede afectar los costos de comunicación. Un algoritmo puede caracterizarse por ser fino, mediano o poco granular, de acuerdo con el tamaño de cada pedazo de tarea (o tamaño del grano).

Si el tamaño del grano es muy grande (poco granular), el paralelismo es reducido por que las tareas se concentran en un solo procesador. Y si el tamaño del grano es muy pequeño (granos finos), se producen grandes gastos en conmutación de contextos y la sobrecarga de los canales de comunicación genera demoras.

1.2.4.3 Asignación de Tareas

La asignación de tareas se entiende con la asignación de granos (como resultado de una partición de tareas) a un sistema paralelo / distribuido bajo una red de interconexión dada.

Si el número de nodos de un grafo de tareas y de un grafo de procesadores es igual a n en cada uno, existen n! diferentes formas de asignar los nodos de G_t (grafo de tareas) sobre los nodos de G_p (grafo de procesadores); y llamamos a cada una de estas formas un mapeo de G_t sobre G_p. Algunos mapeos son mejores que otros en términos de tiempos totales de ejecución. Algunos supuestos típicos sobre G_p son:

- No existen limites en la capacidad de memoria.
- Cada PE tiene la misma capacidad.

- El hecho de concurrencia en la red no es tenido en cuenta, aunque la distancia entre dos tareas que se comunican (o relacionadas por precedencia) sigue siendo un factor de demoras en la comunicación.

Se debe anotar que la planeación de tareas no se tiene que dar en los dos pasos (partición y asignación de tareas). Este acercamiento de dos pasos es solo para simplificar el proceso de planeación.

1.2.5 Sistemas dinámicos de distribución de procesamiento

La limitación de la distribución de carga estática es que asigna tareas a los procesadores de manera uno-para-todos y requiere un conocimiento a priori del comportamiento del programa. Muchos modelos ignoran los efectos de interferencia en un sistema compuesto por procesos de comunicación y los efectos del desarrollo de los procesos de computación. Por el contrario, el balance de la carga dinámica asume poco o ningún conocimiento acerca de los parámetros en tiempo de ejecución, tales como tiempo de ejecución de tareas o demoras de comunicación. Aunque el balance de carga dinámica se demora también al redistribuir los procesos para lograr los objetivos de rendimiento, mejora el rendimiento del sistema, brindando una mejor utilización de todos los recursos que hacen parte de él.

Vamos a considerar un ejemplo donde una estructura de búsqueda llamado árbol de juego es generado mientras se está buscando. Este acercamiento es muy usado en el diseño de juegos de computador en los que hay dos jugadores. Cada jugador toma un turno moviendo hasta que uno de ellos gana, basado en un conjunto de reglas predefinido. En un árbol de juego los nodos corresponden a las posiciones del tablero y las ramas corresponden a los movimientos. Los nodos en un camino corresponden a los movimientos alternos entre jugadores.

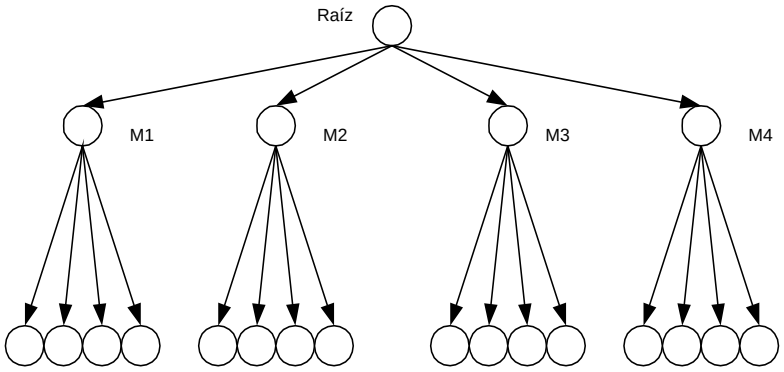


Figura 7. Árbol de juego

En la Figura 7, el nodo raíz corresponde a la posición actual en el tablero. Suponga que el jugador A debe tomar una decisión. Hay 4 posibles movimientos M_i ($1 \leq i \leq 4$) para el siguiente paso. Un buen jugador normalmente calcula los futuros pasos profundizando en el árbol de búsqueda para evaluar el beneficio de cada movimiento. Si hay 4 procesadores en el sistema, la distribución estática asignará una rama M_i a cada procesador. Sin embargo, ciertas ramas pueden corresponder a malos movimientos y esas ramas serán terminadas en pocos pasos abajo del árbol de búsqueda. En este punto el problema con la distribución estática se hace aparente.

El procesador que está explorando el subárbol que tiene el mal movimiento se expande en menos nodos que los otros procesadores. Debido a este desbalance en la carga de trabajo, podemos tener una situación donde uno o más procesadores están desocupados mientras que otros están ocupados buscando el buen movimiento. Un balance de carga dinámico redistribuirá la carga en los diferentes procesadores. En este caso el algoritmo de balance de carga transfiere la carga de los procesadores más cargados a los menos cargados.

1.2.5.1 Clasificación de algoritmos de distribución de carga dinámica

Un algoritmo de distribución de carga dinámica debe ser general, adaptable, estable, escalable, tolerante a fallas y transparente a las aplicaciones. Están clasificados así:

- Global vs. Local: En un algoritmo de balance de carga local, la carga de trabajo es transferida entre nodos individuales vecinos. En uno global, los procesadores se esfuerzan por calcular la carga en todo el sistema y ajustar su propia carga relativa al valor global.
- Centralizado vs. Descentralizado: En un algoritmo centralizado, un controlador central almacena el estado del sistema y toma decisiones acerca del balance de carga. En uno descentralizado, el mecanismo de control es físicamente distribuido a cada nodo en el sistema.
- No-Cooperativo vs. Cooperativo: En un algoritmo no-cooperativo, cada nodo decide su localización y políticas sin conocer los estados de los otros nodos en el sistema. Los nodos se coordinan con los otros para tomar decisiones de balance de carga en el modelo Cooperativo.
- Adaptativo vs. No-Adaptativo: En el modelo adaptativo, las políticas de balance de carga son modificadas cuando el estado del sistema cambia; estas políticas no cambian en el modelo no-adaptativo.

1.2.5.2 Decisiones en el Diseño de Sistemas de Balance Dinámico

1.2.5.2.1 Algoritmos Estáticos vs. Dinámicos

Cuando los procesos están bien definidos en términos de sus tiempos de ejecución y requerimientos de recursos, y sus intervalos de llegada son conocidos previamente, el balance de carga puede ser implementado con relativa facilidad.

Para los problemas que involucran intervalos desconocidos de llegada de procesos a los procesadores, es imposible obtener una estrategia determinística que permita la migración de los procesos a otros procesadores. Algunos algoritmos de distribución de carga dinámica manejan tareas

ininterrumpidas, esto es, los nodos solo pueden distribuir las tareas recién llegadas, y cuando ya han sido asignadas no se tocan. Otros algoritmos tienen las tareas interrumpibles (también llamados de migración de tareas) y pueden redistribuir una tarea que ya se este ejecutando.

Los modelos con tareas ininterrumpidas pueden ser además clasificados en planeados-una-vez y de planeación múltiple. En el primer caso, una vez que una tarea ha sido asignada a un procesador, ésta no puede ser redistribuida aún si se encuentra en la cola de espera. En el caso de planeación múltiple, una tarea sí puede ser redistribuida si no se está ejecutando y está en la cola de espera. Normalmente los siguientes pasos están incluidos en una migración de tareas: (a) Suspender el proceso a migrar en el nodo fuente, (b) transferir el proceso junto con su estado actual al nodo destino y (c) reiniciar la ejecución del proceso en el nodo destino.

Dos tipos de procesos están definidos, los genéricos y los regulares. Un proceso genérico puede ser asignado individualmente a cada nodo. Un proceso regular solo puede ser asignado localmente. Esto es, los procesos genéricos pueden ser utilizados para implementar el movimiento dinámico de carga mientras que los regulares no pueden ser compartidos o migrados.

1.2.5.2.2 Varias Políticas de Información

Deciden la información necesaria para un balance de carga eficiente. Por ejemplo, es esencial que un procesador sobrecargado sepa los posibles nodos destino. Es más, los demás procesadores en el sistema también necesitan saber acerca de los procesadores sobrecargados.

- Información aperiódica vs. periódica: una forma de recolectar información del sistema es hacer una petición a cada procesador sobre sus niveles de carga, otra es que cada uno de estos emita (es decir, haga una difusión o *broadcasting*) estos niveles a los demás. Cualquiera de estos dos métodos puede ser utilizado, pero solo cuando se encuentren situaciones críticas

como un procesador sobrecargado o libre. Alternativamente, puede ser utilizado en forma regular por cada uno de los procesadores. La segunda forma, que corresponde a la periódica, generará mas gastos si la carga no cambia rápidamente, sin embargo, asegura que todos lo procesadores conocen los niveles de carga de los demás con mayor certeza en un momento dado. El período de tiempo entre dos intercambios de la información de estado es un parámetro muy importante. Este puede cambiar dinámicamente en respuesta al rendimiento del sistema. Por otra parte, el intercambio de información aperiódico (que es el primer caso) genera menos carga en los canales de comunicación, pero puede darse el caso de tener un procesador que pierda un tiempo valioso tratando de encontrar un donador o receptor adecuado de carga de trabajo.

- Información Global vs. Local: la recolección de información global puede ser muy costosa para un sistema grande. Una alternativa es recolectar información local dentro de un cierto rango. Encontrar el procesador con carga mínima local, puede ser más conveniente que encontrar el mínimo global. Esto será mas importante aún si la distancia entre los procesadores es grande y los costos de comunicación sobre grandes distancias son prohibitivos. También en casos donde se envían grandes archivos a los procesadores destino como parte del proceso de migración de carga, será más conveniente enviarlos a un procesador que esté más cerca, así exista uno con menor carga pero más lejos. Esta estrategia conduce a que los procesadores tengan menos información sobre la carga del sistema.
- Información de Carga del Sistema: Generalmente la carga de trabajo de un procesador se mide basada en el porcentaje de uso de la CPU, cantidad de memoria utilizada, y número de procesos. También se pueden considerar otras variables físicas de comportamiento, como su velocidad, tamaño de la memoria principal, actividad de paginación, razón de actividades en proceso y de entrada/salida, son otros factores que determinan el nivel de carga. En algunos casos, cuando un proceso es ejecutado repetidamente, la información de las ejecuciones iniciales puede dar una idea de los niveles de carga esperados posteriormente.

1.2.5.2.3 Algoritmos Centralizados vs. Descentralizados

Los algoritmos centralizados tienen un procesador central que recibe la información de carga de los otros procesadores del sistema. En algunos sistemas centralizados el procesador central, luego de obtener un panorama general del estado de carga del sistema, envía esta información a los demás procesadores, para que éstos tomen decisiones sobre migrar sus procesos a otros nodos o aceptar procesos de otros nodos. Es posible diseñar un sistema que le permita a un procesador, en el caso de migrar un trabajo, solicitarle al procesador central un candidato basado en la información de estado que éste ha recolectado. En este caso, el procesador central no necesitaría enviar la información general de estado a los demás procesadores, y de esta forma disminuiría la carga de los canales de comunicación. Un único procesador central también puede producir el hecho de una falla total del sistema en términos de migración de carga cuando se presentan fallas en él.

Los algoritmos descentralizados son implementados por cada procesador individual, enviando los cambios en su estado de carga a todos los procesadores o a los de su vecindario. Estos a su vez a sus vecinos, y así sucesivamente. Luego de un tiempo, todos los procesadores tendrán una información global del estado de carga del sistema.

Este tipo de algoritmo es más tolerante a fallas que los centralizados, pero la responsabilidad de comunicar el estado de carga del sistema se ha pasado del procesador central a cada uno de los que componen el sistema. Por lo tanto, el tiempo total que gasta un procesador en recolectar la información de carga del sistema se incrementa.

1.2.5.2.4 Políticas de Iniciación de Migración

Existen dos formas de iniciar un balance de carga. Cuando un procesador sobrecargado es responsable de buscar otros procesadores potenciales que puedan aceptar su carga extra, el algoritmo es denominado como iniciado por

el que envía. Cuando un procesador libre obtiene una tarea de un procesador sobrecargado, el algoritmo se denomina como iniciado por el receptor. Una combinación de los dos es denominada iniciada simétricamente.

1.2.5.2.5 Replicación de recursos

Cuando una tarea tiene que ser migrada, los archivos y datos relacionados también tienen que ser migrados al procesador de destino. Si éstos son grandes, la sobrecarga generada sobre los canales de comunicación será sustancial. Para reducir estas cargas, se puede replicar toda la información requerida por las tareas sobre todos los procesadores del sistema. Pero esto requerirá de mas espacio de almacenamiento en cada nodo, y un mecanismo de actualización de esta información.

1.2.5.2.6 Clases de Procesos

Los procesos pertenecientes a un tipo genérico tales como los de procesamiento de texto, programas de simulación, programas del sistema, programas interactivos, etc. pueden ser tratados de manera particular, puesto que tienen características similares. Si los procesos que se estén ejecutando son completamente diferentes entre si, deben ser tratados de forma individual. En estos casos, las estrategias de balance de carga pueden variar dependiendo de los tipos de procesos a consideración. Aun así, la mayoría de algoritmos de balance de carga consideran una única clase de proceso.

1.2.5.2.7 Control de Lazo Abierto vs. Lazo Cerrado

Un algoritmo con control de lazo abierto no tiene en cuenta actividades pasadas para tomar sus decisiones. En cambio uno de lazo cerrado evita que el sistema se aleje de la funcionalidad, deseada encausando sus decisiones para que progrese en la manera requerida. Estos algoritmos de lazo cerrado necesitan ser bien implementados para evitar que se desestabilicen y produzcan sobrecarga en el sistema.

1.2.5.2.8 Utilización de Hardware vs. Software

El balance de carga puede ser realizado utilizando software que lleve un seguimiento de los procesos de los diferentes nodos. Sistemas que realicen ese control basados en hardware también pueden ser utilizados para tal fin. Algunas arquitecturas comprometen varios servidores de archivos basados en LAN trabajando juntos para implementar un almacenamiento de archivos global y distribuido del sistema. Los algoritmos de balance de carga y asignación de usuarios son responsabilidades de un coordinador de alto nivel que puede ser centralizado o descentralizado.

1.2.5.3 Parámetros Utilizados para el Balance de Carga

La formación y selección de un algoritmo de balance de carga depende de una serie de componentes cuantificables. Estos componentes son considerados como parámetros del sistema y se numeran a continuación:

1.2.5.3.1 Tamaño del sistema

El numero de procesadores en el sistema es uno de los parámetros que más influye en las decisiones de balance de carga. Una mayor cantidad de procesadores hace más probable que un procesador sobrecargado pueda encontrar otro libre y así migrar sus tareas. Este efecto positivo compite con el efecto negativo producido por una mayor sobrecarga en las comunicaciones debido al mayor tamaño del sistema.

1.2.5.3.2 Carga del sistema

Normalmente la carga del sistema es medida por el porcentaje de uso de la CPU como ya se mencionó anteriormente. Si el sistema esta altamente cargado y más procesos llegan a un procesador, cualquier migración de procesos de un nodo altamente cargado probablemente disparará una cascada de migraciones hacia otros procesadores. Esto puede conducir a un fenómeno denominado desperdicio de procesos que ocurre cuando una tarea es migrada desde un procesador resultando en que el procesador destino migra otra vez esa tarea. En tales casos los procesos que entran no deberían ser migrados.

1.2.5.3.3 Intensidad de tráfico del sistema

Los procesos que llegan a los nodos pueden seguir cualquier patrón aleatorio, si los procesadores pueden determinar su intensidad de tráfico y compararla con la de otros procesadores, es más fácil estimar el nivel de carga instantáneo del sistema. Esto permite a los procesadores tomar decisiones mas acertadas con respecto a la migración de tareas.

1.2.5.3.4 Umbral de migración

El umbral tiene que ser determinado heurísticamente, ya sea por experimentación dinámica de las características del sistema para decidir un nivel máximo o mínimo apropiado. Si los procesadores se clasifican por su nivel de carga (bajo, mediano o alto) como se hace en varios sistemas, el número exacto de procesos en cada una de estas categorías debe ser determinado. El nivel de carga utilizado para disparar la migración de procesos en el sistema es un parámetro crítico puesto que puede conducir a una pérdida de balance si se escoge inapropiadamente.

1.2.5.3.5 Tamaño de tarea

Normalmente es inapropiado migrar una tarea muy pequeña. De la misma manera, procesos que involucren la transferencia de grandes cantidades de datos son mejor manejados por el procesador local sin hacer uso de la migración. Por lo tanto, el tamaño óptimo de una tarea para migración se convierte en otro parámetro importante. No es fácil determinar el tamaño de una tarea; sin embargo, este puede ser estimado basado en el requerimiento de recursos, el tipo de tarea, los requerimientos de memoria y numero de archivos que se necesitan. Se ha demostrado con varios sistemas que los procesos grandes se benefician sustancialmente de un balance de carga, mientras que los pequeños no como resultado de la excesiva comunicación.

1.2.5.3.4 Costos por sobrecarga

Si la distancia entre el procesador que envía y el que recibe es larga, el tiempo de transferencia y comunicación aumenta. Si un procesador esta más cerca al que envía y tiene más carga (pero aún esta poco cargado) que un procesador que esta más lejos, será una mejor opción de migración si se tienen en cuenta los costos de migración.

1.2.5.3.5 Tiempo de respuesta

Para estimar la carga de un procesador, necesitamos saber el tiempo de respuesta de un proceso. El tiempo de respuesta no es evidente a menos que se recolecten algunos datos después de su ejecución. Pero hay problemas cuando no se tiene información de algunos procesos.

1.2.5.3.6 Horizonte de balance de carga

La distancia a los nodos vecinos que puedan ser considerados como destinos de una tarea es llamada horizonte. Este parámetro asigna el número de nodos vecinos a ser probados con el fin de encontrar un destino conveniente. En los casos donde la actividad necesaria para encontrar este horizonte es costosa, se tiene que escoger entre la pérdida de eficiencia por no encontrar un sitio óptimo para la tarea y el costo de prueba. Por lo tanto seleccionando el horizonte, la carga puede ser balanceada dentro de él, pero es posible que externamente exista un mejor candidato para la migración.

1.2.5.3.7 Demanda de recursos

La demanda de una tarea sobre los recursos del sistema puede influenciar su migración. Los procesos que necesitan muchos recursos para su ejecución estarán esperando constantemente que los recursos estén disponibles. Esto puede afectar el tiempo de respuesta del sistema. En este sentido la demanda de recursos refleja la carga del procesador.

1.2.5.4 Otros Aspectos Relevantes

Algunos otros aspectos gobiernan el diseño de un sistema de balance de carga.

1.2.5.4.1 Archivos de código y datos

Las operaciones de balance de carga se facilitan en sistemas que tienen servidores de archivos dedicados. Los archivos de datos requeridos serán enviados por el servidor de archivos al procesador de destino. Cuando es imposible tener un servidor de archivos dedicado, se agrega un sobrecosto al mover los archivos requeridos.

1.2.5.4.2 Estabilidad del sistema

Cuando un procesador es identificado como destino por varios no sobrecargados, es probable que éste se vea también sobrecargado. Esta condición hará que el procesador destino también inicie una actividad de balance de carga. Un mecanismo que evite que posteriores procesos lleguen a un procesador sobrecargado es necesario.

1.2.5.4.3 Arquitectura del sistema

La heterogeneidad de arquitecturas y configuraciones complica el problema del balance de carga. La heterogeneidad puede derivarse de la diferencia de los radios de llegada de tareas a procesadores del mismo tipo. El sistema será también heterogéneo si los procesadores tienen radios de procesamiento de tareas diferentes entre sí. La forma en que el sistema esta interconectado también influye en el algoritmo de balance de carga. Aunque al incrementar el tamaño del sistema también se incrementa la posibilidad de encontrar un procesador libre que acepte más carga, esto es un problema potencial para los sistemas conectados en bus. En este caso, más procesadores significan mas utilización del bus de datos y esto se convierte en un cuello de botella.

1.3 ARQUITECTURAS DE *SOFTWARE* PARA LA CONSTRUCCIÓN DE SISTEMAS DISTRIBUIDOS

1.3.1 DCE-RPC

Existen básicamente dos modos de distribuir la ejecución de un programa. Un modo es implementar la aplicación en cuestión como un conjunto de procesos cooperativos pero separados. Los procesos individuales pueden ser ubicados en nodos diferentes e interactuar por medio de varios mecanismos distribuidos para comunicación y sincronización entre procesos.

Otro método es dividir un único programa monolítico y permitir que sus trozos se ejecuten en diferentes nodos de un sistema distribuido. Los procedimientos son fronteras naturales para la división de un programa ya que son en gran medida entidades lógicas auto contenidas que tienen interfaces y puntos de entrada / salida bien definidos. Además, hay que proporcionar un mecanismo de llamada a procedimientos remotos (RPC, *Remote Procedure Call*, llamada a procedimientos remotos) para hacer que el programa dividido funcione en un entorno distribuido.

Conceptualmente, la llamada a procedimiento remoto obedece al mecanismo bien entendido de la transferencia de control y datos dentro de dos o más computadores. Cuando se invoca un procedimiento remoto, el entorno invocador queda suspendido y los parámetros son transferidos al nodo donde el procedimiento va a ser ejecutado, ya sea en el mismo equipo o en el otro. Cuando el procedimiento finaliza y genera sus resultados el nodo invocado los devuelve al nodo invocador.

La distribución del cálculo asistida por llamadas a procedimientos remotos tiene una serie de propiedades atractivas. Una de ellas es su semántica limpia y simple, que facilita la implementación correcta de los cómputos distribuidos.

A diferencia de otras alternativas, las implementaciones de procedimientos remotos hacen la distribución transparente a los programadores. Utilizando un modelo familiar de flujo de datos y de control, los programadores de aplicación se ven liberados de la codificación y el control explícitos de primitivas relativas a la distribución, tales como paso de mensajes. Otro beneficio de la transparencia de distribución que RPC proporciona es que las aplicaciones existentes desarrolladas para un solo computador pueden ser portadas con comparativa facilidad a un entorno distribuido. Una tercera ventaja es la generalidad: los procedimientos son uno de los mecanismos más habituales y mejor comprendidos para comunicación entre partes de un algoritmo.

Los procedimientos remotos son apropiados para el modelo cliente / servidor de computación distribuida. Las rutinas clientes pueden solicitar los servicios invocando remotamente la ejecución de los procedimientos adecuados en el servidor.

Los servidores pueden proporcionar servicios comunes por medio de los procedimientos públicos que una serie de clientes potenciales pueden invocar. Este concepto es similar al de las librerías dinámicas (DLLs, *Dynamic Link Libraries*) en computadores personales. En ambos entornos, las rutinas publicas deben ser reentrantes o estar protegidas frente a expropiaciones mediante alguna norma de control de concurrencia, como por ejemplo exclusión mutua, semáforos, etc.

Tres cuestiones importantes en la implementación del mecanismo de llamada a procedimientos remotos son:

- Transferencia de control
- Vinculación
- Transferencia de datos

1.3.1.1 Transferencia de Control

Presumiblemente, una cierta forma de abstracción de memoria compartida global podría servir como facilidad para implementar las llamadas a procedimientos remotos en un sistema distribuido. Este método tiene dos dificultades potenciales: 1) puede requerir la modificación de los lenguajes de programación con objeto de soportar direcciones remotas y 2) es difícil de implementar con suficiente eficacia para las RPC. Los métodos presentados a continuación suponen la no existencia de memoria compartida.

La estructura de un programa para llamada a procedimiento remoto suele basarse en el concepto de sustitutos de usuario (*stubs*) La Figura 8 ilustra los componentes básicos de un RPC.

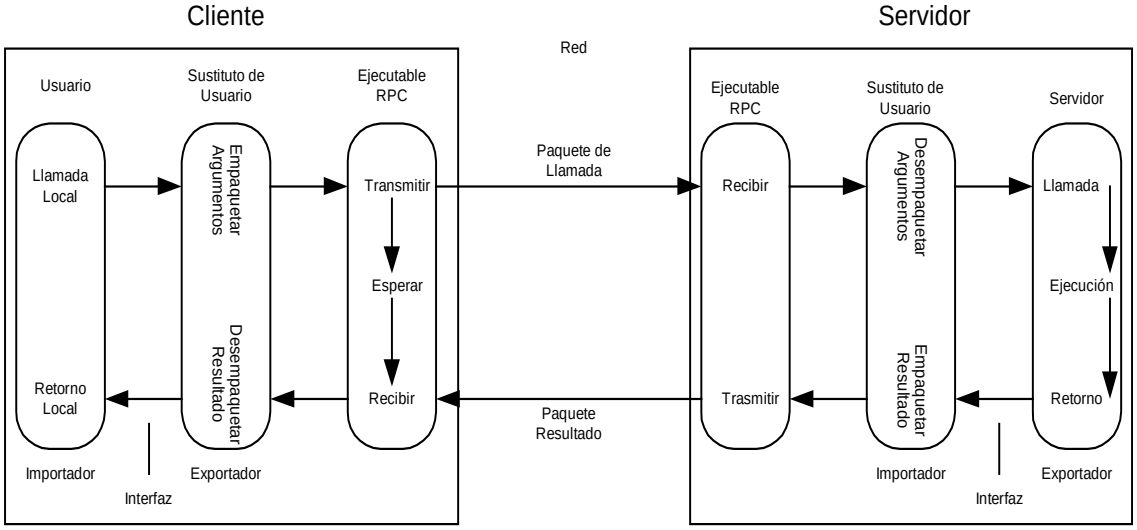


Figura 8. Componentes básicos de un RPC

Además de la rutina de usuario (invocador), la máquina invocadora contiene un sustituto de usuario y el paquete ejecutable RPC. La máquina invocada tiene sus propias rutinas ejecutables RPC, el sustituto servidor y la rutina invocada designada como servidor. La llamada a procedimiento remoto del usuario es enlazada en la máquina usuario al sustituto del usuario. Por tanto, la invocación del usuario al procedimiento remoto se convierte en una llamada

local al procedimiento correspondiente en el sustituto de usuario. El sustituto de usuario es responsable de incluir una especificación del procedimiento destino y sus argumentos en los paquetes de comunicación. El sustituto de usuario invoca entonces al ejecutable RPC local a través de una llamada a procedimiento local (LPC – *Local Procedure Call*) para transmitir esos paquetes con fiabilidad a la máquina invocada.

El ejecutable RPC es responsable de la entrega fiable de los paquetes. Sus funciones incluyen retransmisiones, reconocimientos, encaminamiento de paquetes y opcionalmente cifrado. Aunque los protocolos de comunicación por red ofrecen servicios similares, muchas implementaciones de ejecutables RPC proporcionan sus propios protocolos de comunicación RPC-RPC personalizados para mejorar la eficiencia.

El ejecutable RPC del servidor (invocado) recibe los paquetes y los transfiere al sustituto del servidor a través de una llamada a procedimiento local. El sustituto servidor desempaqueta los datos y actúa como agente de los usuarios remotos. En particular, el sustituto servidor efectúa una llamada local al procedimiento adecuado y suministra los parámetros recibidos. Cuando el procedimiento invocado completa la ejecución, devuelve el control y los resultados al sustituto servidor. El sustituto servidor transfiere los resultados al ejecutable RPC servidor para su transmisión a la máquina invocadora. El ejecutable RPC del invocador transfiere los paquetes resultados al sustituto usuario. Tras desempaquetar, el sustituto usuario completa el proceso efectuando un retorno local a la rutina invocadora.

Como indica la implementación genérica, la percepción del usuario en tiempo de ejecución de la llamada a procedimiento remoto es difícilmente diferenciable de la de una llamada a procedimiento local. Si el código del invocador y del servidor se encuentran dentro de una misma máquina y están ligadas directamente sin los sustitutos, el programa seguirá funcionando. Excepto por

la vinculación multimáquina, la semántica de RPC es la misma que la de una llamada a procedimiento en una sola máquina.

En términos de flujo de control, el invocador queda suspendido hasta que el procedimiento invocado termina y regresa. Como cabe esperar, existe completa sincronía entre el invocador y el invocado. A la terminación de la llamada, el invocador continua su ejecución desde el punto inmediatamente después de la llamada. Se supone que el procedimiento servidor se ejecuta en su totalidad antes de devolver control. Las llamadas a procedimientos remotos pueden ser anidadas; es decir, un procedimiento servidor puede invocar a otros procedimientos. La imitación que hace RPC de la llamada a procedimiento normal es intencionada. Gran parte del atractivo de las llamadas a procedimientos remotos se debe a su familiar semántica y comportamiento.

1.3.1.2 Vinculación

Parte de la complejidad añadida de las RPC se debe a la necesidad de suministrar sustitutos de usuario y de servidor. Muchas implementaciones RPC proporcionan herramientas automáticas tales como lenguajes de interfaz (como el IDL – *Interface Definition Language*, lenguaje de definición de interfaces, usado en CORBA y DCOM) para facilitar la generación de sustitutos.

En lenguajes de programación con soporte para compilación separada, como Ada y Modula, los procedimientos que implementan una interfaz se dice que exportan esa interfaz. Un módulo de programa que llama a los procedimientos de una interfaz se dice que importa la interfaz. La comprobación de tipos a través de fronteras de módulo durante la compilación separada puede ser forzada haciendo que un módulo sea compilado con todas las definiciones de interfaz (exportadas por sus propietarios) que éste importe. En general, el sustituto de usuario importa la interfaz y el sustituto de servidor lo exporta. Los generadores automáticos de sustitutos son utilidades de software que pueden utilizar estas interfaces para generar los sustitutos adecuados con poca o ninguna intervención del usuario. La implementación de sustitutos es mucho

más fácil en lenguajes de programación y entornos que soportan compilación separada.

Así, bien en tiempo de compilación o bien en tiempo de ejecución, el nodo invocador inicia la vinculación comunicándose con el servidor de vinculaciones. En general, es preferible la vinculación tardía (en tiempo de ejecución – *Late Binding*) en los sistemas distribuidos debido a su flexibilidad. La vinculación temprana (en tiempo de compilación – *Early Binding*) tiene la desventaja de inscribir en el código una instancia especifica del servidor demasiado pronto. La vinculación en tiempo de ejecución retrasa este compromiso hasta que se efectúa realmente la llamada; como resultado, el sistema puede optimizar el rendimiento asignando los recursos sobre la base del estado en que se halle el sistema durante la ejecución real de la RPC. Por ejemplo, el sistema puede elegir emplear un servidor poco cargado, o aumentar la robustez ofertando suplentes adecuados para servidores caídos.

Antes de la vinculación del invocador, el exportador debe comunicarse con el servidor de vinculaciones para registrar su interfaz. Todas estas interacciones tienen lugar con anterioridad a la secuencia RPC representada en la Figura 8.

1.3.1.3 Flujo de Datos

Extendiendo la noción de llamada a procedimiento local, se supone que RPC transfiere todos los parámetros en el momento de la llamada. Dado que el invocador y el invocado se ejecutan en nodos diferentes y en espacios de direcciones separados, todos los parámetros deben ser transferidos explícitamente y por valor. En sistemas sin memoria compartida, las variables globales y las llamadas por referencia no pueden ser soportadas por el mecanismo RPC. Algunos aspectos de la llamada por referencia pueden ser simulados transfiriendo la variable en cuestión y haciendo que el invocado opere sobre ella y devuelva el valor modificado como resultado. La transferencia de argumentos que contienen punteros no tiene significado en sistemas sin memoria compartida global.

Una de las principales cuestiones de las implementaciones RPC es el soporte de la heterogeneidad. Las diferencias en software, hardware, o ambas, pueden conducir a representaciones incompatibles de los tipos de datos en máquinas diferentes dentro de un sistema distribuido. Algunos de los problemas comunes con tipos de datos primitivos incluyen la ordenación incompatible de los *bytes*, las representaciones de punto flotante y la codificación de los caracteres (ASCII vs. *Unicode*).

El uso de estándares, como por ejemplo el estándar IEEE de punto flotante, y el que algunos procesadores soporten ordenación de *bytes* seleccionables por programa en sus diseños, sirve de ayuda. No obstante, un RPC viable diseñado para un entorno heterogéneo debe proporcionar algunos medios para abordar el problema.

Un método es facilitar un formato de representación de datos uniforme global para la red. Cada emisor convierte sus formatos de datos locales a la representación uniforme para su subsiguiente decodificación en el receptor. Este método es obviamente ineficiente cuando son dos máquinas homogéneas las que se comunican en un sistema heterogéneo. Otra posibilidad es proporcionar un lenguaje de programación común que resuelva las diferencias de representación compilando en el lenguaje máquina nativo de cada nodo y proporcionando las rutinas de traducción en tiempo de ejecución cuando sea necesario (p. ej. Java).

1.3.1.4 RPC frente a paso de mensajes

Los mensajes son un mecanismo de comunicación y sincronización entre procesos adecuado para su uso por procesos cooperativos pero separados y en gran medida autónomos. Por tanto, ambos mecanismos sirven a propósitos en cierta medida diferentes y no son sustitutivos directos el uno del otro.

Sin embargo, cuando se utiliza uno de estos mecanismos para otro propósito distinto de su propósito principal, existe cierto solapamiento y puede ser necesario realizar una elección entre ellos. Por ejemplo, este puede ser el caso cuando se utiliza la facilidad de distribución para transferencia de datos masivos, tales como archivos, o cuando se está decidiendo el estilo de programación y el mecanismo de soporte para comunicación entre procesos a emplear en una nueva aplicación distribuida. Algunas de las principales consideraciones de interés aquí son:

- Transferencia del control y sincronía.
- Vinculación.
- Transferencia de datos.
- Tolerancia a fallos.

Siguiendo estrictamente la semántica de una llamada a procedimiento en un sistema mono computador, las RPC son síncronicas. Los mensajes proporcionan ambas variedades, síncronica y asíncronica, de la relación emisor / receptor. Los mensajes asíncronos pueden mejorar la concurrencia a costa de tener que ocuparse del almacenamiento de los mensajes que han sido enviados pero aún no recibidos. La imposición de un limite sobre el numero de mensajes pendientes tiende a resolver el problema de la implementación, pero también complica la semántica de la primitiva de envió asíncrono.

Las RPC se apoyan en la vinculación multimáquina para el adecuado procesamiento de las invocaciones remotas. El paso de mensajes no requiere generalmente vinculación. Solo requiere la identificación de un puerto destino o de un receptor mediante el mecanismo de denominación en tiempo de ejecución.

En términos de transferencias de datos, las RPC imponen una cierta caracterización de tipo de los parámetros y típicamente requieren que todos los parámetros sean entregados en el momento de la llamada. Los mensajes son menos estructurados y permiten que el emisor y el receptor negocien

libremente el formato de los datos. Al ser esencialmente flujos de bytes, los mensajes tienden a ser más fáciles de adaptar a entornos heterogéneos.

Los fallos son difíciles de abordar en ambos entornos. Sin embargo, los mensajes tienen la propiedad conveniente de que un receptor puede actuar sobre un mensaje que es lógicamente independiente del fallo o la terminación del emisor. Si, por ejemplo, la operación es actualizar una variable compartida, aún puede completarla a pesar del fallo del emisor después de enviar el mensaje. En el modelo RPC, el fallo del cliente crea un proceso huérfano que típicamente resulta abortado por el servidor.

Finalmente, una cuestión no directamente relacionada con el diseño puede ser la preferencia del programador. Se ha argumentado que algunos programadores encuentran las primitivas de mensajes poco naturales y confusas, y prefieren usar el mecanismo RPC por razones de conveniencia.

1.3.2 DCOM

En pocas palabras, DCOM (*Distributed Component Object Model*) es la tecnología de desarrollo de sistemas distribuidos propuesto por Microsoft e implementado en todos sus sistemas operativos de 32 bits. Este modelo permite el desarrollo de programas que, ubicados en diferentes máquinas conectadas entre sí a través de una red de área local (LAN) o de área amplia (WAN), interactúan entre sí para llevar a cabo las tareas que tienen a cargo.

Antes de presentar en detalle la descripción y las características de DCOM, es necesario exponer brevemente el modelo sobre el cual está construido DCOM, el cual se denomina COM.

1.3.2.1 COM: La raíz de DCOM

COM (*Component Object Model*) es, al igual que DCOM, un modelo de desarrollo propuesto por Microsoft. Al construir programas utilizando COM, es

posible que una aplicación exponga su funcionalidad a otras aplicaciones, de modo que éstas pueden utilizarla para llevar a cabo sus tareas, sin que sea necesario reescribir o copiar el código de la aplicación utilizada. Esto es posible porque una aplicación COM expone su funcionalidad a través de un conjunto específico de procedimientos, denominado interfaz. Por otro lado, cualquier aplicación puede acceder a la funcionalidad expuesta por otras, a través de las interfaces que éstas exponen. Dicho de otra manera, es posible construir aplicaciones que trabajan como "servidores de funcionalidad", y los programas que acceden a esa funcionalidad son clientes de la primera.

COM provee las siguientes características a las aplicaciones que son construidas con base en este modelo:

- Independencia del lenguaje de programación: Los programas COM no necesitan estar escritos en un lenguaje de programación en particular. Por ejemplo, una aplicación desarrollada en C++ puede ser accedida por otra escrita en Java o Visual Basic.
- Manejo robusto de versiones: Si una aplicación COM es actualizada, las aplicaciones que son clientes de ella no tienen que ser modificadas o reconstruidas para que sigan funcionando con la versión actualizada del servidor. Esto es posible debido a que un servidor expone su nueva funcionalidad a través de una nueva interfaz, que puede ser utilizada por los nuevos clientes mientras que los antiguos siguen trabajando sin cambios.
- Invocación por demanda de las aplicaciones: No es necesario que una aplicación servidora esté cargada previamente en la máquina para que los clientes puedan acceder a ella. En cambio, los servidores son cargados solamente cuando es necesario, y si todos los clientes ya han dejado de utilizar al servidor, éste es descargado del sistema, lo que conduce al ahorro de recursos y al mejor rendimiento del mismo.
- Orientación a objetos: COM, al ser un modelo orientado a objetos, provee las características de encapsulamiento, herencia y polimorfismo, y lo hace

sin importar el lenguaje en que se desarrollan las aplicaciones. Esto es posible debido a la flexibilidad que se puede obtener al desarrollar interfaces.

- Manejo unificado de errores: Todas las aplicaciones construidas sobre COM emplean el mismo mecanismo de comunicación de errores, lo que facilita el manejo de las condiciones de error.
- Independencia de la localización: La aplicación cliente no tienen que saber dónde se encuentra ubicada físicamente la aplicación servidora que está utilizando. Es decir, el servidor puede residir en el mismo proceso que el cliente, en otro proceso o en otra máquina, y el cliente que lo utiliza accede de la misma forma en cualquiera de los casos. Este punto será retomado más adelante, en la presentación de DCOM.
- Manejo unificado de seguridad: COM permite utilizar las características de seguridad propias del sistema operativo (por ejemplo, las listas de grupos, usuarios y claves de acceso de Windows NT o un sistema basado en llaves públicas suministrado por una aplicación auxiliar para Windows NT), con el fin de centralizar la autenticación de las aplicaciones clientes y servidores. Así, es posible restringir el acceso a las aplicaciones servidores, y controlar la forma en que los clientes las utilizan.
- Manejo de eventos: Es posible establecer canales de comunicación entre clientes y servidores, donde estos últimos pueden alertar a los primeros sobre sucesos que han ocurrido, como la terminación de una operación larga o una condición de error. COM provee un mecanismo conocido como *connection point* (punto de conexión), a través del cual el servidor puede acceder al cliente e informarle de tales sucesos.

COM está construido sobre RPC, el servicio de comunicación entre aplicaciones que ya había sido tratado anteriormente, por lo que cumple el estándar DCE para el desarrollo de sistemas distribuidos. Además, está implementado en todos los sistemas operativos Win32 como un conjunto de funciones declaradas dentro del API de Windows y una declaración de interfaces estándar

que todas las aplicaciones COM deben manejar. De esta forma, no es necesario distribuir *software* adicional para utilizar COM en las aplicaciones Windows.

1.3.2.2 Características de DCOM

Una de las características más importantes de COM es la independencia de la ubicación física de los programas que interactúan como clientes y servidores. Cuando una aplicación cliente necesita llamar funciones que se encuentran en la interfaz expuesta por otra aplicación que está instalada en el mismo equipo, el sistema operativo se encarga de proveer la conexión entre las dos aplicaciones, convirtiendo, a través de DCE - RPC, una llamada a un proceso remoto en una a un proceso local (LPC), de modo que ellas no necesiten saber cómo se ha establecido la comunicación entre ambas. El servicio de seguridad suministrado por el sistema operativo se encarga de realizar la autenticación de clientes y servidores. La Figura 9 ilustra la forma en que las dos aplicaciones se conectan.

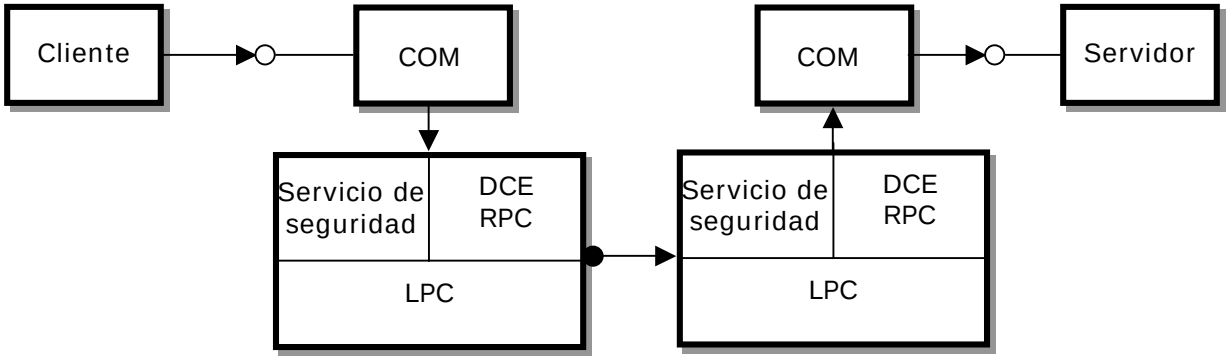


Figura 9. Interacción entre dos aplicaciones COM que están en el mismo equipo

Cuando las aplicaciones se encuentran en máquinas diferentes, DCOM se encarga de reemplazar la comunicación entre dos procesos por un protocolo de red. El cliente no nota que la línea de comunicación entre él y el servidor que está utilizando se ha tornado más larga, ya que DCOM se encarga de llevar a cabo este cambio y ocultarlo al cliente.

La Figura 10 resume la arquitectura de DCOM: El sistema operativo provee los servicios de comunicación entre aplicaciones, y utilizando DCE-RPC para establecer la comunicación, convierte la información que fluye entre las aplicaciones en paquetes válidos para la red en la cual se encuentran los equipos donde están instaladas las aplicaciones. Al igual que en el ejemplo anterior, el servicio de seguridad del sistema operativo se encarga de realizar la autenticación.

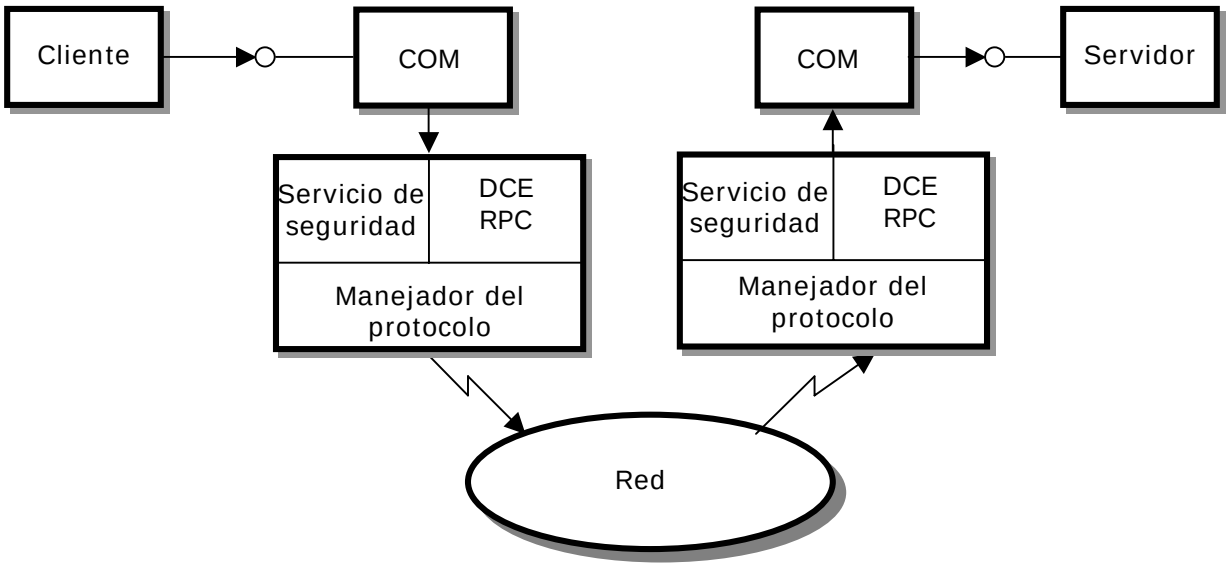


Figura 10. Interacción de dos aplicaciones COM que están en diferentes equipos.

Por otro lado, las conexiones de red son frágiles por naturaleza. Muchos factores, como fallas de los equipos o de las redes, o tiempos de espera demasiado largos, pueden romper una conexión ya establecida entre dos aplicaciones, con los problemas que esto puede generar. DCOM utiliza un método propio de verificación de la disponibilidad de las aplicaciones configuradas como servidores, y cuando alguna deja de estar disponible, entonces avisa al cliente de esto, evitando que deje de funcionar o se comporte de manera inesperada. Del mismo modo, si la falla ocurre donde se encuentra el cliente, la conexión con el servidor es anulada, y los demás

clientes no se enteran de la falla porque el servidor sigue trabajando normalmente.

DCOM posibilita el escalamiento de las aplicaciones construidas sobre este modelo. En primer lugar, aprovecha el soporte a máquinas con varios procesadores que ofrece Windows NT, y cuando una aplicación servidora es instalada en una máquina de este tipo, DCOM se encarga de distribuir su funcionamiento entre los diferentes procesadores, liberando al desarrollador de aplicaciones de esta tarea, que es de gran dificultad. En segundo lugar, es posible instalar una o varias aplicaciones en varios equipos que obran como servidores, para que los clientes las puedan acceder y así obtengan tiempos de respuesta que un solo servidor no puede suministrar. A medida que el número de clientes aumenta, también lo hace la carga en los servidores, y la forma de atender este problema consiste en añadir servidores adicionales y reconfigurar los demás servidores, para que puedan distribuirse la carga entre sí y entregar tiempos de respuesta aceptables para sus clientes. No es necesario reinstalar o reconstruir las aplicaciones que están instaladas en los servidores.

En resumen, DCOM como arquitectura para el desarrollo de aplicaciones distribuidas ofrece todas las características de seguridad, extensibilidad, escalabilidad y eficiencia necesarias para producir las aplicaciones que las necesidades actuales requieren. Además, su enfoque orientado a objetos y el soporte que ofrece a las tecnologías de programación existentes la han posicionado como una alternativa realmente viable en el ambiente Windows.

1.4 ARQUITECTURAS COMERCIALES DE COMPUTACIÓN GRÁFICA

En la actualidad, el mercado de la computación gráfica experimenta un crecimiento sin precedentes. Esto ocurre gracias a un factor de suma importancia: la masificación de equipos de cómputo con la capacidad de procesamiento necesaria para manejar aplicaciones con gran cantidad y calidad de contenidos de imagen y sonido. A pesar de que desde los años 70

ya se estaba investigando en el campo de la computación gráfica, aún no existía un mercado que permitiera aprovechar los conocimientos que se habían adquirido hasta ese entonces. Pero en los últimos diez años, con el advenimiento de procesadores más veloces, dispositivos especializados en el manejo de gráficos, y plataformas de desarrollo de aplicaciones multimedia orientadas a aprovechar estas nuevas tecnologías, se ha presentado un gran salto en este campo.

El hecho de que se hayan desarrollado varias arquitecturas para el desarrollo de aplicaciones con alto contenido gráfico es un factor muy importante. En un computador personal actual, el sistema operativo es el encargado de manejar todo lo relacionado con la presentación de datos a través del dispositivo de salida estándar, que es el monitor. Esto significa que cualquier aplicación que desee dibujar algo en él puede usar el sistema operativo como intermediario, sin importar que este último no sea muy eficiente o no provea las características esperadas por la aplicación. Y lo que ocurre normalmente es precisamente esto, el soporte gráfico incluido en los sistemas operativos para computadores personales es de propósito general, y está orientado a proveer los servicios básicos que la mayoría de aplicaciones necesitan.

Esta fue la justificación principal para la creación de las arquitecturas de desarrollo de computación gráfica. La característica primordial que ofrecen es permitir el acceso directo a los recursos de *hardware* que manejan la presentación visual de los programas; en otras palabras, brindan acceso directo al área de memoria que usan las tarjetas de video de los computadores actuales, la cual es más rápida que la memoria RAM que se emplea usualmente. Por otro lado, ofrecen servicios especializados para la computación gráfica, siendo el más importante de ellos el manejo por omisión de tres dimensiones (lo que es primordial para la construcción de aplicaciones con gráficos dinámicos y en movimiento) y la manipulación de texturas, sombras, luces, colores, etc.

Aunque se han desarrollado varias arquitecturas que satisfacen los objetivos ya mencionados, dos de ellas son las predominantes actualmente: OpenGL y DirectX. Ambas son arquitecturas que permiten construir gráficos de alta calidad y realismo, aunque su enfoque está dirigido al procedimiento a seguir para obtener tales gráficos, y no en la descripción de su apariencia. Por ejemplo, ni OpenGL o DirectX proveen mecanismos para pintar directamente figuras geométricas como esferas o cuadrados, si no que tienen un conjunto de instrucciones que, empleadas conjuntamente, puede producir figuras de esta clase.

En los dos siguientes apartados se explican las características propias de cada una de estas plataformas.

1.4.1 OpenGL

Es un modelo propuesto por la compañía Silicon Graphics, que fue diseñado inicialmente para los equipos que esta empresa ofrece. Fue adaptado paulatinamente por muchas empresas, y actualmente existe un consorcio conocido como OpenGL ARB (*Architecture Review Board,* Comité de Revisión de la arquitectura OpenGL), integrado por Silicon Graphics, Microsoft, Intel, IBM, y Digital, entre otros; el cual se encarga de promover la estandarización del modelo y de impulsar su evolución. Actualmente se encuentra en la versión 1.2.

El uso de esta arquitectura está orientado a las aplicaciones de diseño y manufactura asistidos por computador (CAD y CAM respectivamente), en las cuales prima el realismo y la exactitud al generar los gráficos. Debido a que estas aplicaciones funcionan en varias clases de sistemas operativos, como Windows, MacOS y diversas versiones de UNIX, OpenGL brinda soporte a todos estos sistemas. Por otro lado, la mayoría de las tarjetas de video existentes en el mercado soportan OpenGL en *hardware*.

Por otro lado, OpenGL tiene arquitectura cliente - servidor, en la cual varios sistemas pueden estar conectados a un solo motor gráfico, que mantiene la información de estado de todos los sistemas que están conectados a él. Los clientes y servidores pueden estar ubicados en el mismo equipo o en equipos diferentes, lo que brinda transparencia de red al modelo. Está orientado exclusivamente a la computación gráfica, por lo que no cuenta con herramientas para capturar entradas del usuario, ni métodos para almacenar información gráfica en archivos.

1.4.2 DirectX

DirectX es una arquitectura de computación gráfica y multimedial desarrollada, supervisada e impulsada por Microsoft. Ha sido diseñado exclusivamente para la plataforma Win32 (Windows 95 / 98 / NT). Su versión más reciente es la 6.1.

Es una arquitectura dirigida primordialmente al desarrollo de juegos. A pesar de que es posible utilizarla en otro tipo de aplicaciones, ha sido diseñada y optimizada para ofrecer el mejor desempeño posible en el desarrollo de aplicaciones gráficas con alto nivel de realismo, actividad, movimiento constante e interacción estrecha con el usuario a través de dispositivos de entrada como teclado, ratón y palancas de juegos (*joysticks*); propiedades todas que se destacan en los juegos. Además, aprovecha las características de los sistemas operativos Windows para el manejo de memoria y de capacidad de procesamiento.

DirectX está compuesto de dos grandes áreas: DirectX Foundation y DirectX Media. La primera provee servicios de bajo nivel y de acceso a *hardware,* al tiempo que mantiene la independencia de las aplicaciones a las características propias de las máquinas donde funcionan. La segunda contiene herramientas de alto nivel, que se pueden integrar con aplicaciones estándar de Windows y con páginas Web.

En lo que respecta a la construcción de gráficas y animaciones, los componentes de DirectX involucrados en esta área son:

- Direct3D RM (Modo Retenido): Sirve para manejar gráficas en tres dimensiones a un alto nivel. Se usa para modelar escenas, construir y organizar objetos, manejar luces y sombras, crear animaciones y transformaciones, etc. Se encuentra dentro de DirectX Media.
- Direct3D IM (Modo Inmediato): Se encarga de manejar gráficas en tres dimensiones a un nivel más bajo que el cubierto por Direct3D RM. Esto lo logra a través de técnicas para el manejo de luces, texturas, transparencias, etc. Hace parte de DirectX Foundation.
- DirectDraw: Especializado en el manejo de gráficas en dos dimensiones, y en la administración y optimización del uso de la memoria de video de los computadores personales. Al igual que Direct3D IM, se encuentra dentro de DirectX Foundation.

La Figura 11 muestra la arquitectura de los dos sistemas de construcción de gráficas que existen actualmente para los computadores personales de tipo PC. A la izquierda están los componentes que provee el sistema operativo Windows, conocidos en su conjunto como GDI, y a la derecha los componentes que provee DirectX.

1.4.2.1 GDI: El servicio gráfico de Windows

GDI (*Graphical Device Interface*, interfaz gráfica de dispositivo) es el servicio usado por todas las aplicaciones Windows para construir gráficas básicas, tales como ventanas, textos, figuras en dos dimensiones, etc. El objetivo de GDI es separar tanto como sea posible el *hardware* de video de la aplicación, para que las aplicaciones Windows puedan construir gráficas básicas sin importar cuál es el *hardware* de video presente en el computador donde se están ejecutando.

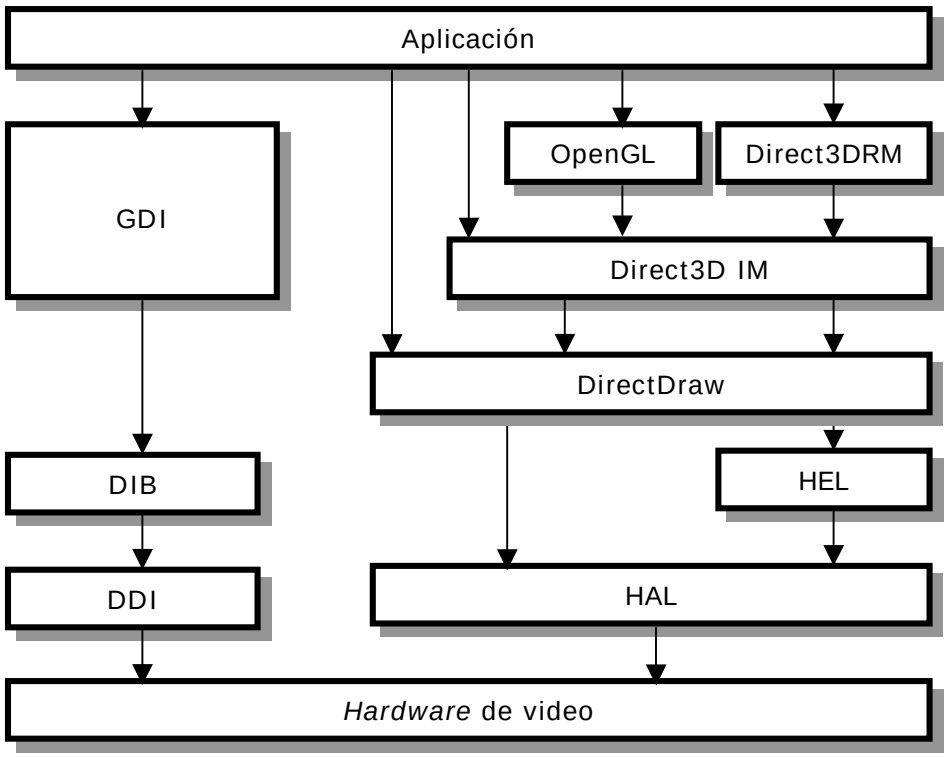


Figura 11. Arquitecturas empleadas para producir gráficas en un PC: GDI y DirectX.

GDI coloca tres capas entre la aplicación y el *hardware* de video, que son:

- GDI propiamente dicho, el cual expone unas interfaces que permiten dibujar las gráficas básicas que ya se han mencionado.
- El manejador de bitmaps o DIB (*Device Independent Bitmap engine* - motor de bitmaps independientes del dispositivo), que transforma las llamadas a las interfaces de GDI en bitmaps, los cuales deben ser transferidos a la memoria de video.
- DDI (*Device Driver Interface* – Interfaz de manejo del dispositivo), que encapsula el *hardware* de video y coloca los bitmaps que recibe del manejador de bitmaps en la memoria de video, para que sean desplegados.

A través de esta arquitectura, GDI permite que una aplicación pueda construir gráficas sin importar cuál es el *hardware* de video disponible y cuáles sus

capacidades. Pero esta ventaja tiene un precio: la construcción de gráficas a través de GDI es un proceso poco eficiente, y no existen servicios de generación de gráficas en tres dimensiones o de escenas gráficas complejas. Por esto se prefiere el uso de DirectX para la construcción de aplicaciones intensivas en el uso de gráficos, que requieren servicios que funcionen eficientemente, y suministren la funcionalidad avanzada que ellas requieren.

1.4.2.2 Direct3D Retained Mode

Direct3D Retained Mode (Direct3D RM) se usa para generar gráficos que deban ser controlados en detalle por la aplicación. Para lograr esto, Direct3D RM realiza las siguientes tareas:

- Manejo de escenas: A partir de marcos donde se colocan los objetos, es posible definir una jerarquía de los elementos que componen una escena. De esta forma, al definir modificaciones, texturas y animaciones un objeto, pueden ser aplicadas a todos los elementos que están debajo del objeto en la jerarquía de marcos. Uno de estos marcos define la cámara, en otras palabras, define el punto y la dirección de observación de la escena.
- Texturas: Además de las funciones de asignación de texturas ya suministradas por Direct3D IM, es posible definir la forma en que una textura envuelve un objeto, para obtener más precisión en su apariencia.
- Luces: El manejo de luces en Direct3D RM es muy amplio, incluye definición de luces de ambiente, direccionales, puntuales y de chorro (*spotlight*), y ubicación de las luces dentro de marcos para incluirlos en la jerarquía de la escena. También es posible modificar la forma en que las luces afectan la apariencia de los objetos, lo que se conoce como calidad de *rendering* (esto será explicado más adelante).
- Animaciones: Es posible animar fácilmente la rotación, la posición y el tamaño de los objetos. También se pueden modificar las caras que componen un objeto, su color, la textura que lo envuelve, las luces que lo iluminan, etc. Los objetos no son animados directamente, para esto son usados los marcos.

Es importante anotar que Direct3D RM, y también Direct3D IM, son servicios que describen los objetos como una serie de caras y no como objetos como tales. Es decir, una esfera en Direct3D está compuesta de un conjunto de triángulos dispuestos para formar una superficie esférica, no por una simple esfera definida por su radio. No existen funciones para construir cubos, conos, tubos, círculos, humo, etc. Estos deben ser construidos a partir de caras.

Direct3D RM provee, un formato de archivo en el cual se pueden almacenar, en formato de texto o binario, todas los componentes, características y configuraciones que describen una escena o animación. Es posible construir una animación y guardarla en un archivo, para que posteriormente pueda ser leída para su generación y/o modificación.

OpenGL es un servicio similar a Direct3D RM, por lo que ambos se encuentran al mismo nivel en la Figura 11. La diferencia entre estos dos servicios es que Direct3D RM ha sido optimizado para funcionar con Direct3D IM y DirectDraw para alcanzar el mejor desempeño posible.

1.4.2.3 Direct3D Immediate Mode

Direct3D Immediate Mode (Direct3D IM) ofrece servicios básicos de dibujo en dos dimensiones, tales como:

- Manejo de la profundidad de la escena (*z-buffering*)
- Construcción de objetos a partir de vértices (puntos en el espacio tridimensional), caras (formadas por vértices) y objetos (formados por caras)
- Transformaciones como desplazamiento , rotación o escalamiento
- Asignación de texturas a los objetos

Las instrucciones de construcción en Direct3D IM se pueden definir de dos formas: lotes de instrucciones conocidas como *execute buffers,* y

DrawPrimitive, una serie de funciones que provee Direct3D para que la aplicación genere los gráficos paso a paso. Cualquiera de las dos formas se puede usar.

Una vez se han construido gráficas a través de Direct3D IM, es imposible tomar control de ellas para modificarlas. En otras palabras, una gráfica construida con un lote de instrucciones o con DrawPrimitive no puede ser controlada una vez ha sido pintada en el *hardware* de video. Aunque esto dificulta la generación de gráficas, el rendimiento obtenido es mayor comparado con el que Direct3D RM ofrece.

1.4.2.4 DirectDraw

DirectDraw, el componente de DirectX que se encuentra en el nivel inferior, es la conexión directa entre la aplicación y el *hardware* de video. Los servicios que ofrece son, básicamente, acceso directo a la memoria de video, definición de superficies que representan la pantalla, de paletas que contienen los colores usados en tales superficies, de secciones para almacenar texturas, etc. También provee funciones para copiar y mover datos de una parte de la memoria de video a otra, de la memoria principal a la de video, y viceversa. No provee servicios para construir gráficas o para definir escenas, esa tarea ha sido encargada a Direct3D IM y a Direct3D RM.

Por otro lado, DirectDraw determina las características del *hardware* de video presente en el computador, y a partir de esa información establece la capa HAL (*Hardware Abstraction Layer* - Capa de abstracción de *hardware*) para presentar el *hardware* de video de manera estándar a la aplicación. Si una característica que la aplicación solicita no está presente entonces la capa HEL (*Hardware Emulation Layer* – Capa de emulación de *hardware*) se encarga de simular, por *software,* la funcionalidad requerida.

Una vez DirectDraw ha establecido las capas HAL y HEL, coloca dos dispositivos a disposición de la aplicación: el *device* y el *viewport*.

- *Device:* Encapsula el sistema de video de un computador y representa la superficie donde se van a dibujar las gráficas. Contiene información de resolución, capacidad, manejo de colores, luces y texturas, etc.
- *Viewport*: Se ubica sobre el *device*, y describe matemáticamente cómo se van a dibujar los objetos construidos con DirectX. Contiene datos que describen el área donde se va a trabajar, tales como la perspectiva, el tamaño del área, el ángulo de visión, etc. Es posible tener en un sistema un solo *device* y varios *viewports*, donde estos últimos representan formas diferentes de ver la escena.

DirectDraw es utilizado por Direct3D IM para traducir las llamadas a sus funciones en operaciones sobre la memoria de video. Y Direct3D IM es utilizado por Direct3D RM para traducir las construcciones gráficas complejas en otras más sencillas, que Direct3D puede manipular y pasar a DirectDraw. De esta forma, es posible utilizar cualquiera de los componentes que DirectX ofrece para generar gráficas, como se ve en la Figura 11. La elección depende del tipo de gráficas que se van a generar y del rendimiento que se desea obtener.

1.4.2.5 Animaciones en DirectX

Para describir completamente una animación en DirectX es necesario contar con la siguiente información:

1. Duración total de la animación, en segundos.
2. Proporción de imágenes por segundo que compondrán la animación. Esta propiedad determina la suavidad del movimiento de los objetos.
3. Tamaño: Se refiere al ancho y el alto de las imágenes que se generarán a partir de la definición de la animación.
4. Calidad de *rendering*: Se refiere a la forma en que se van a pintar los objetos que se encuentran en la animación. Puede ser de las siguientes formas:

- *Wireframe:* Sólo se dibujarán las líneas de contorno de las caras (triángulos) que componen los objetos.
- *Unlit Flat:* Los objetos son rellenados con colores planos.
- *Flat:* Es similar al anterior, pero para cada cara se calcula un color según la cantidad de luz que recibe, y toda la cara se pinta con ese color.
- *Gouraud:* Se utiliza un algoritmo de cálculo de colores para producir un sombreado suave en las caras de los objetos.

5. Modelo de color: Se relaciona con el manejo de las luces de colores. Puede ser de dos valores:
    - Mono: Las luces son monocromáticas (blancas o grises).
    - RGB: Las luces pueden ser de colores.
6. Interpolación de texturas: Se refiere a la forma en que las texturas son colocadas sobre las caras de los objetos. Cada punto en una cara es aproximado a un punto en la textura de acuerdo a alguno de los siguientes métodos:
    - Puntual: Los puntos de la textura se aproximan según el más punto de textura cercano al punto en la cara.
    - Bilineal: Los cuatro puntos de la textura más próximos al punto en la cara son interpolados para obtener el valor correspondiente al punto.
7. *Dithering*: Es un efecto que se aplica a las caras de un objeto, para suavizar el efecto de sombreado producido por el método Gouraud.
8. Archivo .x: En este archivo se encuentran definidos todos los objetos que conforman la animación. Aquí se describen los objetos, las texturas, las luces, los movimientos y las transformaciones que sufren los objetos lo largo de la animación. Cuando se lee una animación de un archivo .x, todos los objetos son colocados automáticamente en un marco.
9. Posición en el eje Z: Es la distancia entre la cámara y el marco donde se encuentran definidos todos los objetos.

2. ANÁLISIS

En pocas palabras, el problema que se va a abordar en el desarrollo de este proyecto consiste en distribuir la ejecución de un trabajo —cuyo tiempo de ejecución es muy grande— en un conjunto de computadores, con el objetivo de reducir tal tiempo de ejecución. Aprovechando el poder de cómputo que ofrecen varios equipos, y teniendo en cuenta las consideraciones de arquitectura del sistema y de distribución de procesamiento que se mencionarán en las siguientes páginas, se pretende alcanzar el objetivo ya mencionado.

El proceso de análisis se ha dividido en tres partes. En la primera se considera cuál es el trabajo más adecuado para enfrentar el problema, en la segunda se plantea la arquitectura del sistema distribuido que se va a utilizar para llevar a cabo los trabajos descritos en la parte anterior, y en la tercera se determina la manera en que se va a manejar la distribución de las tareas que componen tales trabajos. Este proceso no fue secuencial, pues a medida que se iban encontrando consideraciones y problemas que se debían atender, los tres puntos anteriores fueron actualizados de acuerdo al resultado del análisis de tales consideraciones.

2.1 TIPO DE TRABAJO A DISTRIBUIR

Generalmente, cuando se construye un sistema para procesamiento distribuido, éste es diseñado para procesar múltiples trabajos del mismo tipo (por ejemplo: procesamiento de imágenes, señales, simulaciones, etc.)

partiéndolos en unidades atómicas (tareas) y distribuyendo su procesamiento sobre las estaciones pertenecientes al sistema. Este modelo se ajusta mucho al denominado modelo MIMD (Múltiples Instrucciones Múltiples Datos), propuesto por Michael Flynn en 1972, en donde cada una de las máquinas es independiente y contiene su propio flujo de instrucciones y datos. Dentro de este modelo están además de los conjuntos de computadores o *farms,* los sistemas SMP (Multiprocesamiento Simétrico).

Esta arquitectura permite el manejo de trabajos que se desean manejar con el sistema objetivo de este proyecto. Tales trabajos deben tener como característica principal la posibilidad de poderse segmentar en unidades independientes que no necesiten de los datos de otras unidades para su procesamiento.

El tamaño (tiempo de procesamiento) de cada una de estas unidades individuales (tareas) es un parámetro importantísimo en el rendimiento general esperado. Si el tamaño de la tarea es muy pequeño en relación con los tiempos de comunicación, entonces las ganancias en tiempo por la distribución del trabajo se desvanecerían. Y en el caso que la tarea sea muy grande, los tiempos obtenidos por la distribución de la carga no mejoraran mucho con relación a los obtenidos con el procesamiento utilizando una sola estación. Por esto, el sistema funcionara mejor con trabajos que puedan ser segmentados en tareas de un tamaño “mediano”. Este tamaño es relativo y su valor óptimo depende del tipo específico de trabajo que se desea dividir.

Un buen ejemplo de este tipo de trabajo lo constituyen las animaciones. Para generar una animación se necesita bastante capacidad de procesamiento, y cuando tal capacidad es suministrada por un solo computador, el tiempo de generación se torna demasiado grande. Por otro lado, la división natural del trabajo que consiste en generar una animación está determinada por las imágenes (*frames*) que la componen. Cada imagen es una parte de la animación, y a partir de los datos que describen la animación en general es

posible generar cualquier imagen, sin necesitar la información de las imágenes vecinas o de la inicial. Además, la capacidad de procesamiento para generar cualquier imagen en una animación tiende a ser constante.

Por todo lo anterior, se decidió implementar la generación de animaciones como ejemplo de proceso distribuido para este proyecto.

2.2 MODELO DEL SISTEMA

2.2.1 Distribución de hardware, datos y control

En primer lugar se debe considerar la forma en que se van a dividir los componentes básicos de un sistema distribuido, ya mencionados en la sección 1.1.3.

Los equipos de computación más populares y accesibles actualmente pueden comportarse como sistemas autónomos, es decir, poseen todo lo necesario para llevar a cabo las tareas más usuales sin necesidad de auxilio externo. Por lo tanto, la forma más natural de aprovechar el poder de cómputo que ofrecen estos equipos consiste en dotarlos con un sistema de comunicación, equivalente a una red que los interconecte. Esto se hace para conservar la configuración inicial de cada equipo intacta (procesador, memoria, almacenamiento de datos, etc.) sin necesidad de construir un sistema especial y específico para cada caso, lo que reduce costos y tiempo de trabajo y permite distribuir lo que más se necesita para el problema que se está considerando: el procesamiento. Teniendo en cuenta todo esto, se ha decidido que la arquitectura de *hardware* de el sistema distribuido en consideración consistirá en una serie de equipos con procesamiento y espacio de memoria exclusivos para cada equipo, comunicados por una red de interconexión (ver Figura 3: *Sistema distribuido con un bloque de memoria para cada procesador*).

Con respecto a la distribución de datos, se ha observado que para este caso la proporción de datos a distribuir es mucho menor respecto al procesamiento. Por lo tanto, se ha adoptado un esquema mixto, en el cual la información que describe la configuración y las características del trabajo a distribuir es copiada a todos los procesadores, mientras que los datos usados para procesar el trabajo son almacenados en un recurso compartido que esté al alcance de todos los equipos en el *farm.* Se pretende lograr un equilibrio entre la velocidad de lectura de los datos y la capacidad de procesamiento, pues una vez ya se han leído tales datos, cada equipo puede proceder a procesar la parte del trabajo que le corresponde, y cuando ha terminado puede recibir la siguiente parte sin necesidad de repasar los datos que ya había leído anteriormente.

Finalmente, el control del sistema distribuido es un factor que siempre tiene consecuencias positivas y negativas, según la forma en que se implemente. Aunque un sistema de control distribuido hace al sistema más robusto y resistente a fallas, el desarrollo del mismo se torna más complicado, y la distribución de datos se vuelve también más compleja, pues la información de estado del sistema tiene que viajar por la red de interconexión periódicamente, toda o en parte. En consecuencia, se ha decidido establecer un sistema de control centralizado, es decir, colocar el sistema de control en un punto específico del sistema, desde el cual se conoce el estado de todos los equipos que hacen parte de él y se aplica un algoritmo de control (también centralizado) para tomar las decisiones de administración con base en el resultado del algoritmo.

Retomando el esquema de fragmentación que se planteó en la sección 1.1.3, se encuentra que el sistema planteado hasta ahora no es completamente distribuido, pues el grado de control es el mínimo. Pero el grado de distribución de *hardware* y de datos es alto (ambos con niveles 3), por lo que se puede decir que el sistema es parcialmente distribuido.

2.2.2 Distribución de procesamiento

Volviendo a la distribución de procesamiento, que es el factor más importante entre los que se están considerando, se describieron dos formas básicas de distribuir la carga entre los equipos que componen el sistema distribuido: estática y dinámica (ver sección 1.2.2).

Aunque la distribución estática de carga permite conocer de antemano el comportamiento del sistema, y obrar con base en esa información para obtener el mejor desempeño, no es fácil estimar la respuesta de los componentes del sistema a la realización de distintos trabajos. Existen varios factores que complican esta tarea, entre los cuales se puede mencionar:

- Dependiendo de la configuración del trabajo, algunas partes del mismo pueden hacerse en menos tiempo que otras. En el caso de las animaciones, es posible que los primeros segundos de la animación se compongan de un solo objeto, moviéndose de manera simple, y repentinamente entran todos los demás personajes, interactuando de manera compleja unos con otros. Obviamente, esta última parte exige más trabajo para ser generada que la primera, y no es fácil determinar *a priori* la proporción de trabajo adicional o el momento en que tal trabajo se requiere. Esto también es válido para otros tipos de tareas, como el manejo de hojas de cálculo o la simulación.
- La configuración de *hardware* de los equipos que hacen parte de un sistema puede ser bastante heterogénea. En particular, la configuración de memoria, medios de almacenamiento y red puede determinar una gran diferencia entre dos equipos cuya capacidad de procesamiento es similar.
- El uso de los computadores que componen el sistema puede ser exclusivo o no. Por ejemplo, un equipo que se encuentra procesando parte de un trabajo puede tener el control del sistema durante un tiempo, para luego cedérselo a otro más adelante. O también, el sistema operativo puede activar una rutina de administración mientras el equipo

está trabajando. Además, existe un fenómeno que es muy común, sobre todo en el medio colombiano, que consiste en la imposibilidad o dificultad de contar con equipos de dedicación exclusiva a una sola tarea. Tarde o temprano será necesario realizar en un equipo algo para lo que no estaba planeado inicialmente.

Todos estos factores complican en gran medida la estimación del comportamiento del sistema, y por consiguiente, una distribución estática de carga. En cambio, si se considera la distribución dinámica, se encuentra que la principal característica de esta arquitectura consiste en la administración del sistema de acuerdo a la forma en que se está realizando el trabajo. A medida que el tiempo transcurre, el centro o centros de control del sistema pueden conocer la forma en que los componentes del *farm* han reaccionado a la ejecución de las tareas que componen el trabajo, y obrar en consecuencia. De esta forma, se evitan los problemas de exclusividad de uso de los equipos, de la configuración de *hardware* de los mismos y de las características únicas y propias de cada trabajo, puntos mencionados unos párrafos atrás.

A pesar de que la distribución dinámica trae consigo problemas de posible subutilización de equipos, y de sobrecarga en la red de comunicación, estos problemas pueden ser enfrentados con una política de distribución cuidadosamente diseñada y con reglas claras para determinar el comportamiento del sistema. Estos dos puntos son abordados en las secciones que se encuentran a continuación.

2.2.3 Esquema de distribución dinámica de carga

De acuerdo a lo expuesto en la sección 1.2.5, existen varios factores que caracterizan un sistema dinámico de distribución de procesamiento. Para el problema particular que se está estudiando, se han decidido las siguientes características para el sistema que atenderá tal problema:

2.2.3.1 Políticas de información

Para conocer la forma en que se está comportando el sistema es necesario leer las variables que describan su estado, y determinar el alcance de las operaciones de lectura de tales variables.

Para este caso se ha decidido recurrir a la recolección periódica de la información de desempeño, ya que esto asegura que el módulo de control conocerá constantemente el estado de las estaciones de procesamiento, y podrá tomar decisiones de acuerdo a tal estado. Debido a la gran influencia del tiempo entre lecturas de información de estado en el desempeño del sistema de comunicación, es necesario implementar un esquema flexible de adquisición del la información de estado, como la posibilidad de modificar el período de lectura o la de conocer de antemano el comportamiento del sistema de comunicación para así tomar decisiones respecto al período de lectura (ver sección 1.2.5.2.2). Además, debido a la ubicación del módulo de control en un punto único y conocido del sistema distribuido, es posible leer la información de desempeño de todos los componentes del sistema distribuido. En términos más precisos, el horizonte del balance de carga es totalmente completo. Esta idea es explicada con detalle en la sección 1.2.5.3.6.

Por otro lado, considerando que el procesamiento es el recurso que finalmente determina el desempeño del sistema distribuido, se ha decidido obtener el porcentaje de uso de la CPU de las estaciones de procesamiento como medida de la carga de trabajo que éstas tienen. A pesar de que variables como la cantidad libre de memoria principal o el tamaño del archivo de intercambio (*swap*) también son variables que describen el desempeño, el procesador es el recurso principal del sistema, y por lo tanto es suficiente conocer el estado del esta variable para hacerse una idea del estado de las estaciones de procesamiento.

2.2.3.2 Cooperación

Una vez el trabajo en consideración ha sido dividido en tareas, éstas pueden ser asignadas a los equipos para que las realicen. De acuerdo a la descripción de trabajos y tareas hecha anteriormente, las tareas que componen una animación son independientes entre sí en el sentido de que no es necesario que los equipos intercambien información de las tareas entre ellos o con el módulo de control para realizar las tareas. Esto nos conduce a un esquema no cooperativo (ver sección 1.2.5.1). La ventaja de este sistema consiste en la menor complejidad de programación de las estaciones de procesamiento, pues no es necesario considerar la transferencia de información diferente al desempeño de las estaciones de trabajo.

2.2.3.3 Manejo de las tareas ya asignadas

Este punto está estrechamente relacionado con los dos anteriores, ya que la política de información del sistema y el esquema de cooperación afectan la forma en que se van a manipular las tareas.

Cuando una tarea ha sido asignada, el modo de ejecución de la misma será ininterrumpido (ver sección 1.2.5.2.1), la estación de trabajo tendrá la responsabilidad de ejecutar esa tarea. El módulo de control no reasignará la tarea hasta que ésta haya sido ejecutada por la estación de procesamiento.

Por otro lado, gracias a la periodicidad de la obtención de la información de desempeño de las estaciones de trabajo, es posible conocer si una estación está trabajando correctamente. Por muchas razones, entre ellas fallas en la ejecución de los programas, exceso de carga en el procesador, o falta de memoria o espacio en disco, es posible que una estación deje de trabajar. En ese caso, ya no será posible obtener la información de procesamiento. Cuando eso ocurra, el módulo de control deberá retirar al equipo afectado del sistema y deberá reasignar la tarea o tareas que el equipo tenía asignadas en el momento de detenerse. No se han considerado otros casos en los cuales se

reasignen tareas ya entregadas a estaciones de trabajo, para simplificar el modelo.

2.3 MODELO DE DISTRIBUCIÓN DE TAREAS

Para que el sistema distribuido pueda balancear de manera transparente y eficiente la carga de procesamiento entre las distintas estaciones de procesamiento, se utilizará una cola centralizada ubicada en el módulo de control, el cual obra como cliente del sistema registrando las peticiones de procesamiento (tareas que componen uno o varios trabajos) en la cola, para que los servidores (estaciones de procesamiento) vayan en busca de trabajos por procesar. De hecho, el módulo de control no asigna directamente la máquina en que se va a procesar una tarea determinada. Todo lo que tiene que hacer es enviar la tarea a la cola de peticiones y luego buscar el resultado en la cola de respuesta correspondiente. Las tareas que se encuentran en proceso en un momento determinado son almacenadas en una lista auxiliar construida para tal efecto.

En vez de invertir tiempo en la implementación de sofisticadas arquitecturas de balanceo dinámico de carga, la idea es diseñar las estaciones de procesamiento para que solo busquen trabajos a la cola central. Una vez que se tiene esto, es fácil añadir múltiples instancias de procesamiento adicionales corriendo en sus respectivas estaciones, apuntando todas a la misma cola. Ya que cada servidor compite con los otros por trabajos en la cola, el balanceo de carga sucede natural e implícitamente: el servidor con la mayor capacidad de procesamiento visitará la cola la mayor cantidad de veces y procesará la mayor cantidad de peticiones. Incluso se puede sofisticar el modelo, incorporando el chequeo de la memoria y carga del CPU, en los servidores de procesamiento, antes de ir por mas trabajos a la cola. Este diseño, a diferencia del modelo centralizado, reparte la responsabilidad de la distribución de carga a los servidores de procesamiento, facilitando su implementación.

Con este diseño, si se adicionan más estaciones de procesamiento, los trabajos son procesados más rápidamente, sin esfuerzo adicional por parte del módulo de control. Si uno de los servidores de procesamiento sale de servicio, el resto de servidores seguirán procesando los trabajos, aunque obviamente, el rendimiento general se verá afectado.

El papel del módulo de control se ve aliviado. No tiene que asignar tareas a las estaciones de procesamiento, pues este proceso es realizado tácitamente por las estaciones, en el momento que acceden la cola de tareas. Con respecto a las tareas, el servidor solamente debe alimentar la cola de tareas con la descripción de las mismas, por lo que el tamaño del sistema no afecta el desempeño del módulo de control.

Por otro lado, el módulo de control logra más capacidad de proceso para supervisar el desempeño de las estaciones a través de la obtención del porcentaje de uso de la CPU de cada una de ellas. Cuando el módulo de control nota que una estación ha dejado de responder, puede consultar la lista de tareas en proceso para encontrar cuáles tareas tenía asignadas la estación que ha fallado, y colocar esas tareas de regreso a la cola de tareas en espera. Este esquema beneficia en gran medida el comportamiento el sistema, ya que el costo por sobrecarga o por fallas es mucho menor.

Finalmente, el esquema de distribución descrito, aunque no es adaptativo en su concepto, sí lo es en su comportamiento. Cuando hay estaciones de procesamiento que no están trabajando correctamente, éstas son relegadas por las estaciones que sí están haciendo bien su trabajo, y el módulo de control puede darse cuenta de esto y reasignar tareas a las estaciones que sí trabajan. De esta forma, la política de distribución de carga cambia en beneficio de las estaciones eficientes, siendo esta la característica principal de un sistema adaptativo (ver sección 1.2.5.1).

2.4 LISTADO DE REQUERIMIENTOS

Como conclusión del proceso de análisis del problema, y para resumir todos los puntos que se han mencionado a lo largo de este capítulo, se presenta a continuación la lista de requerimientos del sistema. En ella se pretende describir detalladamente qué debe hacer el sistema, y cómo se debe comportar según el estado en que se encuentre.

2.4.1 Manejo de las estaciones de procesamiento (*farm*)

2.4.1.1 Registro y desregistro de equipos

El proceso de registro de equipos se lleva a cabo de forma manual por el operador del módulo de control (administrador). Este inserta la ubicación de la estación de procesamiento, en forma de dirección IP, o nombre del equipo en la red; a continuación, el administrador accede al módulo de interfaz (*Wrapper*) en la estación de procesamiento nueva, y si todo sale bien, entonces el equipo es registrado como componente del *farm* y agregado a la lista de estaciones de procesamiento. Para retirar un equipo del *farm,* basta con eliminar la entrada correspondiente en la lista de estaciones. Ambas operaciones deben poder hacerse en todo momento, tanto cuando el sistema está inactivo como cuando hay trabajos en proceso.

Cuando el administrador va a terminar su ejecución manda un mensaje de "desregistrar" a todo el *farm.* También guarda la lista de estaciones que componen el *farm*, para así registrarlas automáticamente cuando se vuelva a arrancar.

2.4.1.2 Habilitación y deshabilitación de equipos

Los procesos de habilitación y deshabilitación consisten, respectivamente, en activar un equipo para que busque tareas en la cola de tareas en espera, y en impedir que el equipo acceda a esta cola. Ambas operaciones también se deben poder hacer en todo momento.

2.4.1.3 Información de desempeño de los equipos

Se puede configurar en el administrador un período de “pulso”, el cual se pasa a los equipos en el *farm* cuando son registrados. Estos envían entonces los mensajes de “pulso”, constituidos por el porcentaje de uso de la CPU, el cual debe ser leído por el administrador.

2.4.1.4 Información de los equipos que se va a mostrar al usuario

Los datos de cada estación que se van a mostrar en la interfaz de administración comprenden:

- Nombre y ubicación
- Porcentaje de uso de la CPU
- Trabajo Actual que se esta procesando
- Tarea que se esta procesando
- Estado del equipo
    - Activo (esta procesando) - Verde
    - Inactivo (puede recibir trabajos pero no tiene) - Amarillo
    - Deshabilitado por el usuario - Rojo
    - Muerto (no responde) – Negro

Cuando se detecta que una Estación está “Muerta” esta no se desregistra, simplemente deja de ejecutar tareas porque ya no puede acceder a la cola de tareas.

2.4.2 Manejo de trabajos

2.4.2.1 Inserción y eliminación de trabajos

Un trabajo describe una animación completa. Cuando se añade un nuevo trabajo al sistema, se deben llenar las siguientes propiedades:

- Nombre
- Duración (en segundos)
- Frecuencia de imágenes por segundo
- Ancho y alto de los imágenes

- Calidad de *rendering*
- Modo de color
- Modo de interpolación de texturas
- Uso de *dithering* (sí/no).
- Posición de la animación en el eje z.
- Ubicación del archivo .x que describe la animación.
- Utilización de compresión para los archivos que contienen las imágenes (sí/no).
- Utilización de línea de comandos para acceder a las estaciones de procesamiento (sí/no).
- Descripción breve de la animación.

La descripción de las características gráficas de una animación se encuentra en la sección 1.4.2.5.

Una vez las propiedades anteriores han sido suministradas por el usuario, el trabajo es añadido a la cola de trabajos. Además se le avisa a todas las estaciones el nombre del nuevo trabajo y todas sus propiedades. Mientras no haya estaciones registradas en el *farm* no se pueden agregar trabajos.

Para eliminar un trabajo, solamente será necesario borrar la entrada correspondiente de la cola de trabajos, y avisar a todas las estaciones registradas sobre la eliminación. Es responsabilidad del usuario eliminar el o los archivos que describen el trabajo.

2.4.2.2 Ejecución de los trabajos

El proceso de *rendering* se hace mientras hayan trabajos en cola. Los trabajos pueden estar en alguno de los siguientes estados:

- En Espera
- En Proceso
- En Pausa (se ha detenido su procesamiento, o no se desea generar todavía)

- Terminado

Luego de terminada una animación se debe llamar una aplicación independiente que se encargue de generar un archivo de video (.AVI) con base en las imágenes (*frames*) obtenidas.

2.4.2.3 Información de los trabajos que se va a mostrar al usuario

Para cada trabajo se van a mostrar los siguientes datos:

- Nombre
- Número de tareas que la componen
- Tareas por segundo (imágenes por segundo)
- Estado del trabajo

Además, la evolución de la ejecución del trabajo también debe mostrarse al usuario, es decir, el tiempo de iniciación, la duración de la ejecución del trabajo, los posibles mensajes de error, etc.

2.4.3 Manejo de tareas

2.4.3.1 Descripción de tareas

Una tarea es cada una de las imágenes que componen la animación. Cada tarea se identifica por la animación a la que pertenece y el instante de tiempo (en segundos) donde se encuentra dentro de la animación. Para cada trabajo se deben manejar las siguientes estructuras de manipulación de tareas:

- Cola de tareas en espera: Contiene las tareas que aún no han sido ejecutadas. Esta lista es guardada cuando el módulo administrador termina su ejecución, para todos los trabajos.
- Lista de tareas en proceso: En ella se encuentran las tareas que están siendo atendidas por las estaciones que están activas en el *farm.*
- Lista de Tareas Terminadas: Almacena los identificadores de las tareas que ya han sido terminadas.

2.4.3.2 Despacho de tareas

El esquema de despacho de trabajos consiste en varias etapas:

- Alimentación de la Cola de Tareas desde la Cola de Trabajos: Las tareas son insertadas en la Cola de Trabajos buscando mantener un tamaño determinado (configurable por el usuario). En el caso que el Trabajo haya sido guardado mientras aun estaba en ejecución entonces el sistema lee la lista de tareas en espera para saber cuáles hacen falta.
- Obtención de Tareas: Cada estación va a la cola de tareas para obtener una tarea. Esta tarea además se inserta en una cola de tareas en proceso, con el nombre de la estación que la esta procesando.
- Cuando la tarea ha sido terminada, la estación avisa al administrador que ya termino la tarea, y este último procede a retirarla de la lista de tareas en proceso y a insertarla en la lista de tareas terminadas.

2.4.4 Arquitectura de las estaciones de procesamiento

En cada estación de procesamiento se ejecutan dos componentes: uno que se encarga de la comunicación entre la estación y el administrador (módulo de interfaz) y otro que realiza las tareas (módulo de procesamiento). Ninguno de los dos componentes requiere interfaz de usuario.

Tanto el módulo de interfaz como el módulo de procesamiento funcionan como procesos independientes. El módulo de interfaz funciona como un proceso enlazado constantemente con el administrador a través de RPC, mientras que el módulo de procesamiento es cualquier proceso que se pueda comunicar con el módulo de procesamiento a través de RPC o línea de comandos. Esto permite, al separar el procesamiento de la comunicación y el control, que puedan ser utilizados programas de terceros como módulos de procesamiento, siempre y cuando éstos tengan una interfaz de línea de comandos.

El módulo de interfaz recibe del módulo administrador el directorio donde esta definido el trabajo, el archivo principal del trabajo y el directorio de colocación de las imágenes generadas, además de todas las características gráficas de un

trabajo mencionadas en el punto 2.4.2.1. Todos estos datos son pasados al módulo de procesamiento para que ejecute las tareas que componen el trabajo. Una vez el módulo de procesamiento termina la tarea, avisa al módulo de interfaz de esto, quien a su vez avisa al administrador para que pueda acceder a la cola de tareas y obtener la siguiente tarea que va a procesar.

El módulo de interfaz también puede parar la ejecución del módulo de procesamiento (bajando el proceso). Si el módulo de procesamiento se bloquea, el módulo de interfaz sigue enviando el mensaje de pulso, por lo que la estación no se considera muerta. Para poder asignar la tarea otra vez, se debe poder deshabilitar la estación de trabajo cuyo módulo de procesamiento se haya bloqueado.

Para generar las animaciones, el módulo de procesamiento utilizará el procedimiento de generación de animaciones descrito a lo largo del punto 1.4.2. Este proceso oficiará finalmente como el servidor de procesamiento del sistema distribuido.

Existen otros parámetros que se deben tener en cuenta en el momento de analizar la construcción de un sistema de distribución de tareas. Estos ya han sido contemplados y predeterminados por la arquitectura del sistema operativo y del modelo de componentes distribuidos.

3. DISEÑO

3.1 FUNCIONAMIENTO GENERAL

La Figura 12 describe el funcionamiento general del sistema, el cual ha sido denominado *Antares*. La estación de administración divide cada uno de los trabajos en tareas. Estas tareas son entonces ejecutadas por las estaciones de procesamiento registradas en el sistema.

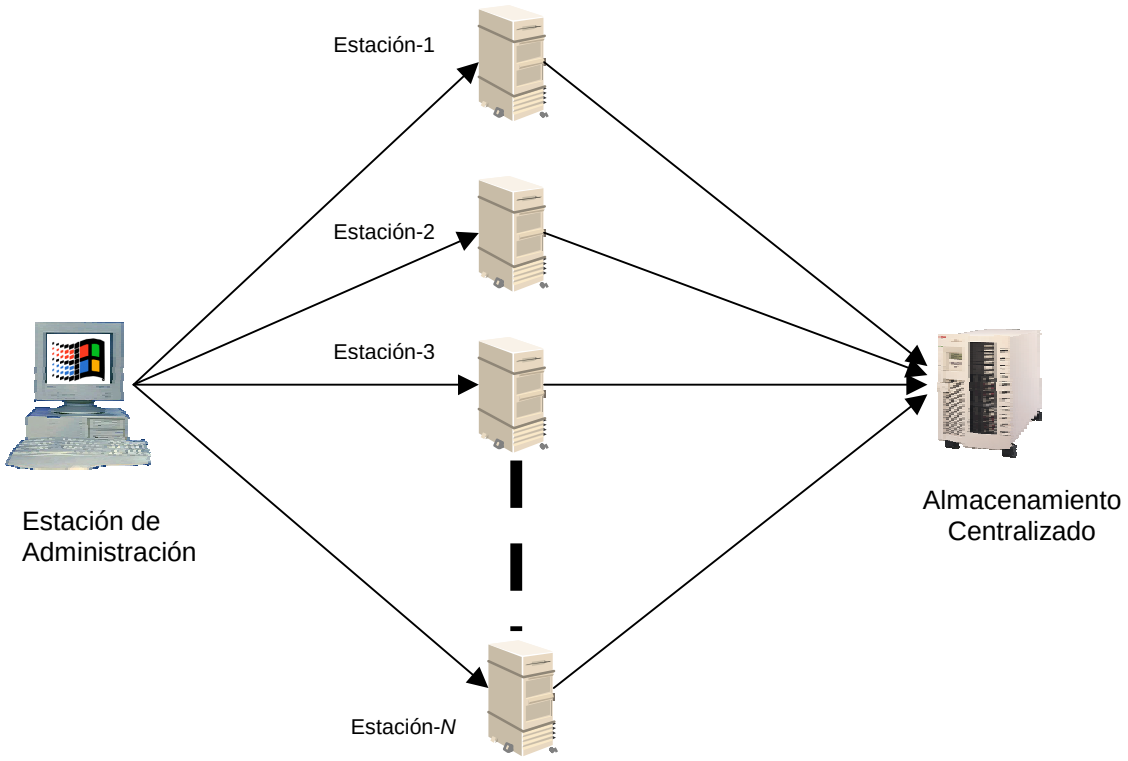


Figura 12. Diagrama de funcionamiento general de *Antares.*

Luego que cada estación de procesamiento ha terminado el trabajo asignado, envía el resultado a la estación central de almacenamiento y avisa a la estación de administracion que se encuentra nuevamente disponible para procesar otro trabajo. Esta operacion continúa hasta que se han procesado todas las tareas que componen un trabajo, y cuando éste ha sido terminado, se procede con los demás trabajos presentes en la cola, hasta terminarlos todos.

3.2 MODELO DE COMPONENTES

La Figura 13 describe el diagrama de componentes de la aplicación y cómo estos interactúan entre sí para implementar la arquitectura previamente descrita.

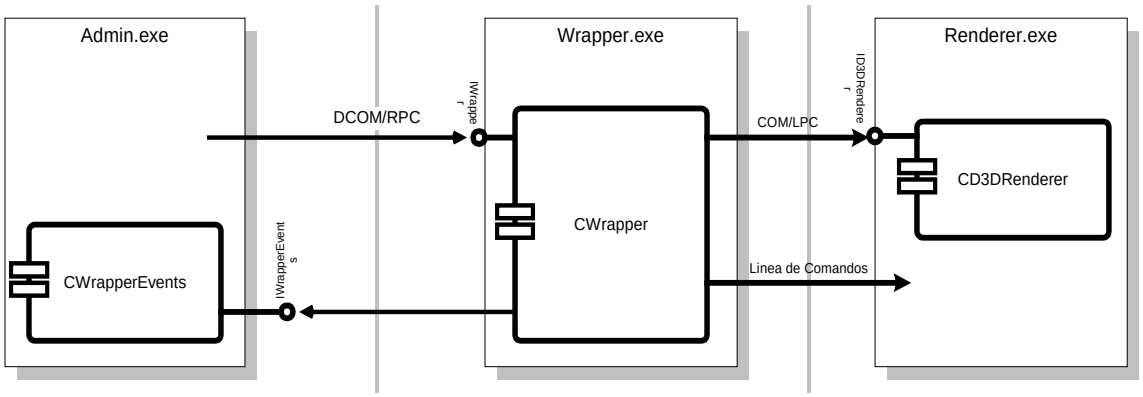


Figura 13. Diagrama de componentes de *Antares.*

Antares esta formado por tres componentes o módulos principales, que a continuación se describen brevemente:

- Módulo administrador (*Admin*): Es el responsable del registro de estaciones de procesamiento, del manejo de las colas de trabajos y de tareas, y de la asignación de las tareas a las estaciones. También esta al

tanto del desempeño de las estaciones; para esto, recibe periódicamente información del estado de cada una de las estaciones que conforman el sistema.

- Módulo de Interfaz (*Wrapper*): Actúa como capa intermedia de abstracción entre el Administrador y el módulo remoto. Se ejecuta en cada una de las estaciones de procesamiento, donde obtiene información sobre el estado del módulo remoto y de la propia estación, transmitiéndola luego al Administrador.
- Módulo de Procesamiento (*Renderer*): Está encargado exclusivamente del procesamiento de cada una de las tareas asignadas que componen un trabajo. Por lo tanto, el diseño del módulo de procesamiento depende del tipo de trabajo que se quiera procesar. El módulo de interfaz puede comunicarse con este ya sea a través de un llamado remoto o con una serie de parámetros pasados en la línea de comandos del módulo de procesamiento en el momento de su ejecución.

En los siguientes puntos se expondrán las diferentes consideraciones de diseño que se tuvieron en cuenta para la construcción de Antares; la explicación detallada de los componentes arriba mencionados se realizará más adelante.

3.3 ESTADOS DE UNA ESTACIÓN DE PROCESAMIENTO

La Figura 14 muestra el diagrama de transición de estados de una estación de procesamiento registrada en el sistema.

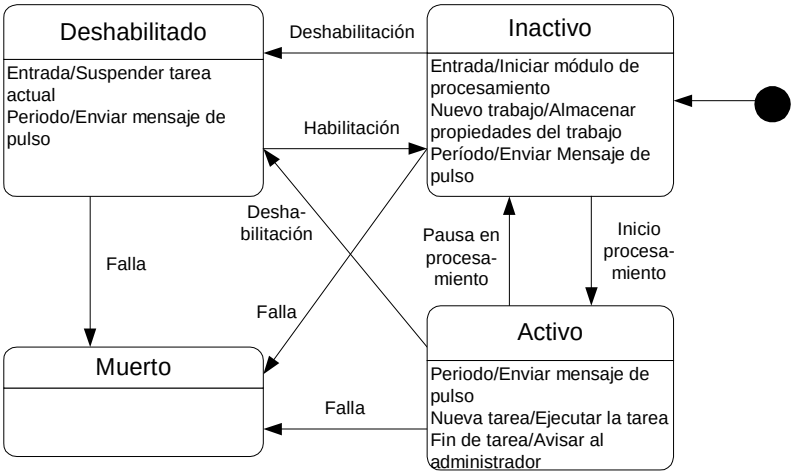


Figura 14. Diagrama de transición de estados de una estación de procesamiento

A continuación se describen cada uno de estos estados:

- **Inactivo:** es el estado en el cual se inicia una estación, e indica que esta en espera de una asignación de tareas.
- **Activo:** indica que la estación esta procesando una tarea asignada. El momento en que la estación termina de procesar una tarea, vuelve al estado Inactivo.
- **Deshabilitado:** representa una estación que se encuentra deshabilitada indefinidamente por el usuario. Este lo puede hacer a través de la interfaz de usuario del Administrador para evitar que se le asignen tareas (por ejemplo: la estación es necesitada temporalmente para otras labores).
- **Muerto:** cuando una estación se registra en el sistema, esta comienza a enviar periódicamente (con un intervalo de tiempo configurable por el usuario) información sobre su estado. A este evento le denominamos "pulso" de la estación. Si el Administrador no recibe esta información durante un intervalo de tiempo, cambia automáticamente el estado de la estación a "muerto", devuelve cualquier tarea asignada a la cola y no le

vuelve a asignar más tareas. Una Estación puede permanecer en este estado temporal o de manera permanente, dependiendo del motivo que originó la no llegada del evento de "pulso". En el caso de que el Administrador vuelva a obtener los eventos de pulso de la estación, este cambiará su estado a inactivo y se le volverán a asignar tareas.

3.4 ESTADOS DE UN TRABAJO

La figura 15 muestra el diagrama de transición de estados de cada trabajo en el sistema.

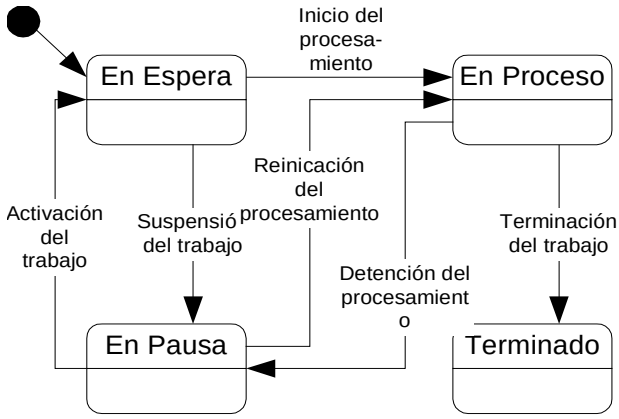


Figura 15. Diagrama de transición de estados de un trabajo

A continuación se describen cada uno de estos estados:

- **En Espera:** Es el estado en el cual se inicia un trabajo e indica que esta en espera de que se comiencen a procesar las tareas que lo componen.
- **En Proceso:** Estado del trabajo mientras esta siendo atendido por las estaciones de procesamiento.
- **En Pausa:** Este estado se puede presentar en dos situaciones: a) El trabajo ha sido detenido cuando ya ha empezado su procesamiento, o b) El trabajo, que estaba en estado de espera, ha sido colocado en pausa

para evitar que pase al estado de proceso. Esto se hace para que los trabajos puedan ser priorizados.

- **Terminado:** un trabajo llega a este estado luego que se culmina el procesamiento de todas las tareas que lo componen. En este momento se puede conocer el resultado general del procesamiento del trabajo (p. ej. ver el video de la animación resultante).

3.5 SUBSISTEMA DE GENERACIÓN DE TAREAS

El corazón del Administrador de Antares lo integra el subsistema encargado de "alimentar" de tareas a las estaciones disponibles (cuyo estado sea inactivo). Este subsistema esta compuesto principalmente por dos hebras (*threads*) que trabajan sobre la estructura de datos en un modelo de colas *Productor de tareas-Consumidor de tareas*. Este modelo provee un poderoso mecanismo para la transferencia de información entre hebras.

La hebra consumidora trata de obtener tareas de la hebra productora, a través de un área compartida (*buffer*). Si las tareas no están disponibles inmediatamente, la hebra consumidora entra en un estado de espera hasta que la hebra productora señale la disponibilidad de tareas por medio de un evento. Igualmente, la hebra productora trata de enviar tareas al consumidor. Si no hay espacio en el buffer para almacenar las tareas, la hebra productora entra en pausa hasta que la hebra consumidora avisa que hay espacio disponible en el buffer. La Figura 16 aclara este concepto.

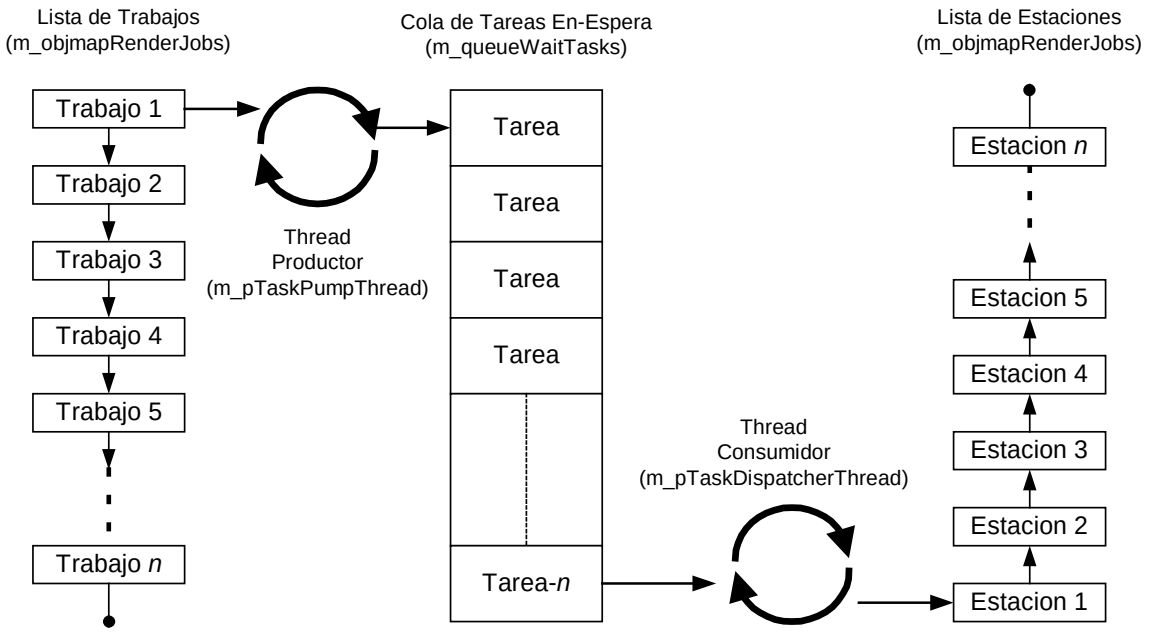


Figura 16. Subsistema de asignación de tareas.

3.6 DISEÑO DEL MÓDULO ADMINISTRADOR

Este componente es una aplicación diseñada bajo el modelo Documento-Vista, que le permite al usuario tener una representación grafica (vista) de los datos que el sistema maneja (documento).

3.6.1 Clases de almacenamiento de datos

Con estas clases se guardará y se manipulará toda la información necesaria para el funcionamiento del administrador.

- **CAdminDoc**: Esta clase representa el documento de la aplicación y contiene principalmente:
    - La información de configuración del sistema,
    - Las listas de estaciones,
    - La cola de trabajos,
    - La cola de tareas en espera de ser procesadas,

    - El subsistema de asignación de tareas, y
    - La lista de eventos definidos para el sistema.
- **CRenderStation**: Representa a cada una de las estaciones registradas en el sistema.
- **CJob**: Esta clase representa cada uno de los trabajos que se desean procesar.
- **CJobTask**: Representa cada una de las tareas (o unidades en las cuales se divide cada trabajo) en espera de ser procesadas.

3.6.2 Clases de presentación de datos y de comandos

Para mostrar la forma en que la aplicación se está comportando, se utilizan tres vistas diferentes del documento, representadas cada una por las siguientes clases:

- **CFarmView**: Lista que muestra las estaciones registradas y algunas de sus propiedades como:
    - Nombre,
    - Nombre del trabajo que está procesando,
    - Identificador de la tarea en proceso,
    - Estado de la estación
    - Porcentaje de Carga en el CPU de la estación en tiempo real. Este indicador nos permite saber que tan cargada esta la Estación y en base a esto tomar la decisión sobre la asignación de tareas.
- **CJobsView**: Lista que muestra los trabajos insertados y algunas de sus propiedades como:
    - Nombre,
    - Número de tareas en las que se subdividió el trabajo,
    - Número de tareas (imágenes) por segundo que tiene el trabajo, y
    - Estado del trabajo.
- **CEventView**: permite ver los diferentes eventos (mensajes de información, advertencia o error) que se presentan durante el funcionamiento del sistema.

La Figura 17 muestra la relación entre el documento de la aplicación (clase *CAdminDoc*) y las vistas que muestran los datos y el estado del mismo (clases *CFarmView, CJobsView y CEventView*).

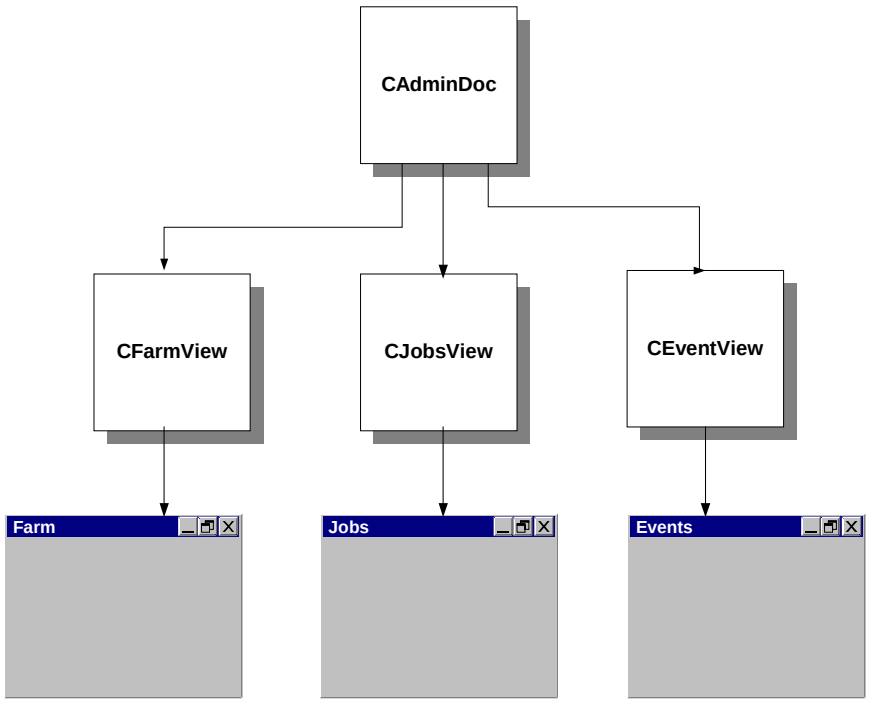


Figura 17. Relación entre el documento y las vistas

Además, las vistas son las clases encargadas de responder a los comandos de usuario. Antes de mostrar la forma en que las vistas responden a estos comandos, es necesario listar la jerarquía de comandos disponibles para el usuario del módulo el Administrador:

- **Sistema**
  - Registrar Estación
  - Eliminar Estación
  - Habilitar
  - Deshabilitar
  - Iniciar Procesamiento
  - Detener Procesamiento

    - Generar Video
- **Configuración**
- **Trabajos**
    - Adicionar
    - Eliminar
    - Suspender (poner en estado de pausa)
    - Reiniciar (poner en estado de espera)
    - Ver propiedades
- **Salir**

El diagrama de clases de la Figura 18 presenta todas las clases que componen el Administrador y la forma en que interactúan entre sí. Es necesario anotar los siguientes puntos respecto al diagrama:

- Los miembros persistentes son aquellos cuyo valor es almacenado en disco una vez el Administrador es cerrado, para conservar su valor y reutilizarlo cuando el Administrador sea ejecutado otra vez.
- Los miembros precedidos de un signo (+) son públicos, mientras que los precedidos por un signo (-) son privados.
- Los estados de las estaciones y de los trabajos están representados por los tipos de datos eStationStatus y eJobStatus.
- La clase CWrapperEvents contiene los eventos que el módulo de interfaz genera en el de administración.
- Algunos de los miembros de las clases describen características propias de una animación. El significado de estos miembros será descrito más adelante.

Este diagrama, y todos los que se presentan a continuación, han sido elaborados con base en el estándar UML.

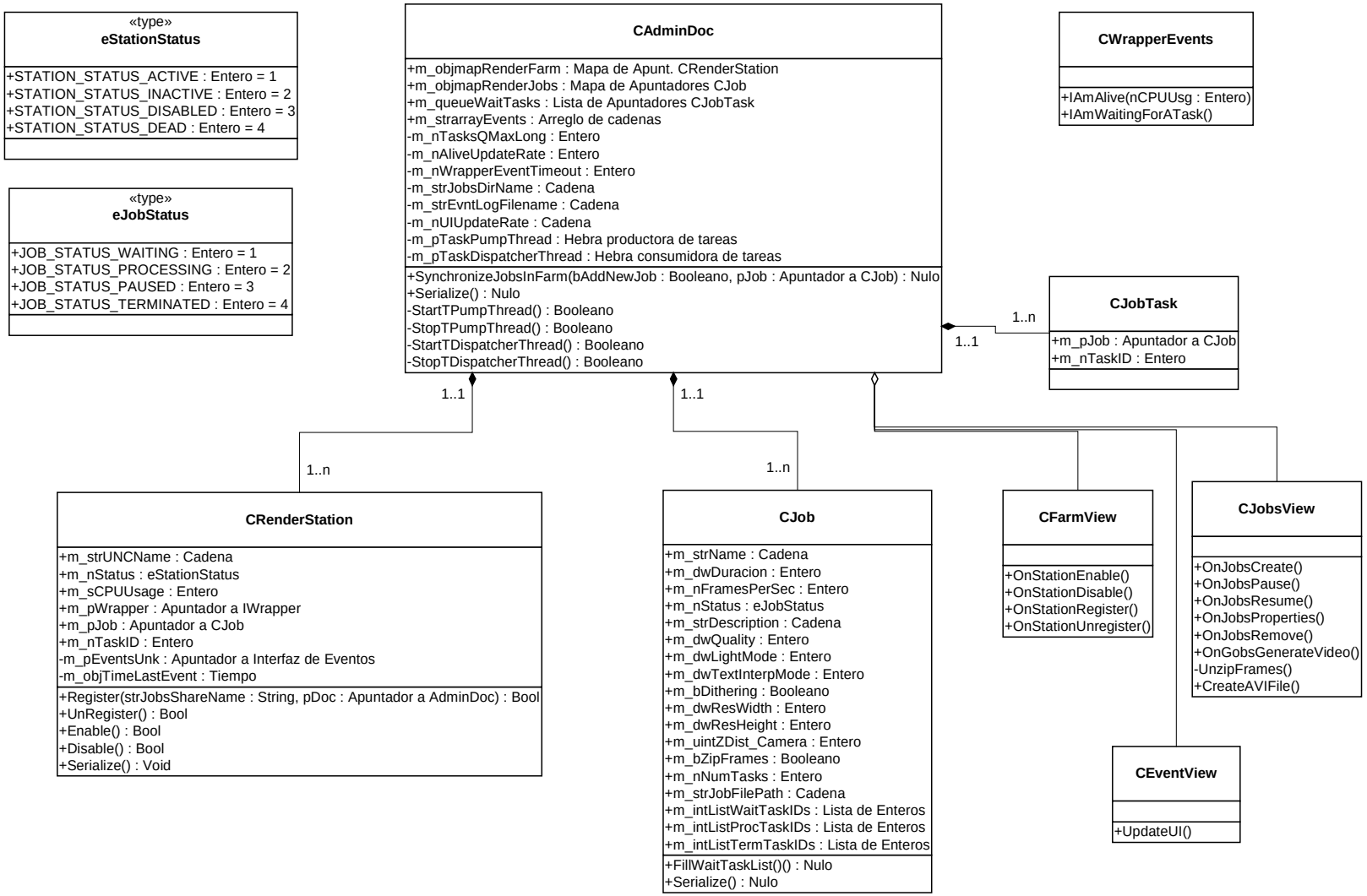


Figura 18. Diagrama de clases del módulo de administración (*Admin*).

3.6.3 Presentación detallada de clases

Las siguiente tablas contienen la descripción de cada uno de los atributos y métodos definidos para las clases presentadas en los puntos anteriores.

Miembro	Descripción
Atributos	
m_objMapRenderFarm	Lista que contiene todas las estaciones que hay en el sistema.
m_objRenderJobs	Lista que contiene todos los trabajos definidos en el sistema
m_queueWaitTasks	Cola con todos los trabajos que están en espera de ser ejecutados
m_strArrayEvents	Arreglo que contiene las cadenas que describen los eventos del sistema
m_nTasksQMaxLong	Longitud máxima de la cola de tareas
m_nAliveUpdateRate	Tiempo de espera entre dos “pulsos” enviados por el módulo de interfaz
m_nWrapperEventTimeOut	Tiempo de espera para recibir eventos del módulo de interfaz
m_strJobsDirName	Directorio de ubicación de los trabajos
m_strEventLogFileName	Nombre del archivo donde se almacenará el registro de eventos
m_nUIUpdateRate	Rata de actualización de las vistas
m_pTaskPumpThread	Apuntador a la hebra productora de tareas
m_pTaskDispatcherThread	Apuntador a la hebra consumidora (despachadora) de tareas
Métodos	
SynchronizeJobsInFarm	Envía a las estaciones de procesamiento los datos de un nuevo trabajo, cuando éste es insertado.
Serialize	Lee y escribe los miembros persistentes de la clase en disco.
StartTPumpThread	Inicia la hebra productora de tareas
StartTDispatcherThread	Inicia la hebra consumidora (despachadora) de tareas
StopTPumpThread	Detiene la hebra productora de tareas
StopTPumpThread	Detiene la hebra consumidora (despachadora) de tareas

Tabla 2. Descripción de miembros de la clase *CAdminDoc*

Miembro	Descripción
Atributos	
m_strUNCName	Nombre que identifica a la estación.
m_nStatus	Estado del sistema (de tipo eStationStatus)
m_sCPUUsage	Porcentaje de uso de la CPU
m_pWrapper	Apuntador a la instancia del *Wrapper* en la estación
m_pJob	Apuntador al trabajo que está ejecutando
m_nTaskID	Identificador de la tarea en proceso
m_pEventsUnk	Apuntador a la interfaz de eventos expuesta por el *Wrapper*
m_objTimeLastEvent	Tiempo en que se ejecutó el último evento
Métodos	
Register	Registra una estación de procesamiento
Unregister	Quita una estación de procesamiento del sistema
Enable	Activa una estación (pasa a estado inactivo)
Disable	Desactiva una estación (pasa a estado de pausa)
Serialize	Lee y escribe los miembros persistentes de la clase en disco

Tabla 3. Descripción de miembros de la clase *CRenderStation*

Miembro	Descripción
Atributos*	
m_strName	Nombre que identifica el trabajo
m_dwDuracion	Duración del trabajo, en segundos
m_nFramesPerSec	Número de trabajos (imágenes) por segundo
m_nStatus	Estado del trabajo (de tipo eJobStatus)
m_strDescripcion	Descripción del trabajo
m_dwQuality	Calidad de la animación a generar
m_dwLightMode	Modo de luces a usar en la animación
m_dwTextInterpMode	Modo de interpolación de texturas
m_bDithering	Uso de *dithering* (sí/no)
m_dwResWidth	Ancho de las imágenes que se van a generar
m_dwResHeight	Alto de las imágenes que se van a generar
m_uiZDist_Camera	Distancia desde la cámara hasta la animación
m_bZipFrames	Compresión de imágenes (sí/no)
m_nNumTasks	Número de tareas que componen el trabajo
m_strJobFilePath	Ubicación del archivo que describe el trabajo
m_intListWaitTaskIDs	Lista de tareas en espera de ser ejecutadas
m_intListProcTaskIDs	Lista de tareas que se están procesando
m_intListtermTaskIDs	Lista de tareas que han sido desactivadas
Métodos	
FillWaitTaskList	Llena la lista de tareas en espera
Serialize	Lee y escribe las características del trabajo en el disco.

Tabla 4. Descripción de miembros de la clase *CJob* (del módulo administrador).

*La descripción detallada de los parámetros puede verse en las secciones 1.4.2.5 y 2.4.2.1.

Miembro	Descripción
Atributos	
m_pJob	Apuntador a CJob (trabajo al que pertenece la tarea)|
m_uiTaskID	Identificador de la tarea

Tabla 5. Descripción de los miembros de la clase *CJobTask*

Miembro	Descripción
Métodos	
IamAlive	Evento periódico ("pulso") que indica que el módulo de interfaz está funcionando correctamente
IAmWaitingForATask	Evento que indica que el módulo de interfaz ha terminado de atender una tarea y que está esperando a que le envíen otra

Tabla 6. Descripción de los miembros de la clase *CWrapperEvents*

Miembro	Descripción
Métodos	
OnStationEnable	Atiende el comando de usuario *Sistema-Habilitar estación*
OnStationDisable	Atiende el comando de usuario *Sistema-Deshabilitar estación*
OnStationRegister	Atiende el comando de usuario *Sistema-Registrar estación*
OnStationDeregister	Atiende el comando de usuario *Sistema-Desregistrar estación*

Tabla 7. Descripción de los miembros de la clase *CFarmView*

Miembro	Descripción
Métodos	
OnJobsCreate	Atiende el comando de usuario *Trabajos-Adicionar*
OnJobsPause	Atiende el comando de usuario *Trabajos-Suspender*
OnJobsResume	Atiende el comando de usuario *Trabajos-Reiniciar*
OnJobsProperties	Atiende el comando de usuario *Trabajos-Ver propiedades*
OnJobsRemove	Atiende el comando de usuario *Trabajos-Eliminar*
OnJobsGenerateVideo	Atiende el comando de usuario *Trabajos-Generar video*.
UnzipFrames	Descomprime las imágenes que conforman una animación, si es necesario (ver atributo *m_bZipFrames* de *CJob*)
CreateAVIFile	Crea un archivo de video de formato AVI con las imágenes que conforman una animación.

Tabla 8. Descripción de los miembros de la clase *CJobsView*

Miembro	Descripción
Métodos	
UpdateUI	Actualiza periódicamente el registro de eventos y la ventana de eventos

Tabla 9. Descripción de eventos de la clase *CEventView*

3.7 DISEÑO DEL MÓDULO DE INTERFAZ

Este componente, denominado *Wrapper*, es una aplicación sin interfaz de usuario, que es instanciada y accedida desde el Administrador a través de RPC. Su trabajo es servir de puente entre el módulo administrador y el módulo remoto, manteniendo una comunicación constante con el primero y verificando que el segundo esté funcionando correctamente.

3.7.1 Descripción y diagrama de clases

Las clases que componen el módulo de interfaz son las siguientes:

- **CWrapper:** Es la clase principal del módulo de interfaz. Almacena una lista con los trabajos registrados en el sistema, y para asegurar su integridad de la lista, el administrador cuenta con un mecanismo de sincronización, que avisa a todos los módulos de interfaz en las estaciones de procesamiento sobre trabajos nuevos o eliminados. Esta clase también expone (vía RPC) varios métodos para que puedan ser llamados desde el Administrador, que sirven para manejar los trabajos, las tareas que los componen y el estado de los módulos de procesamiento.
- **CJob:** Es la clase encargada de almacenar la información correspondiente a cada una de los Trabajos presentes es el sistema. Esta clase tiene la mismas propiedades que la del mismo nombre en el Administrador. El Wrapper mantiene una lista de instancias de esta clase, una para cada trabajo.

- **CD3DRendererEvents**: Sirve de comunicación entre el módulo de procesamiento y el de interfaz, para avisarle a este último de los eventos que han ocurrido en el primero.

En la Figura 19 se muestra el diagrama de clases del módulo de interfaz.

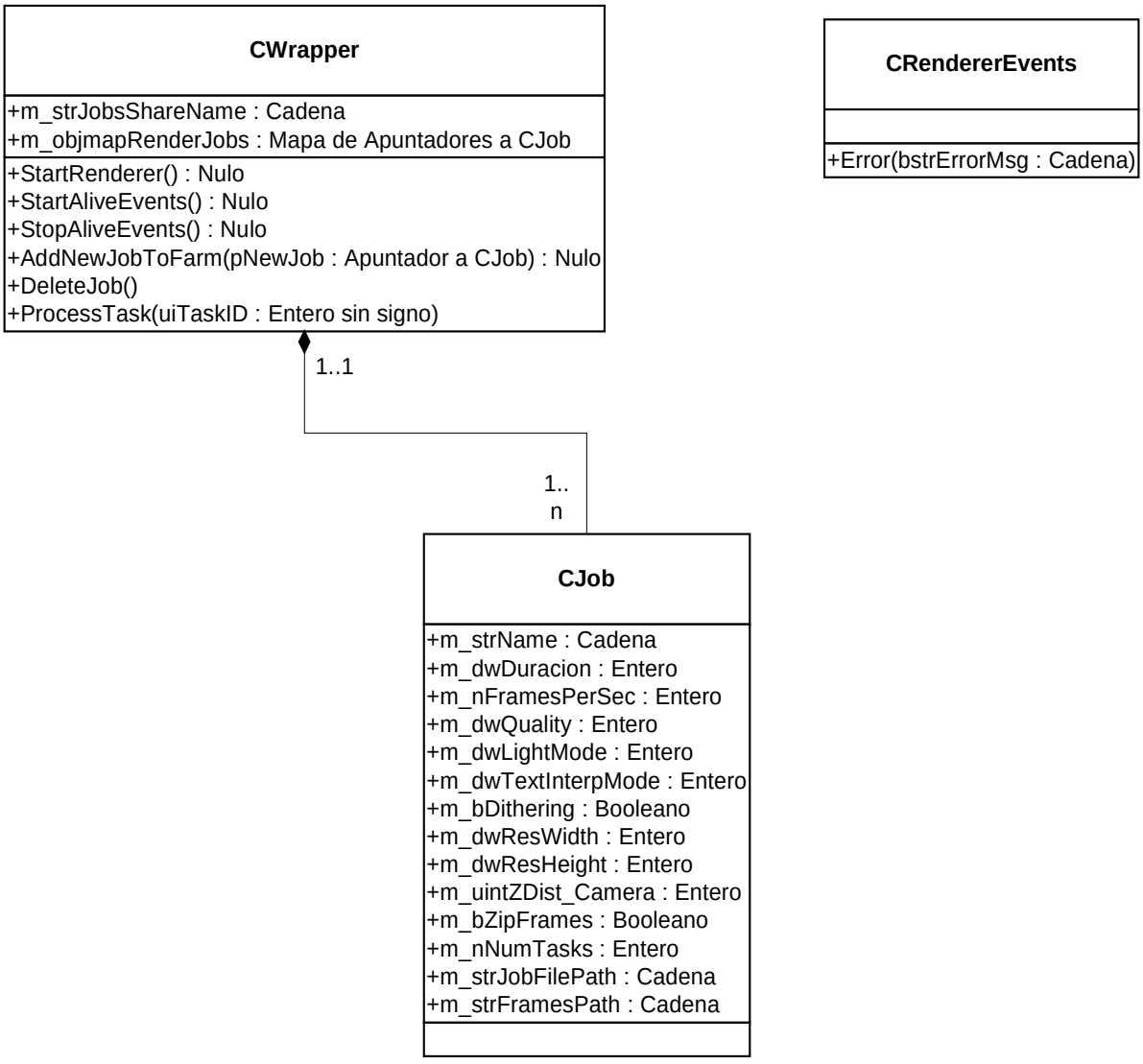


Figura 19. Diagrama de clases del módulo de interfaz (*Wrapper*).

3.7.2 Presentación detallada de clases

Las siguiente tablas contienen la descripción de cada uno de los atributos y métodos definidos para las clases presentadas en el punto anterior.

Miembro	Descripción
Atributos	
m_strJobsShareName	Directorio compartido donde están definidos los trabajos
m_objMapRenderJobs	Lista de objetos de tipo CJob
Métodos	
StartRenderer	Crea una instancia del módulo de procesamiento
StartAliveEvents	Comienza a enviar mensajes de "pulso" al administrador
StopAliveEvents	Detiene el envío de mensajes de "pulso" al administrador
AddNewJobToFarm	Añade un nuevo trabajo a la lista *m_objMapRenderjobs*
DeleteJob	Borra un trabajo de la lista *m_objMapRenderjobs*
ProcessTask	Recibe una tarea y se la entrega al módulo de procesamiento

Tabla 10. Descripción de los miembros de la clase *CWrapper*

Miembro	Descripción
Atributos	
m_strName	Nombre que identifica el trabajo
m_dwDuracion	Duración del trabajo, en segundos
m_nFramesPerSec	Número de trabajos (*frames*) por segundo
m_nStatus	Estado del trabajo (de tipo eJobStatus)
m_strDescripcion	Descripción del trabajo
m_dwQuality	Calidad de la animación a generar
m_dwLightMode	Modo de luces a usar en la animación
m_dwTextInterpMode	Modo de interpolación de texturas a usar en la animación
m_bDithering	Uso de *dithering* (sí/no)
m_dwResWidth	Ancho de las imágenes que se van a generar
m_dwResHeight	Alto de las imágenes que se van a generar
m_uiZDist_Camera	Distancia de la cámara a la animación
m_bZipFrames	Uso de compresión de imágenes (sí/no)
m_nNumTasks	Número de tareas que componen el trabajo
m_strFramesPath	Directorio donde se colocarán las imágenes generadas
m_strJobFilePath	Ubicación del archivo que describe el trabajo

Tabla 11. Descripción de miembros de la clase *CJob* (del módulo de interfaz)

Miembro	Descripción
Métodos	
Error	Envía una cadena que describe el error que ha ocurrido en el módulo de procesamiento, para que el de interfaz lo envíe, a su vez, al administrador.

Tabla 12. Descripción de miembros de la clase *CRendererEvents*

3.8 DISEÑO DEL MÓDULO DE PROCESAMIENTO

Este componente, denominado *Renderer,* es al igual que el módulo de interfaz, una aplicación sin interfaz de usuario, que es instanciada y accedida desde el módulo de interfaz a través de RPC o de línea de comandos, según el valor de una opción configurable en el módulo administrador. Está encargada de realizar las tareas que son generadas por el módulo administrador y recibidas por el de interfaz. Cuando la tarea ha sido terminada, avisa al módulo de interfaz de esto, para que este último avise al administrador de que está esperando una tarea nueva.

De acuerdo a lo expuesto en el análisis, la tarea que tendrá a cargo el módulo de procesamiento será producir las diferentes imágenes o *frames* que conforman una animación. Para ello, antes de empezar a procesar tareas debe realizar la inicialización del ambiente de procesamiento (en otras palabras, de las estructuras de datos y de los procedimientos que se usarán en la generación de las animaciones), y de las características del trabajo que se va a atender.

Una vez hecho esto, podrá recibir tareas, que irá ejecutando secuencialmente a medida que le sean asignadas al módulo de interfaz. Cuando todas las tareas se hayan terminado, el módulo de procesamiento destruirá el ambiente de trabajo y quedará en espera de que lleguen nuevas tareas.

3.8.1 Descripción y diagrama de clases

Las clases que componen el módulo de procesamiento son:

- **CD3DRenderer**: Mantiene toda la información que define una animación, y a solicitud del módulo de interfaz, genera las imágenes que la constituyen. La secuencia de tareas que realiza esta clase es:
    - Iniciar el ambiente de procesamiento con los valores que el administrador le indique (a través del módulo de interfaz).

- Recibir del módulo de interfaz los datos que definen un trabajo, tales como su ubicación, sus características y el número de tareas que lo componen.
- Generar secuencialmente las imágenes que constituyen la animación.
- Grabar las imágenes generadas en archivos, y comprimirlos si el módulo de interfaz así lo indica.
- Una vez se han terminado todos los trabajos, borrar la configuración del ambiente de procesamiento.

- **CJob:** Es la clase encargada de almacenar la información correspondiente a cada una de los trabajos presentes es el sistema. Esta clase tiene la mismas propiedades que la del mismo nombre en el módulo de interfaz. El módulo de procesamiento mantiene una sola instancia de esta clase, correspondiente al trabajo actual.

La figura 20 presenta el diagrama de clases del módulo de procesamiento.

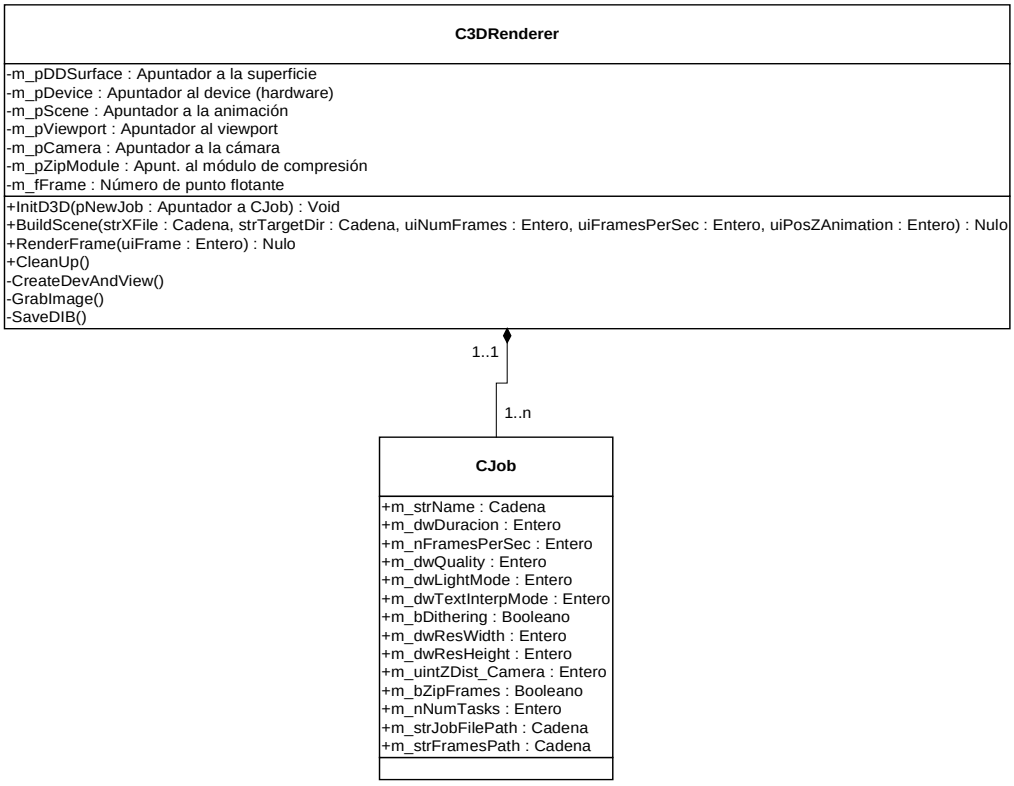


Figura 20: Diagrama de clases del módulo de procesamiento (*Renderer*)

3.8.2 Presentación detallada de clases

Las siguiente tablas contienen la descripción de cada uno de los atributos y métodos definidos para las clases presentadas en el punto anterior.

Miembro	Descripción
Atributos	
m_pDDSurface	Apuntador a la superficie donde se va a realizar el *rendering*
m_pDevice	Apuntador al *device* (representación del *hardware* de video presente en la estación de procesamiento)
m_pScene	Apuntador a la escena que define la animación a generar.
m_pViewport	Apuntador al *viewport* (descripción de la forma en que se va a generar la animación)
m_pCamera	Apuntador a la cámara (punto desde el cual se observará la escena).
m_pZipModule	Apuntador al módulo de compresión de archivos
m_fFrame	Número que indica la imagen que se está generando
Métodos	
InitD3D	Inicia el ambiente de procesamiento
BuildScene	Lee la animación a generar y construye una escena para alojar la animación
RenderFrame	Genera una imagen de la animación, creando el archivo correspondiente (comprimido, si es necesario).
Cleanup	Destruye el ambiente de procesamiento
CreateDevAndView	A partir del *hardware* de video disponible, construye el *device* y el *viewport*.
GrabImage	Lee la imagen generada de la memoria y la coloca en un bitmap definido también en memoria.
SaveDIB	Graba un bitmap en un archivo.

Tabla 13. Descripción de los miembros de la clase *CD3DRenderer*

Miembro	Descripción
Atributos	
m_strName	Nombre que identifica el trabajo
m_dwDuracion	Duración del trabajo, en segundos
m_nFramesPerSec	Número de trabajos (imágenes) por segundo
m_nStatus	Estado del trabajo (de tipo eJobStatus)
m_strDescripcion	Descripción del trabajo
m_dwQuality	Calidad de la animación a generar
m_dwLightMode	Modo de luces a usar en la animación
m_dwTextInterpMode	Modo de interpolación de texturas a usar en la animación
m_bDithering	Uso de *dithering* (sí/no)
m_dwResWidth	Ancho de las imágenes que se van a generar
m_dwResHeight	Alto de las imágenes que se van a generar
m_uiZDist_Camera	Distancia de la cámara a la animación
m_bZipFrames	Compresión de imágenes
m_nNumTasks	Número de tareas que componen el trabajo
m_strFramesPath	Directorio donde se colocarán las imágenes generadas
m_strJobFilePath	Ubicación del archivo que describe el trabajo

Tabla 14. Descripción de los miembros de la clase *CJob* (del módulo de interfaz)

4. IMPLEMENTACIÓN

En este capítulo se pretende explicar la forma en que ha sido construido el conjunto de componentes de *software* descritos anteriormente en el diseño, que resuelven el problema ya planteado en el análisis. Se hará especial énfasis en las plataformas de computación y herramientas de desarrollo utilizadas para este propósito, justificando la elección de cada una de ellas.

4.1 PLATAFORMA DE COMPUTACIÓN

4.1.1 Sistema operativo

El ambiente que se ha seleccionado para la construcción del *software* en cuestión es Windows de 32 bits, lo que comprende los sistemas operativos Windows 95, Windows 98, Windows NT y Windows 2000. Entre las razones que motivaron esta elección se encuentran:

- Es la plataforma de computación más popular y extendida, lo que implica amplísimas oportunidades de soporte, experimentación y utilización de las aplicaciones. Además, posee una base de conocimiento muy amplia, compuesta por bibliografía, herramientas de desarrollo, grupos de interés en Internet, etc.
- El soporte gráfico que provee es poderoso y versátil, pues permite construir aplicaciones gráficas de muy diversas clases y alcances. Se pueden hacer gráficas usando funciones del sistema operativo (API gráfico de Windows),

arquitecturas de computación gráfica (OpenGL y DirectX), o funciones propias que accedan directamente al *hardware* de video.

- El *software* aplicativo y de servicios es desarrollado en su gran mayoría para esta plataforma, por lo que es posible encontrar una gran cantidad y variedad de soluciones que pueden ser usadas en el desarrollo de aplicaciones.

Debido a los requerimientos de seguridad, rendimiento y manejo del sistema asignados al módulo de administración, se ha decidido que éste funcione únicamente bajo Windows NT Server versión 4.0, con la actualización conocida como Service Pack 4. Específicamente, es necesario que el equipo donde está instalado sea un controlador primario de dominio, es decir, un equipo que tenga la capacidad de administrar el acceso y los permisos de todos los equipos que componen el *farm*.

Los componentes restantes que hacen parte del sistema (módulos de interfaz y de procesamiento) han sido diseñados para y probados en cualquiera de los sistemas operativos Win32, incluyendo la versión beta 3 de Windows 2000 Professional, que es equivalente a la versión 5.0 de Windows NT Workstation. En el caso de los equipos que tengan instalado Windows 95, es necesario instalar adicionalmente la extensión DCOM. Esto será explicado en detalle a continuación.

4.1.2 Arquitectura de software para sistemas distribuidos

La selección de la plataforma Windows conduce también a la elección de la arquitectura para sistemas distribuidos utilizada para la construcción de la aplicación. DCOM ha sido la arquitectura seleccionada por las siguientes razones:

- Es una arquitectura que, como ya se explicó en el marco teórico, está incluido en y estrechamente relacionado con los sistemas operativos Windows.

- Ofrece las características de invocación remota de componentes y aplicaciones, manejo de eventos, seguridad y orientación a objetos necesarias para el desarrollo de *Antares*.
- De manera similar a lo que ocurre con la plataforma Windows, existe una gran cantidad y variedad de aplicaciones que soportan y están construidas sobre COM.

La primera versión de Windows 95 lanzada al mercado incluía soporte a COM, pero no a DCOM. Versiones posteriores del producto (específicamente, la versión W95b) proveyeron este soporte. Entonces, si se va a incluir en el *farm* un equipo que funcione bajo Windows 95, debe verificarse la versión del sistema operativo, y si es anterior a la versión W95b se debe instalar el paquete de soporte a DCOM, el cual consiste en un conjunto de librerías y definiciones que se encargan de incluir los servicios de DCOM en el sistema. Este paquete es gratuito y puede ser obtenido en el sitio Web de Microsoft.

Para el caso especial en que la estación de procesamiento funcione sobre un sistema operativo Windows 95 o 98, un componente adicional es necesario debido a las limitaciones de DCOM sobre este sistema operativo, que no permiten la creación remota de componentes. Su nombre es *WrapperMon* y su función principal es la de mantener siempre en ejecución una instancia del módulo de interfaz (*Wrapper*), para que esta pueda ser accedida por el administrador. Cabe recordar que estas limitantes de DCOM no están presentes en sistemas operativos como Windows NT 4 o Windows 2000, y por lo tanto *WrapperMon* no es indispensable para el funcionamiento correcto de *Antares*.

4.1.3 Arquitectura de computación gráfica

De las dos alternativas expuestas en el marco teórico para la selección de la arquitectura de computación gráfica, se seleccionó DirectX debido a los siguientes factores:

- El soporte de esta arquitectura por parte de los fabricantes de *hardware* es muy amplio. Existen toda clase de tarjetas aceleradoras de gráficos que realizan funciones de DirectX (específicamente, DirectDraw), y es posible obtener *software* de control con extensiones para DirectX, con el fin de manejar estas tarjetas y las que ya existen en el mercado.
- DirectX tiene separadas las funciones de manejo de *hardware*, acceso a la memoria de video, manejo de primitivas gráficas y construcción de escenas y animaciones. Esto facilita el desarrollo de aplicaciones y permite corregir errores y afrontar dificultades de manera directa y puntual.
- Todos los componentes de DirectX necesarios para el desarrollo del sistema utilizan COM como lenguaje de comunicación entre DirectX y los programas que lo utilizan. De esta forma, se preserva la unidad en la arquitectura del sistema, y se obtienen las ventajas que COM ofrece para el desarrollo de aplicaciones.
- DirectX ofrece la posibilidad de almacenar animaciones en archivos, lo que es de suma importancia para el sistema en desarrollo. Sería poco práctico tener que construir una animación desde cero todas las veces que se fuera a generar.

La versión de DirectX utilizada en *Antares* es la 3.0. A pesar de que la versión actual es la 7.0, los sistemas operativos Windows NT versión 4.0 no soportan. En cambio, Windows 95, Windows 98 y Windows 2000 sí la soportan, pero, con el fin de conseguir la mayor interoperabilidad posible para la aplicación, se ha utilizado la versión 3.0 para el desarrollo. Esto no representa problema para los equipos que soportan la versión 6.0, puesto que las mejoras progresivas que se le han hecho y se le harán a DirectX son implementadas en nuevas interfaces (ver sección 1.3.2.1) de modo que los sistemas antiguos puedan trabajar con las interfaces presentes en versiones anteriores.

4.2 HERRAMIENTAS DE DESARROLLO

4.2.1 Lenguaje de programación

Para construir el código fuente del sistema, se decidió utilizar C++ como lenguaje de programación. La elección es lógica, debido a las características ampliamente conocidas del lenguaje, como son su enfoque de alto nivel, la orientación a objetos que provee, la uniformidad del código, la velocidad de ejecución de los programas construidos con él, la facilidad que ofrece para la construcción de extensiones propietarias, etc.

Además, aunque es posible utilizar COM con varios lenguajes de programación, como C, Java o Basic, solamente con C++ se tiene acceso a todo el poder y funcionalidad que esta arquitectura ofrece.

4.2.2 *Software* de desarrollo

La elección de las herramientas de desarrollo no termina con la selección del lenguaje de programación. Existen muchos programas que se pueden utilizar para la construcción de programas en C++, desde aplicaciones gratuitas que trabajan con interfaz de línea de comandos, hasta programas muy completos que ofrecen interfaz gráfica, editores de texto poderosos, ayudas para la depuración y la verificación del funcionamiento de los programas, soporte para toda clase de tecnologías y aplicaciones, etc.

Para la construcción de *Antares* se ha seleccionado Visual C++ 6.0, de Microsoft. Entre las características que esta herramienta ofrece para el desarrollo de aplicaciones se encuentran:

- El compilador y enlazador de C++ incluidos con esta herramienta producen programas estables, confiables y rápidos. Además, incluyen soporte para el manejo y seguimiento de errores, la depuración y la puesta a punto de los programas.

- Visual C++ contiene extensiones que facilitan el desarrollo de aplicaciones que usen COM y DCOM. Entre ellas se encuentran extensiones al lenguaje, generación de estructuras y clases que encapsulan las interfaces de un componente, manejo de errores específicos de COM, etc.
- La herramienta es estable, pues controla la ejecución de los programas mientras se están depurando, lo que evita bloqueos del sistema cuando hay errores de memoria o de apuntadores. Además, las características de depuración que incluye permiten descubrir este tipo de errores antes de que ocurran.
- Es posible corregir el programa en tiempo de depuración, sin detener su ejecución. Esto reduce el tiempo de depuración, pues ya no será necesario parar el programa en cuestión, corregirlo, compilarlo, construirlo, ejecutarlo y llevarlo al estado en que se encontraba cada vez que se encuentran errores.
- El soporte que Visual C++ ofrece es muy amplio. Consiste, primordialmente, en la biblioteca de documentos, manuales y artículos MSDN (*Microsoft Developer Network,* red de desarrollo de Microsoft), y el amplio soporte que se ofrece en el sitio Web de Microsoft, sitios Web independientes, libros y revistas, congresos, etc.

4.3 OTRAS HERRAMIENTAS

Como se explicó en el marco teórico, una de las grandes ventajas de COM es que permite utilizar en una aplicación cualquiera los servicios y funciones que otra tiene, sin tener que reescribir código. Durante el desarrollo de *Antares* se utilizó una aplicación de esta forma.

XceedZip, componente COM desarrollado por la compañía XceedSoft, es una librería dinámica (DLL) que ofrece funciones de compresión y descompresión de archivos de formato .zip. Este componente ofrece, entre otras, las siguientes funciones:

- Compresión y descompresión de archivos a demanda y de manera concurrente, sin que la aplicación que está usando XceedZip interrumpa su ejecución.
- Manejo de los archivos que hacen parte de un archivo .zip, incluyendo actualización, borrado, corrección de errores, inclusión de archivos nuevos y aviso cuando el archivo a descomprimir ya existe en el directorio de destino.
- Compresión y descompresión de directorios enteros, conservando los datos y ubicación de cada archivo en el directorio.
- Definición de eventos que avisan cuándo se ha terminado una operación de compresión o descompresión.
- Compresión y descompresión de datos que se encuentran en memoria.

El componente fue utilizado para comprimir las imágenes que componen una animación. Estos son archivos muy grandes, cuyo tamaño fácilmente puede sobrepasar 1 MB. Entonces, para minimizar el tráfico de red que implica colocar las imágenes en el directorio de destino, los archivos son comprimidos por el módulo de procesamiento antes de ser copiados, y descomprimidos por el administrador una vez llegan a su destino.

5. PRUEBAS Y EVALUACIÓN DEL DESEMPEÑO

Se llevaron a cabo una serie de pruebas de tiempos de procesamiento, involucrando varias máquinas interconectadas en red y un trabajo (animación) de ejemplo.

Los equipos de prueba tuvieron las siguientes características:

Nombre del Equipo	Procesador	Sistema Operativo
Wakko	Pentium II 350 Mhz	NT 4 Workstation
Bugs	Pentium II Celeron 333 Mhz	NT 4 Workstation
Elmer	Pentium II Celeron 333 Mhz	NT 4 Workstation
Taz	Pentium MMX 266 Mhz	NT 4 Workstation
Pinky	Pentium 166 Mhz	NT 4 Workstation
Marvin	Pentium II Celeron 333 Mhz	NT 4 Workstation
Brain	Pentium II Celeron 333 Mhz	NT 4 Server
Lucas	Pentium II 300 Mhz	NT 4 Workstation
Virginia2K	Pentium MMX 200 Mhz	Windows 2000 Professional/ Windows 95

Tabla 15. Características de los equipos que conforman el sistema (*farm*)

Como se aprecia en la tabla anterior, sólo se tuvo en cuenta el tipo y velocidad del procesador, puesto que es la característica que mas influye sobre los resultados de nuestras pruebas, con respecto al desempeño de las estaciones de procesamiento. Otras características que también influyen, son el tipo de tarjeta de red (ISA o PCI o de otro tipo), la velocidad de transmisión la red (10 o 100 Mbits/seg., etc.), tipo de interfaz de comunicación del disco con el procesador (IDE, EIDE, SCSI, etc. – ver Glosario). Para nuestro caso todos

estos equipos contaron con tarjeta de Red ISA de 10 Mbits/seg. e Interfaz de Disco EIDE.

Con respecto al *software,* las estaciones de procesamiento cuentan con sistema operativo Windows NT 4.0 Workstation, aunque por sus requerimientos el Administrador se ejecutó sobre Windows NT 4.0 Server (Estación "*Brain*").

Para tener mas idea sobre la capacidad de procesamiento individual de cada una de las estaciones, se realizó una prueba consistente en el procesamiento de la tarea de prueba por cada uno de los equipos en el *farm*, de manera exclusiva. Se obtuvieron los siguientes resultados:

Nombre del Equipo	Tiempo de Procesamiento (seg.)
Wakko	472
Bugs	586
Elmer	595
Taz	671
Pinky	662
Marvin	580
Brain	572
Lucas	501
Virginia2K	696

Tabla 16. Tiempo de procesamiento de la tarea de ejemplo por cada uno de los equipos

Es necesario anotar que todos los equipos incluidos en la tabla anterior tenían instalado el sistema operativo Windows NT Workstation 4.0, a excepción de Virginia2K, que tenía Windows 2000 Professional (Beta 3).

La animación de prueba tuvo las siguientes características:

- Nombre del Trabajo: Tiger

- Resolución en Pixeles (Alto x Ancho): 1600x1200
- Numero de imágenes (*frames*): 100
- Compresión de imágenes: Sí
- Calidad del *Rendering*: *Gouraud*
- Modo de Luces: RGB
- Modo de Interpolación de Texturas: Lineal
- *Dithering*: Sí
- Distancia de la Cámara (en el eje Z positivo): 15

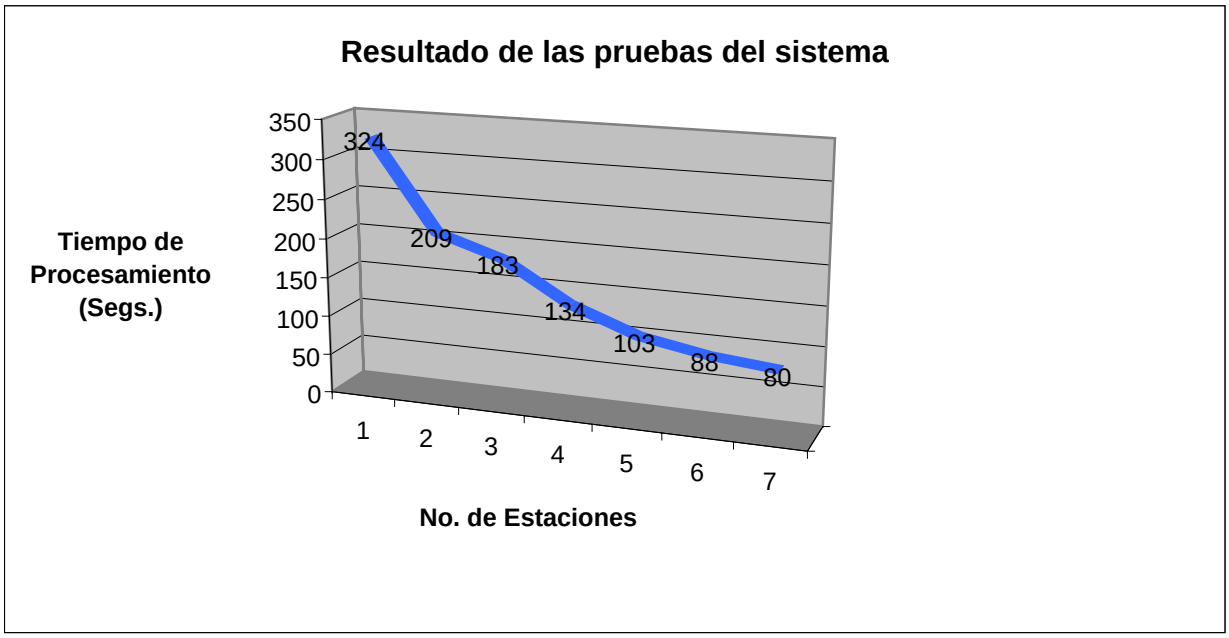
La Tabla 17 resume los tiempos de resultado que se obtuvieron luego que procesar el anterior trabajo añadiendo cada una de las estaciones al *farm*. La marca en cada equipo indica si el equipo estaba incluido en el *farm* en el momento de la prueba.

No. De Estaciones en el farm	*Wakko*	*Bugs*	*Elmer*	*Taz*	*Pinky*	*Marvin*	*Lucas*	*Tiempo Procesamiento (mm:ss)*
1	☑							5:24
2	☑	☑						3:29
3	☑	☑	☑					3:03
4	☑	☑		☑		☑		2:14
5	☑	☑		☑	☑	☑		1:43
6	☑	☑	☑		☑	☑	☑	1:28
7	☑	☑	☑	☑	☑	☑	☑	1:20

Tabla 17. Tiempo de procesamiento de la tarea de ejemplo por los equipos en el sistema, en diferentes configuraciones.

En la tabla anterior observamos cómo no se lograron algunas veces tiempos de procesamiento parecidos con tener el mismo número de máquinas en el sistema, esto se debió a la heterogeneidad en el poder computacional de cada una de las estaciones de prueba.

En la siguiente gráfica podemos apreciar mucho mejor la disminución de tiempos de procesamiento conforme se añaden máquinas al sistema:



Se realizaron otras pruebas adicionales en un ambiente diferente (Equipos/Red), con las siguientes características:

Nombre del Equipo	Procesador	Sistema Operativo
Estacion1	2 Pentium Pro 200 Mhz	NT 4 Workstation
Estacion2	Pentium II 400 Mhz	NT 4 Workstation
Estacion3	2 Pentium II 400 Mhz	NT 4 Workstation
Estacion4	2 Pentium II 400 Mhz	NT 4 Workstation
Bigmac	Pentium MMX 166 Mhz	NT 4 Server

Con respecto al *software,* las estaciones de procesamiento cuentan con sistema operativo Windows NT 4.0 Workstation, aunque por sus requerimientos el Administrador se ejecutó sobre Windows NT 4.0 Server (Estación "*Bigmac*").

Esta segunda prueba se hizo con el mismo trabajo de la anterior (*Tiger*) y se obtuvieron los siguientes resultados:

Nombre del Equipo	Tiempo de Procesamiento (seg.)
Estacion1	642
Estacion2	453
Estacion3	449
Estacion4	438

Cabe notar que aunque 3 de las estaciones son sistemas multiprocesador, no se notaron ganancias considerables en los tiempos de procesamiento, esto debido a que el sistema no hace uso de ello.

6. CONCLUSIONES

1. La conclusión más importante que se obtuvo como resultado de la ejecución de este proyecto fue que se logró una reducción considerable en los tiempos de procesamiento de trabajos a través del empleo de una arquitectura distribuida. Este era el propósito principal de este proyecto y se cumplió a cabalidad, por eso su importancia.

2. Como se previó en el análisis y se observó en las pruebas de desempeño, las reducciones en el tiempo de procesamiento de los trabajos fueron más altas cuando se utilizó un número reducido de equipos. Esto ocurrió porque los canales de comunicación no se encontraban saturados con información de desempeño y con la transmisión de los resultados de los trabajos. A medida que se añaden más equipos al sistema, los costos de comunicación van absorbiendo los beneficios de tiempo obtenidos por la distribución; a partir de una cantidad de equipos determinada (ocho equipos), los beneficios se tornan despreciables.

3. El modelo de distribución de procesamiento, basado en un esquema de colas de tareas, probó ser sencillo en su concepto y efectivo en su desempeño. Gracias a este modelo el balanceo de carga ocurre de forma natural y no fue necesario recurrir a algoritmos más sofisticados de distribución de procesamiento como otros considerados en el marco teórico.

4. Una consecuencia de la conclusión anterior es que, con la arquitectura de distribución de procesamiento empleada en este proyecto, se plantea la posibilidad de construir un sistema computacional tolerante a

fallas, característica obligatoria en las aplicaciones actuales tales como sitios de Internet o sistemas de misión crítica. Muchas de las bases teóricas empleadas para el desarrollo de *Antares* son indispensables para la construcción de tales sistemas.

5. Gracias a su característica de tolerancia a fallas, el sistema tiene una propiedad muy valiosa: si alguna estación sale de operación por alguna razón, el sistema puede seguir funcionando sin mayor contratiempo. Además, es posible sustituir dinámicamente la estación que ha fallado por una nueva, y el sistema seguirá operando sin necesidad de reconfigurar el modelo de distribución de tareas, o de asignar explícitamente las tareas que la máquina nueva debe procesar.

6. La plataforma de computación seleccionada —que comprende el sistema operativo, los APIs y herramientas de desarrollo, el modelo de objetos distribuidos y la arquitectura de computación gráfica— suministraron el soporte necesario para la construcción y el funcionamiento de *Antares*. Esto demuestra la validez de la plataforma como soporte al desarrollo de aplicaciones robustas y de alto nivel.

7. Una de las herramientas más útiles a la hora de desarrollar un sistema con las características de Antares es el uso de múltiples hilos de procesamiento (*multithreading*). Fue utilizado en el módulo administrador para manejar la asignación de tareas, sin que se alterara la fluidez y la funcionalidad de la interfaz de usuario. Además, en el módulo de interfaz permitió implementar la asincronía necesaria para la comunicación con el administrador de manera independiente a la ejecución del módulo de procesamiento.

8. Con respecto al desempeño del sistema, también se notó que la configuración de las estaciones de procesamiento es determinante en el rendimiento del sistema. A pesar de que la capacidad de procesamiento

es la variable que influye en mayor medida en el comportamiento de las estaciones que hacen parte del sistema, otros factores como la velocidad del disco duro y la configuración del bus de datos afectaron el desempeño de las estaciones. Por ejemplo, un computador de escritorio y uno portátil con el mismo procesador pueden tener tiempos de respuesta muy diferentes, lo que ocurre debido a que el equipo de escritorio está optimizado para el desempeño, mientras que el portátil está orientado a ahorrar energía y proveer una funcionalidad básica.

9. También se observó que la calidad de la tarjeta de video —que es proporcional a su precio— y la cantidad de memoria de video, indispensables para el funcionamiento adecuado de juegos y otras aplicaciones gráficas, no fueron determinantes en este caso. Esto se debe, en parte, debido a la arquitectura del módulo de procesamiento, que empleaba la memoria principal para generar las imágenes, ignorando la memoria de video.

10. Otro factor determinante en el rendimiento del sistema es la calidad de *rendering.* Si se emplea inicialmente una calidad baja, en la cual sólo se dibujen los contornos de los objetos, se obtiene un muy buen tiempo de respuesta, apropiado para verificar el comportamiento de la animación. De esta forma, se puede generar posteriormente la animación con la mayor calidad posible y con la seguridad de que el resultado va a ser el esperado.

11. Con respecto al rendimiento en computación gráfica, se encontró que los equipos que tenían instalado el sistema operativo Windows 95/98 tenían el mejor desempeño para construir las imágenes. Esto ocurre porque estos sistemas operativos están optimizados para la computación gráfica y la multimedia, lo que se nota en la versión de DirectX disponible para estos sistemas (6.0) en comparación con la versión disponible para Windows NT (3.0).

12. La utilización de un módulo que actúa como intermediario entre los módulos administrador y de procesamiento no afecta el desempeño global del sistema, por el contrario, facilitan su desarrollo y permiten extender su utilización en otras aplicaciones. Los módulos de interfaz y de procesamiento fueron separados para poder procesar cualquier clase de trabajo y a pesar de que se encuentran en la misma máquina, la comunicación entre los procesos consume tiempo, el cual se halla en el orden de los milisegundos. Considerando que el tiempo de ejecución de las tareas está en el orden de los segundos o minutos, el tiempo empleado en la comunicación es despreciable.

13. Considerado la estabilidad y confiabilidad como parámetros relevantes, se obtienen mejores resultados utilizando Windows NT. Las tecnologías utilizadas por *Antares* y por las herramientas de desarrollo empleadas en su construcción se comportaron de una manera mucho más estable y robusta sobre este sistema operativo.

14. Se tuvo la posibilidad de usar la versión preliminar de Windows 2000 Professional (equivalente a Windows NT Workstation) y se encontró que este sistema operativo combina el soporte para el procesamiento gráfico de Windows 95/98 con la estabilidad y eficiencia de Windows NT. Las pruebas de *rendering* realizadas con este sistema fueron las que dieron mejores resultados. La configuración ideal del sistema implicaría el uso de Windows 2000 en todas las estaciones.

15. El módulo de interfaz fue diseñado para que pudiera comunicarse con el de procesamiento ya sea usando DCOM o línea de comandos. A través de la línea de comandos el módulo de interfaz puede utilizar como módulo de procesamiento una gran cantidad de aplicaciones disponibles en el mercado. Sin embargo, este beneficio se veía opacado por la velocidad de procesamiento de tareas, que es considerablemente

más lenta que la obtenida utilizando DCOM. Otro punto en contra es la disminución del control que el módulo de interfaz tiene sobre el de procesamiento mientras éste procesa la tarea, pues sólo puede iniciarlo y esperar a que termine su ejecución. El uso de DCOM tiene la limitación de que el módulo de procesamiento tiene que implementarse de acuerdo al protocolo planteado y utilizado por el módulo de interfaz.

16. Por ser el desempeño de los canales de intercomunicación del sistema su principal cuello de botella, se pueden lograr mejoras considerables en el rendimiento utilizando configuraciones de red de alto desempeño: 100 Mbits/seg, GigaEthernet, etc.

17. Una solución al problema de congestión de los canales de comunicación está en la compresión de las imágenes producto de la ejecución. Sin importar el costo extra de procesamiento que se incurre utilizando este método, la gran reducción del tamaño alivia en gran medida el transporte de datos a través de la red.

18. A pesar de que *Antares* internamente es un sistema complejo, se logró implementar una interfaz de usuario sencilla, que provee una vista en tiempo real del estado del sistema. También provee un conjunto sencillo de comandos para manejar las estaciones de procesamiento y los trabajos. Esto permite que el usuario vea el sistema como si fuera un gran computador, siendo en realidad un conjunto de equipos adecuadamente conectados entre sí.

7. RECOMENDACIONES

A pesar de todo los esfuerzos que se hicieron durante todas las fases que comprendieron la ejecución de este proyecto, hay múltiples detalles que sería muy interesante mejorar o adicionar. Cabe anotar los siguientes:

1. Reducir la cantidad de información que se intercambia entre los componentes del sistema y así disminuir la congestión de las líneas de comunicación.

2. Extender las capacidades de tolerancia a fallos del sistema. Esto incluye un manejo mas complejo de los eventos de falla en las comunicaciones o en el *hardware* / *software* de las estaciones.

3. Implementar una característica de medición inicial del rendimiento del sistema a través de un trabajo de referencia. Los tiempos de resultado de este trabajo de prueba ofrecerían mucha información sobre los tiempos de procesamiento de los trabajos “reales” e incluso del rendimiento de cada una de las estaciones, para poder generar un modelo de asignación de tareas basado en prioridades y en el rendimiento de las estaciones.

4. Realizar pruebas sobre los límites de rendimiento de la arquitectura de Antares. Y buscar un método para manejar estos limites y restricciones. En principio, los límites podrían ser el tamaño del sistema, la cantidad y granularidad de las tareas, el consumo de recursos de almacenamiento y

de comunicación, la disponibilidad de programas que trabajen como módulos de procesamiento, etc.

5. Con respecto a la disponibilidad de módulos de procesamiento, existe la posibilidad de ampliar la portabilidad del sistema, extendiendo la vía de comunicación entre los módulos de interfaz y de procesamiento (que actualmente se limita a DCOM y línea de comandos) a Automation. Esta tecnología permite acceder la funcionalidad de una aplicación sin modificar el código del cliente que la usa, tal como la línea de comandos, con las ventajas de velocidad y seguridad que COM ofrece. Una gran cantidad de aplicaciones de productividad, como las que componen Microsoft Office, emplean Automation para exponer su funcionalidad.

6. Implementar "puentes" que le permitan a los componentes basados en COM interactuar con similares desarrollados en un lenguaje independiente de plataforma como lo es Java, y de esta manera incrementar la portabilidad del sistema. También se puede implementar el sistema empleando otra arquitectura de objetos distribuidos, tal como CORBA o Enterprise Java Beans, con el fin de comparar el desempeño de tales arquitecturas.

7. Debido a las características de orientación a objetos y componentes con los que *Antares* fue diseñado y construido, es factible que el sistema evolucione a partir de mejoras en sus características o la implementación de algunas de las recomendaciones anteriores, como resultado de lo realizado en otros trabajos de grado. Y también es factible instalar un sistema basado en *Antares* en la Universidad Nacional, para aprovechar las ventajas del procesamiento distribuido como solución de bajo costo a problemas de gran complejidad computacional encontrados en las múltiples disciplinas del saber.

BIBLIOGRAFÍA

ARMSTRONG, Tom. Active Template Library: A Developer´s Guide. New York: M&T Books, 1998. 355p.

ANDERSON, Jerry. ActiveX Programming with Visual C++ 5.0. Chicago, QUE Books, 1998. 250 p.

BOX, Don. Essential COM. Los Angeles, Addison-Wesley Longman Inc., 1998. 440 p.

BOX, Don, BROWN, Keith, et. al. Effective COM: 50 Ways to Improve your COM and MTS-based Applications. New York, A-W Professional, 1998. 208 p.

BLOOMER, John. Distributed Computing and the OSF/DCE. En: Dr. Dobb´s Journal. Vol. –1, No. 227 (febr. 1995); p 10-15.

BROCKSCHMIDT, Kraig. Inside OLE: The Fast Track to Building Powerful Object-Oriented Applications. Redmond, Microsoft Press, 1997. 376 p.

CHAPPELL, David. Understanding ActiveX and OLE: A Guide for Developers and Managers. Redmond, Microsoft Press, 1996. 328 p.

COULOURIS, George, Jean Dollimore, Tim Kindberg. Distributed Systems, Concepts and Design. Los Angeles, Addison-Wesley Publishing Company, 1993. 600 p.

CROCKETT, Frank. MFC Developer's Guide. Redmond, Microsoft Press, 1997. 532 p.

EDDON, Guy. RPC for NT. New York, R&D Publications, 1994. 420 p.

EDDON, Guy, Henry Eddon. Inside Distributed COM. Redmond, Microsoft Press, 1998. 552 p.

FLANIGAN, Pat y JAWEN, Karim. NCSA Symera. En: Dr. Dobb´s Journal No. 291 (Nov 1998); p 20-24.

GRIMES, Richard. Professional DCOM Programming. Birmingham, WROX Press, 1997. 565 p.

HOLZNER, Steven. Advanced Visual C++ 5. Philadelphia, M&T Books, 1997. 654 p.

JALOTE Pankaj, Fault Tolerance in Distributed Systems. Boston, Prentice Hall, 1994. 432 p.

KRAMER, Jeff y MORRIS, Sloman. Distributed Systems and Computer Networks. Boston, Prentice Hall, 1994. 651 p.

KRUGLINSKI, David. Programming Visual C++. Redmond, Microsoft Press, 1998. 450p.

MULLENDER, Sape. Distributed Systems. Houston, ACM Press, 1995. 570 p.

MCCONNELL, Steve. Code Complete, A Practical Handbook of Software Construction. Microsoft Press, 1993. 870 p.

MILENKOVIC, Milan. Sistemas Operativos, Conceptos y Diseño. 2da. Edicion. McGraw-Hill, 1994. 540 p.

BLOOMER, John. Distributed Computing and the OSF/DCE. En: Dr. Dobb´s Journal. Vol. –1, No. 227 (febr. 1995); p 10-15.

PANT, Lalit. Thread Communication in Parallel Algorithms. En: Dr. Dobb´s Journal No. 298 (abr. 1999); p 30-36.

PINNOCK, Jonathan. Professional DCOM Application Development. Birmingham, WROX Press, 1998. 479 p.

ROGERSON, Dale. Inside COM: Microsoft's Component Object Model. Redmond, Microsoft Press, 1997. 376 p.

TANNENBAUM, Andrew S. Distributed Operating Systems. Boston, Prentice Hall, 1995. 620 p.

TANNENBAUM, Andrew S. Modern Operating Systems. Boston, Prentice Hall, 1995. 590 p.

TANNENBAUM, Todd y LITZKOW, Michael. The Condor Distributed Processing System. En: Dr. Dobb´s Journal No. 227 (feb 1995); p 42-46.

THOMPSON, Nigel. 3D Graphics Programming for Windows 95. Redmond, Microsoft Press, 1996. 320 p.

VARIOS autores. Automation Programmer's Reference. Microsoft Press, 1997. 568 p.

VARIOS autores. DCE 1.1: Especificacion RPC del Open Group CAE. En: The Open Group. http://www.opengroup.org/pubs/catalog/c706.htm (ago. 1997).

VARIOS autores. Especificacion del Modelo de Objetos Componentes (*Component Object Model – COM*). En: Microsoft Corporation http://www.microsoft.com/com/resources/specs.asp (ene. 1999).

VARIOS autores. Especificacion del Protocolo COM Distribuido (Distributed Component Object Model - DCOM) version 1.0. En: Microsoft Corporation. http://www.microsoft.com/com/resources/specs.asp (ene. 1998).

VARIOS autores. MSDN: Microsoft Developer Network. En: MSDN Online, http://msdn.microsoft.com (jul. 1999).

WU, Jie. Distributed Systems Design. New York, CRC Press, 1999. 340 p.

YOURDON, Edward. Object-Oriented Systems Design: An Integrated Approach. Yourdon Press – Prentice Hall. 1994. 672 p.

ZARATIAN, Beck. Visual C++ Owner's Manual, Version 5.0. Microsoft Press, 1997. 870 p.

ANEXOS

ANEXO A: REQUERIMIENTOS DEL SISTEMA

ADMINISTRADOR

La estación donde se va a instalar este módulo debe cumplir con las siguientes características:

- PC con un procesador Intel Pentium o posterior. Pentium Pro o posterior recomendado.
- Sistema Operativo Windows NT Server 4.0 o Windows 2000 Server, configurado como PDC (Controlador Primario de Dominio).
- Service Pack 3 o posterior para Windows NT 4.0.
- 32 MB de memoria RAM. 64 MB recomendados.
- 5 MB de espacio libre en disco duro requeridos para la instalación.
- Unidad de CD-ROM.
- Tarjeta de Red configurada en Windows NT y protocolo de red TCP/IP.

ESTACIÓN DE PROCESAMIENTO

La estaciones que trabajarán como estaciones de procesamiento deben cumplir con las siguientes características:

- PC con un procesador Intel Pentium o posterior. Pentium Pro o posterior recomendado.

- Sistema operativo Windows NT 4.0, Windows 2000, Windows 95 o 98. Configurado y autenticado como estación que pertenezca al dominio de Windows NT que controla el equipo de administración. Se recomienda Windows NT 4.0 o Windows 2000.
- Service Pack 3 o posterior si la estación utiliza Windows NT 4.0.
- Actualización de DCOM si la estación tiene Windows 95 versión a.
- 32 MB de memoria RAM. 64 MB recomendados.
- 5 MB de espacio libre en disco duro requeridos para la instalación.
- Tarjeta y servicio de red configurados para funcionar en la red (cliente para redes Microsoft y protocolo TCP/IP).

ESTACIÓN DE ALMACENAMIENTO DE IMÁGENES YA GENERADAS

- Sistema operativo Windows NT 4.0, Windows 2000, Windows 95 o 98.
- Service Pack 3 o posterior si la estación utiliza Windows NT 4.0.
- Actualización de DCOM si la estación tiene Windows 95 versión a.
- 32 MB de memoria RAM. 64 MB recomendados.
- 1 GB de espacio en disco como mínimo
- Tarjeta y servicio de red configurados para funcionar en la red (cliente para redes Microsoft y protocolo TCP/IP).

ANEXO B: GUÍA DE INSTALACIÓN

1. Inicie Windows e introduzca el CD de instalación de *Antares* en la unidad de CD-ROM.
2. Espere a que se ejecute automáticamente el instalador de *Antares*. Si luego de unos segundos este no se activa, se puede ejecutar manualmente usando el archivo **Instalar.exe**, ubicado en la raíz del CD de instalación.
3. Siga las instrucciones que aparecen en la pantalla dependiendo si quiere instalar el módulo de Administración o el módulo de estación de Procesamiento.

Se debe tener en cuenta que un *farm* debe estar compuesto al menos por:

- Una estación con el módulo de administración instalado.
- Una estación con los módulos de interfaz y procesamiento. Debe ser una estación independiente a la de administración. El módulo de procesamiento no puede ejecutarse en la misma estación donde funciona el módulo de administración.

Será posible insertar nuevas estaciones al *farm* en cualquier momento con solo instalar en ellas el módulo de procesamiento. Además, si la estación de procesamiento funcione sobre un sistema operativo Windows 95 o 98, el componente auxiliar *WrapperMon* es necesario para que la estación de procesamiento funcione correctamente (ver sección 4.1.2: *Arquitectura de software para sistemas distribuidos*).

ANEXO C: GUÍA DE UTILIZACIÓN

Inicie el Administrador de Antares utilizando el menú de Inicio (*Start*):

Figura 21. Menú de inicio del administrador Antares

A continuación podrá apreciar la Interfaz de Usuario del Administrador (Figura 22), con el menú de comandos, la barra de herramientas y las tres vistas: Estaciones, Trabajos y Eventos. Es posible cambiar la dimensión de cada una de las vistas y el ancho de cada una de sus columnas

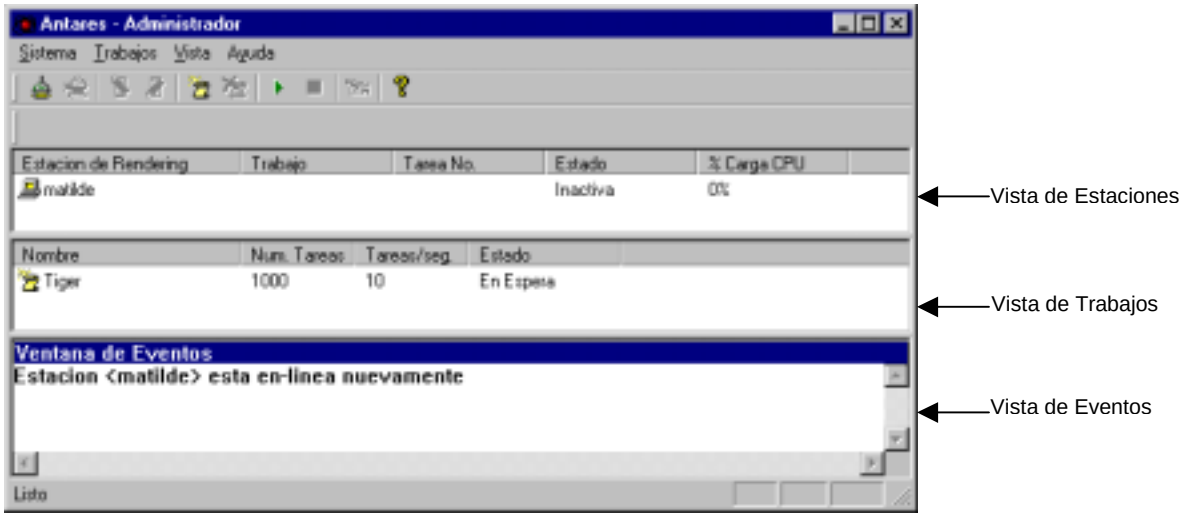


Figura 22. Ventana principal del Administrador

La barra de herramientas de la aplicación puede verse en la Figura 23.

1 2 3 4 5 6 7 8 9 10 11

1. Registrar estación
2. Desregistrar estación
3. Habilitar estación
4. Deshabilitar estación
5. Insertar trabajo
6. Eliminar trabajo
7. Ver propiedades del trabajo
8. Iniciar procesamiento
9. Detener procesamiento
10. Generar Video
11. Ayuda

Figura 23. Barra de herramientas del Administrador

OPCIONES DE CONFIGURACIÓN

Usted podrá acceder a las opciones de configuración del administrador a través de la opción Configuración del menú Sistema, o presione Ctrl-C. El cuadro de diálogo que aparece a continuación se puede ver en la Figura 24.

A continuación se describen las diferentes opciones de configuración. Cabe anotar que cualquier cambio en estas opciones es almacenado en la base de datos del registro de Windows (*Registry*).

- Tiempo entre Refresco de Estado: permite configurar el intervalo de tiempo en milisegundos entre cada uno de los eventos de “Pulso”, recibidos de cada una de las estaciones registradas en el sistema. Debe tener un especial cuidado al cambiar el valor por defecto (3000 ms), puesto que si el valor es muy pequeño causará una carga innecesaria en la red y el Administrador. Y si este valor es muy grande, no tendrá una información muy actualizada del estado de cada estación en el sistema.

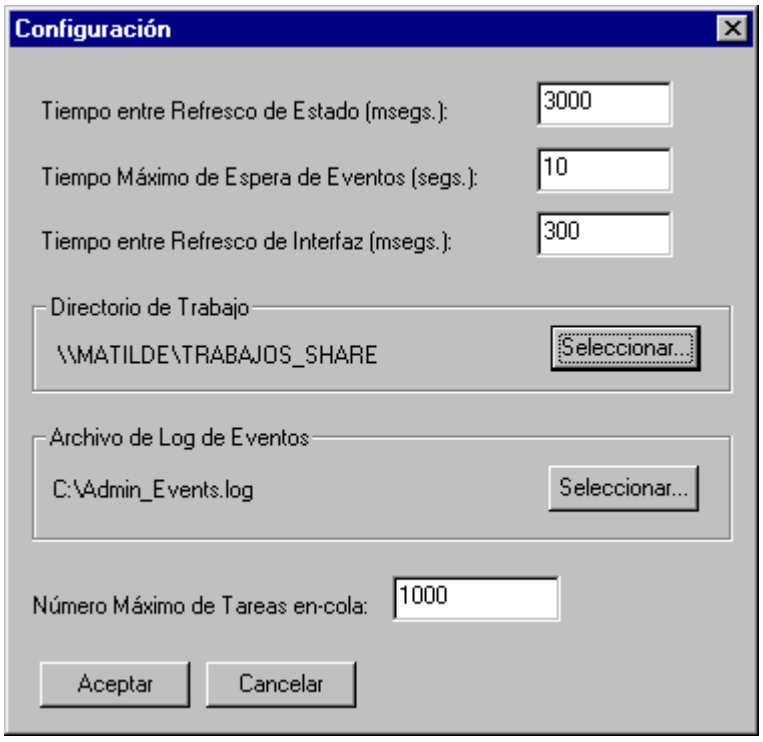


Figura 24. Cuadro de diálogo de opciones de configuración

- Tiempo Máximo de Espera de Eventos: permite configurar el tiempo (en segundos) que el Administrador espera por un evento de "Pulso" antes de cambiar el estado de una estación a "muerto". Por obvias razones este tiempo no puede ser inferior al insertado en el parámetro anterior. Aconsejamos no cambiar su valor por defecto (10 seg.) en lo posible, a menos que note congestión en la red y necesite aumentar este valor.
- Tiempo entre Refresco de Interfaz: configura el intervalo de tiempo en ms. entre actualizaciones sucesivas de la información que muestran las vistas de Estaciones y Trabajos. Si cambia su valor por defecto (300 ms.), tenga en cuenta que un valor muy pequeño puede ocasionar parpadeo excesivo en las vistas, y un valor muy grande, que se esté mostrando información desactualizada en las vistas.
- Directorio de Trabajo: permite elegir el directorio donde se encuentran los trabajos a ser procesados. Para esto muestra una lista de todos los directorios compartidos que encuentran en la red.

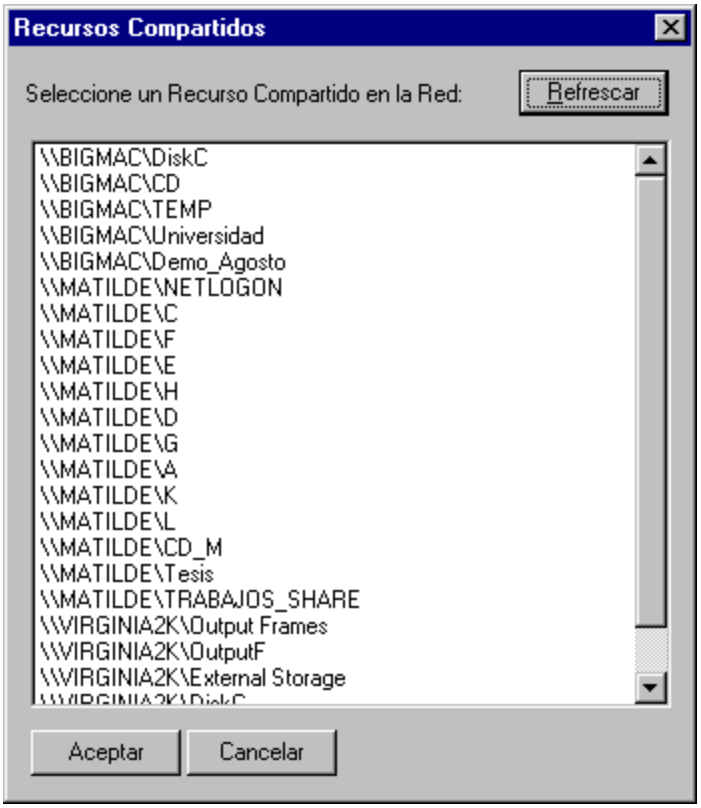


Figura 25. Cuadro de diálogo de selección del directorio de trabajo

- Archivo de Registro de Eventos : permite indicar el nombre y ubicación del archivo que almacenara la bitácora (log) de los eventos sucedidos en el sistema.
- Numero Máximo de Tareas en la Cola: permite asignarle un tamaño máximo a la cola de tareas en espera. El valor de este parámetro determina el rendimiento de las hebras productora y consumidora de tareas (ver sección 3.5: *Subsistema de generación de tareas*); si se desea cambiar el valor por defecto debe observarse cuidadosamente el rendimiento del sistema una vez se hayan hecho los cambios.

Deberá reiniciar el Administrador para que este reconozca cualquier cambio en estos parámetros, a excepción del directorio de trabajo.

INSERTAR UNA NUEVA ESTACIÓN EN EL SISTEMA

Usted podrá registrar una nueva estación en el sistema a través de la opción Registrar Estación del menú Sistema, haciendo clic en el botón en forma de conector de red en la barra de herramientas, o presionando Ctrl-R. El Administrador mantiene una lista de las Estaciones que han sido registradas en una ocasión anterior, por lo que se puede digitar el nombre de la estación o seleccionarla de la lista. Luego de hacer clic en el botón Aceptar, el Administrador se conectará con el módulo de Antares que reside en la estación y lo preparará para que reciba Tareas.

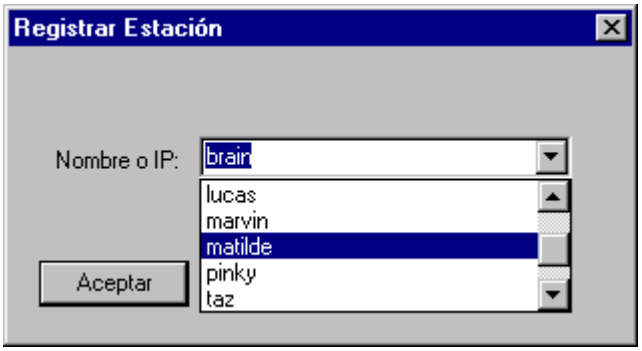


Figura 26. Cuadro de diálogo de registro de una estación

Si hay estaciones registradas en el momento de salir de la aplicación, esta configuración será almacenada en disco (ver Tabla 3. Descripción de miembros de la clase *CRenderStation*), y de esta manera la próxima vez que el Administrador se ejecute, estas estaciones serán registradas automáticamente al inicio de la aplicación.

ELIMINAR UNA ESTACIÓN DEL SISTEMA

Se puede eliminar una estación del sistema a través de la opción Eliminar Estación del menú Sistema, con el botón en forma de conector de red con una X roja, o presionando la tecla Del habiendo seleccionado la Estación a eliminar en la Vista de Estaciones. También podrá hacer esto a través del menú de

contexto de la Estación seleccionada, que aparece cuando se hace clic con el botón derecho del *mouse* sobre una estación. También se pueden eliminar varias estaciones al mismo tiempo, seleccionándolas en la vista haciendo clic sobre ellas mientras se mantiene presionado el botón Ctrl.

Cuando se elimina una estación del *farm*, el Administrador le retira todas las tareas que estaba realizando, coloca estas tareas de regreso a la cola de asignación de tareas y termina las instancias de los módulos de interfaz y procesamiento que se estaban ejecutando en la estación.

DESHABILITAR Y HABILITAR TEMPORALMENTE UNA ESTACIÓN

Para evitar que se le asignen trabajos a una estación especifica se debe utilizar esta opción, disponible en la opción Deshabilitar en el menú Estación, o en el botón con una palanca hacia abajo (Deshabilitar) o hacia arriba (Habilitar) en la barra de herramientas. Estas opciones también se encuentran en el menú de contexto de la estación seleccionada.

Si una estación es deshabilitada cuando está procesando una tarea, esta tarea es cancelada y colocada de nuevo en la cola de tareas en espera. Esto es útil cuando el módulo de procesamiento de una estación se queda bloqueado por alguna razón mientras procesa una tarea en particular.

INSERTAR UN NUEVO TRABAJO

Para insertar un nuevo trabajo se utiliza la opción Adicionar del menú Trabajos, presionando Ctrl-A o haciendo clic en el botón con una hoja de papel en la barra de herramientas. Esto le permitirá elegir el archivo .X (animación de DirectX) que representa su trabajo usando un cuadro de diálogo estándar de Windows, y a continuación la aplicación le mostrará un cuadro de diálogo que

le permitirá ajustar las propiedades del nuevo trabajo. Este cuadro de diálogo se puede ver en la Figura 27.

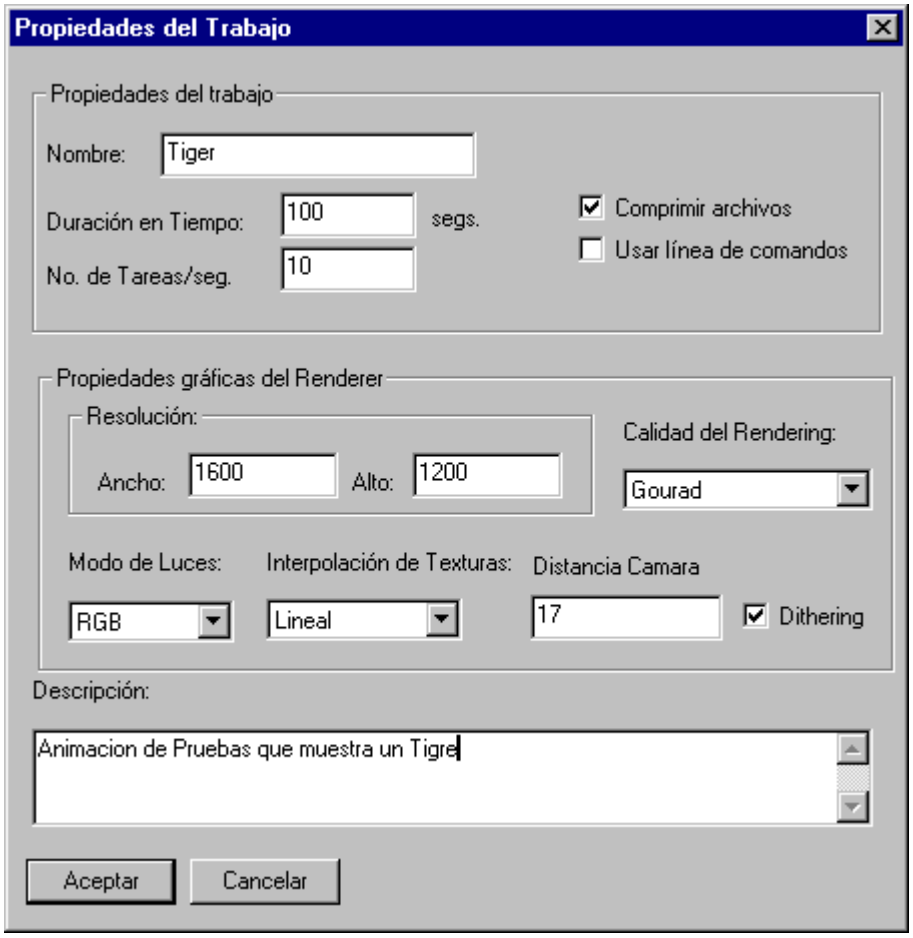


Figura 27. Cuadro de diálogo de propiedades de un nuevo trabajo

Si hay trabajos insertados en el momento de salir de la aplicación, esta configuración también será almacenada en disco (ver *Tabla 4. Descripción de miembros de la clase CJob (del módulo administrador)*) y de esta manera la próxima vez que el Administrador se ejecute, estos trabajos serán insertados automáticamente al inicio de la aplicación.

Antes de añadir un trabajo es necesario crear un nuevo directorio con el nombre del archivo .X dentro del directorio de trabajos que se eligió en las opciones de Configuración del Sistema. Si el archivo de trabajo requiere de

archivos adicionales de texturas, estos deben ser colocados dentro de un directorio llamado “texturas”. También debe existir un directorio llamado “frames” que contendrá los archivos de resultado (por lo general, archivos gráficos con formato .bmp o archivos comprimidos con extensión .zip) del procesamiento de cada Tarea por las Estaciones. La siguiente figura ilustra un ejemplo de esta estructura.

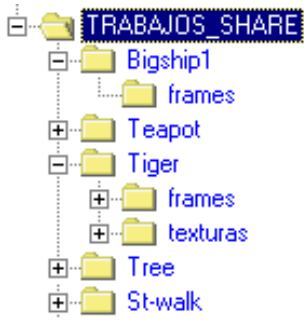


Figura 28. Estructura de directorio que se debe usar para organizar los trabajos

VER Y MODIFICAR LAS PROPIEDADES DE UN TRABAJO

Se pueden ver y editar las propiedades de un trabajo usando la opción Propiedades del menú Trabajos, o haciendo doble clic sobre el trabajo seleccionado en la Vista de Trabajos. También se puede acceder a esta opción a través del menú de contexto del trabajo seleccionado. El cuadro de diálogo que aparece a continuación es similar al de la Figura 27.

Usted podrá cambiar las propiedades de un trabajo solo si no se ha comenzado a ser procesado por las estaciones. Para el caso de estar en ejecución, solo podrá observar sus propiedades.

ELIMINAR UN TRABAJO

Para eliminar un trabajo se puede usar la opción Eliminar del menú Trabajos, o el botón con una hoja de papel y una X roja en la barra de herramientas. También es posible acceder a esta opción con el menú de contexto del Trabajo seleccionado. Se pueden eliminar varios trabajos al mismo tiempo, seleccionándolos primero y luego usando cualquiera de las opciones ya mencionadas.

Un trabajo puede ser eliminado en cualquier momento, sin importar el estado en que se encuentre. Si el trabajo ya se estaba ejecutando, todas las tareas son suspendidas, y l sistema comienza a procesar el siguiente trabajo, si lo hay. Pero las imágenes que alcanzaron a generarse se conservan en el directorio *frames* del trabajo eliminado, por lo que es posible observar el progreso del trabajo antes de que fuera eliminado.

SUSPENDER Y REINICIAR EL PROCESAMIENTO DE UN TRABAJO

Se puede evitar que un trabajo siga o llegue a ser procesado a través de la opción Suspender del menú Trabajos. También se puede utilizar la opción Reiniciar para que el trabajo seleccionado pueda ser procesado. Estas opciones también se encuentran en el menú de contexto del trabajo.

Si se suspende un trabajo antes de que sea procesado, cuando la cola de trabajos llegue al trabajo suspendido, éste se quedará en estado de espera y los trabajos que se encuentran detrás de él serán procesados. Cuando el trabajo sea reiniciado, entonces deberá esperar a que el trabajo que se está ejecutando se termine.

INICIAR Y DETENER EL PROCESAMIENTO DE LOS TRABAJOS

Habiendo registrado las estaciones de procesamiento y los trabajos, se procede a iniciar el procesamiento de estos últimos. Para esto se utiliza la opción Iniciar Procesamiento del menú Sistema. También se puede detener el procesamiento de estos utilizando la opción Detener Procesamiento, y si se desea, reiniciarlo nuevamente. Los botones correspondientes en la barra de herramientas son el botón con un triángulo (Iniciar) y con un cuadrado (Detener).

Detener el procesamiento de los trabajos tiene el siguiente efecto: el trabajo que se está ejecutando es suspendido, y las tareas que hacen parte del trabajo son canceladas y colocadas en la cola de trabajos en espera.

GENERAR EL VIDEO DE LA ANIMACIÓN

Una vez se han terminado todas las tareas que componen un trabajo, en el directorio *frames* se encuentran todos los archivos que componen la animación. Para obtener un video que permita ver la animación como tal, se puede usar la opción Generar Video en el menú Sistema, o el botón con una cámara en la barra de herramientas, o usando la combinación de teclas Ctrl-G. El administrador construirá un archivo de video en formato .AVI con todas los imágenes de la animación, y lo colocará en el mismo directorio donde aquellos se encuentran. El tiempo invertido en esta operación depende del tamaño de las imágenes, de la cantidad de espacio en disco y de la capacidad de procesamiento del administrador.

AYUDA

Finalmente, con esta opción se puede obtener información de la aplicación: versión, autores, *copyright,* fecha, etc.