69

Universidad Nacional Autónoma de México

FACULTAD DE INGENIERIA

DESARROLLO Y ADECUACION DE HERRAMIENTAS DE SOFTWARE PARA LA INGENIERIA ESTRUCTURAL UTILIZANDO MULTIMEDIOS

T E S I S

Que para obtener el Título de

INGENIERO EN COMPUTACION

p r e s e n t a n

Yolanda Beatríz Pérez Gutiérrez

José Juan Contreras Ventura

Director de Tesis: Dr. Gustavo Ayala Milián

México, D. F. 1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

AGRADECIMIENTOS

A mi esposa Claudia por nuestro amor y el apoyo que he recibido de su parte.

A mis padres y hermanos que me han acompañado y estimulado durante gran parte de mi existencia.

A mis suegros.

A los profesores de la Facultad de Ingeniería que nos han transmitido sus conocimientos y experiencias.

José Juan

A mis padres, hermanos y a Hugo por su amor y apoyo incondicional.

A los profesores de la Facultad de Ingeniería para que continúen con entusiasmo formando profesionales de alta calidad.

Yolanda

Al Dr. Gustavo Ayala por permitirnos colaborar en el proyecto SOCRATES dentro del Instituto de Ingeniería y dirigir esta tesis.

Al Dr. Marco A. Murray por su colaboración en la producción del documento y por brindarnos facilidades para la difusión del tema dentro y fuera de la UNAM.

A Rodolfo, Marco y Aimée por su ayuda en la búsqueda de información en algunos puntos de la tesis.

Yolanda y José Juan

CONTENIDO

INTRODUCCION

La evolución de los métodos de enseñanza de la ingeniería ha seguido muy de cerca el desarrollo de los equipos de cómputo y de las herramientas de programación. Ejemplo de esto es el proyecto Sócrates, desarrollado en la Universidad de Cornell en el estado de Nueva York, EUA con el patrocinio de la Fundación Nacional de Ciencia de los Estados Unidos de Norteamérica con el propósito de desarrollar herramientas de software de utilidad en la enseñanza de la ingeniería consistente con la evolución y disponibilidad de equipos de cómputo en las escuelas de ingeniería. Este proyecto, del cual la Facultad de Ingeniería de la UNAM es miembro activo, ha tenido gran éxito en diversas universidades en países de primer mundo originando la creación de nuevos grupos de desarrollo y ampliando su alcance a otras áreas de ingeniería y ciencia.

El equipo empleado para la programación original de las aplicaciones del proyecto SOCRATES fueron estaciones de trabajo bajo el sistema operativo VMS. La programación se realizó con el lenguaje FORTRAN usando como kernel gráfico las librerías HOOPS.

El objetivo inicial del proyecto SOCRATES dentro de la Facultad de Ingeniería de la UNAM es desarrollar métodos para la enseñanza de la ingeniería estructural asistida por computadoras, en primera instancia a nivel de cursos de posgrado para su posterior aplicación a los de nivel licenciatura. Para esto se han trazado objetivos particulares mediante los cuales se conseguirá desarrollar y/o adecuar métodos y herramientas de software para la enseñanza de la ingeniería estructural en equipos de cómputo que, con costo limitado, ofrezcan el poder de cálculo y las capacidades de visualización requeridas por esta actividad.

Los programas presentados en esta tesis fueron creados para la enseñanza del método de las rigideces de la ingeniería estructural. UNAM-GISMO y UNAM-GISMA son acrónimos de los programas originales llamados en Inglés CU-GISMA (Cornell University - Graphical Interpreter for Structural Matrix Analysis) y CU-GISMO (Cornell University - Graphical Interpreter for Structural Matrix Operations). UNAM-GISMO es un intérprete gráfico de operaciones estructurales matriciales que muestra por etapas el proceso completo de análisis estructural cubriendo: definición, construcción de rigidez, estudio de parámetros, resolución y presentación de resultados. En todas las etapas se utilizan recursos de visualización para facilitar su comprensión. Dadas la limitaciones de presentación en pantalla de grandes estructuras y almacenar y manipular su correspondiente matriz de rigidez, el programa UNAM-GISMO restringe su operación al manejo de estructuras relativamente pequeñas en todos sus niveles. El programa UNAM-GISMA, sin embargo, tiene la capacidad de manejar estructuras más complejas, las etapas de análisis estructural que abarca este sistema son: definición, resolución y presentación de resultados. A diferencia de UNAM-GISMO este sistema carece de las etapas de construcción de rigidez y estudio de parámetros debido a las limitaciones mencionadas previamente.

El objetivo de esta tesis es presentar los procedimientos de adecuación e inclusión de multimedios para los programas UNAM-GISMO y UNAM-GISMA, así como una pequeña introducción a las interfaces y librerías gráficas en estaciones de trabajo UNIX. La etapa de adecuación incluyó la migración de los programas a la plataforma UNIX y la generación de rutinas en el lenguaje C para interactuar con el sistema operativo. En la inclusión de multimedios se generaron archivos de audio y aplicaciones gráficas para usarse en el sistema de ayuda de los programas UNAM-GISMO y UNAM-GISMA.

Se encuentra que los programas UNAM-GISMO y UNAM-GISMA han sido adecuados y traducidos al español para funcionar en estaciones de trabajo UNIX, las cuales puede utilizarse como servidores para que otras computadoras (PC's, MAC's, SUN, HP, etc.) puedan ejecutar los programas a través del protocolo gráfico X11.

CAPITULO 1 ANTECEDENTES DE LOS PROGRAMAS GISMA Y GISMO

1.1 PROYECTO SOCRATES

En Estados Unidos existe una organización llamada "National Engineering Education Coalition" (Coalición Nacional de la Educación en Ingeniería) que tiene como propósito primordial el incrementar la calidad de la educación en ingeniería y el número de egresados en esta área. Los fines anteriores se pueden lograr mediante la disponibilidad de utilerías de software y hardware para el manejo electrónico de información. Las Universidades miembros de esta coalición son: California Polytechnic State University en San Luis Obispo Misuri, Cornell University en Nueva York, Hampton University en Misuri, Iowa State University en Iowa, Southern University en California, Stanford University en California, Tuskegee University en Alabama, University of California en Berkeley y la UNAM en México.

La Universidad de Cornell participa dentro de la coalición con una continuación del proyecto SOCRATES que surgió con el propósito de utilizar el potencial de cómputo en el proceso de la enseñanza de la ingeniería mediante sistemas de aprendizaje de alta calidad y un apropiado contenido pedagógico. La existencia del proyecto SOCRATES ha permitido crear y distribuir sistemas de aprendizaje basados en exhibiciones gráficas para ser utilizados como auxiliares en el proceso de aprendizaje.

1.1.1 Introducción

En el año de 1980 se crea la coordinación de CADIF (Computer Aided Design Instructional Facility) en la Universidad de Cornell, esta coordinación se encarga de administrar los recursos de cómputo a los estudiantes y a desarrollar utilerías de software basadas en presentaciones gráficas como auxiliares para el aprendizaje.

En un principio lo que motivo la creación del software fue la terminación de los tediosos cálculos que tenían que hacer los estudiantes para resolver los problemas-ejercicio típicos en ingeniería, sin embargo el fin real del software es el de desarrollar la habilidad de "pensar como ingenieros" a los estudiantes, es decir, mediante visualización de sistemas y datos comenzar a proporcionar bases intuitivas para la descripción matemática de fenómenos reales.

Con la puesta en marcha de los programas, los estudiantes de un curso de análisis de estructuras eran capaces de terminar el trabajo de todo un semestre en tan sólo ocho semanas, dejando al profesor más tiempo para plantear nuevos problemas. Además de la rapidez en la solución de problemas, el estudiante puede explorar por iniciativa propia una gran variedad de casos y ejercicios variando algunos parámetros en el planteamiento original de los problemas-ejercicio.

En 1986 el departamento de educación de EUA funda el proyecto SOCRATES como medio para impulsar y documentar el software existente, escribir nuevas aplicaciones y hacer posible el ejecutar los programas en nuevas plataformas de hardware (estaciones de trabajo). Para 1990 SOCRATES llega a ser parte de la Coalición Nacional de la Educación en Ingeniería con la intención de revolucionar la enseñanza de la ingeniería a través de los avances tecnológicos en la información.

1.1.2 Características de las aplicaciones de SOCRATES

Interactividad .- Esta característica se refiere a la forma en que el estudiante explora e interactúa con el programa.

Visualización .- Esta característica aprovecha la capacidad del cerebro humano para procesar y discriminar información visual. La exhibición gráfica es esencial para comprender patrones y memorizar más rápidamente procesos.

Simulación .- Son muchos los fenómenos en la ingeniería que no pueden ser observados directamente debido a diversas causas; pueden ser muy rápidos, demasiado lentos, en muy alta o larga escala, porque ocurren dentro de cuerpos sólidos, etc. La simulación por computadora no solamente hace a estos procesos asequibles, sino que permite operaciones que son físicamente imposibles, como: la evolución en el tiempo, repetición instantánea, las comparaciones simultáneas de casos y la comparación de modelos matemáticos con el mundo real. La simulación es una herramienta que complementa el análisis y los experimentos físicos, cuando se vuelve común y fácil ayuda al estudiante a comprender los límites de la computación y el uso de los métodos numéricos en un problema específico.

Poder de cómputo .- El poder de cómputo debe ofrecerse a los estudiantes para encontrar soluciones reales a problemas de gran complejidad numérica (grandes cantidades de información).

Fácil de aprender .- En la industria se puede justificar que una persona gaste algunas semanas de tiempo aprendiendo el manejo de un sistema que usará por varios años. En el caso de estos sistemas de aprendizaje un estudiante con quince minutos de instrucción es capaz de hacer uso productivo del programa.

Fácil de usar .- Existen programas llamados "fáciles de aprender" que, sin embargo poseen graves errores, pues algunos abortan su ejecución por acciones no previstas del usuario, por lo que no pueden usarse rápida y eficientemente. Los programas de SOCRATES tienen incorporados rutinas para retornos en cualquier parte del sistema, submenúes, comandos de llave cuando es posible, comandos de ayuda a lo largo de todas las pantallas del programa, diagramas que muestran a petición del usuario el grado de avance en un problema, etc.

Robusto .- Los programas que no operan satisfactoriamente, en general generan errores, dan resultados inconsistentes, pueden no actualizar apropiadamente la exhibición de una imagen, etc. Un buen sistema debe haber pasado por un largo proceso de pruebas antes de ser utilizado para los fines que se creó.

Bien documentado .- Este punto se refiere a los manuales en línea, manual de usuario, manual técnico y copias comentadas del código fuente. La documentación debe estar bien elaborada para permitir a otras personas el fácil uso y adecuaciones de los programas.

Transportable .- El programa debe poderse instalar, ejecutar y mantener en cualquier tipo de equipo de cómputo disponible en el centro de enseñanza.

Adaptable .- Algunas escuelas no están en posición de escoger el tipo de computadoras a usarse en sus centros de enseñanza. En el mercado existen numerosos modelos de estaciones de trabajo y computadoras personales con tiempos de vida de pocos años, por lo que se requiere que todos los programas de SOCRATES estén escritos usando técnicas de portabilidad estándar en la industria.

Pedagógicamente apropiado .- Un software comercial orientado a la resolución de problemas de ingeniería puede ser de excelente calidad, sin embargo no es necesariamente apropiado para su uso en la enseñanza por algunas de las siguientes razones:

- Requiere de un considerable tiempo de aprendizaje.
- Está orientado a la solución automática de problemas y no da explicaciones o ilustraciones de los principios usados para llegar a los resultados.
- No puede ser modificado (por ejemplo para comparar algún otro algoritmo alternativo).
- Su compra y mantenimiento son extremadamente costosos.

1.1.3 Proceso de desarrollo de aplicaciones

Las ideas en las que se basa el proyecto SOCRATES han sido proporcionadas por los profesores que lo crearon y desarrollaron, y en algunos casos por los mismos estudiantes (al momento de usar los programas). Las prioridades asignadas en el desarrollo del proyecto fueron:

- Planeación para dar servicio a un gran número de usuarios.
- Direccionar nuevos campos o tópicos,
- Sacar provecho de las experiencias particulares de los miembros del equipo de desarrollo,
- Hacer un uso efectivo del hardware y
- Dar oportunidades para el desarrollo de nuevas técnicas.

Uno de los primeros pasos para iniciar el proyecto fue el diseño de reuniones entre los profesores de ingeniería y los programadores, las reuniones fueron el intérprete para que los académicos intercambiaran ideas en el dialecto los programadores/técnicos. En estas reuniones se definieron propósitos, lineamientos, supervisiones y criterios de programación para los programas del proyecto. El proceso de trabajo fue mediante demostraciones de los programas en desarrollo a los supervisores (frecuentemente profesores), para sugerir, por parte de los programadores opciones y características que ayudaran a mejorar la idea original del profesor.

Al inicio de la elaboración del programa se realizaron prototipos del funcionamiento general y el tipo de interfaz que se planeaba presentar al usuario. Una vez que la interfaz y el funcionamiento estaban claramente definidos se procedió a la elaboración del manual de usuario. Finalmente, se revisó el manual y una vez aprobado éste se procedió a la codificación del programa.

Al momento de agregar una nueva función al programa ésta se probaba continuamente y se hacían demostraciones a los supervisores del proyecto. Hacia el final del desarrollo de una aplicación se realizaban pruebas con estudiantes y usuarios no familiarizados con el programa. En este punto de desarrollo los supervisores estaban ya familiarizados con el programa y tenían nociones para hacer uso de éste en las aulas.

1.1.4 Impacto de los programas de SOCRATES

El 10 % de los académicos en la escuela de ingeniería de la Universidad de Cornell estaban entusiasmados con el uso de los programas en estaciones de trabajo como material de apoyo para sus clases. La mayoría de los académicos pertenecían a las áreas de la Ingeniería Civil y Eléctrica, estos hacían uso de los programas como un verdadero sistema de ayuda gráfico e interactivo. Todos los estudiantes de ingeniería en Cornell entraban en contacto con los programas en algún momento de sus carreras. Los comentarios de los estudiantes sobre sus primeras experiencias con los programas fueron:

"Hacen posible el cubrir más material en menos tiempo"
"Facilitan el rápido aprendizaje de conceptos"
"Provocan un cuestionamiento más profundo del tema"
"Estimulan el interés del estudiante"
"Pueden ser extremadamente frustrantes si no son robustos"

Algunas industrias comentan que encuentran muy valiosa la experiencia en cómputo interactivo que brinda el desarrollo de estos programas porque ven futuros empleados con

excelentes conocimientos de CAD en estaciones de trabajo. En algunos casos varias compañías han planeado y establecido diseños y laboratorios de entrenamiento para sus respectivas compañías basados en la experiencia del diseño con gráficas interactivas.

1.2 PROYECTO SOCRATES EN LA UNAM

El trabajo desarrollado en la Facultad de Ingeniería por un equipo integrado de diferentes especialidades que abarcan las referidas a la computación y la ingeniería estructural durante el año de 1993 y los primeros meses de 1994 consistió en dar los primeros pasos para adecuar los programas GISMA y GISMO elaborados dentro del proyecto SOCRATES, para su uso en el sistema de enseñanza a niveles de licenciatura y maestría de la U.N.A.M. con las perspectivas de extender su utilización a otras instituciones del país.

El empleo inicial de estos programas será en proyectos de tesis de grado que requieran de este tipo de herramientas y cursos de posgrado, sin embargo una vez integrados estos en el contenido curricular y probados por los estudiantes de maestría se usarán en cursos de nivel licenciatura, siendo en este nivel donde se podrán apreciar sus verdaderos beneficios al mejorar las características de los cursos y facilitar la enseñanza de las asignaturas que traten el análisis estructural no solo en la UNAM sino en cualquier centro universitario del país.

CAPITULO 2 MULTIMEDIOS

2.1 MULTIMEDIOS

Los multimedios en computación hacen referencia a sistemas que tienen como elemento controlador la computadora para manejar elementos tales como texto, audio, video, animación y gráficos. Su uso está enfocado a la producción de sistemas de aprendizaje, sistemas de comunicación o de entretenimiento.

Con la tecnología existente es posible la elaboración de multimedios, empleando las computadoras y diversos dispositivos de almacenamiento adecuados para este fin. Las aplicaciones de multimedios han aparecido y siguen en continuo desarrollo, estas incluyen video notas, edición de video, recorridos de tópicos, simulaciones, juegos de aventura, audio libros, sistemas tutoriales auxiliares para el aprendizaje, kioskos de información, estaciones de conocimiento, y presentaciones.

Los desarrolladores de software y usuarios han comenzado el desarrollo de aplicaciones con convenciones y conceptos conocidos: televisión, libros, películas, juegos de computadora, aprendizaje auxiliado por computadora y han descubierto nuevas formas de uso para aprovechar el ambiente que dan los multimedios. Los sistemas de programas autores y algunos conceptos como hipermedios (multimedios estructurados como el hipertexto) y laboratorios de aprendizaje son originados por la forma de uso de esta tecnología.

La tecnología de multimedios seguramente cambiará la forma de pensar y de resolver problemas. En años recientes los procesadores de palabras, hojas de cálculo y las bases de datos han cambiado la forma de pensar con respecto a la escritura, las operaciones aritméticas en forma tabular y el acceso a la información. La tecnología de multimedios hace posible nuevas formas de análisis de información y nuevas habilidades que podrán ser usadas en otras disciplinas.

Los componentes de la tecnología de multimedios han cambiado con el tiempo. En un principio una aplicación típica de multimedios requería de controladores especiales para videodiscos y/o discos ópticos, actualmente estos dispositivos están disponibles en módulos y hasta en las mismas computadoras. Los multimedios serán más rápidos y menos costosos mientras más sean usados para diversas actividades.

Debido a su enorme auge, los multimedios se han clasificado en diferentes categorías dependiendo de los métodos utilizados en su desarrollo. A continuación se presentan los más importantes.

2.1.1 Programas Autor

Los programas autor se forman por un proceso de manipulación de las aplicaciones de multimedios para la creación de material que será visto por otras personas. Los programas autor utilizan una gran cantidad herramientas de cómputo como por ejemplo: los editores de texto, programas pará hacer presentaciones, captura y manejo de imágenes, edición de archivos de audio, etc.

Mediante los sistemas autor se pueden desarrollar sistemas para entrenamiento de personal, presentaciones y aplicaciones de tipo corporativo, tal es el caso de las notas de audio usadas en el correo electrónico.

2.1.2 Playback

Es una forma de mostrar las aplicaciones de multimedios. Mediante "playback" se puede incluir en una presentación interactiva tomas de video o animaciones mostradas en pantalla sin que interfieran en la interacción sistema-usuario, es decir, los procesos en "playback" corren en la aplicación sin que se origine un retardo en la selección de una opción de la aplicación.

2.1.3 Tiempo Real

Una operación en tiempo real no tiene retardo entre la comunicación de datos y la respuesta originada por estos. Un ejemplo sería el siguiente:

Al estar trabajando en una estación de trabajo se hace un cambio de información en la pantalla, en ese momento un colaborador en otra estación de trabajo ve el cambio de manera inmediata en su pantalla.

2.1.4 Almacenamiento y reenvío

El almacenamiento y reenvío es un método para manejar la información de multimedios a distancia. La información es creada en un punto en determinado tiempo, y recibida en otro. Un ejemplo es el correo electrónico de nuestros días.

2.1.5 Colaboración

Este concepto involucra a dos o más personas trabajando juntas en tiempo real o en almacenamiento y reenvío. Las aplicaciones en este concepto permiten el trabajo de varios usuarios en red para compartir recursos tales como pantallas, pizarras y la participación en video conferencias.

2.1.6 Edición

La edición en multimedios se da en todos los tipos de recursos de comunicación disponibles (sonido, video, etc.); se hacen operaciones de recorte, copia, inserción y cambio de tamaño en: voz, animación, música, imágenes de video, gráficas y texto.

2.1.7 Sincronización.

La sincronización es un proceso preciso ejecutado en tiempo real (por debajo del tiempo de un milisegundo); al momento de correr una aplicación de multimedios en red se transmite información de audio e imágenes; en la transmisión el tiempo es un factor crítico, pues, no se desean tener retardos al momento de mostrar texto en la pantalla de la computadora o a la hora de estar escuchando información de audio.

2.1.8 Comunicación en un solo sentido.

La comunicación en un solo sentido es hecha para ser manipulada por un solo usuario final. Algunos ejemplos de este tipo de comunicación son: los materiales de entrenamiento y de documentación.

2.1.9 Granularidad

La granularidad es el periodo de tiempo que transcurre desde que se selecciona una acción dentro de un programa hasta que se puede realizar otro tipo de interacción con él. Mientras más corto sea este tiempo se suele decir que la granularidad es fina, es decir, que la respuesta del sistema a las acciones del usuario son más rápidas y permiten una mejor interacción.

2.1.10 Comunicación entre fuentes variadas.

Este tipo de comunicación se hace entre dos o más personas, la transmisión de datos no tiene un sentido definido (todos se pueden comunicar con todos). Algunos ejemplos de este tipo de comunicación son las video conferencias, en ellas una persona expone un tema y las demás están en la posibilidad de hacer preguntas y/o responder al conferencista.

2.1.11 Documento compuesto

En este documento se pueden incorporar diferentes tipos de datos como pueden ser: texto, audio, video, imágenes y gráficas.

2.1.12 Video conferencias

En las video conferencias la comunicación es en varios sentidos entre los participantes, la información la componen frecuentemente datos de audio y video. El uso de las computadoras para hacer una conferencia es reciente, en su realización se necesita de equipo destinado solamente a esta actividad así como recursos de video, audio y una buena sincronización de datos.

2.1.13 Hipertexto

Se le denomina hipertexto a la información exhibida en forma de palabra escrita cuya organización está referenciada por la selección de palabras clave, estas palabras tienen la peculiaridad de aparecer frecuentemente remarcadas y al momento de seleccionarlas por algún dispositivo periférico se presenta información que hace referencia a ésta.

2.1.14 Hipermedios

Este concepto es una mezcla de sistemas autor y playback. Con software de hipermedios se pueden recuperar múltiples capas de documentos compuestos y hacer una búsqueda de cualquier tópico. Un ejemplo de ambiente hipermedios sería el siguiente:

Un documento en hipermedios puede estar formado por video, imágenes y palabras clave que al momento de ser seleccionarlas por medio del ratón recuperen otro documento compuesto o bien ejecuten otra aplicación en particular. Se suele decir que hipermedios es una extensión del hipertexto con más recursos.

2.1.15 Tiempo de ejecución

Un ambiente de tiempo de ejecución es aquel en el que se utiliza material de multimedios ya elaborado. Solamente se puede ejecutar la aplicación, no se puede editar ésta para modificar su contenido.

2.1.16 Dispositivos de control

Mediante los dispositivos de control se pueden manejar una gran variedad de periféricos, tales como: cámaras de video, discos ópticos, videodiscos, etc.

2.1.17 Telefonía

Se refiere a la integración del audio en las estaciones de trabajo. En estas máquinas se pueden integrar datos de audio a bases de datos y al correo electrónico. Tal es el caso de las actuales estaciones de trabajo de SUN, las cuales ya incluyen una utilería para enviar correo electrónico con audio.

2.2 POSIBILIDADES DE LOS MULTIMEDIOS

El campo de acción de los multimedios es enorme, ya que encuentran aplicación en los más diversos campos de la vida, llámense ciencias sociales, naturales, matemáticas, sistemas financieros y económicos, entretenimiento, sistemas de aprendizaje, etc.. La utilización de los multimedios en la vida cotidiana ha hecho cambiar la forma hacer las cosas en muchos aspectos.

La funcionalidad que ofrecen los multimedios ha cambiado la forma de realizar actividades tan simples como rentar un película o ir de compras, lo cual ahora con las nuevas tecnologías es posible hacer desde la comodidad del hogar a través de la televisión interactiva que se encuentra enlazada mediante fibra óptica a una red comercial de supermercados.

Una de las áreas con mayor desarrollo ha sido el área de entretenimiento, actualmente existen juegos de video que dan la impresión al usuario de ser partícipe de la acción y los eventos que en ellos ocurren. El sonido, video, animación, lo introducen a un mundo que en ocasiones parece real, donde la realidad y la fantasía empiezan a entremezclarse y surge el concepto de realidad virtual.

Una de las áreas más importantes y que ha revolucionado los métodos tradicionales para la transmisión de conocimientos es la ayuda que brindan los multimedios al proceso de enseñanza-aprendizaje en cualquiera de las áreas de conocimiento. Para las nuevas generaciones las formas de aprendizaje son más sencillas y visuales con más posibilidades de las que se tenían antes de los multimedios.

2.2.1 Multimedios en la educación

¿Porque son importantes los multimedios en la educación?

Los multimedios pueden mejorar la calidad de la educación. Durante mucho tiempo las personas han pensado en cuál sería la forma más fácil de tener acceso a información rica en imágenes y sonido, ahora mediante los multimedios se está resolviendo el acceso a este tipo de información.

Los educadores pueden abandonar sus pizarrones, gises y libros para dar paso a nuevos materiales en el proceso enseñanza-aprendizaje. Mediante discos ópticos de enseñanza, imágenes y una secuencia de presentación para exponer un tema se tiene como límite la imaginación de los educadores. Los estudiantes que no tienen facilidad de expresión escrita tienen ahora una nueva forma de comunicación y de aprendizaje.

2.2.2 Proyecto Shakespeare

Existe actualmente un proyecto de desarrollo de un sistema de multimedios para el aprendizaje del arte y temas de humanidades. El proyecto tiene el nombre de "Shakespeare" y se está desarrollando en la Universidad de Stanford [1], en los Estados Unidos. El proyecto tiene como objetivo ser un auxiliar en el aprendizaje del teatro y de las obras del autor inglés Shakespeare. Con el sistema se pretende que los alumnos estudien, hagan creaciones y disfruten los temas teatrales. El sistema Shakespeare esta formado en hardware por computadoras y video discos. El director del proyecto es el profesor Larry Friedlander, quien ha dado clases de Shakespeare y actuación por más de veinticinco años.

El sistema tiene como objetivo mostrar al estudiante escenas de teatro, por ejemplo, de la obra Hamlet. En Hamlet se pretende mostrar el vestuario de los personajes, los escenarios y la secuencia de acontecimientos en el transcurso de la obra. Con este sistema se pueden definir nuevos vestuarios, escenarios y hasta variaciones al diálogo en cada una de las obras teatrales o escenas que forman el sistema.

Este proyecto es muy ambicioso, pues tratar de mostrar conceptos abstractos como son los movimientos del cuerpo, expresiones corporales, modas de ropa, etc. Estas metas requieren de mucho trabajo por parte de los diseñadores y programadores del proyecto.

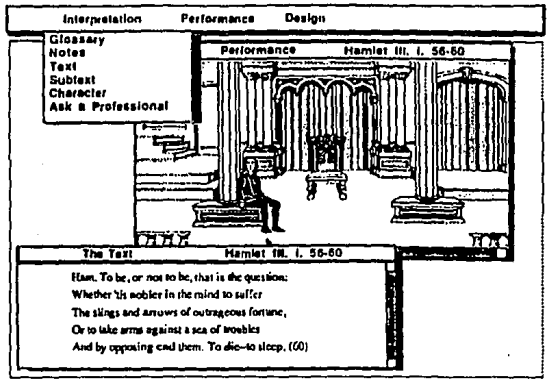

Fig. 2.1 Proyecto Shakespeare

Conjuntando las capacidades de la televisión con el control que proporcionan las computadoras es posible crear un programa para televisión que muestre una sensación de movimiento a través de una historia o aventura, es decir se pueden realizar programas de tipo educativo que expliquen conceptos de una materia en texto y al estar repasando su significado introducir el video para un mejor entendimiento.

Una de las mayores aplicaciones de los multimedios en la educación es el diseño de sistemas de aprendizaje por medio de la televisión. En estos se puede presentar a un maestro auxiliado de las facilidades de audio y video para la exposición de un tema. Esto es muy bueno, por ejemplo: si se está explicando el concepto de sonido, se daría la definición con palabra y después se presentaría audio y hasta la forma de las ondas del sonido.

2.2.4 Laboratorio de aprendizaje

La principal idea del "laboratorio de aprendizaje" [1] es conjuntar diversa información de un tema contenida en diferentes medios (texto, video, voz, etc.) para hacer su manipulación de forma interactiva. Esta forma de trabajo puede conducir al diseño de sistemas tutoriales inteligentes, los cuales pueden tener la capacidad de mostrar información que auxilie a las personas en el aprendizaje de un tema. Mediante este laboratorio se facilita la capacidad de memorizar una idea si ésta es recibida por audio y video al mismo tiempo. Esto es porque existen más cosas que la mente puede relacionar. Los "laboratorios de aprendizaje" pueden ser manejados a través de videodiscos interactivos o mediante televisores con pantallas sensibles al tacto.

Una de las primeras producciones en televisión que incorporaron las ideas del laboratorio de aprendizaje fue la del proyecto "Beyond Einstein". El tema del proyecto es el de la Física y en especial de la teoría de Einstein. La forma de interactuar con el usuario es por recorridos, texto, diagramas, preguntas y mediante herramientas para avance o retroceso de imágenes,

etc. Inicialmente la producción se desarrollo en cinta magnética, pero ya está disponible en disco compacto interactivo (CD-I).

Fig. 2.2 Proyecto "Beyond Einstein" (aplicación por televisión)

La figura anterior es del proyecto " Beyond Einstein", el cual es un antecedente de los laboratorios de aprendizaje.

2.3 MULTIMEDIOS EN SISTEMAS DE APRENDIZAJE

Los multimedios han generado una rápida evolución de los métodos tradicionales de enseñanza-aprendizaje.

La interactividad en cualquier sistema de aprendizaje en donde intervienen los multimedios juega un papel crucial, ya que lo que hace realmente interesante a los multimedios no son sólo las presentaciones de animaciones, video digital o sonido estéreo, sino precisamente la interacción. La capacidad de hacer interacción con estos medios es de gran importancia, ya que si no existiera, el concepto de multimedios perdería claridad y podría considerarse que ha existido desde siempre ó al menos desde la introducción del cine o la televisión, los cuales combinan imágenes y sonido, pero difieren de los multimedios reales precisamente en que el usuario no tiene el control de la presentación y se reduce tan sólo a criterios como verla o no verla, encenderla o apagarla. Es más importante el que el usuario tenga el control de la presentación, que el hecho mismo de la incorporación de la computadora como dispositivo en donde ocurre la integración de los medios.

La interactividad debe ser algo más que la capacidad de encender o apagar la computadora, moverse de pantalla en pantalla de manera lineal o hacer ocasionalmente elecciones en un menú mediante el ratón o el teclado; la definición de interactividad es un poco más difícil, Andy Lippman [2], del Multimedia Lab del MIT (Massachusetts Institute of Technology) ha propuesto que la interactividad puede definirse como la actividad mutua y simultánea entre los participantes trabajando hacia conseguir una meta, usualmente aunque no necesariamente común. Y esta definición tiene cuando menos los siguientes elementos importantes: la capacidad de interrupción, la granularidad fina, la capacidad limitada de predecir acciones, la existencia de acciones por omisión u opciones prefijadas y la apariencia de continuidad ilimitada, términos que se definen a continuación.

La capacidad de interrupción es la clave para diferenciar la interactividad de simplemente estar alternando entre participantes; aquí la metáfora es la de la conversación o el diálogo, en que los participantes pueden libremente interrumpir a su interlocutor tanto para aclarar como para comentar sobre lo que se dialoga, características que dan a la conversación humana no sólo su riqueza sino su eficiencia comunicativa. Esto es diferente del simplemente ir "tomando turnos" de manera preestablecida sobre rutas fijas.

A su vez, la capacidad de interrupción requiere de que **"la granularidad"** sea fina, esto es, que la duración de las unidades de interacción sea pequeña, por ejemplo si uno sólo puede participar después de una hora, en realidad se asistió a una conferencia y no a una conversación. La granularidad fina permite la capacidad de interrupción.

Mediante **la capacidad limitada de predecir acciones** y la inexistencia de **omisiones** Lippman parece tener en mente la idea de que en la interacción real no hay una ruta preestablecida, ni una secuencia de eventos "óptima o prefijada"; puede haber sorpresas en una conversación real, en la que se pueden seguir caminos no previstos. No hay mucha interacción cuando sólo se tienen cuatro opciones a seguir en orden. Y si a cada paso se puede prever lo que sucederá, la interacción se vuelve aburrida y deja de tener sentido.

La **apariencia de continuidad ilimitada** en un programa y su planeación deben estar estrechamente relacionadas. Si ante el menor error, el sistema responde con un mensaje de "system error", o "bad command", o simplemente se queda "colgado" o "se cae", la interacción no existe. El sistema debe estar construido de forma tal que cuando el usuario haga demandas que no pueden satisfacerse, la interacción no se interrumpa de manera abrupta. Es por ello también que, en el mejor de los casos, el sistema debe dar la impresión de poseer continuidad ilimitada (aunque se sabe que no puede ser realmente ilimitada, ya que dependerá de los límites reales físicos de los medios de distribución).

Según Judda Schwartz [2] de la Universidad de Harvard, la interacción nunca puede ser impositiva -el programa ideal es aquél que no le "pregunta" o "le pide" al usuario algo en particular, sino que le permite hacer, y le orienta sobre lo que puede hacer.

Así, la interactividad es el acceso aleatorio a información (sin ruta preestablecida ú óptima), a voluntad del usuario, con un sistema de granularidad fina, interrumpible, con apariencia de

continuidad ilimitada, y a través de una interfaz amigable. Con respecto a la interactividad Gándara [2] afirma que "la interfaz es la base" y el elemento clave para que un sistema de multimedios sea realmente interactivo. Esta característica es especialmente importante en las aplicaciones educativas.

2.3.1 Ventajas educativas de los multimedios

En el Congreso Metropolitano Para Estudiantes de Ingeniería efectuado en el museo tecnológico de la CFE en 1993 [2] se presentaron cuatro características que hacen de los multimedios una verdadera opción educativa:

1.- No todo el aprendizaje ocurre en el mismo canal perceptual.

Algunos aprendizajes involucran experiencias auditivas (por ejemplo el estudio de música ó idiomas); otros requieren experiencias visuales (el reconocimiento de estilos en el arte, la adquisición de información por vía del texto, etc); otros más, finalmente, requieren de experiencias motrices - de manipulación directa de objetos o situaciones (por ejemplo, aprender a nadar o manejar un vehículo). Muchas experiencias involucran a más de un sentido o canal perceptual, a veces de manera directa y a veces como refuerzo de lo que ocurre en otros canales.

2.- No todas las personas aprenden igual.

Cada día se incrementa la evidencia a favor de que no todos aprendemos igual; la característica en la escuela tradicional de que todo el aprendizaje sea por la vía de oír o leer parece carecer de un fundamento cognoscitivo sólido. El tipo de instrucción normal fuerza a todos los estudiantes a aprender de una misma manera. De ahí viene la frustración de aquellas personas que, ya sea por una situación de deficiencias perceptuales en un canal particular, o por la vía de tener un estilo de presentaciones estandarizadas no puedan aprender igual. En realidad, hay personas que aprenderían muy bien si pudieran manipular objetos o practicar en vez de aprender teoría; y en otros casos, hay estudiantes que avanzarían mucho más rápidamente mediante ilustraciones y ejemplos gráficos que mediante textos. Es por lo anterior que la instrucción tradicional encajona el estilo de aprendizaje de diferentes individuos.

3.- No todos los humanos tienen los mismos antecedentes sobre lo que aprenden.

Es típico que al diferenciar un nivel promedio para toda una clase se logra que algunos estudiantes se aburran al encontrar el material demasiado sencillo, mientras que otros se desalientan por que lo encuentran demasiado difícil. En particular, en el caso de la instrucción no formal o no escolarizada, como es el caso de la capacitación para el trabajo, los estudiantes generalmente tienen antecedentes académicos muy diversos, por lo que tratar

MATEMATICAS

El uso de elementos gráficos en los métodos de enseñanza de las matemáticas básicas, es una técnica muy utilizada, pero cuando un alumno empieza a adentrarse en matemáticas más complicadas como el cálculo, el uso de imágenes desaparece, a excepción de la teoría de graficación cuando se dibujan funciones en dos dimensiones. Esto esta cambiando rápidamente, el uso de computadoras gráficas en el proceso de enseñanza-aprendizaje es una técnica muy aceptada en el salón de clases. Con el advenimiento de calculadoras baratas con capacidades de graficación y programas para la visualización bidimensional y tridimensional de funciones, la concepción de esta asignatura por parte de los alumnos ha mejorado notablemente.

ARTE

Actualmente en el área de arte y de diseño, las gráficas y las imágenes con computadora se enseñan como cursos separados. En las escuelas de arte, las herramientas de software se usan principalmente para investigar aplicaciones de computadora para graficación en 2D y 3D, animación y diseño gráfico interactivo. Las herramientas de software son elaboradas para crear productos finales como ilustraciones, impresión de imágenes y animaciones. Dentro de la carrera de arte, las herramientas de software y el arte mismo son difíciles de separar. Sin embargo, usando esas herramientas, el estudiante puede olvidarse un poco del producto final y dedicarse a explorar el proceso de diseño gráfico.

Los multimedios han provisto formas más eficientes para aprender conceptos básicos como teoría de colores y aplicaciones, relaciones espaciales, y diseño de aplicaciones en los que intervienen múltiples medios (animaciones, texto, imágenes, etc.). Algunas adiciones al repertorio de herramientas disponibles para el arte están surgiendo como la interrelación de diseño dimensional, lo cual ha forzado a separar estas herramientas en dibujo (2D), modelado (3D) y movimiento de imágenes (4D).

GRAFICACION

Por la experiencia es bien sabido que los estudiantes aprenden más poniendo en práctica los conceptos adquiridos teóricamente. En las gráficas por computadora pueden leer extensivamente sobre alguna técnica o algoritmo particular, y aún no entenderlo por completo. Proporcionar herramientas para practicar lo leído en los libros origina un entendimiento inmediato e intuitivo de un concepto gráfico complejo.

Los estudiantes de computación que estudian graficación por computadora pasan mucho tiempo programando algoritmos de graficación para poder tener una mejor comprensión de la técnica. Muchas veces el excesivo código que hay que generar para un algoritmo de graficación hace que el alumno pierda el objetivo del curso. Un programa interactivo que permita visualizar un objeto a través de distintos algoritmos de graficación sería más útil para el entendimiento intuitivo de los algoritmos.

Como podemos ver, los sistemas de aprendizaje a través de multimedios están ya en todas las áreas del conocimiento humano, proporcionando a maestros y estudiantes un nuevo camino para el proceso enseñanza-aprendizaje, que contempla mayores y mejores posibilidades de lograrlo.

CAPITULO 3 INTERFACES GRAFICAS

3.1 INTERFACES GRAFICAS

Las terminales de texto se han usado por largo tiempo para la comunicación entre los usuarios y las computadoras. En este tipo de sistemas un intérprete de comandos es el encargado de establecer la comunicación entre la máquina y el usuario, las peticiones se realizan por el teclado y el resultado de ellas se conoce mediante texto que aparece en el monitor. Este tipo de terminales están cambiado gracias al avance de la tecnología, ahora con computadoras más veloces y con terminales de alta resolución y varios dispositivos de entrada de datos como el ratón, monitores sensibles al tacto y lápiz óptico, se facilita el uso de los recursos de las computadoras a través de una comunicación más rápida entre los usuarios y las computadoras.

La forma de interactuar con estos nuevos recursos ha sido mediante el desarrollo de interfaces gráficas. Al igual que el intérprete de comandos una interfaz gráfica se utiliza como el medio de comunicación entre el usuario y la computadora. La mayoría de los comandos que se utilizaban en el intérprete se sustituyen por movimientos del ratón sobre un conjunto de iconos, los cuales representan la ejecución de una serie de herramientas para manipular los recursos de la computadora.

Una de las primeras interfaces gráficas introducidas al mercado por las computadoras Apple fue WIMP (windows, icons, menus, pointing-devices); esta fue bien recibida por la gente debido a la sencillez de manejo que resultaba mediante el uso de iconos y el ratón.

Desde la aparición de WIMP, los fabricantes de computadoras personales como IBM sintieron la necesidad de imitar esta interfaz, por lo que en colaboración con Microsoft se creó a WINDOWS. WINDOWS es una interfaz gráfica para computadoras IBM-PC o compatibles que corre bajo el sistema operativo MS-DOS.

De la misma forma que en computadoras personales se desarrollan interfaces gráficas para el usuario también en las computadoras con más poder de cómputo como las estaciones de trabajo se han desarrollado varias interfaces gráficas como por ejemplo: Motif, AIX de IBM, OpenWindows de SUN y VUE (visual user environment) de HP. Todas estas están basadas en el protocolo gráfico XWindow.

3.2 XWINDOW

Inicialmente XWindow [4] tenía el nombre de W y fue creado en el proyecto Athena del MIT con apoyo de DEC (Digital Equipment Corporation). W era un sistema de ventanas síncrono de red que corría bajo el sistema operativo Standford V (1982). Posteriormente DEC portó el sistema W a UNIX y en el mismo año 1983 el MIT aumentó su portabilidad y funcionalidad mediante transmisiones asíncronas y le dió el nombre de X. En septiembre de 1985 el MIT lo da a conocer como su primera versión pública, el X versión 9. Esta versión ofrece una interfaz completamente independiente de la plataforma de hardware. En 1986 Jim Gettys de DEC, Robert Scheifler y Ron Newman del MIT completan la mayor parte de desarrollo de X dándole el nombre de X versión 10.

Debido a la popularidad de X y sus características de portabilidad se inició su comercialización. En 1987 se crea el consorcio X; este organismo proporciona las normas y especificaciones del protocolo gráfico para que los desarrolladores de hardware puedan transportar X a sus equipos. Es en esta época cuando a X se le da el nombre que actualmente tiene, el de X versión 11 o simplemente X11.

3.2.1 Arquitectura de sistema para X11

La arquitectura que posee X11 es la de cliente-servidor. El servidor es el encargado de controlar las exhibiciones gráficas entre los programas de aplicación además de los dispositivos de entrada/salida. Cuando se inicia una sesión de X se inicializa el proceso de servidor X (X server). Cada exhibición de X requiere de un servidor dedicado para controlar su operación. El servidor coordina la entrada de datos desde dispositivos como el teclado y el ratón o bien otra aplicación, es también responsable de actualizar la imagen en la pantalla, la inclusión de texto y gráficas.

El cliente X es una aplicación que utiliza los servicios del ambiente X. El cliente se comunica con el servidor a través del protocolo de red X, el cual es un conjunto de funciones de bajo nivel (Xlib y Xtoolkit). La interacción entre el cliente y el servidor se caracteriza porque el cliente hace peticiones al servidor y el servidor notifica al cliente de eventos como movimientos del ratón, presión de botones, etc. Un servidor puede controlar varios clientes y consecuentemente un cliente puede conectarse a otros servidores. Por lo general un cliente puede desplegarse en cualquier estación de trabajo que posea un servidor X y que acepte conexiones de diferentes clientes.

Arquitectura del Sistema

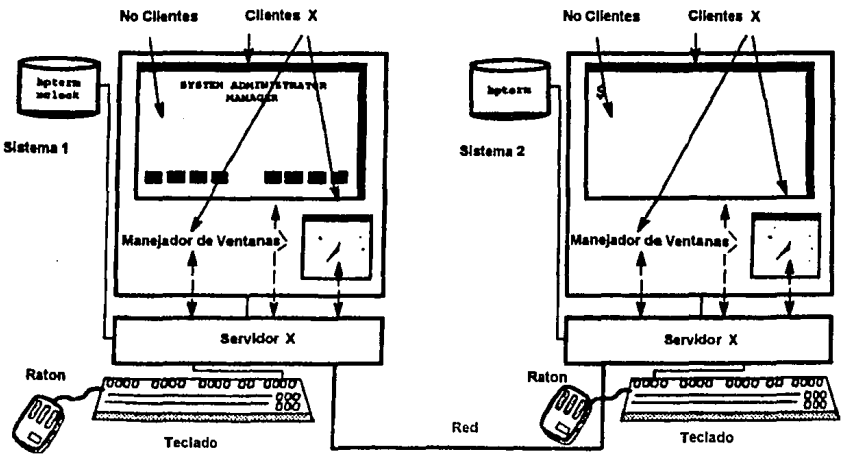

Fig. 3.1 Arquitectura de X11

El sistema es reconocido ampliamente como el estándar de la industria para los sistemas de ventanas basados en redes de computadoras. La más grande ventaja de X es que las aplicaciones escritas para una plataforma en particular, corren en casi cualquier otra plataforma sin modificaciones.

En el mercado, los vendedores ofrecen el sistema X como un paquete que incluye a Motif u otro sistema de widgets (herramientas gráficas).

X presenta una ventana raíz en la que es posible exhibir ventanas mas pequeñas. X permite al usuario correr más de una aplicación o cliente a la vez, donde cada aplicación puede tener cualquier número de ventanas.

Componentes de una Interfaz X

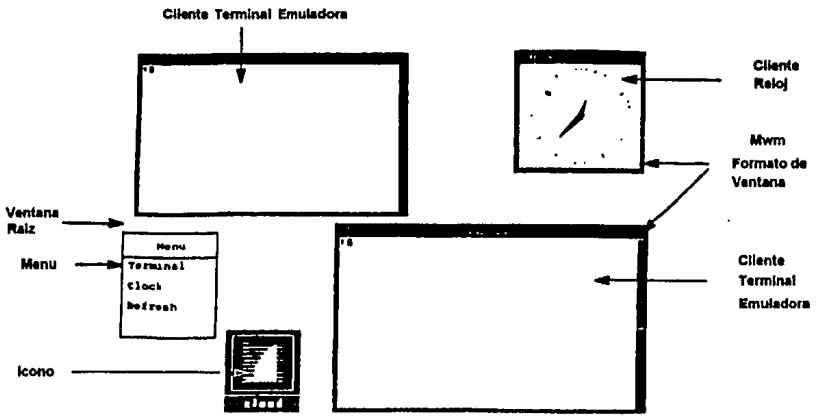

Fig. 3.2 Componentes de X

X posee un conjunto de librerías de funciones escritas en C, a éstas se les conoce en conjunto como Xlibs y forman el corazón de X. La programación haciendo uso exclusivo de las funciones Xlib puede ser aburrida y tediosa, es por esto que los diseñadores de X agregaron un segundo grupo de funciones llamado Intrínsecos Xt o Xtoolkit que hacen más sencilla la programación.

3.2.1.1 El Administrador de Ventanas

El sistema X concentra el control de ventanas en una aplicación cliente especial llamada administrador de ventanas. El administrador de ventanas es en realidad otro cliente que corre en X (aunque es un cliente especial) que le permite al usuario controlar el tamaño y la posición de las ventanas en la pantalla.

El administrador de ventanas más conocido es el de OSF/Motif y se llama Mwm (Motif Window Manager). Mwm permite el movimiento o cambio de tamaño de las ventanas, reacomoda el orden de éstas en la ventana raíz, permite crear ventanas adicionales, conversión de aplicaciones a iconos y otras funciones.

Otros ejemplos de administradores de ventanas en uso son: twn (tom window manager), olwm (OPEN LOOK window manager), awn (ardent window manager) y la rtl (tiled window manager).

3.2.1.2 Terminal Emuladora Xterm

X esta diseñado para exhibir solamente imágenes gráficas en forma de mapas de bits (bitmaps), por esta razón, uno de los clientes más importantes es una terminal emuladora, la Xterm. La Xterm permite entrar a un sistema multiusuario y correr aplicaciones designadas para usarse sobre una terminal alfanumérica estándar. Cualquier cosa que se pueda hacer sobre esta terminal, se puede hacer sobre una Xterm.

La Xterm es una de las terminales emuladoras más práctica, emula una terminal DEC®VT102 o una terminal Textronix®4014.

Al ejecutar varios procesos Xterm sobre X11, es equivalente a trabajar con múltiples terminales en una sola pantalla. Si se tiene más de una Xterm activa, se pueden correr varios programas al mismo tiempo. Por ejemplo en una Xterm se puede estar ejecutando un proceso de transferencia de datos, en otra un proceso de edición, en otra mas consultando el correo, etc.

3.2.1.3 El Administrador de pantalla

El administrador de pantalla, Xdm (X display manager), es un cliente diseñado para iniciar el servidor X11 automáticamente (desde el archivo UNIX /etc/rc System Startup) y conservarlo en ejecución permanente.

En su más básica implementación el administrador de pantalla emula los programas getty y/o login; los cuales muestran en pantalla el indicador para pedir el login sobre una terminal, conservando el proceso de servidor en espera de datos para la cuenta y la clave del usuario. Dirige la sesión estándar para entrar en el sistema.

Actualmente Xdm tiene nuevas características. El usuario puede especificar el entorno de una sesión, automáticamente se ejecutan varias aplicaciones y se establecen recursos personales tales como teclado, el ratón y características de la pantalla.

3.2.2 Consorcio X

El consorcio X (Xconsorcio) fue creado en el verano de 1984, es parte del Laboratory for Computer Science del MIT. Su principal función es la promoción de cooperación entre industrias de cómputo para la creación de interfaces de software estándar para todas las capas del sistema X Window. El papel del MIT es suministrar una arquitectura neutral para los vendedores y la dirección administrativa para hacerlo. El consorcio X es financiado económicamente con ingresos de cualquier organización. Existen dos clases de ingresos: Miembro (para grandes organizaciones) y Afiliados (para pequeñas organizaciones).

Las actividades del consorcio X son supervisadas por el Comité Directivo del MIT, el cual establece la vigilancia y suministra estrategias directivas revisando las actividades del consorcio. El proceso de estandarización incluye una revisión pública y una demostración de pruebas de conceptos, típicamente establecidas con público e implementación portátil de especificaciones.

3.3 MOTIF

Es una interfaz gráfica desarrollada por la Fundación de Software Abierto (Open Software Foundation) en EUA, más conocida como OSF. Esta interfaz gráfica esta construida en X. El propósito de OSF desde su fundación en 1988 es desarrollar un ambiente de interfaz gráfica para el usuario estándar.

Motif consta de cuatro partes:

- Ayuda en línea para usuario.
- Una herramienta API (interfaz de programación para aplicaciones) de rutinas en C que ayuda a construir aplicaciones que conforman la ayuda en línea.
- El administrador de ventana construye aplicaciones con cualquier XWindow API, y las aplicaciones Motif corren bajo cualquier administrador de ventanas.
- Un lenguaje diseñado para el fácil desarrollo de interfaz de usuario.

Motif da la misma apariencia a todas las aplicaciones (cliente) que corren en ella, su administrador de ventanas indica colores, tipo de marco y atributos de las ventanas. Si una aplicación especifica una apariencia diferente a la de Motif se tendrá la apariencia definida por ésta aplicación.

Motif utiliza un conjunto de componentes llamados widgets y gadgets para lograr la interfaz gráfica de usuario. Los widgets y gadgets son en apariencia los botones de desplazamiento en las ventanas, los botones de elección de opciones y las cajas de dialogo. Motif en combinación con X permite que el usuario programe aplicaciones mediante una interfaz gráfica que funciona en prácticamente todas las estaciones de trabajo comerciales.

A continuación se presentan el nivel que ocupa Motif dentro de un sistema UNIX:

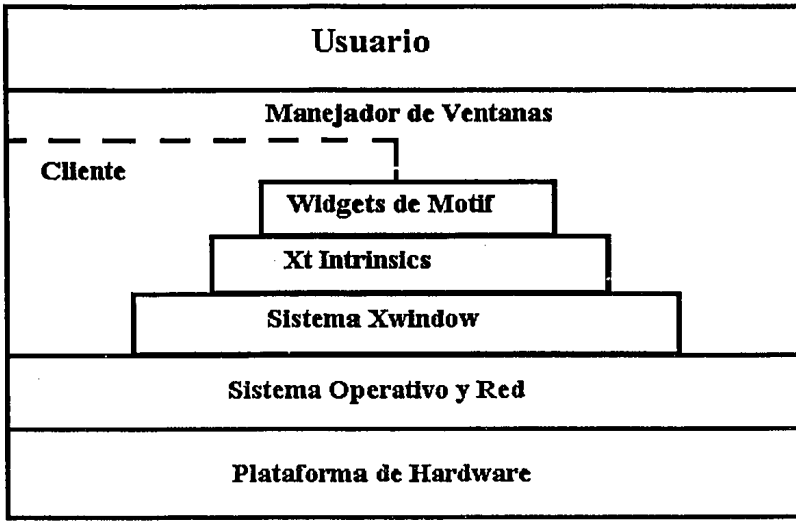

Fig. 3.3 Capas de un sistema basado en X

3.3.1 Widgets y Gadgets

Como se menciono en el párrafo anterior un widget es un componente de la interfaz de usuario. En realidad los widgets son estructuras de datos y procedimientos que al exhibirse en pantalla poseen su propia ventana. Un gadget es un widget que no posee su propia ventana.

Los widgets y gadgets de Motif fueron diseñados con los principios de programación orientada a objetos, es decir, están organizados en conjuntos de distintas clases. Una clase es un conjunto de objetos con características similares. Uno de los aspectos más importantes de la programación orientada a objetos es la herencia, cualquier clase puede heredar características de una clase mas elevada. Por ejemplo, el widget "pushbutton" puede heredar características del widget etiqueta, el widget primitiva y el widget núcleo si consideramos a estos últimos superclases del "pushbutton". Muchos widgets tienen el aspecto de ventanas, aunque algunos no son visibles.

Un gadget no tiene su propia ventana, pero puede exhibir en pantalla su contenido en su correspondiente ventana madre. Esta característica los hace más eficientes que los widgets en aplicaciones donde se tienen que mostrar, por ejemplo, un gran número de "pushbuttons".

3.3.1.1 Jerarquía

La arquitectura orientada a objetos de Motif agrupa a los widgets y gadgets en distintas clases. Cada clase tiene estructuras de datos y procedimientos que operan sobre los datos. Las clases se estructuran en un árbol jerárquico que tiene a las superclases en la parte más alta del árbol.

Cualquier clase correspondiente a un widget puede heredar alguno o todos los recursos y atributos de una clase más alta en la jerarquía. La clase base de los widgets es la clase "core", ésta clase contiene recursos que se heredan a todas las demás clases. Un recurso es un nombre de datos o variables que afectan algún atributo del gadget o widget.

3.4 OPEN LOOK

Open Look es la interfaz gráfica desarrollada por la compañía SUN MicroSystems Inc. Está basada en el ambiente de ventas OpenWindows de la misma compañía, incluye el sistema X11/NeWS, que es la versión postscript de X.

Algunas herramientas de OpenWindows son:

ToolTalk

Es una utilería para aplicaciones que requieren de actualización automática de información. ToolTalk es un auxiliar para el desarrollo de DOE (Distributed Objects Everywhere). Este concepto es una visión de un nuevo paradigma de cómputo en el cual las aplicaciones modulares son creadas con objetos distribuidos de múltiples plataformas.

Solaris DeskSet.

Es un conjunto de quince aplicaciones y utilerías basadas en la interfaz gráfica de usuario OPEN LOOK. Un ejemplo de aplicación es el manejador de archivos (File Manager). La versión 3.0 de OpenWindows incluye dentro del DeskSet aplicaciones como el MailTool (manejador del correo), calendario, editor de texto, herramientas de Shell, calculadora y reloj. Las características más recientes del DeskSet incluyen correo de Multimedia, herramientas de audio, manejador de archivos de red, manejador de calendario para trabajos en grupo y ayuda en linea.

Tecnología orientada a objetos.

La tecnología orientada a objetos permite reducir la complejidad en la construcción de aplicaciones sofisticadas de forma distribuida, de esta manera se puede operar sobre plataformas múltiples. Esta tecnología ayuda a las aplicaciones que requieren de una gran

cantidad de recursos a utilizar los recursos de diferentes estaciones de trabajo. El proyecto DOE [5] de SUN pretende a sacar al mercado tecnología de objetos distribuidos.

Multimedia.

En las estaciones de trabajo de SUN el usuario tiene las capacidades de interactuar con audio, video, texto, gráficos, FAX y aplicaciones de telefonía integrada.

3.4.1 Arquitectura de OpenWindows

OpenWindows es un sistema basado en red con ventanas distribuidas que brindan un ambiente completamente interactivo para los usuarios y los desarrolladores de aplicaciones. Por el hecho de que OpenWindows es un sistema de ventanas basado en red, una aplicación puede estar corriendo en una máquina mientras la interfaz gráfica dirigida al usuario se exhibe en otra máquina.

OpenWindows se basa en un modelo de cómputo cliente-servidor. La aplicación de software se almacena en un servidor y el escritorio de trabajo del usuario es el cliente del servidor. A través del ambiente OpenWindows los usuarios pueden tener acceso remoto a aplicaciones desde cualquier nodo de la red e interactuar con aplicaciones en pantallas locales.

OpenWindows esta formado por siete capas, las cuales dan al usuario la interfaz gráfica para utilizar los recursos del sistema:

Interfaz Gráfica de Usuario Aplicaciones Windows	OPEN LOOK Interfaz Gráfica de Usuario Ambiente DeskSet
Herramientas de Prototipo para Interfaz de Usuario Utilerias para Interfaz	Openwindows Guia de programación Xview, OPENLOOK Intrinsics
Servidor de Ventanas y Manejador de Ventanas Modelo de Imagenes	X11/NeWS, Manejador de ventanas OPEN LOOK, Xlib, PostScript, OpenFonts
Sistema Operativo	UNIX

Fig. 3.4 Capas de OpenWindows

La primera y principal capa es el sistema operativo (SO) que tiene una interfaz con el hardware de la computadora. El SO para todas las plataformas Solaris es SunOS (UNIX de SUN).

La segunda capa contiene los modelos que usa el sistema de ventanas. Estos modelos controlan la forma en que el sistema de ventanas hace posible la exhibición de imágenes en pantalla. Xlib, PostScript y OpenFonts son los modelos. Xlib se basa en puntos (pixels) y hace el despliegue punto por punto. PostScript se basa en un modelo de esténcil y pintado que es independiente de la resolución y del dispositivo, usa un sistema de coordenadas mundiales y mapea la imagen en pantalla sin importar el número de puntos. OpenFonts ofrece fonts escalables.

La tercera capa contiene los elementos del sistema de ventanas: el servidor de ventanas y el administrador de ventanas. Estos forman el corazón de OpenWindows. Los servidores permiten a los programas cliente usar el sistema estándar XWindow para el exhibir gráficos o hacer exposiciones en PostScript mediante NeWS.

La cuarta capa contiene las herramientas para la interfaz de usuario. Una herramienta es una interfaz de programación de alto nivel que tiene elementos para construir una interfaz gráfica de usuario, hace el desarrollo de aplicaciones más sencillo y asegura que la interfaz de usuario sea consistente. Estas herramientas incluyen el XView Toolkit, el OPEN LOOK Intrinsics Toolkit (OLIT) y el NeWS Toolkit (TNT).

La quinta capa contiene la guía para el programador en OpenWindows, ofrece representaciones gráficas de los elementos de la interfaz de usuario (OPEN LOOK). Esta herramienta permite a los desarrolladores de software crear, compilar y evaluar una interfaz de usuario OPEN LOOK sin haber escrito código.

La sexta capa contiene aplicaciones de ventanas que son usadas por los usuarios de OPEN LOOK. En esta capa encontramos el ambiente "DeskSet" que está formado por herramientas como: administradores de archivos, editores de textos, editores de iconos y otras más que ayudan al usuario a crear aplicaciones y manejar los recursos de la máquina.

La séptima capa contiene la interfaz gráfica de usuario el OPEN LOOK GUI, ésta es la parte más alta en las capas y mediante ella el usuario de OPEN LOOK dispone de todo el software y las herramientas existentes en la estación de trabajo.

Open Windows cuenta con un servidor de ventanas simple para crear y administrar ventanas: X11/NeWS. Este último ofrece Xlib y gráficos PostScript en un solo administrador de ventanas combinando el sistema de ventanas de X11 con NeWS. El administrador de ventanas estándar de OpenWindows es OPEN LOOK.

3.4.1.1 OPEN LOOK Toolkits.

Los diseñadores pueden implementar el OPEN LOOK GUI usando Xlib, pero muchos usan al toolkit que fue escrito para el sistema de ventanas OPEN LOOK. El toolkit ofrece un conjunto de rutinas que implementan consistentemente las diversas interfaces que están descritas en las especificaciones funcionales de la interfaz gráfica de usuario OPEN LOOK. Hay tres herramientas disponibles para OPEN LOOK y son las siguientes: OPEN LOOK Intrinsics, XView Toolkit, y el TNT.

3.4.1.2 OPEN LOOK Intrinsics Toolkit.

OLIT consiste de widgets y de Intrínsecos Xt. La base de esta herramienta es el Intrínseco Xt, un conjunto de rutinas en C que monitorean los eventos relativos a la interacción con el usuario y dan el código correcto para manejar esos eventos. Los Intrínsecos Xt ofrecen un marco que proporciona la habilidad de construir una interfaz de usuario completa.

Otra capa de esta herramienta son los widgets. Cada widget es un conjunto de datos y código que le permiten al usuario implantar fácilmente abstracciones de interfaz gráfica tales como "pushbuttons" y barras de desplazamiento. Un widget define un área en la pantalla con la semántica necesaria para formar un objeto.

OLIT ofrece un conjunto definido de widgets que se pueden usar para definir la presentación de la pantalla de interfaz y manejar interacciones con los usuarios finales. También se pueden desarrollar nuevos widgets. Tanto los Intrínsecos Xt como el conjunto de widgets están construidos e integrados en la parte alta de la librería gráfica Xlib.

3.4.1.3 XView Toolkit.

Es un conjunto de herramientas y rutinas de X11 disponibles en SUN que permiten al usuario implementar el OPEN LOOK GUI. La interfaz de programación para aplicaciones (API) XView se deriva de SunView (antigua interfaz de las máquinas SUN) y ofrece una ruta de emigración estable y rápida para aplicaciones de SunView a la interfaz de usuario OPEN LOOK. XView ofrece la única ruta de emigración de un sistema de ventanas basado en kernel a un sistema de ventanas basado en red. Todos los botones, menúes, controles y ventanas que son creados con este toolkit cumplen con las especificaciones de OPEN LOOK. El programador de aplicaciones se debe asegurar que cada control y presentación de objetos dentro de la interfaz de usuarios se apeguen a las reglas especificadas en el OPEN LOOK Graphical User Interface Aplication Style Guidelines.

3.4.2 Comparación entre Microsoft Windows 3.0 y Solaris Open Windows.

La principal diferencia es que Windows 3.0 es un sistema de ventanas para un solo usuario y Open Windows es un sistema de ventanas basado en un protocolo gráfico de red. Mientras Open Windows permite a la aplicación correr en una máquina distinta de la que presenta el despliegue, Windows 3.0 necesita que la aplicación resida, se ejecute y se exhiba en la misma máquina.

Windows 3.0 se carga igual que cualquier aplicación de MS-DOS y una vez cargada trabaja de manera similar a un sistema operativo administrando memoria, el teclado, el ratón, el despliegue de video, la impresora y los puertos seriales. Pero en esto, MS-DOS es el que maneja la entrada y salida de archivos para Windows 3.0. Los tres principales componentes de Microsoft Windows son: el kernel, el GDI (interfaz de dispositivos gráficos) y los módulos de usuario. El kernel realiza los servicios del sistema tales como multitareas, administración de memoria y recursos. El GDI trabaja con los gráficos y los módulos de usuario ofrecen los demás servicios. Los módulos de usuario y GDI se comunican con los manejadores de dispositivos de Windows 3.0 quienes se encargan de manejar el hardware. Estos tres módulos juntos conforman el API para Microsoft Windows 3.0 y contienen herramientas como funciones, mensajes y estructuras de datos utilizadas para crear aplicaciones para Windows 3.0.

Microsoft Windows 3.0 es un sistema monousuario, por esto sus aplicaciones deben compartir recursos del sistema como el CPU, la memoria y los dispositivos. Cada aplicación gasta mucho tiempo y código administrando los recursos del sistema, mientras el kernel, GDI y los módulos de usuario ayudan a minimizar la cantidad de código que requiere cada aplicación. Los módulos hacen esto porque son como librerías ligadas dinámicamente que permiten a las aplicaciones compartir código.

En Open Windows los recursos del sistema son abundantes y para diferentes usuarios; no se necesitará gastar grandes cantidades de código y tiempo tratando de administrar los recursos del sistema.

4.1 LA LIBRERIA XLIB

Xlib es un conjunto de funciones de bajo nivel que proveen acceso a gráficos y funciones para programar una interfaz gráfica.

Xlib provee funciones y macros para dibujar líneas, crear ventanas, desplegar texto, y definir colores. Xlib esconde muchos de los tediosos detalles acerca de las conexiones con un servidor X, el mantenimiento de los enlaces de red, el actual formato de los fonts X y otros. La programación con Xlib es provechosa para el programador por que hace a su programa transportable, esto quiere decir que su programa podrá compilarse lo mismo sobre una estación de trabajo Sun, HP, Silicon Graphics o sobre una IBM PS/2 corriendo con sistema operativo UNIX.

Xlib es la base de muchas cajas de herramientas X conocidas como "Xtoolkits", de manera similar estas lo son para Motif y Open Look, como se muestra en la siguiente figura:

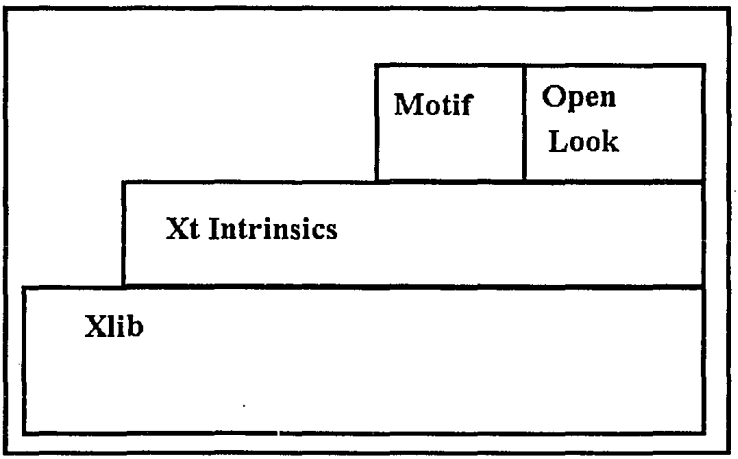

Fig. 4.1 . Niveles de funciones en OPEN LOOK

Los Xtoolkits pueden acelerar rápidamente la creación de aplicaciones X. Si se planea desarrollar una aplicación de tipo comercial se requerirá hacer uso de las librerías Xlib para alcanzar una excelente presentación del programa.

4.1.1 Programación con Intrínsecos Xt

Los Intrínsecos Xt son una librería gráfica que permite la elaboración de interfaces al usuario dentro del sistema X Window. Los Intrínsecos Xt (frecuentemente referidos como Xt) siguen la misma filosofía de Xlib, proveen mecanismos que no afectan la forma y características de una aplicación para interfaz al usuario. Xt es usada como un área de trabajo que permite a los programadores crear interfaces mediante la combinación de un número extenso de componentes. Estos componentes son conocidos como widgets y son funciones definidas para la programación de las siguientes herramientas: barras de desplazamiento, botones, menúes y cajas de diálogo. Un widget está formado por una sola ventana X11 y procesos que operan sobre esta ventana. Los Intrínsecos Xt definen un modelo de arquitectura para widgets que permite a los programadores extender sus herramientas para la creación de nuevos tipos de widgets.

4.1.2 Convenciones de Nombres

Cada procedimiento de interfaz para programación en X11 sigue convenciones de nombre. Los nombres para todas las funciones y estructuras disponibles al usuario de Xlib comienzan con la letra X. Cuando las funciones llamadas están compuestas por más de una palabra la primera letra de cada palabra estará en mayúsculas, por ejemplo:

```
XCreateWindow()      /* Creación de una ventana en X11 */
XDrawString()        /* Definición de una cadena en pantalla*/
```

Todas las macros de Xlib siguen estas mismas convenciones pero no comienzan con la letra X. Un ejemplo de esta excepción es:

```
DisplayWidth()       /* Ancho de la ventana */
ButtonPressMask      /* Mascara para estado del ratón */
```

Las librerías Xlib permiten también que algunas convenciones sean más fáciles de recordar, por ejemplo: el orden de los argumentos para cada función de Xlib. El primer argumento está referido como un apuntador a la estructura del Display. Si una función requiere un identificador de recurso como un argumento, el recurso ID inmediatamente sigue el recurso de Display. Los Drawables (estructuras que pueden exhibirse en pantalla) preceden a todos los demás recursos. Cuando los parámetros de una función incluyen especificaciones de tamaño y localización, "x" siempre precede a "y", ancho siempre precede a la altura, y un par (x,y) de coordenadas siempre precede al par (ancho,altura). Tenemos por ejemplo la siguiente función:

XDrawRectangle (display, drawable, gc, x,y, ancho, altura)

Como especifica la convención del orden de los argumentos, el primero de ellos es un apuntador a la estructura Display. Los siguientes dos argumentos son recursos; el primero de estos es un Drawable, precediendo a un segundo recurso, el contexto gráfico. Finalmente el par (x,y) de localización precede al ancho y largo de un rectángulo.

Los Intrínsecos Xt usan convenciones de nombre similares a las usadas por Xlib. Todas las funciones y macros comienzan con las letras Xt en contraparte a Xlib donde se diferencian las macros y funciones. Por ejemplo:

```
XtCreateWidget()        /* función */
XtSetArg()              /* macro */
```

Los Intrínsecos Xt usan cadenas constantes para especificar nombres de recursos. Estas cadenas caen en tres categorías: cadenas de nombre de recurso, cadenas de clase de recurso, y cadenas de presentación de recurso. Por convención, Xt define las cadenas de nombre de recurso por la adición del prefijo XtN a la cadena. Ejemplo:

```
#define XtNWidth   "width"
```

para cadenas clase de recurso se usa el prefijo XtC:

```
#define XtCBackground   "background"
```

y para cadenas de presentación de recurso se une el prefijo XtR en la cadena:

```
#define XtRInt   "int"
```

4.1.3 El modelo de programación Xtoolkit

Un Xtoolkit provee aplicaciones de programación con un modelo específico para escribir aplicaciones. Todas las aplicaciones X son normalmente designadas para ser manejadas a través de eventos. Las aplicaciones Xlib usualmente pueden ser interrumpidas a través de un evento. La sentencia de interrupción observa el tipo de cada evento y ejecuta alguna acción basada en la información del evento. Si la aplicación usa ventanas múltiples, el programa también determina en cual ventana ocurre un evento y toma una acción predeterminada.

Para aplicaciones de ventanas múltiples la especificación de interrupción puede llegar a ser larga y muy compleja. El Xtoolkit esconde estos procesos y proporciona al programador componentes que facilitan la detección de eventos, además de permitirle la simplificación de su trabajo. Todas las aplicaciones de Xtoolkits ejecutan los siguientes pasos básicos:

a) Inicialización de Intrínsecos. Este paso establece comunicación con el servidor X, asignando recursos e inicializando la capa de Intrínsecos.
b) Creación de widgets. Todos los programas crean uno o más widgets para construir un programa con interfaz al usuario.
c) Registro de llamadas y manejadores de eventos. Las llamadas y manejadores de eventos responden a acciones del usuario y eventos que ocurren dentro de cada widget.
d) Creación de todos los widgets. Manejo de un widget y creación de su respectiva ventana X.
e) Ejecución de eventos de lazo. Un evento de lazo se emplea para regresar indicaciones de un evento X al manejador de eventos apropiado dentro de una aplicación o bien llamar a la función asociada con el widget seleccionado.

4.2 HOOPS

Actualmente, las interfaces gráficas del usuario han tomado un gran auge. El modo texto utilizado 5 años atrás se convierte rápidamente en una forma de presentación obsoleta, la información exhibida gráficamente a través de zonas rectangulares llamadas ventanas, barras de desplazamiento, iconos y elementos gráficos, se observa en la mayoría de las aplicaciones actuales.

Debido a la diversidad de requerimientos por parte de los usuarios finales, han surgido en el mercado un conjunto de interfaces gráficas cuyas características varían dependiendo de la plataforma en la que se estén utilizando.

Por eso podemos encontrar la interfaz gráfica Windows de Microsoft para correr en PC's con DOS, la interfaz gráfica de Macintosh, la cual es considerada la pionera en cuanto a interfaces gráficas se refiere y que está diseñada para correr en máquinas Apple. Motif que corre en equipos UNIX y un sinnúmero de interfaces que son descritas con mas detalle en el capítulo correspondiente a interfaces gráficas.

Si tomamos en cuenta que para un desarrollador de software es muy importante que la aplicación que desarrolle cubra la mayor cantidad de mercado posible, tendría que desarrollar su aplicación para correr en cada una de las interfaces gráficas existentes, lo cual es realmente difícil, ya que para dominar cada una de las formas de programar en cada una de las diferentes interfaces se requiere de gran esfuerzo. Por eso, la selección de la interfaz gráfica en la que se va a desarrollar una aplicación cobra suma importancia.

Lógicamente, el desarrollador va a buscar aquella interfaz que ofrezca más opciones de transportación, es decir, que la aplicación pueda ejecutarse tanto en una PC como en un equipo UNIX, sin tener necesidad de corregir grandes cantidades de código.

Si tomamos en cuenta que el sistema operativo UNIX, ha alcanzado niveles de popularidad francamente inesperados, una interfaz gráfica capaz de correr en sistemas UNIX, podría ser una buena opción.

La interfaz gráfica para UNIX más popular es Motif; ésta fue desarrollada mediante un conjunto de funciones gráficas que ofrecen rutinas para manipular ventanas, crear y manipular objetos gráficos; estas funciones son conocidas como X-Windows.

La programación de aplicaciones mediante X-Windows no es algo que se aprenda con facilidad; lograr destreza en la programación con X-Windows requiere de gran esfuerzo y gran cantidad de libros leídos. Por ello se decidió crear otro conjunto de funciones basadas en las funciones ya existentes de X-Windows que simplificarán aún más la programación de aplicaciones con interfaz gráfica y que son conocidas con el nombre de HOOPS.

Se considera de suma importancia, hacer referencia a la librería gráfica HOOPS dentro de éste documento, ya que fue utilizado para el desarrollo de los programas UNAM-GISMO y UNAM-GISMA; por lo tanto, la necesidad de comprender dichos programas para la adaptación de multimedios y las correcciones hechas a los programas nos llevan al estudio y entendimiento de HOOPS.

Un ejemplo importante de la aplicación de HOOPS es el programa de diseño asistido por computadora, AUTOCAD, el cual actualmente goza de una posición comercial privilegiada.

A continuación se presenta una descripción breve de lo que es HOOPS, para qué sirve, sus ventajas y limitaciones.

HOOPS es un sistema para la creación de aplicaciones gráficas interactivas. Se trata de un conjunto de funciones prefabricadas y agrupadas para formar una librería, la cual puede ser incluida dentro de programas realizados en C ó Fortran, estas librerías se basan en las funciones de las librerías de XWindow pero a más alto nivel.

Debido a esto, un programa realizado con HOOPS se puede transportar a otra plataforma, ya que puede correr en cualquier máquina que posee el ambiente XWindow.

Para el almacenamiento de datos gráficos, HOOPS utiliza una base de datos, en la cual se guarda información acerca de cuáles objetos se deben dibujar, en qué posición deben ser dibujados y como deben ser sombreados.

La organización de la base de datos de HOOPS es jerárquica, por lo que se asemeja a una estructura de árbol característica de la organización de archivos.

Los datos almacenados son elementos gráficos, cámaras, luces, sombras y atributos de modelado, ademaás de información específica de cada aplicación.

Para una organización interna de los datos, HOOPS utiliza módulos de información llamados SEGMENTOS, que se caracterizan porque agrupan a varios elementos relacionados entre sí dentro de un solo objeto. Cada segmento puede a su vez, almacenar otros segmentos, lo que da como resultado la estructura de árbol mencionada anteriormente.

Toda la información almacenada en la base de datos se puede cambiar. La geometría se puede editar, los atributos se pueden modificar y la jerarquía se puede replantear. Después de realizar una serie de cambios dentro de la base de datos, se le puede decir al sistema que actualice el despliegue para reflejar en pantalla los cambios hechos a la base de datos.

Para un aprendizaje más rápido y eficiente de estas librerías es conveniente tener conocimientos previos de C ó FORTRAN.

A continuación se muestra un programa sencillo hecho con las librerías gráficas HOOPS, se puede observar que la única dificultad que existe es la de saber llamar a las funciones correctas, pasando los parámetros correspondientes a cada función; esto provoca que para un usuario que conozca un lenguaje de programación como C, sea relativamente sencillo aprender a desarrollar aplicaciones con HOOPS.

```
/* Despliegue de Hola Mundo */

main()
{
        HC_Open_Segment("?Picture");
                HC_Insert_Text (0.0, 0.0, 0.0, "Hola Mundo");
        HC_Close_Segment ();
        HC_Pause();
}
```

Es importante aclarar que cuando una sentencia empieza con HC, significa que se está haciendo una llamada a una función desde una aplicación en C y si la sentencia empieza con HF, se trata de una aplicación en Fortran.

4.2.1 Compilación y Ligado.

Después de haber capturado algún programa utilizando un editor de texto, es necesario compilarlo y ligarlo, para esto se debe ejecutar la siguiente instrucción en un Shell de UNIX:

```
% cc -o miprog.c -lhoops
```

También es importante declarar una variable de ambiente llamada HOOPS_PICTURE en la cual se va a indicar en que lugar físico (monitor) se va a realizar el despliegue de los gráficos.

4.2.2 Conceptos básicos de HOOPS

4.2.2.1 Segmentos.

Un segmento es la unidad fundamental de organización en la base de datos de HOOPS. Esta es una colección de atributos, geometrías y otros segmentos, localizados juntos para que su manipulación sea más sencilla.

4.2.2.2 Geometría.

La Geometría se refiere a las líneas, polígonos, texto y marcas, las cuales representan los bloques constructores del escenario gráfico.

4.2.2.3 Atributos de Sombreado.

Los atributos de sombreado incluyen toda la información perteneciente a la apariencia del objeto como lo son los colores, patrones de sombreado y tamaños. La geometría le dice al sistema qué crear, mientras los atributos le dicen cómo crearlo.

4.2.3 Construyendo un programa

La forma óptima para atacar un problema de graficación es dividirlo en pequeñas partes y después ir atendiendo a cada una de ellas, de otra forma, el problema podría llegar a ser tan grande como para salirse de nuestros alcances de entendimiento. De la misma forma, pueden atacarse los problemas con HOOPS, dividiendo un problema grande en segmentos y subsegmentos que mejoren la ejecución del programa además de facilitar las tareas de comprensión y mantenimiento de los sistemas, para ello se recomienda la creación de un menú, que por sí mismo separe las tareas en diferentes módulos.

4.2.4 Coordenadas mundiales y cámaras

La geometría de HOOPS se encuentra definida con el sistema de coordenadas cartesianas tridimensionales, a esto se le conoce como coordenadas mundiales. Es posible definir objetos en cualquier punto definido dentro de este espacio tridimensional, y por medio de la cámara proyectarlo sobre la pantalla. La cámara es el medio por el cual se proyecta un objeto de dos o tres dimensiones en un segmento de la pantalla.

4.2.5 Definición de un objeto

La definición de un objeto con HOOPS es sencilla, únicamente hay que especificar el número de vértices, y las coordenadas de los vértices que formarán al objeto.

4.2.6 Interacción con el usuario

Los programas hechos con HOOPS, permiten una alta interacción con el usuario final de la aplicación, ya que por medio del ratón se pueden hacer selecciones de las opciones mostradas en los menúes.

4.2.7 Ventanas Múltiples

En HOOPS es posible tener ventanas múltiples y en cada una de ellas exhibir el objeto desde diferentes puntos de vista, esto es una gran ayuda visual, porque permite apreciar la estructura de un objeto desde todos sus ángulos.

4.2.8 Iluminación

Para dar una apariencia más real a un objeto es necesario introducir luces para iluminarlo. Estas luces pueden ser de diversos colores y pueden ser proyectadas hacia el objeto desde diferentes puntos dentro del espacio cartesiano tridimensional.

En síntesis HOOPS es una herramienta ideal para el desarrollo de aplicaciones gráficas sin importar cuán complejas sean éstas. La facilidad de uso, la portabilidad a diferentes plataformas de trabajo, la interacción con el usuario y su enorme funcionalidad, hacen que HOOPS sea una excelente herramienta para desarrollar cualquier tipo de aplicación gráfica.

CAPITULO 5 PROGRAMAS UNAM-GISMA Y UNAM-GISMO

El objetivo de este capítulo es dar una orientación al usuario. La definición y descripción detallada de los comandos se pueden obtener directamente utilizando la opción de AYUDA del programa. La mejor forma para aprender a usar los programas UNAM-GISMO y UNAM-GISMA, no es leyendo este capítulo, sino usando los programas directamente.

5.1 DESCRIPCION GENERAL

Los programas UNAM-GISMA y UNAM-GISMO son las versiones UNIX en español de los programas originales CU-GISMA y CU-GISMO de la Universidad de Cornell. El número de módulos y comandos en los programas GISMA y GISMO de ambas Universidades son los mismos, con excepción de las utilerías de audio disponibles en los programas de la UNAM.

El programa UNAM-GISMO está formado por seis módulos principales: DEFINICION DEL PROBLEMA, CONSTRUCCION DE RIGIDECES, DEFINICION DE PARAMETROS, RESOLUCION, PRESENTACION DE RESULTADOS, y GUARDAR EL PROBLEMA EN LIBRERIA. El programa UNAM-GISMA solamente posee los módulos de DEFINICION DEL PROBLEMA, RESOLUCION y GUARDAR EL PROBLEMA EN LIBRERIA. Sus características se describen a continuación.

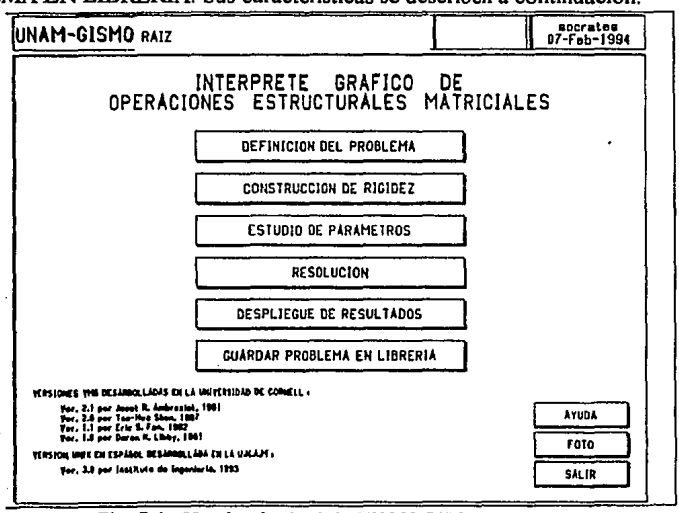

Fig. 5.1 Menú principal de UNAM-GISMO

El módulo de DEFINICION DEL PROBLEMA permite al usuario definir una estructura, ya sea construyendo gráficamente una nueva, o seleccionando una de las existentes dentro de la librería del programa. Una nueva estructura se puede crear con las instrucciones de preprocesamiento incluidas en el programa, pudiéndose modificar la geometría, propiedades de los elementos, condiciones de apoyo o fijación de los nodos, y las cargas. Una estructura guardada en la librería puede ser modificada gráficamente en cualquier momento.

Una vez definida una estructura, el usuario puede proceder a analizarla, empezando con el módulo de CONSTRUCCION DE RIGIDECES. Este módulo permite revisar los grados de libertad para comparar las matrices de rigidez locales de los diferentes elementos, y particularmente, ensamblar la matriz de rigidez global y el vector de cargas, manipulando los arreglos simbólicos gráfica e interactivamente. En ciertos casos especiales la matriz de rigidez global se puede ensamblar de manera automática mediante esta opción.

El módulo de DEFINICION DE PARAMETROS ofrece la opción de comprender la formación de la matriz de rigidez global. Al seleccionar un elemento, este se mostrará resaltado en la matriz de rigidez y viceversa. Otra opción permite al usuario reordenar la secuencia de numeración de los nodos para observar los cambios correspondientes en el ancho de banda de la matriz de rigidez global. Si se desea, se puede evitar la ejecución del módulo DEFINICION DE PARAMETROS ya que éste no es necesario en el procedimiento de análisis.

En esta etapa el módulo de RESOLUCION puede invocarse para calcular los desplazamientos globales de la estructura, las reacciones en los apoyos, y las fuerzas internas en los elementos. La presentación de las ecuaciones necesarias para calcular los desplazamientos y las fuerzas en los apoyos puede separarse del sistema completo de ecuaciones. Esta presentación simbólica de las ecuaciones, puede resolverse automáticamente. Las fuerzas en los elementos pueden determinarse ya sea mediante un procedimiento interactivo simbólico, o seleccionando la opción de realizar los cálculos automáticamente.

Una vez que se han encontrado todas las incógnitas, su presentación gráfica se puede activar mediante el módulo PRESENTACION DE RESULTADOS. En esta parte, el usuario puede revisar la geometría deformada de la estructura, los diagramas de fuerzas axiales, cortantes, y momentos flexionantes de todos los elementos. El factor de escala de estos dibujos puede modificarse de forma interactiva. Si se desea, se puede optar por la presentación numérica de desplazamientos, reacciones, y fuerzas en los elementos deseados. Adicionalmente, existe una opción para generar una impresión de la salida completa de resultados.

La última opción, GUARDA PROBLEMA EN LIBRERIA, se puede activar una vez que la estructura ha sido definida. Este módulo, independiente del análisis, permite el manejo de los archivos en la librería del usuario. El usuario puede renombrar la estructura, guardarla con el mismo nombre, o borrar estructuras que ya existen en la librería.

5.1.1 Formato de las páginas del menú

Los formatos de las páginas de los menúes existentes en los seis módulos son básicamente los mismos. El formato básico de la página del menú se muestra en la Fig 5.2. Esta página está dividida en las áreas descritas a continuación, cada una de las cuales contiene texto y objetos diferentes, según sea su uso:

1) El área del título muestra los nombres del programa y la página actual del menú.

2) El área del nombre del archivo muestra el nombre actual de la estructura.

3) El área de proceso muestra el nombre del usuario y la fecha.

4) El área principal de presentación se usa para mostrar la estructura, las matrices, el texto de AYUDA, y otros objetos. Esta área a veces se subdivide en subáreas.

5) El área inferior muestra diferentes mensajes e información, que puede dar al usuario instrucciones en todo momento durante la ejecución del programa.

6) El área de menú muestra los comandos que se encuentran disponibles en la página que muestra la pantalla del menú.

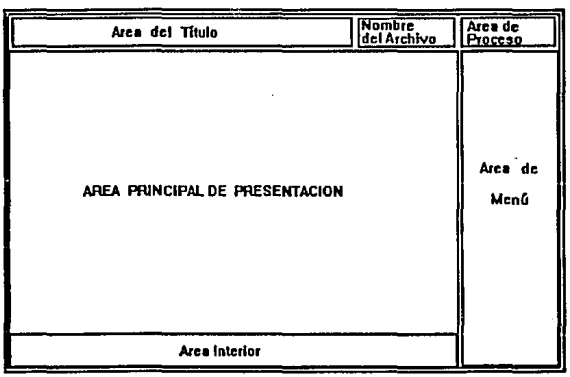

Fig. 5.2 Formato de las páginas de menú de UNAM-GISMO

Los comandos mostrados en el área de menú pueden ser categorizados en tres tipos: comandos de variables, comandos de manipulación y comandos comunes. Las instrucciones de variables son específicas para la funciones de una página de un menú particular, por lo

que existe una diferente para cada página. Los comandos de manipulación incluyen ROT Z, PAN, ZOOM, AUMENTA, REINICIA y LLENA, que aparecen en algunas páginas de menú y son usadas para manipular la vista mostrada de la estructura. Los comandos comunes incluyen AYUDA, FOTO, RAIZ o REGRESA, y SALIR, aparecen en cada página de menú.

5.1.2 Mecanismos de entrada y control

Durante la ejecución de GISMO existen tres modos de entrada:

1) Uso del ratón (como: apuntar, seleccionar, invocar, etc). Esta operación consiste en mover el cursor con el ratón hasta cierta localización y presionar alguno de los botones del mismo. Para controlar el flujo del programa, la localización puede ser un comando de un menú. Alternativamente, para identificación o entrada, la localización puede ser un área en la pantalla, una entidad de la estructura o del teclado mostrado en pantalla.

2) La información alfabética, como nombres de archivos, puede introducirse desde el teclado.

3) Los datos numéricos pueden ser introducidos desde el teclado. También se pueden introducir apuntándoles al teclado mostrado en la pantalla.

5.1.3 Comandos de manipulación

En algunas páginas en los módulos de DEFINICION DEL PROBLEMA y PRESENTACION DE RESULTADOS, los comandos de manipulación permiten al usuario cambiar la presentación de la vista de la estructura. Estos comandos incluyen ROT Z, PAN, ZOOM, AUMENTA, REGRESA y LLENA.

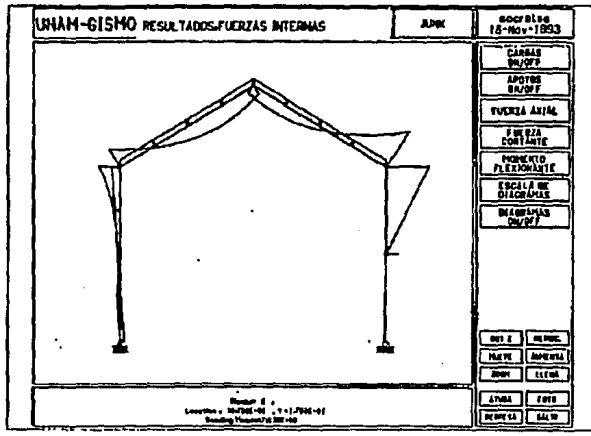

Fig. 5.3.1 Comando Aumenta

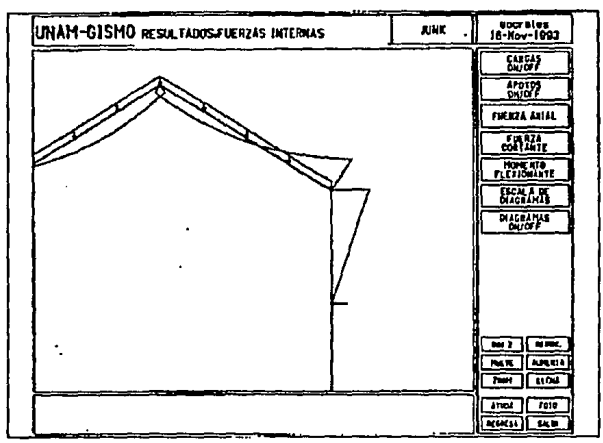

Fig. 5.3.2 Comando Aumenta

El comando ROT Z permite girar la estructura alrededor del eje z, que es normal a la pantalla. El área principal de la pantalla se usa como un potenciómetro que controla linealmente la magnitud de la rotación. La estructura rota en sentido antihorario si el ratón está en la mitad derecha del área mostrada, y horario si está en la mitad izquierda.

El comando PAN permite mover la estructura dentro del área principal. El área principal se usa como un potenciómetro que controla linealmente la magnitud de la traslación. La

dirección de la traslación es la misma que la localización dada con el ratón, relativa al centro del área.

El comando ZOOM permite reducir o agrandar el tamaño de la estructura mostrada. El área principal se usa como un potenciómetro que controla linealmente la magnitud del zoom. El tamaño mostrado crece si el ratón se instala en el lado derecho del área disponible, y se reduce si se instala en el lado izquierdo.

El comando AUMENTA permite determinar un rectángulo con una mayor área disponible, para ser amplificada. El usuario determina esta área rectangular, al seleccionar cualquiera de dos esquinas opuestas, Figs 5.3.1 y 5.3.2.

El comando REGRESA permite recuperar la orientación, posición, y el tamaño originales de la estructura.

El comando LLENA permite iniciar el contenido en la mayor área que esta disponible en el tamaño de la pantalla con una manipulación diversa de comandos a lo largo de los botones de la pantalla, Fig 5.4.

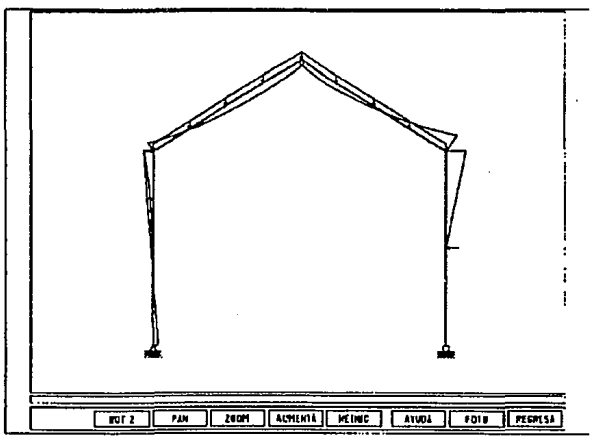

Fig. 5.4 La Página llena

5.1.4 Comandos comunes

En el programa, siempre aparecen cuatro comandos en cada página sobre el menú. Estos son llamados comandos usuales, y son los siguientes: AYUDA, FOTO, REGRESA (o en algunos casos RAIZ), y SALIR.

El comando AYUDA permite proporcionar en la pantalla, texto de ayuda mediante la descripción del uso de todos los comandos disponibles en cada una de las páginas de menú. El usuario puede seleccionar este comando para pedir el texto de AYUDA, y entonces elegir cualquier comando o área disponible para leer el texto de ayuda correspondiente. Presionando nuevamente el comando AYUDA, se sale de este menú.

El comando FOTO permite enviar una copia completa de lo mostrado en la pantalla a una impresora láser. El usuario deberá teclear este comando dos veces para confirmar su deseo de hacer una copia completa.

Debido al alto costo involucrado y al tiempo involucrado en la generación de copias completas de la pantalla, se recomienda al usuario que emplee este comando sólo cuando sea necesario.

El comando REGRESA permite el control directo de respaldos, anterior a la página del menú, lo cual es un nivel arriba, durante el comando RAIZ, que permite volver a la ejecución del programa para el control de página de la RAIZ. Uno de estos dos comandos debe aparecer en la pantalla para proporcionar formas de continuar la operación del programa al ejecutarlo.

El comando SALIR permite al usuario aceptar la salida de el programa. Para el uso de este comando, el usuario tecleará dos veces para confirmar su intento.

5.2 ALCANCES Y LIMITACIONES

El programa GISMO tiene los siguientes alcances y limitaciones:

1) Se pueden manejar vigas, armaduras y marcos planos.

2) Se pueden manejar estructuras de hasta 30 grados de libertad. Esta limitación es equivalente a un máximo de 15 nodos en armaduras o vigas, o 10 para marcos. Esta limitante se debe a la imposibilidad de presentar claramente en pantalla matrices de orden superior a los 30 grados de libertad.

3) El análisis está limitado a un caso de carga incluyendo combinaciones de cargas en nodos, y en elementos. Las cargas distribuidas aplicadas a los elementos de las armaduras, son convertidas a cargas equivalentes en los nodos.

4) El análisis es elástico, y se suponen deformaciones pequeñas en las secciones. Todos los elementos se idealizan como una línea recta con las mismas propiedades en toda su longitud. Todos los nodos se representan por puntos, los de las armaduras se suponen como articulaciones sin fricción ni rigidez rotacional.

5.3 OPERACION DE LOS PROGRAMAS

La forma de operación de los programas GISMO y GISMA se muestra en la Fig 5.5. Después de que se ha elegido un color, se pasa a la página de control PRINCIPAL, Fig 5.6. En ella se muestran los seis módulos del programa. La página INTRODUCCION se obtiene presionando el ratón en la opción AYUDA, Fig 5.7.

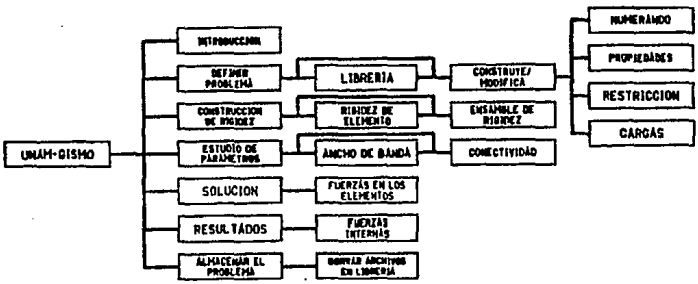

Fig. 5.5 Estructura de operación de UNAM-GISMO

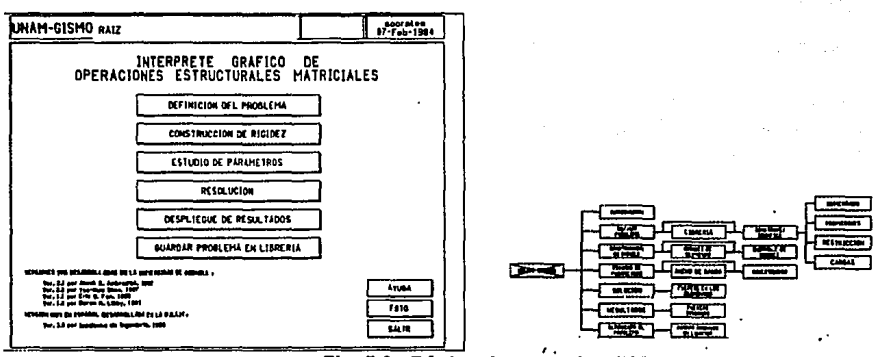

Fig. 5.6 Página de Control PRINCIPAL

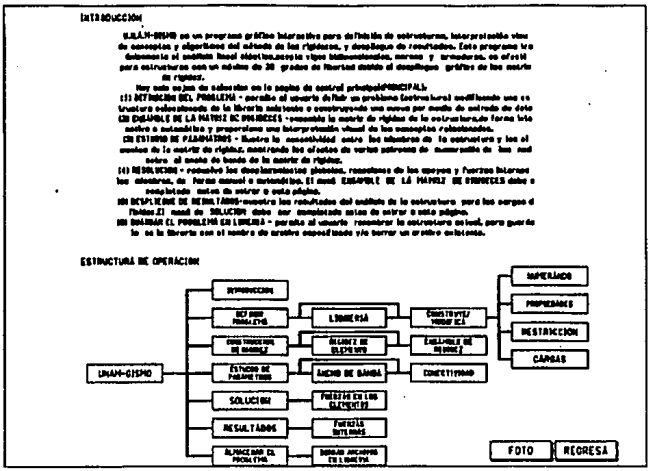

Fig. 5.7 La página INTRODUCCION

5.3.1 Comando DEFINICION DE PROBLEMA - CONSTRUYE/ MODIFICA

El primer paso para la utilización de los programas GISMO o GISMA consiste en crear una estructura, ya sea construyéndola desde el principio, o leyendo alguna de las ya existentes en la librería. Si se desea modificar alguna, se puede utilizar la opción DEFINICION DEL PROBLEMA y el submenú CONSTRUYE.

Los programas GISMO y GISMA suponen que todas las unidades en las que se definen los parámetros de cada estructura son consistentes.

Para crear una nueva estructura se invoca el comando CONSTRUYE/MODIFICA en la página de DEFINICION, Fig 5.8, para obtener una serie de opciones de modificación.

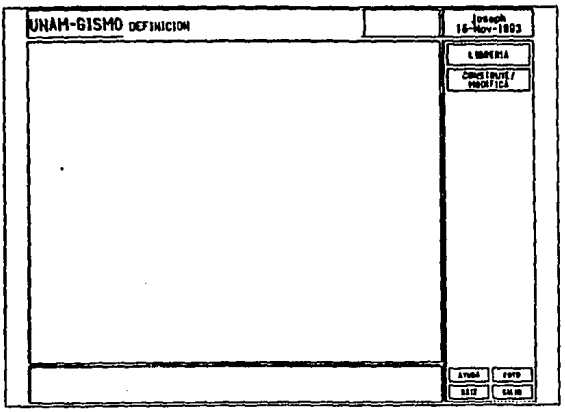

Fig 5.8 Selección de definición de una estructura

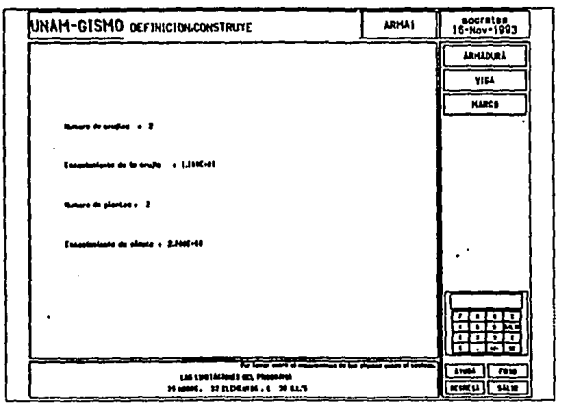

Fig. 5.9 Entrada de datos

5.3.1.1 Tipo de estructura, escala, y malla.

Una vez que se elige un tipo de estructura (viga, armadura, o marco, como en la Fig 5.9), se debe introducir el nombre de la misma desde el teclado, y definir el tamaño y la escala de la misma. En la definición del tamaño y la escala se definen el número de crujías y pisos. Los números creados con la pantalla se introducen al sistema presionando la tecla OK. Con toda esta información se generan los nodos y los elementos de la estructura.

Para terminar con el modo de introducción de datos con el ratón desde la pantalla, se presiona la opción SALIDA.

5.3.1.2 Modificaciones de la geometría.

Terminada la introducción de datos de una estructura, aparece la página CONSTRUYE, Fig 5.10. Esta página ofrece las siguientes opciones:

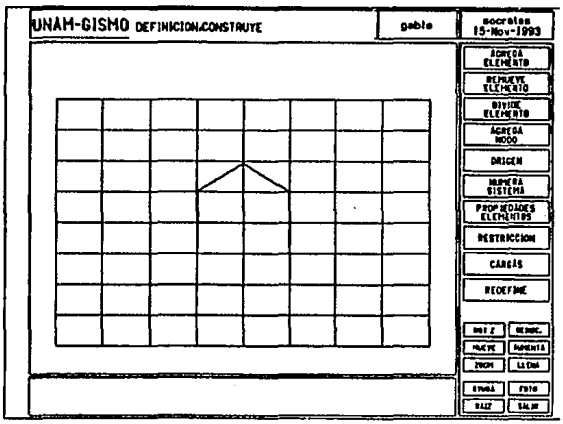

Fig. 5.10 Resultado malla fondo y malla estructural

- AGREGA ELEMENTO, permite incluir nuevos elementos presionando con el ratón donde se ubicarán los nodos del elemento. Si no se especifica el tipo de propiedad, a los nuevos elementos se les asigna la propiedad tipo 1.

- REMUEVE ELEMENTO, remueve elementos de la estructura modificada son renumerados automáticamente.

- DIVIDE ELEMENTO, puede seccionar un elemento en dos o tres partes de igual longitud, dependiendo de la instrucción activada (2 ó 3). Los nuevos elementos y nodos se numeran automáticamente. La propiedad por omisión asignada a los nuevos elementos es la número 1.

- AGREGA NODO, permite crear nuevos nodos. Puede activarse de dos formas; utilizando el comando MALLA o mediante PANTALLA. En la primera se utiliza el ratón, presionando sobre un punto de intersección de la malla se creará un nuevo nodo. Con la segunda opción se deben teclear las coordenadas de los nodos.

- ORIGEN permite cambiar la posición del origen del sistema global de coordenadas.

- REDEFINE borra la estructura actual y permite crear una nueva.

5.3.1.3 Comando NUMERA SISTEMA

El comando NUMERA SISTEMA lleva al menú de la Fig 5.11, donde se pueden ver los números de nodos presionando NUMEROS DE NODOS, los de elementos presionando NUMEROS DE ELEMENTOS, y el sistema de coordenadas con COORDENADAS DE NODOS. En este menú se puede redefinir la numeración de los nodos.

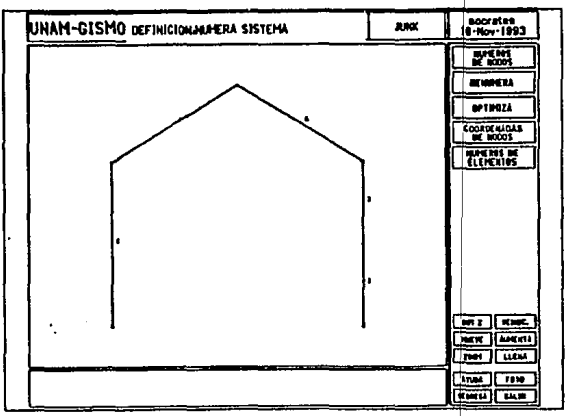

Fig 5.11 Numera Sistema

El comando RENUMERA permite cambiar la numeración de los nodos.

El comando OPTIMIZA se emplea para reducir el ancho de banda de la matriz de rigidez.

5.3.1.4 Comandos PROPIEDADES DE ELEMENTOS y TABLA DE PROPIEDADES

La elección del menú de PROPIEDADES, lleva al mismo submenú, Fig. 6.8, en él se pueden asignar las propiedades de los elementos E, A, e I.

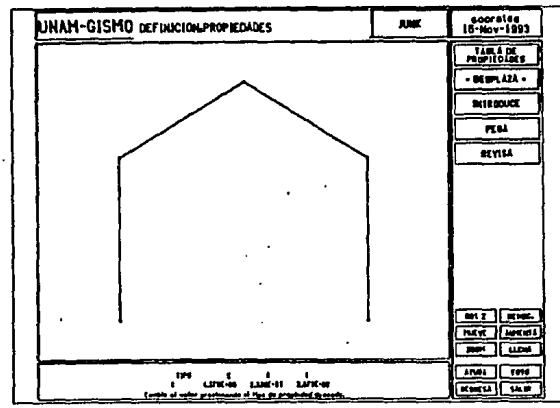

Fig. 5.12 Hoja de propiedades

El comando TECLEA, crea un nuevo tipo de propiedad copiando los valores la propiedad actual. Este nuevo tipo se muestra en el área inferior de la pantalla.

El comando DESPLAZA, permite ver los tipos de propiedades existentes.

El usuario puede revisar las propiedades de un elemento con el comando REVISA.

El tipo de propiedad mostrado puede ser asignado a un elemento con el comando PEGA.
El comando TABLA DE PROPIEDADES, permite el manejo de las propiedades de los elementos, Fig 5.13.

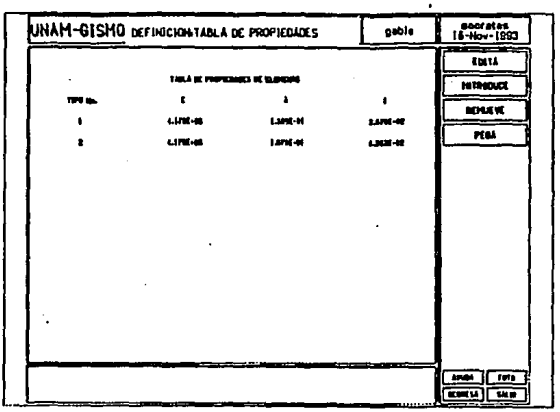

Fig. 5.13 Tabla de propiedades

5.3.1.5 Comando GRADO DE RESTRICCION.

El comando GRADO DE RESTRICCION lleva al menú mostrado en la Fig 5.14, en el cual es posible asignar las condiciones de frontera de la estructura. Cada grado de libertad puede definirse como LIBRE o FIJO. El grado de libertad por omisión es libre.

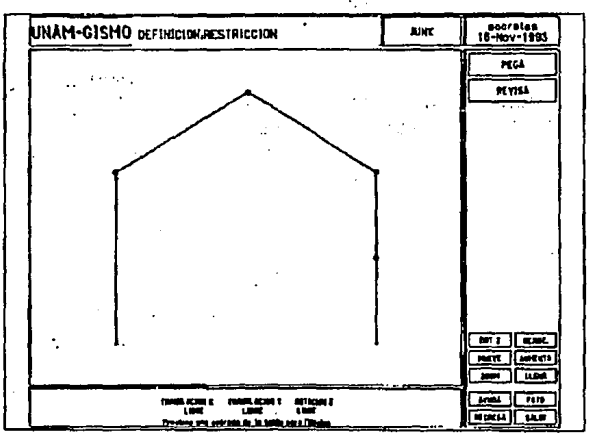

Fig. 5.14 Hoja de restricción

5.3.1.6 Comandos CARGAS y TABLA DE CARGAS.

Estos comandos llevan al menú de la Fig 5.15, con él se pueden asignar, revisar, y borrar diferentes tipos de carga para los nodos y elementos. Para poder ver los tipos de cargas existentes se utiliza el comando DESPLAZA. Los tipos de carga pueden ser: cargas concentradas, uniformemente distribuidas y cargas lineales.

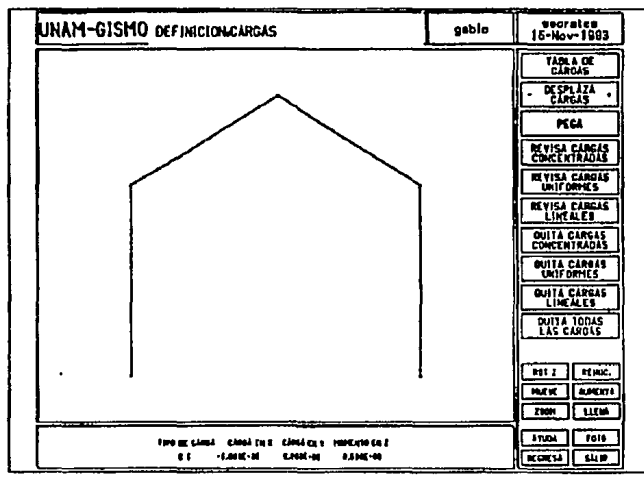

Fig. 5.15 Hoja de cargas

Con el comando PEGA se pueden aplicar las cargas a los nodos o elementos deseados.

Los comandos REVISA CARGAS CONCENTRADAS, REVISA CARGAS UNIFORMES, y REVISA CARGAS LINEALES, permiten revisar las cargas aplicadas a los nodos o elementos.

Similarmente existen tres comandos para eliminar cargas en los nodos y elementos. Estos son QUITA CARGAS CONCENTRADAS, QUITA CARGAS UNIFORMES, y QUITA CARGAS LINEALES. Finalmente el comando DESCARGA TODO, permite borrar todas las cargas de la estructura. Este comando necesita confirmarse oprimiendo dos veces el ratón. Los tipos de carga existentes se pueden modificar, o agregar nuevos con el menú TABLA DE CARGAS, como se muestra en la Fig 5.16.

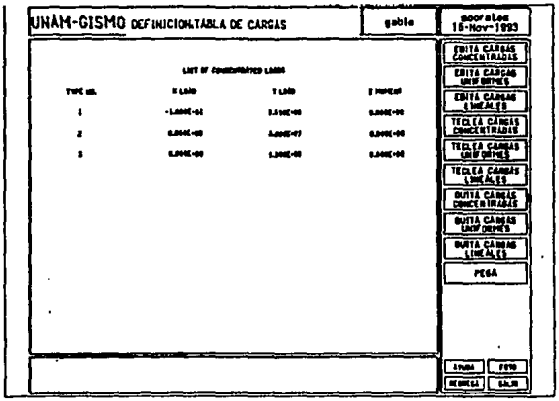

Fig. 5.16 Tabla de distribución uniforme de carga

5.3.2 Comando CONSTRUCCION DE RIGIDECES.

Una vez definida completamente la estructura, se procede a ensamblar la matriz de rigidez global y el vector de cargas con el menú CONSTRUYE RIGIDECES, Fig 5.17.

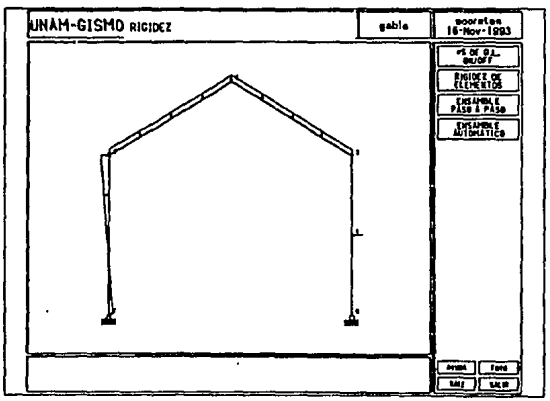

Fig 5.17 Hoja de rigidez

El comando NUMEROS DE GDL muestra la figura de la estructura indicando sus grados de libertad.

El comando RIGIDEZ DE ELEMENTO lleva al menú ELEMENTO, el cuál permite formar la matriz de rigidez local de los diferentes elementos estructurales.

El comando ENSAMBLE PASO A PASO lleva al menú ENSAMBLE, que permite ensamblar la matriz de rigidez global y el vector de cargas paso a paso.

El comando ENSAMBLE AUTOMATICO realiza el ensamble de la matriz de rigidez y el vector de cargas en un solo paso.

5.3.2.1 Comando RIGIDEZ DE ELEMENTO.

El menú ELEMENTO muestra la pantalla dividida como se presenta en la Fig 5.18. En ella se puede seleccionar un elemento de la estructura presionando el ratón, a éste se le denomina "elemento activo". En otra parte de la pantalla se muestra la matriz de rigidez local de este elemento.

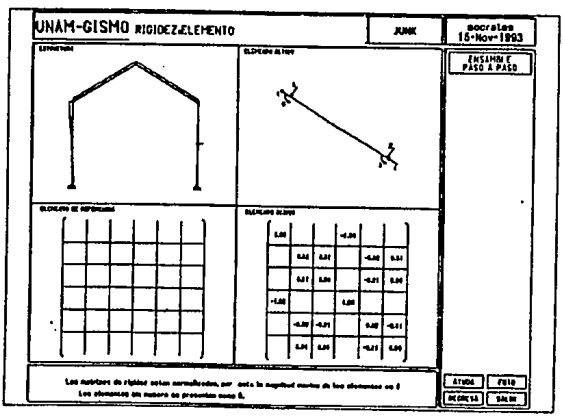

Fig. 5.18 Comparación de matriz de rigidez de elemento

5.3.2.2 Comando ENSAMBLE PASO A PASO.

El despliegue correspondiente a los comandos ENSAMBLE está formado por cuatro ventanas, Fig 5.19.

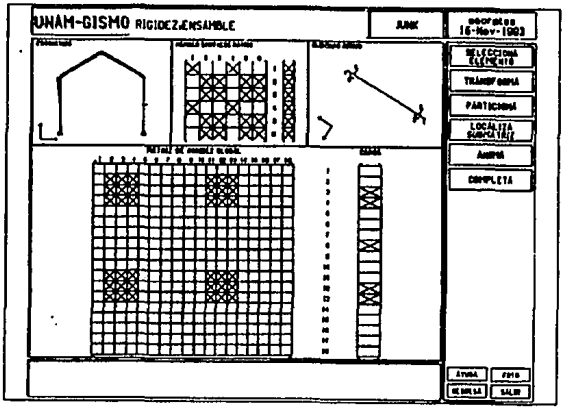

Fig. 5.19 Ensamble matriz de rigidez global

El comando SELECCIONA ELEMENTOS permite seleccionar un elemento para ensamblar su matriz de rigidez local en la matriz global de la estructura. Al elemento seleccionado se le aplica el comando TRANSFORMA, para transformar sus matrices locales a coordenadas globales.

Después que la transformación se ha compietado el usuario debe dividir la matriz del elemento en submatrices con el comando PARTICION.

El comando ANIMA muestra la manipulación matricial requerida para ensamblar completamente la matriz local del elemento seleccionado. El comando COMPLETA, realiza automáticamente la formulación de la matriz global de rigidez y del vector de cargas.

5.3.3 Comando ESTUDIO DE PARAMETROS.

Bajo el comando ESTUDIO DE PARAMETROS se tiene acceso a las opciones INCIDENCIAS, o ANCHO DE BANDA, Fig 5.20.

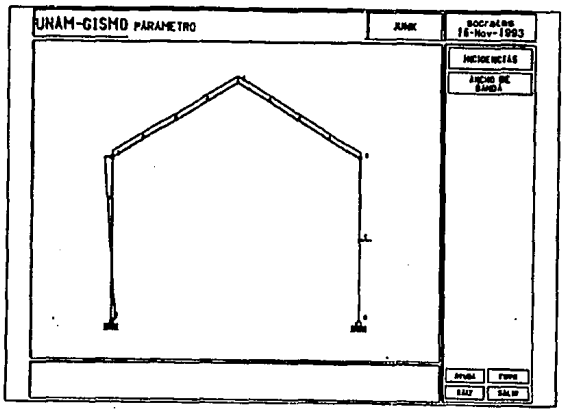

Fig. 5.20 Hoja de parámetros

5.3.3.1 Comando INCIDENCIAS.

El comando INCIDENCIAS lleva al menú mostrado en la Fig 5.21, en el que los grados de libertad y la matriz global de rigidez pueden ser estudiados.

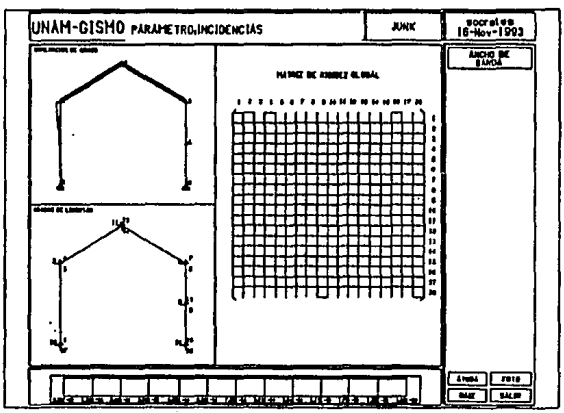

Fig. 5.21 Conectividades estructurales

5.3.3.2 Comando ANCHO DE BANDA.

Este comando lleva al menú mostrado en la Fig 5.22.

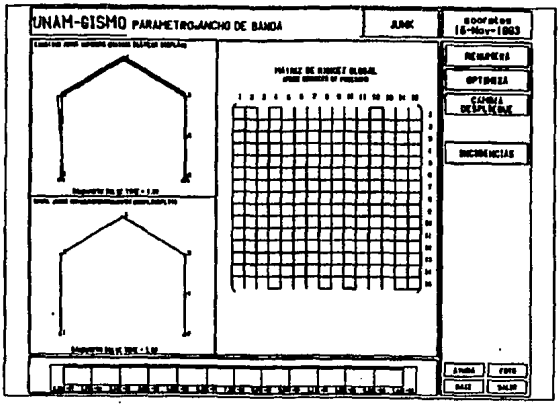

Fig 5.22 Matriz de rigidez y ancho de banda

El comando RENUMERA permite asignar nuevos números a los nodos. La instrucción OPTIMIZA renumera automáticamente los nodos, de acuerdo con los resultados de un algoritmo de reducción del ancho de banda de la matriz.

5.3.4 Comando RESOLUCION.

El comando RESOLUCION muestra un menú como el de la Fig 5.23.

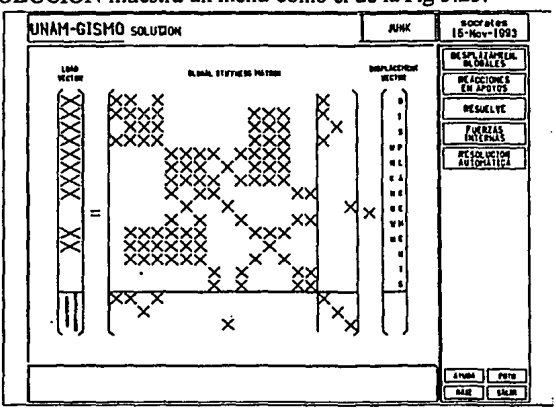

Fig. 5.23 Ecuaciones de rigidez global

5.3.4.1 Solución para variables globales.

El comando DESPLAZAMIENTOS GLOBALES muestra las ecuaciones requeridas para calcular los desplazamientos en coordenadas globales. El comando REACCIONES EN APOYOS, muestra las fuerzas en los nodos restringidos de la estructura.

El comando RESUELVE soluciona el sistema de ecuaciones.

El comando FUERZAS INTERNAS lleva al menú ELEMENTOS.

El comando SOLUCION AUTOMATICA determina automáticamente los desplazamientos globales, reacciones en apoyos y fuerzas en los elementos.

5.3.4.2 Comando FUERZAS INTERNAS.

Este menú correspondiente se muestra en la Fig 5.24. Para utilizar el comando se selecciona un elemento con el comando SELECCIONA ELEMENTO, del que se obtienen sus fuerzas internas que se muestran con diferente color.

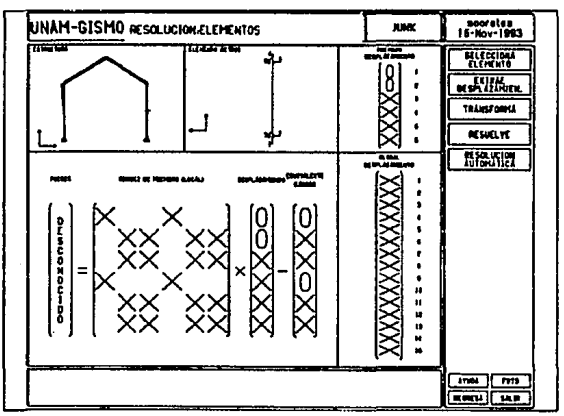

Fig. 5.24 Determinación de fuerzas en elementos

El comando CALCULA DESPLAZAMIENTOS permite obtener los desplazamientos en coordenadas locales. Para transformarlos se debe utilizar el comando TRANSFORMA.

Con el comando RESUELVE se obtienen las fuerzas internas de los elementos.

El comando SOLUCION AUTOMATICA automáticamente obtiene las fuerzas de todos los elementos.

5.3.5 Comando DESPLIEGUE DE RESULTADOS.

Este menú es capaz de mostrar gráficamente los resultados obtenidos del análisis.

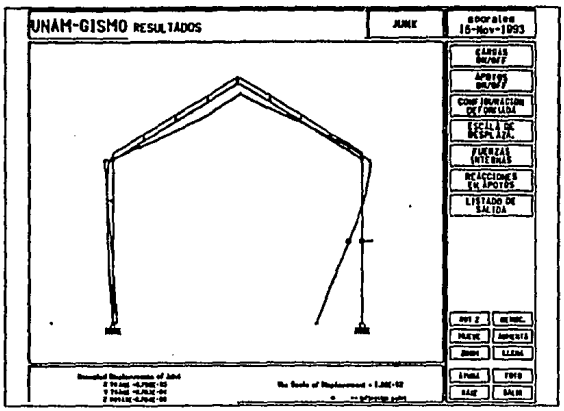

Fig. 5.25 Configuración deformada

5.3.5.1 Comando CONFIGURACION DEFORMADA.

El comando CONFIGURACION DEFORMADA, permite observar la deformación de la estructura. Dentro de este comando se pueden utilizar las opciones CARGAS ON/OFF, y APOYOS ON/OFF, Fig 5.25.

5.3.5.2 Comando FUERZAS INTERNAS.

Este comando permite observar los diagramas de fuerzas de los elementos, Fig 5.26.

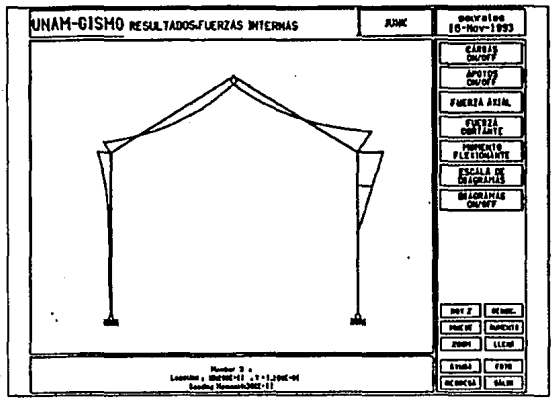

Fig. 5.26 Diagrama de momento flexionante

El comando FUERZA AXIAL muestra los diagramas de tensión o compresión en dos colores diferentes.

Para armaduras existe la opción INTENSIDAD DE FUERZA AXIAL.

Los comandos FUERZA CORTANTE y MOMENTO FLEXIONANTE muestra los diagramas de cortantes y momentos de los elementos.

Si se desea cambiar el tamaño de los diagramas se utiliza el comando ESCALA DIAGRAMA. El comando DIAGRAMAS ON/OFF, permite seleccionar dos comandos UNO o TODOS, para modificar el tamaño de los diagramas de los elementos seleccionados.

Los comandos CARGAS ON/OFF y APOYOS ON/OFF muestran las cargas y reacciones en la pantalla.

5.3.5.3 Comando REACCIONES EN LOS APOYOS.

El comando REACCIONES EN LOS APOYOS muestra las fuerzas en los apoyos mediante flechas. Su valor numérico se puede obtener presionando el ratón sobre el apoyo deseado.

5.3.5.4 Comando LISTADO DE SALIDA.

La impresión de los resultados del análisis se obtiene invocando el comando LISTADO DE SALIDA.

5.3.6 Comando GUARDAR PROBLEMA EN LIBRERIA.

El comando GUARDA PROBLEMA EN LIBRERIA permite guardar archivos en la librería, Fig 5.27.

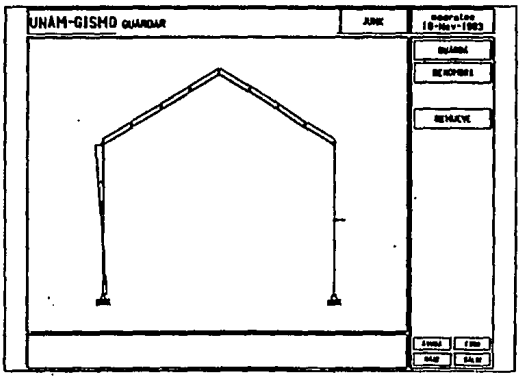

Fig. 5.27 Manipulación de archivos de datos

5.3.6.1 Comando GUARDA Y RENOMBRA.

El comando GUARDA permite almacenar en la biblioteca la estructura deseada.

El comando RENOMBRA permite cambiar el nombre de una estructura.

5.3.6.2 Comando REMOVER.

Este comando elimina archivos de la biblioteca, Fig. 5.28. Con él se pueden activar dos comandos DIAGRAMA y DIRECTORIO que muestran las figuras y los nombres de las estructuras respectivamente.

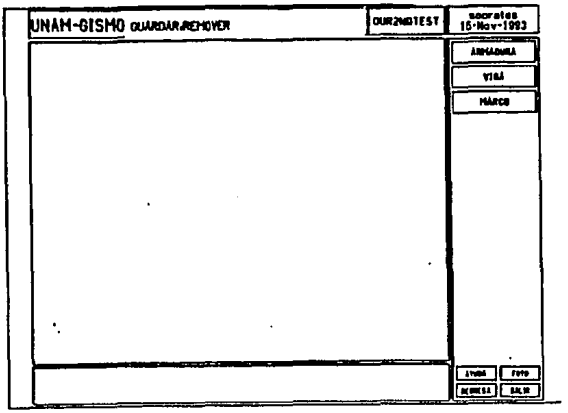

Fig. 5.28 Hoja para remover

5.3.7 Comando DEFINICION DEL PROBLEMA - LIBRERIA.

La selección de librería incluye dos comandos DIAGRAMA y DIRECTORIO, Fig 5.29

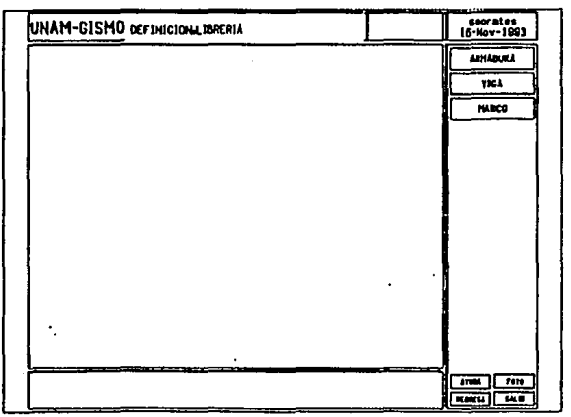

Fig. 5.29 Selección del modo para el tipo de estructura

5.3.7.1 Comando DIAGRAMA.

El comando DIAGRAMA, muestra las estructuras almacenadas en la librería, Fig 5.30.

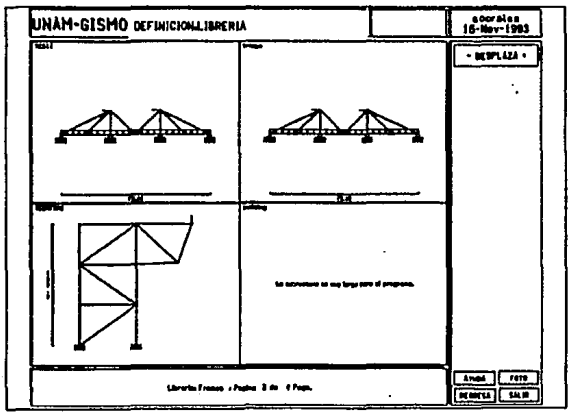

Fig. 5.30 Selección de una estructura por su diagrama

5.3.7.2 Comando DIRECTORIO.

El comando DIRECTORIO permite ver los nombres de las estructuras almacenadas en la librería, Fig 5.31.

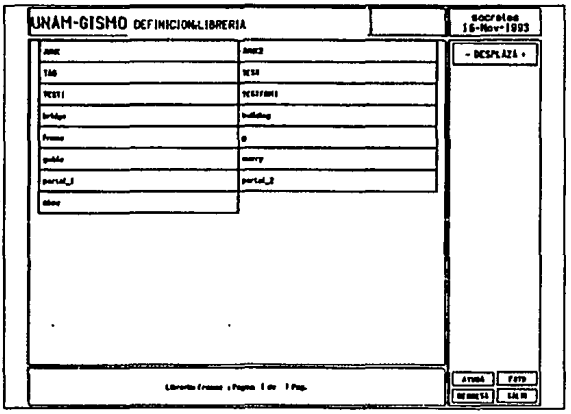

Fig. 5.31 Selección de una estructura por su nombre

5.3.7.3 Comando Selección.

Seleccionando DIAGRAMA o DIRECTORIO, se puede ver la estructura almacenada en la opción DEFINICION DEL PROBLEMA, Fig 5.32.

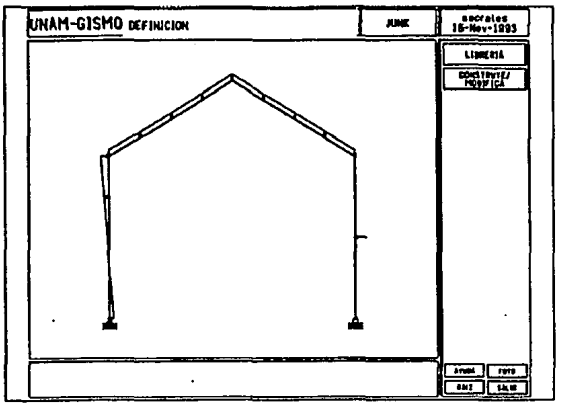

Fig. 5.32 Elección de una estructura

CAPITULO 6 ADECUACION DE LOS PROGRAMAS GISMA Y GISMO

6.1 ANTECEDENTES

Inicialmente los programas GISMA y GISMO se encontraban instalados en una estación de trabajo VAXStation II de DIGITAL con sistema operativo VMS. En esta máquina se comenzó a conocer y estudiar el funcionamiento del los programas, las librerías de HOOPS y el compilador Fortran 77.

VMS es un sistema operativo propio de la compañía DIGITAL, por lo cuál sólo corre en máquinas DIGITAL. Lo anterior traía como consecuencia limitaciones en cuanto a la portabilidad de la aplicación. Para resolver este problema se decidió migrar la aplicación a plataformas UNIX y continuar ahí con el desarrollo de los programas. UNIX es la plataforma más popular para estaciones de trabajo y posee un gran número de herramientas de comunicación.

La plataforma VMS usada en un principio se utilizó exclusivamente para analizar el funcionamiento de los programas. Este análisis incluyó la estructura de programas, las llamadas al sistema operativo, el estudio de Fortran 77 y el uso de las librerías HOOPS.

6.2 CICLO DE VIDA DEL SISTEMA

6.2.1 Requerimientos

Cuando se recibieron los programas GISMO y GISMA en la Facultad de Ingeniería, surgió la necesidad de adaptarlos a una serie de requerimientos planteados para lograr su operación de acuerdo con las necesidades de la situación. Entre los requerimientos más importantes que se plantearon están los siguientes:

- **Adaptar el sistema al español.**

- **Adaptar el sistema a los recursos con los que cuenta la Facultad de Ingeniería**

- **Agregar una aportación original al sistema de aprendizaje con el objetivo de mejorar la forma de transmisión de conocimientos a través de la computadora.**

6.2.2 Análisis

Adaptar el sistema al español.
Es muy cierto que en la actualidad la mayoría de los estudiantes de ingeniería tienen conocimientos del idioma inglés, pero también es muy cierto que el inglés no es nuestra lengua materna y por lo tanto es un obstáculo para que el proceso de aprendizaje se dé de una manera natural y sencilla. El tener que aprender nuevos conceptos en un idioma distinto al nuestro crea barreras en el proceso de aprendizaje.

Para este punto se realizó un análisis sobre los programas fuentes para conocer aquellas funciones involucradas con el despliegue de información en pantalla; como: mensajes de ayuda, mensajes de error, texto de botones y texto de menúes. Estas funciones de despliegue son funciones HOOPS, por lo que se requirió del estudio de ellas para conocer su comportamiento y saber en que parte deberían de ser modificadas. Al momento del análisis se observó que el texto de ayuda y muchos de los conceptos que se manejaban era muy orientados hacia el área de estructuras por lo que se solicitó ayuda de personal especializado en el área de estructuras para la traducción de los programas. Una vez que un programa había sido traducido, se ligaba a los demás para probar que la funcionalidad de éste no había sido modificada por algún error de teclado. Los programas GISMO y GISMA fueron traducidos al español en su totalidad.

Adaptar el sistema a los recursos con los que cuenta la Facultad de Ingeniería.
Para la ejecución de este sistema se necesitaba una estación de trabajo; el uso de PC's fué descartado debido a que los cálculos que se realizan durante la ejecución de los programas GISMO y GISMA son muchos, por lo que la capacidad de procesamiento de datos en una PC sería insuficiente. Lo anterior provocaría que las respuestas del sistema fueran muy lentas y se perdería la interactividad que es uno de los puntos más importantes de cualquier sistema de aprendizaje. La Facultad de Ingeniería cuenta con estaciones de trabajo de distintos proveedores y había que hacer la selección del equipo en el que se instalaría la aplicación. Tomando en cuenta que el sistema UNIX es un sistema muy abierto y muy popular en el mundo y que la mayoría de estaciones de trabajo dentro del la UNAM cuentan con sistema operativo UNIX, se decidió portar la aplicación de un ambiente VMS a un ambiente UNIX. Lo anterior requirió de un análisis muy fuerte de ambos sistemas operativos, principalmente para descubrir las diferencias entre ellos. Tomando en cuenta que la aplicación había sido desarrollada en HOOPS y que el HOOPS para VMS y UNIX es exactamente el mismo, sólo se tuvieron que considerar aquellas funciones que tenían acceso a comandos propios del sistema operativo. Las principales diferencias entre ambos sistemas operativos que afectaban directamente la correcta operación de la aplicación fueron el manejo de archivos, las llamadas a procesos, intercambio de información entre procesos y las funciones para la obtención de fecha, hora y nombre del usuario. Se decidió utilizar una plataforma SUN por ser el equipo con el que se contaba, además de ser muy popular dentro de la UNAM y barato en comparación de otras marcas. Los programas GISMO y GISMA, para ser ligados correctamente, requiere de librerías especiales

dependiendo del sistema operativo y de la plataforma en la que se desee correr, por lo cuál fué necesario obtener las librerías HOOPS propias para la plataforma escogida.

Agregar una aportación original al sistema de aprendizaje con el objetivo de mejorar la forma de transmisión de conocimientos a través de la computadora.
Idear una adaptación original que agregará valor a la aplicación ya existente. Formas más eficientes de transmisión de conocimientos a través de la utilización de multimedios. Se analizaron los recursos con los que cuenta la plataforma SUN para creación de aplicaciones con multimedios. Actualmente las mayoría de las estaciones de trabajo de distintos proveedores brindan muchas facilidades para crear aplicaciones con multimedios. Decidimos que la inclusión de audio dentro de la aplicación existente permitiría al usuario final aprender de manera más eficiente la teoría en forma auditiva. El usuario final puede estar practicando en la creación de una nueva estructura mientras escucha audio referente a los conceptos más importantes en la teoría de estructuras. Por lo que se decidió incluir un nuevo botón dentro de la aplicación original que tendría dos estados: encendido y apagado. Cuando este botón estuviera encendido, además de tener una ayuda desplegada en pantalla acerca del tópico de interés, se prodría tener ayuda en forma de voz describiendo en más detalle el concepto. Esto implicaría analizar el programa fuente para saber en qué punto de éste se podrían incluir lo programas para la ejecución de los archivos de voz.

6.2.3 Desarrollo

En el mercado existen una gran variedad de proveedores de hardware que ofrecen máquinas con sistema operativo UNIX. En nuestro caso particular decidimos portar la aplicación a plataforma UNIX de SUN, ya que este équipo es muy popular dentro de la UNAM y gran número de dependencias cuentan con el, además se requirió adquirir una serie de librerías gráficas para UNIX; una vez tomada esta decisión se comenzó con la adecuación, traducción e inclusión de nuevas herramientas a los programas GISMA y GISMO. El plan de trabajo para la adecuación estuvo formado por las siguientes etapas:

1.- Adecuación de los programas a UNIX
2.- Creación de nuevas llamadas al sistema operativo
3.- Traducción y depuración de rutinas
4.- Inclusión de audio

6.2.3.1 Adecuación de los programas a UNIX

Los programas GISMA y GISMO están formados por varios subprogramas que realizan una función específica, por ejemplo; despliegue de menúes, análisis, detección del ratón, despliegue de páginas de ayuda y animación de un comando. El programa GISMA está formado por 159 subprogramas y el programa GISMO por 205. La forma de organización de estos programas es mediante los siguientes módulos:

define/	Programas para definición del problema (estructura)
include/	Archivos de definición de parámetros de GISMA y GISMO
menu/	Programas para el manejo de menúes
save/	Programas para las rutinas de guardar problema
vms-calls/	Programas que implementan llamadas del sistema VMS
general/	Programas de uso general
keypad/	Programas para el manejo del teclado gráfico
param/	Programas para el estudio de parámetros
solut/	Programas para las etapas de solución
analyze/	Programas para análisis y solución de ecuaciones
main/	Programas principales
result/	Programas para exhibición gráfica de resultados
stiff/	Programas para la etapa de rigidez

Los programas de cada uno de los módulos son los siguientes:

analyze/

auto_optimize.F	*member_load.F*	*solve_force.F*
formulate_load.F	*member_stiffness.F*	*solve_reaction.F*
formulate_stiffness.F	*solve_displ.F*	

define/

DEFbuild.F	*div_member.F*	*member_prop.F*
add_joint.F	*draw_grids.F*	*number_sys.F*
add_member.F	*edit_load.F*	*optimize.F*
attach_fix.F	*edit_prop_table.F*	*origin.F*
attach_load.F	*edit_prop_type.F*	*prop_list.F*
build.F	*fixity.F*	*prop_table.F*
check_fix.F	*get_struc.F*	*prop_type.F*
check_loads.F	*grid_joint.F*	*redefine.F*
check_newjoint.F	*info.F*	*renumber.F*
clear_data.F	*keyin.F*	*scroll_load.F*
compress_data.F	*library.F*	*select_file.F*
definition.F	*load_list.F*	*setup_data.F*
del_member.F	*load_table.F*	*show_fix.F*
delete_load.F	*load_type.F*	*switch_data.F*
delete_prop_type.F	*loads.F*	*unload.F*

general/

angle.F
bell.F
check_fn.F
com_manip.F
dehilite.F
display_view.F
draw_arrowhead.F
draw_axis.F
draw_cross.F
draw_dofs.F
draw_lib_diag.F
draw_lib_fn.F
draw_loads.F
draw_mem_g_dof.F
draw_member.F
draw_struc.F
draw_supports.F
file_name.F
full.F
help.F
help2.F
hilite.F
hit_test.F
input_filename.F
inv_transform.F
lib_fn.F
magnify.F
message.F
mouse_input.F
mouse_rest.F
pan.F
photo.F
prompt.F
quit.F
read_file.F
reset.F
rotate.F
scroll.F
store_data.F
string_length.F
transform.F
write_file.F
write_numbers.F
zoom.F

keypad/

kp_create.F
kp_pen_down.F
kp_display.F
kp_input.F

main/

auto_process.F
init.F
check_limit.F
process_info.F
gisma.F
tyle_lib.F
gismo_help_diag.F
build_gismo_help_flags.F
intro.F
creation.F
setup_parameters.F
build_gisma_help_flags.F
init_message.F
color_def.F
select_color.F
gisma_help_diag.F

menu/

draw_menu.F
draw_win_shades.F
full_page.F
intro_menu.F
main_menu.F
menu_creator.F
menu_name.F
win_on_full.F
win_on_general.F
win_on_gisma_main.F
win_on_gismo_main.F

param/

band.F
bandcom.F
bandwidth.F
color_scale.F
conn.F
connectivity.F
draw_bandwidth.F
draw_global_stiff.F
element_connect.F
hit_element.F
member_connect.F
new_bandcom.F
nodal_connect.F
optimize.F
parameter_study.F
reduce_band.F
renumber.F
stiff_matrix_display.F

result/

axial_intensity.F
calcinfl.F
change_loads.F
check_color_scale.F
cubsolve.F
diagrams_on_off.F
diagrams_scale.F
display_results.F
draw_axial.F
draw_displaced_shape.F
draw_moment.F
draw_reactions.F
draw_shear.F
gisma_output.F
gisma_output_listing.F
gismo_output.F
gismo_output_listing.F
init_shape.F
internal_forces.F
shape_scale.F
show_axial_value.F
show_displacement.F
show_force.F
show_reaction.F

save/

delete_file.F	*save_problem.F*	*lib_save.F*
structure_list.F		

solut/

auto_solve.F	*global_displ_vector.F*	*solve_int_force.F*
display_global_matrix.F	*int_force.F*	*solvedisp.F*
draw_global_matrices.F	*member_displ_vector.F*	*solvforce.F*
draw_mem_matrices.F	*old_extract.F*	*solvreact.F*
extract.F	*select_member.F*	*transform_disp.F*
gausselim.F	*solution.F*	*transmem.F*
getband.F	*solve.F*	

stiff/

animate.F	*draw_mem_matrix.F*	*rotation.F*
assemble_global.F	*draw_mem_stiff.F*	*scalstiff.F*
auto_assembly.F	*find_part.F*	*select.F*
autodisp.F	*global_matrix_color.F*	*setup_global.F*
autoreact.F	*globmat.F*	*stepwise_assembly.F*
compare_mem_stiff.F	*locate_submat.F*	*stiff_construction.F*
dofnum.F	*matoper.F*	*submat.F*
dofs_display.F	*member_stiff.F*	*transform.F*
draw_global_matrices.F	*partition.F*	
draw_global_matrix.F	*redraw_mem_matrix.F*	

vms-calls/
vms-calls.c

La forma en que se manejan los archivos en VMS difiere de la de UNIX por ello fué necesario adecuar todos los programas para hacer uso de la sintaxis correspondiente al sistema operativo usado. La forma de hacer esto fue mediante instrucciones al preprocesador de Fortran para incluir o excluir código dependiendo del sistema operativo en que se estuviera trabajando en ese momento:

```
#ifdef unix
#include <hf.h>                    /* Librerías gráficas HOOPS*/
#endif
```

```
#ifdef unix
#ifdef GISMA_COMPILATION  /* variable definida al momento de compilación*/
    INCLUDE '../include/gisma/global_declaration.inc'
    INCLUDE '../include/gisma/limitations.inc'
    INCLUDE '../include/gisma/common_parameter.inc'
    INCLUDE '../include/gisma/common_solution.inc'
    INCLUDE '../include/gisma/common_stiffness.inc'
#elif GISMO_COMPILATION    /* variable definida al momento de compilación*/
```

```
    INCLUDE '../include/gismo/global_declaration.inc'
    INCLUDE '../include/gismo/limitations.inc'
    INCLUDE '../include/gismo/common_parameter.inc'
    INCLUDE '../include/gismo/common_solution.inc'
    INCLUDE '../include/gismo/common_stiffness.inc'
#endif
    INCLUDE '../include/common_graph.inc'
    INCLUDE '../include/common_definition.inc'
#elif vms
    INCLUDE 'GLOBAL_DECLARATION'
    INCLUDE 'LIMITATIONS'
    INCLUDE 'COMMON_PARAMETER'
    INCLUDE 'COMMON_SOLUTION'
    INCLUDE 'COMMON_STIFFNESS'
    INCLUDE 'COMMON_GRAPH'
    INCLUDE 'COMMON_DEFINITION'
#endif
```

Las directrices #ifdef y #endif especifican un bloque de código. Este bloque únicamente se compila cuando las variables de validación correspondientes adquieren un valor positivo. La definición de las etiquetas "unix" y "vms" es automática, estas etiquetas toman su valor al momento de la compilacións de un programa. Por ejemplo en cualquier compilador de C o Fortran en UNIX se genera la definición de "unix" y se le asigna el valor de 1. Las variables GISMA_COMPILATION y GISMO_COMPILATION son definidas al momento de compilar el programa:

Con la asignación de todas estas variables, nos aseguramos que al momento de compilar, se compilen únicamente aquellos segmentos de código que pertenecen a la plataforma en la que se esta trabajando.

```
f77 -c help.F -DGISMO_COMPILATION
```

En la instrucción anterior la opción -c es para generar únicamente el programa objeto, la opción -D define la variable GISMO_COMPILATION para ser usada en todos los programas de GISMO que sean compilados con el compilador FORTRAN 77.

A todos los programas de GISMA y GISMO se les hicieron este tipo de adecuaciones.

Debido a que la sintaxis de instrucciones de HOOPS y F77 son iguales para el sistema operativo VMS que para UNIX no se tuvieron que hacer mayores cambios a los programas.

6.2.3.2 Creación de nuevas llamadas al sistema operativo

Algunas rutinas de los programas GISMA y GISMO hacen referencia a funciones propias del sistema operativo VMS, debido a que en el sistema operativo UNIX estas funciones no existen, tienen variantes de nombre ó la obtención de datos es diferente se crearon nuevas rutinas para reemplazar a estas llamadas del sistema.

La solución óptima para incluir estas rutinas no fue especificando el uso de éstas en los programas que forman GISMA y GISMO, sino que únicamente se creó un nuevo programa en C que redefinía las rutinas de VMS. La razón principal por la que se decidió crear nuevas rutinas en lugar de alterar la ya existentes fue porque las adaptaciónes se harían de forma más rápida y sencilla y el manteniemiento a los programas sería menos costoso aunque el código se vió incrementado en pequeña medida.

Las llamadas al sistema que se redefienieron en UNIX fueron:

lib_getjpi_	Obtiene el nombre del usuario en caracteres
lib_wait_	Especifica tiempo de retardo en segundos
lib_date_time_	Obtiene la fecha actual en caracteres
lib_find_file_	Detección de la existencia de un archivo
lib_delete_file_	Borra el archivo especificado

6.2.3.2.1 Programa con las llamadas al sistema redefinidas en UNIX

El archivo con las llamadas redefinidas se presenta a continuación:

```
/* VMS-CALLS.C: SunOS version of several VMS LIB$ system calls */
/* this program was developed at UNAM*/

#include <stdio.h>
#include <time.h>
#include <errno.h>


lib_getjpi_(code, pid, pnam, res_val, res_length)
     char *res_val;
     int *res_length;
     {
     cuserid(res_val); /*función de UNIX para obtener el nombre del usuario en res_val*/
     *res_length = strlen(res_val);
     return 1;
     }

lib_wait_(x)
     double x;
     {
     usleep((unsigned)(x*1000)); /* suspensión de la ejecución por X milisegundos */
```

```
}

lib_date_time_(buf)
     char *buf;
     {
     time_t clock;

     time(&clock);  /* obtiene la hora del sistema*/
                    /* Conversión del tiempo a  tiempo local (localtime) y posteriormente a cadena
                      de caractéres especificando formato*/
     return strftime(buf, 23, "%d-%h-%Y %k:%M:%S ", localtime(&clock));
     }

lib_find_file_(file_spec, result_spec, context)
     char *file_spec, *result_spec;
     int *context;
     {
     static char result_buffer[1024];
     int end = 0;

     shell_echo(file_spec, result_buffer, 1024);  /* ejecuta subproceso con el comando echo */
     for (;;)
      switch (result_buffer[end])
               {
          case ' ':                          /* se recibio una palabra */
               *result_spec = '\0';
               *context = ++end;             /* asignación del la longuitud*/
               goto FIN;

          case '\0':                         /* se recibio una cadena vacia*/
               *result_spec = '\0';
               *context = 0;                 /* longuitud cero*/
               goto FIN;

          default:
               *result_spec++ = result_buffer[end++];
               break;
               }
FIN:
     }

lib_delete_file_(fname)     /* borra el archivo "fname" */
     char *fname;
     {
     int stat;

     fprintf(stderr, "borrando el archivo: %s; ", fname);
     stat = unlink(fname);
     fprintf(stderr, "estatus: %i\n", stat);
     perror(NULL);
     }


#define BUFLEN  1024
```

```
static FILE   *fp_fromchild, *fp_tochild;
static int    fromchild, tochild;

static start_shell()
    {
  int pipefrom[2], pipeto[2];
  int c, numfds;

  if (pipe(pipefrom) < 0 || pipe(pipeto) < 0) /*Creación de un canal de comunicación (pipe) y
                                                retorno de dos descriptores de archivo para lectura
                                                (pipefrom[0]) y escritura (pipefrom[1])*/
        {
        perror("filefind: pipe");          /* si hubo error lo describe y direcciona a la salida de
                                              errores */
        exit(1);                           /* salida del sistema*/
        }

  switch (fork()) /*Creación de un nuevo proceso*/
        {
  case -1:                                 /* error en la creación del proceso */
        perror("start_shell: fork");
        exit(1);

  case 0:                                  /* Este es el proceso hijo */
        dup2(pipeto[0], 0);                /* especifica valor
                                              deseado  para (pipeto[0])*/
        dup2(pipefrom[1], 1);              /* especifica valor deseado para pipefrom[1] */
        numfds = getdtablesize();          /*Obtiene estado de la tabla de descriptores de archivo*/
        for (c = 3; c < numfds; c++)       /* cierra todos los descriptores de archivo */
             close(c);

        execl("/usr/bin/sh", "sh", 0);     /*ejecuta un shell y pasa el contro al subproceso*/
        perror("subshell: exec");          /* Mensaje al sistema de error*/
        exit(1);

  default:                                 /* Este es el proceso padre */
        close(pipefrom[1]);                /* desactiva apuntador */
        close(pipeto[0]);                  /* desactiva apuntador */
        fromchild = pipefrom[0];
        tochild = pipeto[1];
        fp_fromchild = fdopen(fromchild, "r");  /*apertura del canal de lectura
                                                  especificado en fromchild*/
        fp_tochild = fdopen(tochild, "w");      /* apertura del canal de escritura
                                                  especificado en tochild */
        setbuf(fp_tochild, NULL);          /* asigna un buffer al apuntador fp_tochild */
        break;
        }
    }

static stop_shell()                        /* salida del shell */
 {
    int stat;
```

```
    fputs("exit\n", fp_tochild);      /* introduce en el subproceso el comando exit*/
      if (wait(&stat) == -1)
           perror("subshell");      /* mensaje al sistema de errores si hay problemas en el subshell*/
      }

static pipe_writer_echo(filespec)  /* manda al subproceso el comando echo con el parámetro
                                      filespec*/
      char *filespec;
      {
    char buf[512];

      strcpy(buf, "echo ");
      strcat(buf, filespec);
      strcat(buf, "\n");
    fputs(buf, fp_tochild);
      }

static pipe_writer(command)    /*  ejecuta el comando "command" en el subshell*/
      char *command;
      {
    char buf[128];
      strcpy(buf, command);
      strcat(buf, "\n");
    fputs(buf, fp_tochild);
      }

static pipe_reader(buf, n)         /* lectura del buffer de datos del subshell*/
      char *buf;
      {
    fgets(buf, n, fp_fromchild);
    buf[strlen(buf) - 1] = '\0';      /* remueve el caracter newline */
      }

shell_echo(filespec, result, n)  /* ejecuta el comando echo en un subproceso */
      char *filespec, *result;
      {
      start_shell();                          /* inicia un nuevo shell */
      pipe_writer_echo(filespec);             /* llamada a la rutina pipe_writer_echo */
      pipe_reader(result, n);                 /* lectura del bufer de datos*/
      stop_shell();                  /* cierra el shell */
      }
shell_command(command, result, n)         /* ejecuta el comando "comando" en un subproceso */
      char *command, *result;
      {
      start_shell();                    /*inicia un subproceso*/
      pipe_writer(command);            /*ejecuta comando*/
      pipe_reader(result, n);          /*lee buffer de datos*/
      stop_shell();              /*cierra subproceso*/
      }
```

6.2.3.2.2 Descripción de funciones

La función ***lib_getjpi*** extrae información referente a el usuario que esta ejecutando los programas. En las variables ***res_val*** y ***res_length*** se regresa el nombre de la cuenta del usuario y su longitud en caracteres respectivamente. En VMS ésta función entrega información adicional en las variables *code, pid* y *pname,* las cuales no son usada en los demás programas de GISMA y GISMO. Por tal motivo se decidió no extraer los datos correspondientes a estas variables dentro de nueva función ***lib_getjpi***.

En la función ***lib_wait*** de VMS se especifica un retardo de "X" segundos en el programa. Para UNIX la función equivalente para el retardo de tiempo es ***usleep***, en ésta se debe de especificar el retardo en milisegundos.

La fecha del sistema en VMS se obtiene mediante la función ***lib_date_time***; ésta función retorna en una cadena de 23 caracteres la fecha y la hora. En UNIX se utiliza la librería ***time.h*** para obtener la fecha y hora del sistema. Mediante la función ***time*** de ésta librería se obtiene en una estructura de datos llamada *time_t* la información de la fecha y hora en la variable ***clock***. Para dar el mismo formato de 23 caracteres en VMS se uso la función ***strftime***, la cual recibió como argumento la función ***localtime (&clock)*** que transforma los datos mundiales de ***clock*** a datos locales.

La función ***lib_find_file_*** en VMS se utiliza para conocer si un archivo se encuentra en el directorio actual. En UNIX no existe una función que pueda ser usada dentro de un programa para proporcionar esta información, sin embargo dentro de un shell de UNIX se usa el comando "echo [argumento]" que muestra en pantalla una lista de archivos que cumplen con la cadena "argumento". Mediante la creación de rutinas que generan subprocesos a partir de un comando e interactuan recibiendo los resultados se usa el comando "echo" a través de la función ***shell_echo***. Al inicio de ***lib_find_file_*** se usa la función ***shell_echo***. Posteriormente se inicia un ciclo iterativo para determinar si se recibió el nombre del archivo buscado. Si fue recibido, el apuntador ***result_spec*** apunta a ésta cadena y la variable ***context*** da la longitud.

Para borrar un archivo dentro de VMS utilizamos la función ***lib_delete_file_***. En UNIX se utiliza la función ***unlink***, la cual borra la liga que referencia al archivo dentro de un directorio.

Ademas de las funciones descritas anteriormente, se crearon las funciones ***start_shell, stop_shell, pipe_writer, pipe_reader, shell_echo*** y ***shell_command***. La función ***start_shell*** crea un subproceso mediante la función ***fork*** y define canales de comunicación entre el subproceso y el proceso padre a través de funciones ***pipe***. El subproceso ejecuta el shell "/usr/bin/sh" y pasa el control al proceso padre; que en este caso el proceso padre es una de las aplicaciones GISMA o GISMO. Las funciones ***pipe_writer*** y ***pipe_reader*** son usadas para transmitir al shell del subproceso los comandos a ejecutar en la variable ***filespec*** y leer los resultados en la variable ***buf***. Finalmente las funciones ***shell_echo*** y ***shell_command***

usan las funciones anteriores descritas en este párrafo para ejecutar el comando "echo" y cualquier otro comando que se especifica en la variable "command" para la función ***shell_command***.

El anterior programa se compila generando solamente su código objeto y se liga después a los 205 programas de UNAM-GISMO o a los 159 programas de UNAM-GISMA para la generación del correspondiente programa ejecutable.

```
cc -c  vms-calls.c                          /* Generación del código objeto */
f77 $(FILES)  -o gismo -lX11 -lhoops -lxgl -lm   /* Se ligan todos los objetos para
                                            generar el ejecutable GISMO,
                                            la variable FILES contiene todos
                                            los programas que son ligados.
                                            Las opciones de compilación que
                                            comienzan con -l son para ligar
                                            librerias del sistema*/
```

6.2.3.3 Traducción y depuración de rutinas

La traducción de las pantallas de menúes y la ayuda del sistema se realizó con la colaboración de estudiantes de la maestría en estructuras de la DEPFI. En esta etapa se modificaron en los programas fuente rutinas que mediante el uso de HOOPS exhiben en pantalla comandos, texto de ayuda y mensajes.

Las funciones modificadas fueron:

CALL HF_INSERT_TEXT

CALL GIS_G_PROMPT

CALL GIS_MENU_PAGE_NAME

Estas funciones proporcionan cadenas de caractéres a HOOPS para exhibirlas en la pantalla. Solamente se tradujeron las cadenas de caractéres y se incluyeron como argumentos de las funciones. Las llamadas a las rutinas GIS_G_PROMPT y GIS_MENU_PAGE_NAME también utilizan la función HF_INSERT_TEXT de HOOPS.

Además de las funciones se tradujeron variables que definen comandos y especificaciones de formato usados en los programas GISMA y GISMO:

```
      CHARACTER *13 CMD_NAME(2,NUMBER)
      DATA CMD_NAME/ 'AGREGA     ', 'ELEMENTO   ',
     &               'REMUEVE    ', 'ELEMENTO   ',
     &               'DIVIDE     ', 'ELEMENTO   ',
     &               'AGREGA     ', 'NODO       ',
```

```
   &          'ORIGEN     ','          ',
   &          'NUMERA     ','SISTEMA    ',
   &          'PROPIEDADES ','ELEMENTOS  ',
   &          'RESTRICCION ','          ',
   &          'CARGAS     ','          ',
   &          'REDEFINE   ','          '/
 2  FORMAT(I4,' NODOS , ',I4,' ELEMENTOS , & ',I4,' G.L.'S')
19 FORMAT(///T61,'FUERZAS DE ELEMENTOS'//T38,'AXIAL')
```

Los mensajes de ayuda de GISMA y GISMO no están contenidos en los programas fuente, se encuentran en archivos contenidos en el directorio help:

```
/help
gisma_intro.hlp        help_gisma_result.hlp    help_solution.hlp*
gismo_intro.hlp        help_gismo_result.hlp    help_stiffness.hlp*
help_common.hlp        help_parameter.hlp
help_definition.hlp    help_save.hlp
```

En cada uno de estos archivos se tradujeron todos los mensajes de ayuda como lo muestra el siguiente fragmento de archivo:

```
** Archivo help_gismo_result.hlp
** Texto de ayuda para el menu resultados
**  página del menu de resultados  :
 -1  4
DESPLIEGUE DE RESULTADOS página de menú :
Esta página de menú permite al usuario inspeccionar los resultados
analizados  incluyendo configuración deformada, fuerzas internas en
los elementos y reacciones en apoyos.

**  Comando CARGAS ON/OFF en  el menu RESULTADOS :
  1  5
Comando CARGAS ON/OFF :
Este  comando  intercambia la pantalla activando y desactivando las
cargas.
Desactivando las cargas en la mayoría de los casos se puede limpiar
la pantalla.

**  Comando APOYOS ON/OFF en el menu RESULTADOS :
  2  5
Comando APOYOS ON/OFF :
Este comando intercambia de la pantalla los apoyos de la estructura
de activados a desactivados.
Desactivando los apoyos en la mayoría de los casos se puede limpiar
la pantalla.

**  Comando CONFIGURACION DEFORMADA en el menu RESULTADOS :
  3  9
Comando CONFIGURACION DEFORMADA :
Este comando exhibe la forma (a escala) de la estructura deformada
```

y los puntos de inflexión de los elementos de la estructura.
El factor de amplificación de la configuración deformada es mostrado en el área de MENSAJES/INFORMACION.
El usuario puede inspeccionar los desplazamientos de los nodos tecleando sobre estos en el esquema de la estructura no deformada.
Notar que las deformaciones intermedias de los elementos son aproximadas.

** Comando ESCALA DE DESPLAZAMIENTOS en el menu RESULTADOS :
4 9
Comando ESCALA DE DESPLAZAMIENTOS :
Este comando permite al usuario variar la escala de la configuración de la estructura deformada usando la ventana del menú principal como un potenciómetro el cual controla linealmente el rango de amplificación.
El actual factor de amplificación es mostrado en el área de MENSAJES/ INFORMACION. La escala de la configuración deformada es aumentada si se teclea el "ratón" en la mitad derecha de esta ventana, y es reducida si se teclea en la mitad izquierda. Nótese que este comando aparece sólo cuando el comando de configuración está activado.

Las líneas que contienen al inicio dos números sirven de referencia a los programas para mapear el texto de ayuda de una pantalla del sistema y su correspondiente comando, ejemplo:

** página del menu de resultados :
-1 4
DESPLIEGUE DE RESULTADOS página de menú :
Esta página de menú permite al usuario inspeccionar los resultados analizados incluyendo configuración deformada, fuerzas internas en los elementos y reacciones en apoyos.

4 9
Comando ESCALA DE DESPLAZAMIENTOS :
Este comando permite al usuario variar la escala de la configuración de la estructura deformada usando la ventana del menú principal como un potenciómetro el cual controla linealmente el rango de amplificación.

El primer párrafo describe el inicio de una nueva pantalla de menú mediante el número "-1". Cabe mencionar que la forma de recuperar el texto de ayuda es mediante comandos para ubicar un apuntador de archivo en el texto que define una pantalla de menú y posteriormente desplazarse hasta su correspondiente comando. El número 4 de este mismo párrafo indica el número de líneas del texto de ayuda para la página de menú.

En el segundo párrafo el número "4" referencia al comando en la pantalla del menú de resultados y el número 9 representa el número de líneas que forman el texto de ayuda para el comando ESCALA DE DESPLAZAMIENTOS. Las líneas que comienzan con el caracter "*" son comentarios dentro de los archivos de ayuda.

6.2.3.4 Inclusión de Audio

Para proporcionar a los programas UNAM-GISMA y UNAM-GISMO mayor facilidad en su manejo se incorporaron rutinas de programas escritos en el lenguaje "C" para añadir recursos de multimedios. La adición de audio fue la característica de multimedios agregada a los programas. Mediante el desarrollo de una aplicación escrita con utilerías de las librerías XVIEW y MULTIMEDIA de SUN se pueden reproducir archivos de audio en el formato de SUN, mediante su control por medio de una ventana con comandos, Fig. 6.1.

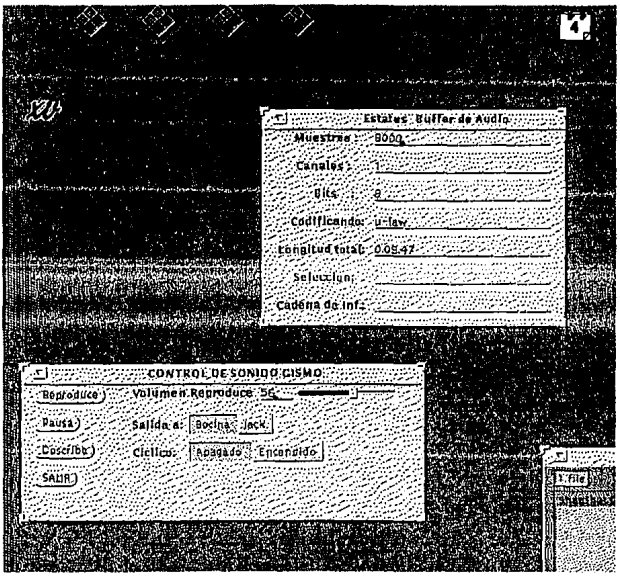

Fig. 6.1 Aplicación de audio

La ventana de control de sonido posee los siguientes comandos para la reproducción de archivos:

REPRODUCE	Inicia la reproducción de un archivo
PAUSA	Detiene la reproducción
DESCRIBE	Presenta en una ventana la descripción del archivo de audio
SALIR	Termina la ejecución del programa

Además de los comandos anteriores se tiene un control deslizable para el volumen, se puede especificar la dirección de la salida del audio (hacia bocina o jack), se tiene un control de repetición del archivo, es decir, se puede estar repitiendo indefinidamente un archivo de audio mediante esta opción.

La inclusión de esta herramienta dentro de los programas UNAM-GISMA y UNAM-GISMO se hizo mediante la inclusión de un botón de comandos en el área de despliegues del modo de ayuda. Fig. 6.2.

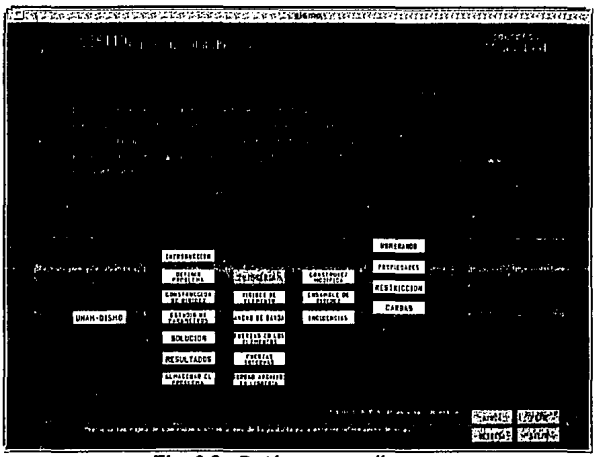

Fig. 6.2 Botón para audio

En la pantalla de ayuda anterior se muestra el botón de sonido incorporado en el área de despliegue de un menú de GISMO. La selección de este botón por medio del ratón ocasiona que cambie de color negro (estado inicial) a amarillo y viceversa. El color amarillo indica que el audio está activo. Con el audio activo, al momento de seleccionar un comando además de aparecer el texto de ayuda en la pantalla de despliegues se ejecuta la aplicación de la Fig. 6.1 y se carga el archivo de sonido correspondiente al texto de ayuda. Si el botón de sonido permanece de color negro no se ejecuta ninguna aplicación de audio y se tiene la ayuda del sistema solamente en texto.

Los programas desarrollados para habilitar las opciones de audio fueron:

read_sound.c	/* rutina para leer datos de los archivos de sonido */
sound.c	/* ejecuta la aplicación de audio*/
sound_window.c	/* despliega en pantalla el botón de sonido*/
gismoids_sound.c	/* aplicación de audio */

6.2.3.4.1 Descripción de los programas de audio

Los programas de sonido se elaborarón mediante las librerias XVIEW y MULTIMEDIA de SUN. XVIEW es un toolkit para programar aplicaciones gráficas por encima de X11. El lenguaje usado para desarrollar estas librerias es C orientado a objetos. También el uso de XVIEW es mediante funciones y estructuras que se orientan al manejo de objetos. Para la libreria MULTIMEDIA de SUN se hace uso de funciones predefinidas con el proposito de manipular los dispositivos de audio de la estación de trabajo.

El listado de los programas necesarios para habilitar las capacidades de audio en los programas UNAM-GISMO y UNAM-GISMA se presentan en el Apéndice A. Todos estos programas fueron desarrollados con el lenguaje de programación C y las librerías gráficas de HOOPS y XVIEW.

read_sound.c

Este programa crea en memoria una estructura de datos que relaciona los menús y sus comandos con su correspondiente archivo de audio. Los archivos de audio fueron elaborados con anterioridad dentro de la misma estación de trabajo mediante la grabación de pequeños comentarios que describen el uso de todos los comandos disponibles en los programas UNAM-GISMO y UNAM-GISMA. Esta grabación se hizo mediante la aplicación de audio SOUNDTOOL que está en todas las estaciones de trabajo SUN y un micrófono profesional de 600 Ω. El formato de los archivos de audio es el ".AU" de SUN, tomando 8000 muestras por segundo y una longitud de datos de 8 bits. El programa **read_sound.c** lee el archivo **lista_sonido.dat** y crea una lista dinámica que crece de acuerdo con el tamaño de este archivo.

La información que contiene el archivo **lista_sonido.dat** es la siguiente:

```
0 1 intro_gisma
1 1 gisma.1-1
1 2 gisma.1-2
.
.
```

En cada renglón el primer número de izquierda a derecha representa al menú actual al momento de ejecutar los programas. El segundo número es el número de comando correspondiente al menú actual. Finalmente los caracteres siguientes son el nombre del archivo de audio. La numeración de los menús y comandos ya esta determinada de acuerdo con los programas originales desarrollados en Cornell.

sound_window.c

Esta es una rutina que usa funciones hecha con HOOPS para crear dentro de los programas UNAM-GISMA y UNAM-GISMO el botón de audio Fig. 6.3. Para habilitar su uso se incluyo dentro del programa **help.F** una llamada invocando a la función ***sound_window*** de este programa.

sound.c

Este programa realiza una búsqueda en la lista de datos generada por el programa **read_sound.c**. Mediante la numeración de menús y comandos se recupera el nombre del archivo de sonido correspondiente y posteriormente se hace una llamada al sistema para ejecutar la aplicación **gismoids_sound** pasándole como argumento el archivo de audio encontrado.

gismoids_sound.c

Este es el programa que genera un nuevo proceso con la apariencia de una ventana con comandos (Fig. 6.1). La aplicación fue hecha mediante funciones y estructuras de datos de SUN muy parecidas a las de MOTIF. Las librerías de SUN usadas fueron: XVIEW para la interfaz gráfica y MULTIMEDIA para el manejo del audio. El dispositivo de audio se encuentra dentro del directorio de dispositivos del sistema y tiene la siguiente ruta: **/dev/audio**.

La operación del programa es mediante los siguientes pasos:

1) La generación de la aplicación cliente (funciones de XVIEW) y su despliegue en pantalla
2) La inicialización del audio a través de la función ***audio_control_init***
3) La generación de un buffer de audio mediante las funciones ***init_buffer*** y ***soundfile_read***
4) La generación y creación de los botones de control para el manejo de opciones mediante
   las funciones: ***main_create_panel, file_create_panel, describe_create_panel, window_fit_height*** y ***xv_set.***
5) Monitoreo de las opciones seleccionadas y ejecución de estas a través de la función ***window_main_loop***

La función **window_main_loop** *monitorea eventos de X11 para saber si se eligió uno de los controles definidos en la función* **main_create_panel**, *si fue así, pasa la ejecución a los procedimientos definidos* dentro de esta misma función.

6.2.3.4.2 Descripción del manejo de audio

Para manejar el audio se sigue la siguiente secuencia dentro del programa:

1.- Verifica el estado del dispositivo de audio
Mediante la función *audio_open* se reserva el uso del dispositivo de audio.

2.- Verificación de la existencia del archivo de audio
En esta etapa se utiliza la llamada a la función *scandir* del programa de audio para revisar mediante la apertura del archivo su existencia.

3.- Lectura del archivo de audio a un buffer de datos
Usando la función *soundfile_read* se almaceman los datos dentro de la estrutura Buffer (estado del audio, tamaño del archivo, apuntador al inicio de datos, apuntador al final de datos y encabezado).

4.- Inicio de la reproducción del archivo
En esta etapa se emplea la función *start_play*. Se indican nuevos valores a la bandera PLAY, se establece el tiempo para terminar con el audio y terminar el proceso.

5.- Reproducción de audio
Esto se hace mediante la función *play_service*. Esta envía constantemente al dispositivo de audio datos para su reproducción.

6.- Suspender el audio
Se hace mediante las funciones *audio_close* para cerrar el dispositivo de audio y *stop_play* para suspender la reproducción.

6.3 DESCRIPCIÓN DE CONCEPTOS Y FUNCIONES UTILIZADAS PARA LA ADECUACIÓN

En la etapa de análisis para la sustitución de funciones de VMS a UNIX se recurrió a manuales en papel y la ayuda en línea que proporciona el comando "man" en los sistemas UNIX. Para reemplazar las funciones ***lib_getjpi_***, ***lib_wait_***, ***lib_date_time_*** *y* ***lib_delete_file_*** en VMS sólo se hizo uso de las equivalentes en UNIX. En cambio para elaborar un proceso parecido a la función ***lib_find_file_*** en VMS se recurrió al uso de funciones de control (fork) y funciones de comunicación entre procesos (pipes).

6.3.2 Función fork

La única forma de crear un proceso a través del kernel de UNIX es mediante la función *fork*. El nuevo proceso que crea la función *fork* es llamado proceso *proceso hijo.* Esta función retorna dos valores: "0", que es valor que retorna al proceso hijo y el identificador del proceso hijo (process ID) que retorna al proceso padre. El motivo por el cual se retorna al proceso padre el identificador del proceso hijo es que el padre puede tener más de un proceso hijo y además no existe función en UNIX que pueda obtener el identificador de los procesos hijo del padre. También el motivo por el cual se retorna "0" al proceso hijo es porque éste solamente puede tener un padre. Después del *fork* el proceso padre e hijo siguen en ejecución a la siguiente instrucción de la llamada al *fork.* El proceso hijo es una copia del padre, tienen los mismos datos pero no están en el mismo espacio de memoria.

En general, no se sabe si el hijo comienza a ejecutar antes que el padre o viceversa. Esto depende del algoritmo usado por el kernel. La única forma de hacer que el proceso padre e hijo se sincronicen es a través de alguna forma de comunicación entre procesos.

6.3.3 Pipes

Los pipes son una forma primitiva para comunicar procesos en UNIX. Tienen las siguientes limitantes:

- Son half-duplex. El flujo de datos es en una sola dirección
- Sólo se usan en procesos que tiene un antecesor común. Se crean mediante un proceso padre.

No obstante estas limitantes éste mecanismo es el más usado entre los diferentes IPCs (Métodos de Comunicación Entre Procesos) de UNIX. El pipe se crea mediante una llamada a la función pipe de UNIX:

```
#include <unistd.h>
int pipe (int filedes[2]) ;
```

En la función se devuelven dos descriptores de archivo en el argumento *filedes*: *filedes*[0] se abre para lectura y *filedes*[1] se abre para escritura. La salida para *filedes*[1] es la entrada para *filedes*[0].

Hay dos formas para dibujar un pipe: en la figura 6.3 se tienen los extremos del pipe conectados a través de un proceso simple. En la figura 6.4 se encuentra el flujo de datos del pipe a través del kernel.

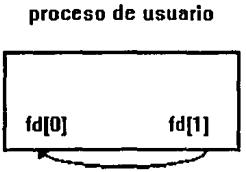

Fig. 6.3 Pipe de UNIX

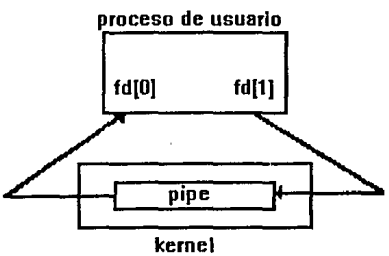

Fig. 6.4 Pipe de UNIX

Los diagramas anteriores muestran una forma de comunicación no usada; en un solo sentido. Es común que el proceso que hace la llamada al pipe también llame a la función *fork*, creando así un canal de comunicación entre el proceso padre y el hijo o viceversa.

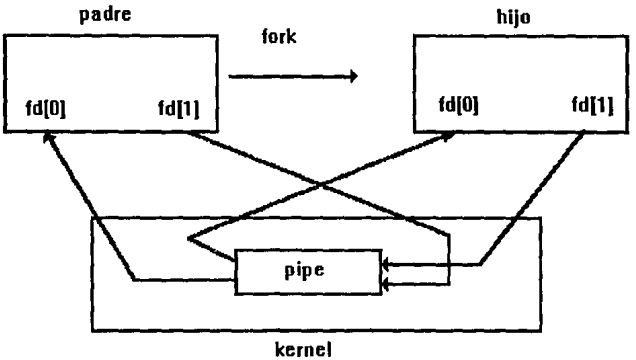

Fig. 6.5 Pipe half-duplex después de un fork

Al momento de usar una función fork y pipe el flujo de datos se especifica de la siguiente forma:

- Si se cierra el descriptor df[0] en el padre y en el hijo se cierra el descriptor df[1], resulta un flujo de datos del padre hacia el hijo.

- Cerrando el descriptor df[1] en el padre y cerrando el df[0] en el hijo se tiene el flujo de datos del proceso hijo al padre.

CONCLUSIONES

La utilización de las interfaces gráficas dirigidas al usuario son en la actualidad una herramienta indispensable para que el usuario disponga de manera sencilla de todas las capacidades y aplicaciones de los equipo de cómputo. Conjuntando los multimedios con las interfaces gráficas se dispone de una gran cantidad de recursos para el desarrollo de un sin número de aplicaciones dirigidas a la presentación de temas de interés humano como los son la educación, capacitación, comunicación, difusión, etc. En el caso particular de los sistemas de aprendizaje se tiene la posibilidad de abarcar múltiples canales de percepción del individuo para la óptima comprensión de un tema, y en algunos, dependiendo del tipo de software, usar menos tiempo para la resolución de problemas.

Es de suma importancia que los avances tecnológicos en materia de cómputo sean aprovechados en la enseñanza, en particular de la ingeniería, debido a que en ésta generalmente se necesita de ayudas adicionales para comprender los conceptos complejos involucrados, por ejemplo: algoritmos de graficación, procesamiento de imágenes, flujo de calor, campos magnéticos, filtros digitales, etc. El proyecto SOCRATES, del cual nuestra Universidad forma parte, es un ejemplo de una respuesta efectiva a esta necesidad. En SOCRATES además de los programas de GISMA y GISMO se disponen de varios temas de ingeniería que se usan como auxiliares en el proceso enseñanza-aprendizaje.

En la Facultad de Ingeniería de la UNAM se dispone de una gran cantidad de recursos de hardware que se utilizan para introducir a los estudiantes en su manejo y para el uso de sistemas específicos. Sin embargo casi no existen programas que ayuden a la comprensión de los temas de las asignaturas. Las personas que desarrollamos esta tesis pensamos que sería de una buena idea continuar con el desarrollo de software en beneficio de las diferentes asignaturas que se imparten en la Facultad y dentro de la UNAM. Es indispensable el disponer de los recursos de cómputo necesarios en benefico de uno de los fines principales de la UNAM que es la enseñanza.

Con respecto a los programas GISMA y GISMO se consiguió la meta de tener estas herramientas traducidas y funcionando en la plataforma UNIX de la compañía SUN, para su uso en la enseñanza. La máquina donde actualmente esta instalado este software puede funcionar como servidor a otras computadoras, sin importar la marca o el modelo que sean. La limitante es tener acceso a RedUNAM y poder importar el protocolo gráfico de X11. Una PC con tarjeta de red y software que la conviertan en un servidor X es suficiente.

Las utilerías de audio sólo son operativas en una estación de trabajo SUN; no están disponibles en la operación del programa por medio de una conexión remota. Esta desventaja puede fácilmente resolverse mediante el uso de las herramientas de multimedios particulares para cada equipo.

Durante el desarrollo de esta tesis se conoció con más detalle el potencial de cómputo y las diferentes herramientas de conectividad que poseen las estaciones de trabajo. Debido a la gran rapidez de los procesadores RISC y a las enormes cantidades de memoria en estas máquinas, las herramientas de software como editores, compiladores y depuradores son más rápidas y tienen mayor versatilidad comparadas con las de una PC. La confiabilidad que brinda una estación de trabajo para el desarrollo de aplicaciones es muy grande, ya que no se corren riesgos de alterar información vital para el sistema como zonas de memoria, registros de memoria, borrar archivos importantes y otras acciones que afectan el desarrollo de sistemas en las PCs debido a lo limitado de su sistema operativo.

El conocimiento del sistema operativo UNIX de SUN y la forma de interactuar con él desde un programa son métodos que se conocieron y aprendieron a usar. Las librerías gráficas HOOPS de Autodesk y XVIEW de SUN son actualmente muy utilizadas para el desarrollo de aplicaciones gráficas en estaciones de trabajo y su forma de uso esta muy relacionada con conceptos básicos de X11. Las herramientas de multimedios en estaciones de trabajo están en pleno crecimiento; recientemente se añadieron nuevas funciones en diferentes plataformas para permitir el uso de capacidades de audio y video a través de red.

Además de los trabajos característicos en el desarrollo de aplicaciones se conocieron tareas de administración, como por ejemplo: la apertura de cuentas, la instalación de software, definición de variables dentro de UNIX, configuración del correo, uso de la cuenta del administrador del sistema (root) y otras.

APENDICE A

```
/* Archivo: read_sound.c
Este archivo contiene procedimientos para leer el archivo lista_sonido.dat
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string.h>
#include <sys/types.h>

FILE *fdat; /*  Apuntador al archivo de datos */


/*Definicion de la estructura que mapea los dialogos de ayuda: */

typedef struct Lista{
        int menu; /* numero menu */
        int indice;/* numero comando */
        char *archivo;      /* archivo de sonido */
        struct Lista *liga; /* siguiente estructura*/
}       ListaDialog;

char archivo_n[30];  /* nombre de archivo */
ListaDialog *ListaInicio; /* apuntador inicio de lista de datos */
ListaDialog *ListaAp;          /* apuntador lista de datos */
ListaDialog *ListaNv;          /* apuntador nueva estructura */
static int longi;       /* tama&o estructura */

/* definicion de rutina */
read_sound() {
/*   Se abre el archivo lista_sonido.dat para leerlo en memoria  */
        if ((fdat = fopen("../sound/lista_sonido.dat", "r")) == NULL){
                fprintf(stderr, "No se puede abrir el archivo de datos.\n");
                exit (1);
        }
        longi=sizeof(ListaDialog); /*tama&o de estructura*/
        ListaInicio=(ListaDialog *)malloc(longi); /*se crea el inicio de lista*/
        fscanf(fdat,"%d %d %s",&ListaInicio->menu,
                &ListaInicio->indice,archivo_n);  /* lectura dato 1*/
        ListaInicio->archivo=(char *)malloc(strlen(archivo_n)+1);
        strcpy(ListaInicio->archivo,archivo_n);
        ListaInicio->liga=NULL;
        ListaAp=ListaInicio;
/*
Lectura del archivo comenzando de la segunda linea de lista_sonido.dat
*/
        while ( !feof(fdat) ){ /* lectura de datos hasta encontrar EOF */
                ListaNv=(ListaDialog *)malloc(longi);
                fscanf(fdat,"%d %d %s",&ListaNv->menu,&ListaNv->indice,
                archivo_n);
                ListaNv->archivo=(char *)malloc(strlen(archivo_n)+1);
                strcpy(ListaNv->archivo,archivo_n);
                ListaNv->liga=NULL;
                ListaAp->liga=ListaNv;
                ListaAp=ListaNv;
        }
/*
Fin de la lectuara del archivo lista_
*/
```

```
/* Archivo : sound.c
        Programa para llamar la aplicacion de audio en GISMA y GISMO
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string.h>
#include <sys/types.h>
#include <hc.h>

/*Definicion de la estructura que mapea los dialogos de ayuda: */
typedef struct Lista{
        int menu;
        int indice;
        char *archivo;
        struct Lista *liga;
}       ListaDialog;

/* variables externas de la lista de datos*/
extern  ListaDialog *ListaInicio;
extern  ListaDialog *ListaAp;
/* variables para almacenar nombres de archivo */
char archivo_n[30];
char archivo_audio[50];

/*definicion de la funci_n:*/
sonido ( MENU_FLAG, WINDOW ) /*se dan numero de menu y numero de comando*/
        int * MENU_FLAG;
        int * WINDOW;
{

/*
Inicio de busqueda en memoria:
*/

ListaAp=ListaInicio;
        while ( !((ListaAp->menu == *MENU_FLAG && ListaAp->indice == *WINDOW))  &&  ListaAp->liga !=
NULL)
                ListaAp=ListaAp->liga;
        /*
         Si se encontro en el archivo de sonidos:
         */
        if (ListaAp->menu == *MENU_FLAG && ListaAp->indice == *WINDOW){
                printf("\n Lectura del archivo: %s ",ListaAp->archivo);
                strcpy(archivo_audio,"sound_gisma ");
                strcat(archivo_audio,ListaAp->archivo);
                strcat(archivo_audio," &");
                system(archivo_audio);
        }

}
```

```
/*
Archivo: sound_window.c
        definición de un bóton para habilitar el sonido
        el los programas GISMA y GISMO
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hc.h>

float  XMAX,XMIN,YMAX,YMIN,X2MAX,X2MIN,Y2MAX,Y2MIN; /* localizacion boton*/

ventanavoz (SONIDO_ESTATUS)
        int * SONIDO_ESTATUS ;  /* Estado para el color de la ventana:
                                   0=inactiva=negro
                                   1=activa=oro */
{
 typedef struct {float x,y,z;} punto; /*estructura punto*/
 punto a,b,c,d;   /* a,b y c definen un arco */
 punto poli[8];   /* poligono para bocina */

        XMAX=0.95;  /* se dan valores a los puntos del boton */
        XMIN=0.75;
        YMAX=0.95;
        YMIN=0.80;
        X2MAX=0.96;
        X2MIN=0.74;
        Y2MAX=0.75;
        Y2MIN=0.60;


        a.x=0.3;   /* se definen puntos para el arco*/
        a.y=0.6;
        a.z=0.0;
        b.x=0.15;
        b.y=0.0;
        b.z=0.0;
        c.x=0.3;
        c.y=-0.6;
        c.z=0.0;
        d.x=0.45;
        d.y=0.0;
        d.z=0.0;
                /* se definen puntos de la bocina*/
        poli[0].x=0.3;      poli[0].y=0.6;      poli[0].z=0.0;
        poli[1].x=-0.3;     poli[1].y=0.3;      poli[1].z=0.0;
        poli[2].x=-0.5;     poli[2].y=0.3;      poli[2].z=0.0;
        poli[3].x=-0.5;     poli[3].y=-0.3;     poli[3].z=0.0;
        poli[4].x=-0.3;     poli[4].y=-0.3;     poli[4].z=0.0;
        poli[5].x=0.3;      poli[5].y=-0.6;     poli[5].z=0.0;
        poli[6].x=0.4;      poli[6].y=0.0;      poli[6].z=0.0;
        poli[7].x=0.3;      poli[7].y=0.6;      poli[7].z=0.0;


/*Definicion de la ventana y el dibujo*/

        HC_Open_Segment("SONIDO"); /* se define el segmento del boton */
```

```
        if (*SONIDO_ESTATUS)  /*eleccion del color de fondo */
                HC_Set_Color("windows = gold, text = green, polylines = blue, faces = white");
        else
                HC_Set_Color("windows = black, text = white, polylines = green, faces = gray");
        /* se definen parametros para dibujo y texto*/
        HC_Set_Line_Weight(2.7);
        HC_Set_Text_Font("name = typewriter, size = 0.01 sru");
        HC_Set_Window(XMIN,XMAX,YMIN,YMAX);
        HC_QSet_Visibility("?HELP/CONTENTS/SONIDO","on");
        HC_Insert_Text(-0.9,-0.8,0.0,"Sonido");
        HC_Insert_Line(0.68,0.3,0.0,0.9,0.60,0.0);
        HC_Insert_Line(0.7,0.0,0.0,1.0,0.0,0.0);
        HC_Insert_Line(0.68,-0.3,0.0,0.9,-0.60,0.0);
        HC_Insert_Circular_Arc(a,b,c);
        HC_Insert_Circular_Arc(a,d,c);
        HC_Insert_Polyline(8,poli);
        HC_Close_Segment();
}
```

```
/*
Archivo gismoids_sound.c
        Aplicación de audio
*/

#if !defined(lint) && !defined(NOID)
static  char sccsid[] = "@(#)joseph.c      1.31        94/04/08 Copyr 1991 Sun Micro";
#endif

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <math.h>
#include <fcntl.h>
#include <stropts.h>
#include <sys/types.h>
#include <sys/param.h>
#include <unistd.h>
#ifdef SYSV
#include <dirent.h>
#include <sys/audioio.h>
#else
#include <sys/dir.h>
#include <sun/audioio.h>
#endif
#include <sys/file.h>
#include <sys/stat.h>
#include <sys/signal.h>
#include <sys/ioctl.h>

#include <xview/xview.h>
#include <xview/notice.h>
#include <xview/panel.h>
#include <xview/canvas.h>
#include <xview/pixwin.h>
#include <xview/scrollbar.h>
#include <xview/tty.h>
#include <xview/seln.h>
#include <xview/cursor.h>
#include <xview/screen.h>
#include <xview/server.h>
#include <xview/cms.h>

#include <multimedia/libaudio.h>
#include <multimedia/audio_device.h>
#include <multimedia/ulaw2linear.h>

Server_image gismoids_icon_image;
Server_image waveform_cursor_image;

static unsigned short gismoids_image_bits[] = {
#include "gismoids.icon"
};

static unsigned short waveform_cursor_bits[] = {
#include "/usr/openwin/share/include/images/ibeam.cursor"
};
```

```
#ifdef DEBUG
#define EVENTP(P, E)   (void) printf("%s: event %d\n", (P), event_id(E))
#define DEBUGF(args)   (void) printf args
#define PERROR(msg)    FPRINTF(stderr, "%s(%d): \"%s\" (%s)\n",      \
                              prog, __LINE__, msg, sys_errlist[errno]);
#else
#define EVENTP(P, E)
#define DEBUGF(args)
#define PERROR(msg)    FPRINTF(stderr, "%s: \"%s\" (%s)\n",   \
                              prog, msg, sys_errlist[errno]);
#endif  /*!DEBUG*/


#define REFRESH_RELOAD  (20)          /* solo renovar cada 20 veces */

#define BUTTON_WIDTH    (8)
#define SCROLLBAR_WIDTH (18)

#define WINDOW_WIDTH    (450)
#define SCOPE_HEIGHT    (256 + SCROLLBAR_WIDTH + 4)

#define SCOPE_WIDTH     (256)
#define SCOPE_MASK      (SCOPE_WIDTH - 1)
#define WAVEFORM_WIDTH  (1600)
#define WAVEFORM_HEIGHT (SCOPE_HEIGHT)

#define INFO_SIZE       (80)          /* maxima long. de string  */
#define MIN_BUFSIZE     (5)           /* minimo bufsize para ciclos */
#define MAX_GAIN        (100)         /* rango de ganancia */

#define NOTICE_FILE     "alert.au"    /* archivo de alerta*/
#define AUDIO_DEV       "/dev/audio"  /* dispositivo de audio */
#define AUDIO_CTLDEV    "/dev/audioctl" /* dispositivo de control de audio */

typedef struct Graph {
  Xv_Window     PW;
  Display       *dsp;
  XID           xid;
  GC            gc;
}           GraphType;

char          *prog;

Frame         Base_frame;
Panel         Main_panel;
Panel_item    Main_play_item;
Panel_item    Main_pause_item;
Panel_item    Main_salir_item;
Panel_item    Main_playgain_item;
Panel_item    Main_playport_item;
Panel_item    Loop_item;

Frame         Describe_frame;
Panel         Describe_panel;
Panel_item    Describe_delta_item;
Panel_item    Describe_length_item;
Panel_item    Describe_channel_item;
Panel_item    Describe_sample_item;
```

```
Panel_item     Describe_bits_item;
Panel_item     Describe_encoding_item;
Panel_item     Describe_info_item;

Panel          File_panel;
Panel_item     File_name_item;
Panel_item     File_directory_item;

Display *wc_display;
Drawable wc_drawable;
GC             wc_gc;
XGCValues wc_gcval;
Xv_Screen wc_screen;
int    wc_screen_no;

Display *sc_display;
Drawable sc_drawable;
GC             sc_gc;
XGCValues sc_gcval;
Xv_Screen sc_screen;
int    sc_screen_no;

Xv_Server gismoids_server;

int        Waveform_update_inhibit = FALSE;
int        Show_describe;      /* Verdadero si el panel de descripcion esta levantado */
int        Loop_flag;          /* Verdadero si se esta en el ciclo*/
int        Wait_flag;          /* OR of PLAY and RECORD bits */
int        Active_flag;        /* OR of PLAY and RECORD bits */
#define PLAY   1
#define RECORD  2

int        Audioctl_fd = -1;   /* descripcion de control de audio */
int        Audio_fd = -1;      /* descripcion de e/s para audio */
int        Alert_fd = -1;      /* descripcion de canal de alerta e/s */
int        Sync_sched = FALSE; /* programacion sincrona */
Audio_info     Audio_state;        /* estructura del manejador de audio */
Audio_hdr      Device_phdr;        /* guarda configuracion play */
Audio_hdr      Device_rhdr;        /* guarda configuracion record */
/*se define la ruta de los archivos de sonido*/
char ruta[80]="/home/dali/mecapli/socrates/socrates/\
                    FUENTES/gismoids/sonidos/gisma/";

struct sound_buffer {
        Audio_hdr          hdr;   /* informacion para decodificacion */
        char               info[INFO_SIZE + 1]; /* cadena de informacion */

        char               directory[MAXPATHLEN];
        char               filename[MAXPATHLEN];

        unsigned char      *data;      /* apuntador a bufer de datos */
        unsigned           alloc_size; /* tama&o de bufer de datos */
        int                draining;
        int                paused;

        struct {
                char       directory[MAXPATHLEN];
                time_t     modified;
#ifdef SYSV
                struct dirent  **files;
```

```
#else
                struct direct  **files;
#endif
                Menu        menu;
        }           menu;  /* menu escondido */

        struct {
                unsigned    start;  /* region para reproduccion */
                unsigned    end;
                unsigned    io_position;
        }           play;
        struct {
                int   cursor_pos;   /* posicion del cursor en el bufer */
                int   start;        /* marca de region seleccionada */
                int   end;
                int   position;
                int   last;
        }           display;
} Buffer;


/* funciones locales */
Notify_value   timer_handler();
Notify_value   sigpoll_async_handler();
Notify_value   sigpoll_sync_handler();
extern int scope_repaint_proc();

/* funciones externas  */
long        lseek();
#ifndef SYSV
int         alphasort();
#endif
extern char    *sys_errlist[];

/* Programa principal */
main(argc, argv, envp)
        int    argc;
        char   **argv;
        char   **envp;
{
        /* guarda el comando invocado para mensages de error  */
        prog = argv[0];
        if (argc != 2){
                printf("\nArgumentos invalidos\n");
                exit(1);
        }
        gismoids_server = xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);

        gismoids_icon_image = xv_create(NULL, SERVER_IMAGE,
          XV_WIDTH, 64,
          XV_HEIGHT, 64,
          SERVER_IMAGE_DEPTH, 1,
          SERVER_IMAGE_BITS, gismoids_image_bits,
          0);

        waveform_cursor_image = xv_create(NULL, SERVER_IMAGE,
          XV_WIDTH, 16,
          XV_HEIGHT, 16,
          SERVER_IMAGE_DEPTH, 1,
          SERVER_IMAGE_BITS, waveform_cursor_bits,
```

```
/* Rutinas que implementan el archivo del tablero de control */


/* Event handler for 'File:' item */
Panel_setting
file_name_proc(item, event)
        Panel_item      item;
        Event           *event;
{
        EVENTP("file_name_proc", event);

        if (event_action(event) == ACTION_MENU && event_is_down(event)) {
                file_name_menu(event);
        } else {
                (void) panel_default_handle_event(item, event);
        }
        return (panel_text_notify(item, event));
}

file_wait_cursor()
{
        xv_set(Base_frame, FRAME_BUSY, TRUE, 0);
}

file_restore_cursor()
{
        xv_set(Base_frame, FRAME_BUSY, FALSE, 0);
}

/* Error handler, just like it says */
file_error_handler(error_str)
        char            *error_str;
{
        generic_notice(error_str);
        file_restore_cursor();
}

file_menu_proc(menu, menu_item)
        Menu            menu;
        Menu_item       menu_item;
{
        xv_set(File_name_item, PANEL_VALUE, xv_get(menu_item, MENU_STRING), 0);
}

/*
 * Rutina selectora de archivo llamada scandir().
 * Regresa TRUE si el nombre de archivo es un archivo de audio o el nombre de
 * archivo termina en ".au" or ".snd".
 */
int
#ifdef SYSV
file_select(entry)
        struct dirent *entry
#else
file_select(entry)
        struct direct *entry;
#endif
{
        char            *ptr;
```

```
        PANEL_SLIDER_WIDTH, MAX_GAIN + 1,
        PANEL_NOTIFY_PROC, main_play_volume_proc,
        PANEL_NOTIFY_LEVEL, PANEL_ALL,
        PANEL_SHOW_RANGE,  FALSE,
        PANEL_SHOW_VALUE,  TRUE,
        0);


    Main_playport_item = xv_create(Main_panel, PANEL_CHOICE,
        XV_X, xv_col(Main_panel, 14),
        XV_Y, xv_row(Main_panel, 1),
        PANEL_LABEL_STRING, "Salida a:",
        PANEL_CHOICE_STRINGS, "Bocina", "Jack", 0,
        PANEL_NOTIFY_PROC, control_output_proc,
        0);

    Loop_item = xv_create(Main_panel, PANEL_CHOICE,
        XV_X, xv_col(Main_panel, 14),
        XV_Y, xv_row(Main_panel, 2),
        PANEL_LABEL_STRING, "Ciclico:  ",
        PANEL_CHOICE_STRINGS, "Apagado ", "Encendido", 0,
        PANEL_NOTIFY_PROC, control_loop_proc,
        0);

/*
    * Establece los valores iniciales para volumen y salida desde Audio_state,
    * el cual ya debera esta inicializado desde el estado del dispositivo
    */
    main_update_panel(TRUE);
    window_fit_width(Main_panel);
}
```

```
        }

        /* Usa el campo play.pause para indicar el estado actual de boton */
        Audio_new.play.pause = Buffer.paused;

        if (Audio_new.play.pause != Audio_state.play.pause) {
                (void) xv_set(Main_pause_item,
                  PANEL_LABEL_STRING,
                  (Buffer.paused ? "Resume" : "Pausa"), 0);
        }

        /* Copia el estado actula del display para el siguiente tiempo */
        Audio_state = Audio_new;
}

/* Crea el tablero de control principal */
main_create_panel()
{
        Main_panel = xv_create(Base_frame, PANEL,
          XV_LEFT_MARGIN,    10,
          XV_RIGHT_MARGIN,   10,
          WIN_ROWS,          6,
          0);

        /* Inicializa los botones que cambian su valor cuando son presionados*/
        Main_play_item = xv_create(Main_panel, PANEL_BUTTON,
          XV_X, xv_col(Main_panel, 0),
          XV_Y, xv_row(Main_panel, 0),
          PANEL_LABEL_STRING, "Reproduce",
          PANEL_NOTIFY_PROC, main_play_proc,
          0);

        Main_pause_item = xv_create(Main_panel, PANEL_BUTTON,
          XV_X, xv_col(Main_panel, 0),
          XV_Y, xv_row(Main_panel, 1),
          PANEL_LABEL_STRING, "Pausa",
          PANEL_NOTIFY_PROC, main_pause_proc,
          0);

        (void) xv_create(Main_panel, PANEL_BUTTON,
          XV_X, xv_col(Main_panel, 0),
          XV_Y, xv_row(Main_panel, 2),
          PANEL_LABEL_STRING, "Describe",
          PANEL_NOTIFY_PROC, main_describe_proc,
          0);

        Main_salir_item = xv_create(Main_panel, PANEL_BUTTON,
          XV_X, xv_col(Main_panel, 0),
          XV_Y, xv_row(Main_panel, 3),
          PANEL_LABEL_STRING, "SALIR",
          PANEL_NOTIFY_PROC, main_salir_proc,
          0);

        Main_playgain_item = xv_create(Main_panel, PANEL_SLIDER,
          XV_X, xv_col(Main_panel, 14),
          XV_Y, xv_row(Main_panel, 0),
          PANEL_LABEL_STRING, "Volumen Reproduce ",
          PANEL_MIN_VALUE, 0,
          PANEL_MAX_VALUE, MAX_GAIN,
          PANEL_SHOW_RANGE,  FALSE,
```

```
{
        EVENTP("control_loop_proc", event);

        Loop_flag = (value != 0);
        if (Loop_flag) {
                /* No poner looping si es muy peque&o el buffer */
                (void) selectcheck();

                /* mantener si ya esta drenando */
                if (Active_flag & PLAY)
                        Buffer.draining = FALSE;
        }
}

/*
 * Actualiza la variable de controles de audio en el tablero principal desde
 * su estado actual
 * LLamado cuando algun cambio de parametro pudo haber ocurrido
 */
main_update_panel(init)
        int         init;
{
        Audio_info     Audio_new;

        /* Solo pone los valores si estos han sido cambiados (prevents flashing) */
#define NEWVAL(X, Y)    {                               \
                        if (Audio_new.X != Audio_state.X)           \
                                (void) xv_set(Y, PANEL_VALUE, Audio_new.X, 0);\
                        }

        if (!audio_readstate(&Audio_new) && !init)
                return;

        NEWVAL(play.gain, Main_playgain_item);
        NEWVAL(play.port, Main_playport_item);

        /* Asegurate que el boton pausa/reactivar esta en sincronia con
         * la realidad */
        /*
         * XXX - Pausa debe salvar el estado y cerrar el dispositivo de audio.
         * con la finalidad de permitir a otras aplicaciones de audio accesar
         * a /dev/audio.
         * Note que, si esto cerrara /dev/audio, 'gaintool' no podria ya
         * actualizar este programa.
         */
        if (Active_flag & PLAY)
                Buffer.paused = Audio_new.play.pause;

        /* Si no esta activo, asume el estado depausa solo si ambas banderas
          estan puestas */
        if (!Active_flag) {
                Buffer.paused = Audio_new.play.pause && Audio_new.record.pause;
        } else if (Active_flag & PLAY) {
                /* apaga el temporizador cuando se esta en pausa; se reestablece*/
                /* con resume*/
                if (Buffer.paused)
                        cancel_timer();
                else
                        set_timer((double)SCOPE_WIDTH /
                          (double)Buffer.hdr.sample_rate);
```

```
/* Convierte la ganancia local a parametros del dispositivo */
double
scale_gain(g)
        int    g;
{
        return ((double)g / (double)MAX_GAIN);
}

/* Convierte la ganancia del dispositivo a el factor de escala local */
int
unscale_gain(g)
        double  g;
{
        return (irint((double)MAX_GAIN * g));
}


/* Activar el cursor de volumen movido */
/*ARGSUSED*/
main_play_volume_proc(item, value, event)
        Panel_item      item;
        int             value;
        Event           *event;
{
        double          gain;

        EVENTP("main_play_volume_proc", event);

        /* Let SIGPOLL handler adjust displayed value */
        gain = scale_gain(value);
        Audio_state.play.gain = ~0;
        (void) audio_set_play_gain(Audioctl_fd, &gain);
}

/* Grabar cursor del volumen movido */
/*ARGSUSED*/

/* 'Output to:' cambiado */
/*ARGSUSED*/
control_output_proc(item, value, event)
        Panel_item      item;
        int             value;
        Event           *event;
{
        unsigned        port;

        EVENTP("control_output_proc", event);

        /* Cambia el puerto de salida */
        Audio_state.play.port = (unsigned) value;
        port = (value == 0 ? AUDIO_SPEAKER : AUDIO_HEADPHONE);
        (void) audio_set_play_port(Audioctl_fd, &port);
}

/* 'Looping:' cambiado */
/*ARGSUSED*/
control_loop_proc(item, value, event)
        Panel_item      item;
        int             value;
        Event           *event;
```

```
    Event *event;
{
        EVENTP("main_pause_proc", event);

        if (!Active_flag)
                return;

        /*Poner el estado del dispositivo de audio del lado contrario al
         *estado actual y permite al manejador SIGPOLL actualizar los botones
         */
        if (Active_flag & PLAY) {
                if (Buffer.paused)
                        (void) audio_resume_play(Audioctl_fd);
                else
                        (void) audio_pause_play(Audioctl_fd);
        }
}

/* Descripcion del boton presionado */
/*ARGSUSED*/
main_describe_proc(item, event)
        Panel_item      item;
        Event           *event;
{
        int             x;

        EVENTP("main_describe_proc", event);

        if (Show_describe) {
                Show_describe = FALSE;
                (void) xv_set(Describe_frame, WIN_SHOW, FALSE, 0);
                return;
        }

        Show_describe = TRUE;
        describe_update_panel(TRUE);

        /* Localiza el tablero descrito en la esquina superior derecha del
          marco*/

        x = (int) xv_get(Base_frame, XV_WIDTH) -
           (int) xv_get(Describe_frame, XV_WIDTH);
        (void) xv_set(Describe_frame,
          WIN_X,          x,
          WIN_Y,          0,
          WIN_SHOW,       TRUE,
          0);
}
/*   Salir boton presionado */

main_salir_proc(item,event)
        Panel_item      item;
        Event           *event;
{
        xv_set(Main_salir_item, PANEL_LABEL_STRING, "Adios", 0);
        exit(0);
}
```

```
}

/* Rutinas que forman el tablero de control principal */

/* Activar/desactivar boton presionado */
/*ARGSUSED*/
main_play_proc(item, event)
        Panel_item      item;
        Event           *event;
{
        unsigned int    i;

        EVENTP("main_play_proc", event);


        if (Wait_flag & PLAY) {
                stop_play();            /* nunca obtiene el dispositivo */
                return;
        } else if (Active_flag & PLAY) {   /* Alto boton presionado */
                stop_play();
                if (Audio_state.play.error) {
                        message_display(" Se detecto Underflow.");
                }
                return;
        }

        /* Apagar el ciclo si la seleccion es muy peque&a */
        if (selectcheck() == 0)
                return;             /* no hay datos en el buffer */

        switch (audio_open(PLAY)) {
        case 0:                 /* abre satisfactoriamente */
                break;

        case 1:                 /* abrir regresando ESPERA */
                Wait_flag |= PLAY;      /* SIGPOLL is sent on close() */
                xv_set(Main_play_item, PANEL_LABEL_STRING, "En_espera", 0);

        /* Pone la bandera de espera, la cual seria puesta si hubieramos
         * hecho un blocking open(), fuera de la opurtunidad para retener
         * el dispositivo de audio.
         */
        i = TRUE;
        (void) audio_set_play_waiting(Audioctl_fd, &i);
                return;

        case -1:                /* error al abrir audio */
        default:
                message_display("Error al inicializar audio.");
                return;
        }

        start_play();
}


/* Pausa/reanudar boton presionado */
/*ARGSUSED*/
main_pause_proc(item, event)
   Panel_item item;
```

```
        0);

    /* Crea ventana base */
    Base_frame = xv_create((Xv_Window) 0, FRAME,
      WIN_ROWS, 35,
      XV_WIDTH, WINDOW_WIDTH,
      XV_LABEL, "CONTROL DE SONIDO GISMOIDS",
      FRAME_ICON,
          xv_create(XV_NULL, ICON,
            ICON_IMAGE, gismoids_icon_image,
            ICON_TRANSPARENT, TRUE,
            NULL),
      0);

    if (Base_frame == (Frame) 0) {
          PERROR("No puedo obtener el recuadro base");
          exit(1);
    }

    /* Captura notificaiones de SIGPOLL asincronamente */
    (void) notify_set_signal_func(Base_frame,
      (Notify_func)sigpoll_async_handler, SIGPOLL, NOTIFY_ASYNC);

    /* Captura eventos para manejar programa  */
    (void) notify_set_event_func(SIGPOLL,
      (Notify_func)sigpoll_sync_handler, NOTIFY_SAFE);

    /* Abre control de audio para volumen */
    audio_control_init();
    init_buffer();



    /* lee el archivo de datos */
    soundfile_read(argv[1]);


    /* Crea el tablero principal */
    main_create_panel();
    file_create_panel();
    /* Crea el tablero pop-up */
    describe_create_panel();
    window_fit_height(Base_frame);
    xv_set(Base_frame, WIN_SHOW, TRUE, 0);

    window_main_loop(Base_frame);

    /*  Limpia la salida de audio y cierra el dispositivo de audio si este esta abierto */
    closedown();
/*NOTREACHED*/
}

/* Detiene y sale */
closedown()
{
    if ((Active_flag & PLAY) || (Wait_flag & PLAY))
          stop_play();
    if ((Active_flag & RECORD) || (Wait_flag & RECORD))
    exit(0);
/*NOTREACHED*/
```

```
if (stat(Buffer.directory, &st) < 0 || !(S_ISDIR(st.st_mode))) {
        file_error_handler("Directorio invalido");
        return;
}

/* Si ya existe un menu, revisa que todavia sirva */
if (Buffer.menu.menu != NULL) {
        if ((strcmp(Buffer.directory, Buffer.menu.directory) != 0) ||
          (st.st_mtime != Buffer.menu.modified)) {
                /* Directorio cambiado...se deshace del menu viejo */
                if (Buffer.menu.files != NULL) {
                        (void) free((char *)Buffer.menu.files);
                        Buffer.menu.files = NULL;
                }
                xv_destroy(Buffer.menu.menu);
                Buffer.menu.menu = NULL;
        }
}

if (Buffer.menu.menu == NULL) {
        /* Cambia el cursor a un reloj de arena y crea un nuevo menu */
        file_wait_cursor();
        Buffer.menu.modified = st.st_mtime;
        STRCPY(Buffer.menu.directory, Buffer.directory);

        Buffer.menu.menu = xv_create(NULL, MENU,
          MENU_TITLE_ITEM, "Audio files:",
          MENU_NOTIFY_PROC, file_menu_proc,
          0);

        count = scandir(Buffer.menu.directory, &Buffer.menu.files,
          file_select, alphasort);
        if (count < 0)
                Buffer.menu.files = NULL;

        /* Si no encuentra archivos, crea un objeto del menu no-seleccionable */
        if (count <= 0) {
                /* Crea un objeto vacio si el layout del menu falla */
                mi = (Menu_item) xv_create(XV_NULL, MENUITEM,
                  MENU_STRING, "",
                  MENU_RELEASE,
                  MENU_RELEASE_IMAGE,
                  MENU_FEEDBACK, FALSE,
                  0, 0);
                (void) xv_set(Buffer.menu.menu,
                  MENU_TITLE_ITEM, "No hay archivos de audio en este directorio",
                  MENU_APPEND_ITEM, mi, 0);
        }

        /* Agrega un objeto menu para cada archivo de audio encontrado */
        for (i = 0; i < count; i++) {
                mi = (Menu_item) xv_create(XV_NULL, MENUITEM,
                  MENU_STRING, Buffer.menu.files[i]->d_name,
                  MENU_RELEASE,
                  MENU_RELEASE_IMAGE,
                  MENU_NOTIFY_PROC, file_menu_proc,
                  0, 0);
                (void) xv_set(Buffer.menu.menu,
                  MENU_APPEND_ITEM, mi, 0);
        }
```

```
        char        tmp[MAXPATHLEN];

        if ((strcmp(entry->d_name, ".") == 0) ||
            (strcmp(entry->d_name, "..") == 0))
                return (FALSE);

        /* Busca los archivos con extension .au o .snd */
        ptr = strrchr(entry->d_name, '.');
        if ((ptr != NULL) &&
            ((strcmp(ptr, ".au") == 0) || (strcmp(ptr, ".snd") == 0)))
                return (TRUE);

        /* Busca el encabezado del archivo de audio*/
        if (Buffer.directory[0] == '\0')
                STRCPY(Buffer.directory, ".");
        SPRINTF(tmp, "%s/%s", Buffer.directory, entry->d_name);
        return (audio_isaudiofile(tmp));
}

#ifdef SYSV
/*
 * revisa la existencia de archivos
 */
scandir(dirname, namelist, select, dcomp)
        char        *dirname;
        struct dirent  *(*namelist[]);
        int         (*select)();
        int         (*dcomp)();
{
        register struct dirent *d, *p, **names;
        register int nitems;
        register char *cp1, *cp2;
        struct stat stb;
        long arraysz;
        DIR *dirp;

        if ((dirp = opendir(dirname)) == NULL)
                return (-1);
        if (fstat(dirp->dd_fd, &stb) < 0)
                return (-1);

        /*
         * estima el tama&o del arreglo tomando el tama&o del directorio
         * y dividiendolo en multiples entradas de tama&o minimo.
         */
        arraysz = (stb.st_size / 24);
        names = (struct dirent **)malloc(arraysz * sizeof (struct dirent *));
        if (names == NULL)
                return (-1);

        nitems = 0;
        while ((d = readdir(dirp)) != NULL) {
                if (select != NULL && !(*select)(d))
                        continue;       /* nombres recien seleccionados*/
                /*
                 * Crea una copia comprimida de los datos
                 */
                p = (struct dirent *)malloc(d->d_reclen);
                if (p == NULL)
                        return (-1);
```

```
                p->d_ino = d->d_ino;
                p->d_reclen = d->d_reclen;
                p->d_off = d->d_off;
                for (cp1 = p->d_name, cp2 = d->d_name; *cp1++ = *cp2++; );
                /*
                 * Se asegura que el arrglo tenga espacio para alojar
                 *  el tama&o maximo
                */
                if (++nitems >= arraysz) {
                        if (fstat(dirp->dd_fd, &stb) < 0)
                                return (-1);    /* por si crece */
                        arraysz = stb.st_size / 12;
                        names = (struct dirent **)realloc((char *)names,
                          arraysz * sizeof (struct dirent *));
                        if (names == NULL)
                                return (-1);
                }
                names[nitems-1] = p;
        }
        closedir(dirp);
        if (nitems && dcomp != NULL)
                qsort(names, nitems, sizeof (struct dirent *), dcomp);
        *namelist = names;
        return (nitems);
}

/*
 * rutina de comparacion por orden alfabetico por si es requerido
*/
alphasort(d1, d2)
        struct dirent **d1, **d2;
{
        return (strcmp((*d1)->d_name, (*d2)->d_name));
}
#endif /* SYSV */

/* Construye y despliega un menu para los archivos de audio en el directorio
 *  actual
*/
file_name_menu(event)
        Event       *event;
{
        int         i;
        int         selection;
        int         count;
        int         columns;
        struct stat    st;
        Menu        menu;
        Menu_item mi;

        EVENTP("file_name_menu", event);

        /* Obtiene el nombre del directorio en caso de que cambie */
        STRCPY(Buffer.directory,
          (char *) xv_get(File_directory_item, PANEL_VALUE));

        if (Buffer.directory[0] == '\0')
                STRCPY(Buffer.directory, ".");

        /* Si es un directorio invalido no sigue buscando */
```

```
                /* Calcula el numero de columnas en el menu. Queremos un menu que
                 * sea aproximada mento un cuadrado. Asumimos que el nombre tipico
                 * de archivo es de 12 caracteres y que la distancia entre caracteres
                 * es el dos veces la distancia entre lineas
                */
                columns = sqrt((double)i / 6);
                xv_set(Buffer.menu.menu, MENU_NCOLS, columns, 0);

                /* Restablece el cursor */
                file_restore_cursor();
        }

        /* Despliega el menu, obtiene la seleccion (si la hay(, y gurada el menu */
        menu_show(Buffer.menu.menu, File_panel, event, 0);
}

file_create_panel()
{
        extern char **environ;

        File_panel = xv_create(Base_frame, PANEL,
          WIN_ROWS,       6,
          WIN_RIGHT_OF,     Main_panel,
          0);

/*      (void) xv_create(File_panel, PANEL_BUTTON,
          XV_X,     xv_col(File_panel, 0),
          XV_Y,     xv_row(File_panel, 0),
          PANEL_LABEL_STRING, "Carga",
          PANEL_NOTIFY_PROC, file_load_proc,
          0);

        (void) xv_create(File_panel, PANEL_BUTTON,
          XV_X,     xv_col(File_panel, 0),
          XV_Y,     xv_row(File_panel, 1),
          PANEL_LABEL_STRING, "Guarda",
          PANEL_NOTIFY_PROC, file_store_proc,
          0);

        (void) xv_create(File_panel, PANEL_BUTTON,
          XV_X,     xv_col(File_panel, 0),
          XV_Y,     xv_row(File_panel, 2),
          PANEL_LABEL_STRING, "Agrega",
          PANEL_NOTIFY_PROC, file_append_proc,
          0);

        File_directory_item = xv_create(File_panel, PANEL_TEXT,
          XV_X,     xv_col(File_panel, 0),
          XV_Y,     xv_row(File_panel, 3),
          PANEL_VALUE, getenv("PWD="),
          PANEL_VALUE_STORED_LENGTH, 80,
          PANEL_VALUE_DISPLAY_LENGTH, 30,
          PANEL_LABEL_STRING, "Directorio: ",
          0);

        File_name_item = xv_create(File_panel, PANEL_TEXT,
          XV_X,     xv_col(File_panel, 0),
          XV_Y,     xv_row(File_panel, 4),
          PANEL_VALUE_STORED_LENGTH, 80,
```

```
            PANEL_VALUE_DISPLAY_LENGTH, 30,
            PANEL_LABEL_STRING, "Archivo: ",
            PANEL_EVENT_PROC,  file_name_proc,
            0);

}

/* Aloja el buffer para almacenar datos */
alloc_buffer(size)
        unsigned int   size;
{
        if (Buffer.data != NULL)
                (void) free((char *)Buffer.data);

        /* Aloja el buffer, comprimiendo si los datos son muchos */
        do {
                Buffer.data = (unsigned char *)malloc(size);
        } while ((Buffer.data == NULL) && (size = size - (size / 8)));

        Buffer.alloc_size = size;
}

/* Inicializa el buffer */
init_buffer()
{
        if (Buffer.data != NULL)
                (void) free((char *)Buffer.data);
        Buffer.data = NULL;
        Buffer.alloc_size = 0;
        Buffer.hdr = Device_phdr;
        Buffer.hdr.data_size = 0;
}

/*
 * Convierte el panel de archivos y directorios en una ruta.
 * Regresa verdadero si no se especifica un nombre de archivo
 */
int
soundfile_path(str)
        char        *str;
{
        STRCPY(Buffer.directory,
          (char *) xv_get(File_directory_item, PANEL_VALUE));
        STRCPY(Buffer.filename, (char *) xv_get(File_name_item, PANEL_VALUE));

        if (Buffer.filename[0] == '\0')
                return (TRUE);

        /* Necesita hacer esto en caso de que el usuario limpie la cadena del
          directorio */
        if (Buffer.directory[0] == '\0')
                STRCPY(Buffer.directory, ".");

        SPRINTF(str, "%s/%s", Buffer.directory, Buffer.filename);
        return (FALSE);
}

/* Abre el archivo de sonido y lo lee en memoria. Regresa verdadero si hay error*/
int
soundfile_read(archivo)
```

```
char *archivo;
{
int         fd;
unsigned    size;
int         valid;
char        msg[256];
struct stat st;


strcat(ruta,archivo);
printf("%s\n",ruta);

if (((fd = open(ruta, O_RDONLY)) < 0) || (fstat(fd, &st) < 0)) {
        SPRINTF(msg, "Can't read '%s' (%s).", ruta, sys_errlist[ermo]);
        message_display(msg);
        return;
}

/*
 * Si el archivo de sonido tiene encabezado, lo lee y ecodifica
 */
valid = (AUDIO_SUCCESS == audio_read_filehdr(fd, &Buffer.hdr,
   Buffer.info, sizeof (Buffer.info)));
if (valid) {
        if (Buffer.hdr.data_size == AUDIO_UNKNOWN_SIZE) {
                /* Calculate the data size, if not already known */
                /* Calcula el tama&o de datos, si aun no se conoce */
                Buffer.hdr.data_size =
                   st.st_size - lseek(fd, 0L, SEEK_CUR);
        }
} else {
        /* Si no hay encabezado, lee el renglon del archivo y asume
          compatibilidad */
        Buffer.hdr  = Device_phdr;     /* utiliza la configuracion del
                                                  dispositivo*/
        (void) lseek(fd, 0L, L_SET);   /* lleva el apuntador al inicio
                                                  del archivo */
        Buffer.hdr.data_size = st.st_size - lseek(fd, 0L, SEEK_CUR);
        Buffer.info[0] = '\0';
}

/* Establece la cadena de informacion a desplegar */


/* Si se tienen datos de salida, se establece la bandera de drenado para
 * que play_service() no trate de accesar un buffer obsoleto. file_update*(
 * apagara la bandera de drenado si la salida permanece activa
*/
if (Active_flag & PLAY)
        Buffer.draining = TRUE;

/* Libera el buffer viejo y aloja un nuevo para almacenar los datos */
size = Buffer.hdr.data_size;
alloc_buffer(size);

/* Lee tantos datos como sea posible y cierra el archivo */
Buffer.hdr.data_size = read(fd,
   (char *)Buffer.data, (int)Buffer.alloc_size);
(void) close(fd);
```

```
        Buffer.display.start = Buffer.play.start = 0;
        Buffer.display.end = Buffer.play.end = Buffer.hdr.data_size;
        file_update();          /* despliega un nuevo archivo */
                /* Si no se puede alojar o cargar el archivo completo, muestra mensaje*/
        if (size != Buffer.hdr.data_size) {
                SPRINTF(msg, "%.2f segundos de datos truncados de '%s'.",
                  audio_bytes_to_secs(&Buffer.hdr,
                  (size - Buffer.hdr.data_size)), ruta);
                message_display(msg);
        }

        /* Si el archivo no es un archivo de datos, despliega advertencia */
        if (!valid) {
                SPRINTF(msg, "'%s' no es un archivo de audio valido  %s\n", ruta,
                  "STORE lo convertira en archivo de audio.");
                message_display(msg);
        } else if ((Buffer.hdr.encoding != AUDIO_ENCODING_ULAW) ||
          (Buffer.hdr.bytes_per_unit != 1) ||
          (Buffer.hdr.samples_per_unit != 1) ||
          (Buffer.hdr.channels != 1)) {
                SPRINTF(msg,
          "'%s' el audio no puede ser desplegado correctmente \n",
                  ruta);
                message_display(msg);
        }
}

/* Escribe o agrega al archivo de audio */
soundfile_write(append)
        int         append;
                                /* Verdadero si agrega a un archivo existente*/
        int         fd;
        int         bytes;
        int         exists;
        int         err;
        char        *info;
        struct stat   st;
        char        msg[256];
        char        path[MAXPATHLEN];
        Audio_hdr     tmphdr;

        bytes = buffer_selected_size();
        if (bytes <= 0)
                return;

        /* Obtiene la ruta del archivo de audio y se asegura que existe
          un nombre de archivo */
        if (soundfile_path(path)) {
                message_display("Archivo no especificado.");
                return;
        }
        exists = (stat(path, &st) == 0);
        if (!exists)
                append = FALSE;

        DEBUGF(("%s (%d bytes) to file >%s<\n",
          (append ? "Agregando" : "Escribiendo"), bytes, Buffer.filename));

        if (append) {
                /* Agrega a un archivo existente */
```

```
                if ((fd = open(path, O_RDWR)) < 0) {
                        SPRINTF(msg, "No puedo abrir '%s' (%s).",
                          Buffer.filename, sys_errlist[errno]);
                        message_display(msg);
                        return;
                }

                /* Asegura que este ya es un archivo de audio */
                if (!S_ISREG(st.st_mode) ||
                  (audio_read_filehdr(fd, &tmphdr, (char *)NULL, 0) !=
                  AUDIO_SUCCESS)) {
                        SPRINTF(msg, "'%s' no es un archivo de audio valido.",
                          Buffer.filename, sys_errlist[errno]);
                        message_display(msg);
                        goto closerr;
                }

                if ((int)lseek(fd, st.st_size, SEEK_SET) < 0)
                        goto writerr;

        } else {
                /* Crea un nuevo archivo */
                if (exists) {
                        if (!message_confirm(
                          "El archivo existente sera sobreescrito."))
                                goto closerr;
                }

                /* Obtiene la cadena de informacion actual*/
                info = (char *) xv_get(Describe_info_item, PANEL_VALUE);

                fd = open(path, O_WRONLY | O_CREAT | O_TRUNC, 0666);

                /* escribe el encabezado del archivo de audio primero */
                tmphdr = Buffer.hdr;
                tmphdr.data_size = bytes;
                if ((fd < 0) || (audio_write_filehdr(fd, &tmphdr,
                  info, ((unsigned)strlen(info) + 1)) != AUDIO_SUCCESS))
                        goto writerr;
        }

        err = (write(fd, (char *)&Buffer.data[Buffer.play.start], bytes) !=
          bytes);
        if (append && !err) {
                if ((tmphdr.data_size != AUDIO_UNKNOWN_SIZE) &&
                  (audio_rewrite_filesize(fd, (tmphdr.data_size + bytes)) !=
                  AUDIO_SUCCESS))
                        err++;
        }

        if (err) {
writerr:
                SPRINTF(msg, "No puedo %s hacia '%s' (%s).",
                  (append ? "agrega" : "escribe"), Buffer.filename,
                  sys_errlist[errno]);
                message_display(msg);
        }
closerr:
        (void) close(fd);
}
```

```
/* Rutinas que implementan la descripcion de archivos en el panel pop-up */

/* Llamado cuando la descripcion pop-up se destruye */
/*ARGSUSED*/
Notify_value
describe_destroy_proc(frame)
        Frame        *frame;
{
        (void) xv_set(Describe_frame, WIN_SHOW, FALSE, 0);
        Show_describe = FALSE;
        return (NOTIFY_DONE);
}

/* Convierte los objetos en de solo lectura */
/*ARGSUSED*/
null_panel_event(item, event)
        Panel_item     item;
        Event        *event;
{
}

/* Inicializa el panel pop-up */
describe_create_panel()
{
#define DWIDTH  25

        Describe_frame = xv_create(Base_frame, FRAME,
          XV_LABEL,          "    Estatus  Buffer de Audio",
          FRAME_SHOW_LABEL,  TRUE,
          FRAME_DONE_PROC,   describe_destroy_proc,
          XV_LEFT_MARGIN,    10,
          XV_WIDTH,          xv_col(Base_frame, 15 + DWIDTH),
          0);

        Describe_panel = xv_create(Describe_frame, PANEL,
          0);

        Describe_sample_item = xv_create(Describe_panel, PANEL_TEXT,
          XV_X,              xv_col(Describe_panel, 0),
          XV_Y,              xv_row(Describe_panel, 0),
          PANEL_VALUE_X,     xv_col(Describe_panel, 15),
          PANEL_VALUE_DISPLAY_LENGTH, DWIDTH,
          PANEL_LABEL_STRING, "Muestreo :  ",
          PANEL_EVENT_PROC,  null_panel_event,
          0);

        Describe_channel_item = xv_create(Describe_panel, PANEL_TEXT,
          XV_X,              xv_col(Describe_panel, 0),
          XV_Y,              xv_row(Describe_panel, 1),
          PANEL_VALUE_X,     xv_col(Describe_panel, 15),
          PANEL_VALUE_DISPLAY_LENGTH, DWIDTH,
          PANEL_LABEL_STRING, "Canales :  ",
          PANEL_EVENT_PROC,  null_panel_event,
          0);

        Describe_bits_item = xv_create(Describe_panel, PANEL_TEXT,
          XV_X,              xv_col(Describe_panel, 0),
          XV_Y,              xv_row(Describe_panel, 2),
          PANEL_VALUE_X,     xv_col(Describe_panel, 15),
```

```
        PANEL_VALUE_DISPLAY_LENGTH, DWIDTH,
        PANEL_LABEL_STRING, "Bits   :  ",
        PANEL_EVENT_PROC,  null_panel_event,
        0);

    Describe_encoding_item = xv_create(Describe_panel, PANEL_TEXT,
        XV_X,           xv_col(Describe_panel, 0),
        XV_Y,           xv_row(Describe_panel, 3),
        PANEL_VALUE_X,    xv_col(Describe_panel, 15),
        PANEL_VALUE_DISPLAY_LENGTH, DWIDTH,
        PANEL_LABEL_STRING, "Codificando: ",
        PANEL_EVENT_PROC,  null_panel_event,
        0);

    Describe_length_item = xv_create(Describe_panel, PANEL_TEXT,
        XV_X,           xv_col(Describe_panel, 0),
        XV_Y,           xv_row(Describe_panel, 4),
        PANEL_VALUE_X,    xv_col(Describe_panel, 15),
        PANEL_VALUE_DISPLAY_LENGTH, DWIDTH,
        PANEL_LABEL_STRING, "Longitud total: ",
        PANEL_EVENT_PROC,  null_panel_event,
        0);

    Describe_delta_item = xv_create(Describe_panel, PANEL_TEXT,
        XV_X,           xv_col(Describe_panel, 0),
        XV_Y,           xv_row(Describe_panel, 5),
        PANEL_VALUE_X,    xv_col(Describe_panel, 15),
        PANEL_VALUE_DISPLAY_LENGTH, DWIDTH,
        PANEL_LABEL_STRING, "Seleccion:  ",
        PANEL_EVENT_PROC,  null_panel_event,
        0);

    Describe_info_item = xv_create(Describe_panel, PANEL_TEXT,
        XV_X,           xv_col(Describe_panel, 0),
        XV_Y,           xv_row(Describe_panel, 6),
        PANEL_VALUE_X,    xv_col(Describe_panel, 15),
        PANEL_VALUE_STORED_LENGTH,  (INFO_SIZE - 1),
        PANEL_VALUE_DISPLAY_LENGTH, DWIDTH,
        PANEL_LABEL_STRING, "Cadena de Inf.: ",
        0);

    /* Pone la caret en la cadena de informacion */
    (void) panel_backup_caret(Describe_panel);

    window_fit(Describe_panel);
    window_fit(Describe_frame);
}

/* Actualiza el panel de descripcion con la descripcion de el archivo que
 * se encuentra en memoria.
 * Si 'init' es verdadero, establece todos los valors. De otro modo, solo
 * establece la longitud
*/
describe_update_panel(init)
    int   init;
{
    char   sample_string[80];
    char   bit_string[80];
    char   channel_string[80];
    char   encoding_string[80];
```

```
    char    length_string[80];

    /* Si no es visibe, no importa */
    if (!Show_describe)
            return;

    if (init) {
            SPRINTF(sample_string, "%d", Buffer.hdr.sample_rate);
            SPRINTF(bit_string, "%d", ((8 * Buffer.hdr.bytes_per_unit) /
              Buffer.hdr.samples_per_unit));
            SPRINTF(channel_string, "%d", Buffer.hdr.channels);

            switch (Buffer.hdr.encoding) {
            case AUDIO_ENCODING_ULAW:
                    STRCPY(encoding_string, "u-law");
                    break;
            case AUDIO_ENCODING_ALAW:
                    STRCPY(encoding_string, "A-law");
                    break;
            default:
                    SPRINTF(encoding_string, "unknown (%d)",
                      Buffer.hdr.encoding);
                    break;
            }

            xv_set(Describe_sample_item,
                    PANEL_VALUE, sample_string,
                    0);

            xv_set(Describe_bits_item,
                    PANEL_VALUE, bit_string,
                    0);

            xv_set(Describe_channel_item,
                    PANEL_VALUE, channel_string,
                    0);

            xv_set(Describe_encoding_item,
                    PANEL_VALUE, encoding_string,
                    0);
    }
    if (Buffer.hdr.data_size == 0) {
            STRCPY(length_string, "(buffer empty)");
    } else {
            char    tmp[AUDIO_MAX_TIMEVAL];

            SPRINTF(length_string, "%s",
              audio_secs_to_str(audio_bytes_to_secs(&Buffer.hdr,
              Buffer.hdr.data_size), tmp, 2));
    }

    xv_set(Describe_length_item, PANEL_VALUE, length_string, 0);
}
```

```
/* Redibuja la pantalla, obteniendo los datos del buffer especificado.
 * Si el cursor esta activo sobre el panel de forma de onda, muestra su
 * posicion.
*/
/* Comienza con la ejecucion de los datos (El dispositivo de audio esta abierto)*/
start_play()
{
        /* Verifica el buffer */
        if (selectcheck() <= 0) {
                stop_play();
                return;
        }
        xv_set(Main_play_item, PANEL_LABEL_STRING, "Alto", 0);

        Buffer.play.io_position = Buffer.play.start;
        Buffer.display.last = -1;
        Buffer.draining = FALSE;

        Active_flag |= PLAY;

        /* Establece el tiempo para apagar */
        set_timer((double)SCOPE_WIDTH / (double)Buffer.hdr.sample_rate);

        /* SIGPOLL apaga play_service() */
        (void) kill(getpid(), SIGPOLL);
}

/* Manipulador asincrono SIGPOLL. No puede llamar a ningun programa SunView */
/*ARGSUSED*/
Notify_value
sigpoll_async_handler(client, sig, when)
        Notify_client        client;
        int                  sig;
        Notify_signal_mode   when;
{
        int                  save_errno;

        DEBUGF(("sigpoll_handler: %s\n",
          (Wait_flag ? "Waiting for open()" : "")));

        save_errno = errno;    /* XXX.- Solucion alterna, bug registrado*/

        if (!Wait_flag) {
                /* El dispositivo esta abierto. Intenta dejar las colas llenas*/
                if (Active_flag & PLAY)
                        play_service();

        }

        /*
        *SIGPOLL es tambien enviado si el estado del dispositivo cambia.
        * Pone el horario para el manipulador sincorono.
        */
        if (!Sync_sched) {
                Sync_sched = TRUE;
                (void) notify_post_event(SIGPOLL, NULL, NOTIFY_SAFE);
        }
```

```
        errno = save_errno;
        return (NOTIFY_DONE);  /* XXX - Solucion alterna bug notificado */
}

/*
* Esta es la rutina que se llama desde el loop principal que escribe sonido
* al dispositivo.
*
* Necesita el siguient estado de datos:
*     - punto de inicio y finalizacion del cursor (calculado en proc)
*     - Posicion actual en el buffer
*/

play_service()
{
        int   start;
        int   end;
        int   outcnt;
        int   rtn;

        if (Buffer.draining)
                return;
again:
        start = Buffer.play.io_position;
        end = Buffer.play.end;

        outcnt = end - start;

        while (outcnt > 0) {
                /* Escribe tantos datos como es posible */
                rtn = write(Audio_fd, (char *)&Buffer.data[start], outcnt);

                if (rtn > 0) {
                        outcnt -= rtn;
                        start += rtn;
                        Buffer.play.io_position = start;
                } else {
                        break;
                }
        }

        /* Busca la condicion de finalizacion del sonido */
        if (outcnt == 0) {
                if (Loop_flag && (Buffer.play.start < Buffer.play.end)) {
                        Buffer.play.io_position = Buffer.play.start;
                        goto again;
                } else if (!Buffer.draining) {
                        Buffer.draining = TRUE; /* drena si no esta en el ciclo */
                        /* Escribe una marca EOF (Fin de Archivo) */
                        (void) write(Audio_fd, (char *)&Buffer.data[0], 0);
                }
        }
}

/* Esta rutina se llaman cuando las marcas del bufer son cambiadas */
play_update_cursor()
{
        Audio_info    tmpinfo;

        /* No se necesita tomar accion si no se esta ejecutando actualmente */
```

```
        if (!(Active_flag & PLAY))
                return;

        /* Si los puntos de inicio y finalizacion son los mismo, se captura aqui*/
        if (Buffer.play.end <= Buffer.play.start) {
                stop_play();
                return;
        }

        /* Apaga el manipulador SIGPOLL por ahora */
        Active_flag &= ~PLAY;

        /* Vacia la cola y empieza de nuevo */
        (void) audio_flush_play(Audio_fd);
        Buffer.draining = FALSE;
        Buffer.play.io_position = Buffer.play.start;
        Active_flag |= PLAY;

        /* Reestablece los contadores de sampleo, error y cuenta de fin de archivo */
        /* SIGPOLL empezara la ejecucion */
        AUDIO_INITINFO(&tmpinfo);
        tmpinfo.play.eof = 0;
        tmpinfo.play.error = 0;
        tmpinfo.play.samples = 0;
        (void) audio_setinfo(Audio_fd, &tmpinfo);
}

/* Esta rutina se usa cuando se detiene la ejecucion del sonido, por */
/* cualquier razon */
stop_play()
{
        unsigned        u;

        Active_flag &= ~PLAY;

        /* Si se esta esperando por la apertura del dispositivo, detiene la */
        /* espera ahora */
        if (Wait_flag & PLAY) {
                Wait_flag &= ~PLAY;
                xv_set(Main_play_item, PANEL_LABEL_STRING, "Reproduce", 0);
                return;
        }

        Buffer.draining = FALSE;

        /* Obtiene la cuenta del ultimo error */
        (void) audio_get_play_error(Audio_fd, &u);
        Audio_state.play.error = u;
        if (!Active_flag) {
                audio_flushclose();
                cancel_timer();
        }
        xv_set(Main_play_item, PANEL_LABEL_STRING, "Reproduce", 0);
}

/* Inicia la operacion de grabar (/dev/audio ya esta abierto) */
/*
 * Manipulador SIGPOLL sincrono ejecutado cuando el manipuladora SIGPOLL
 * detecta algo a hacer sincronamente, como actualizar pantallas
*/
```

```
/*ARGSUSED*/
Notify_value
sigpoll_sync_handler(client, event, arg, when)
        Notify_client        client;
        Notify_event         event;
        Notify_arg           arg;
        Notify_event_type    when;
{
        Sync_sched = FALSE;
                        /* Bandera del manipulador asincrono */
        /* Esperando para que se abra el dispositivo */
        if (Wait_flag) {
                if (!audio_open(Wait_flag)) {
                        /* Dispositivo abierto... empieza transferencia */
                        if (Wait_flag & PLAY) {
                                Wait_flag &= ~PLAY;
                                start_play();
                        }
                }
        }

        /* Obtiene el estatus actual de audio y actualiza la pantalla */
        main_update_panel(FALSE);

        /*
         * Detecta si la salida esta completa. La bandera play.eof se
         * incrementa cuando una escritura de tama&o cero ha sido procesada */
        if ((Active_flag & PLAY) && Audio_state.play.eof && Buffer.draining) {
                /* La salida se completo */
                stop_play();
#ifdef notdef
                if (Audio_state.play.error) {
                        message_display("No hay datos de sonido a ejecutar.");
                }
#endif
        }
        return (NOTIFY_DONE);
}
```

```
/* Maniputador de tiempo ejecutado en cualquier momento */
/*ARGSUSED*/
Notify_value
timer_handler(client, which)
        Notify_client        client;
        int                  which;
{
        if (Active_flag & PLAY)

        return (NOTIFY_DONE);
}

/* Establece un temporizador periodico para detectar la disponibilidad de*/
/*  el dispositivo abierto */
set_timer(time)
        double               time;
{
        struct itimerval     timer;
        int                  secs;
        int                  usecs;

        DEBUGF(("init timer (%.2f seconds)\n", time));

        secs = (int)time;
        usecs = (int)((time - (double)secs) * 1000000.);

        timer.it_value.tv_usec = usecs;
        timer.it_value.tv_sec = secs;
        timer.it_interval.tv_usec = usecs;
        timer.it_interval.tv_sec = secs;
        (void) notify_set_itimer_func(Base_frame, (Notify_func)timer_handler,
          ITIMER_REAL, &timer, ((struct itimerval *)0));
}

/* Cancela temporizador periodico */
cancel_timer()
{
        (void) notify_set_itimer_func(Base_frame, (Notify_func)timer_handler,
          ITIMER_REAL, ((struct itimerval *)0), ((struct itimerval *)0));

        DEBUGF(("cancel timer\n"));
}

/* Actualiza el desplegado solo si este esta cambiando (previene flasheo) */
/*VARARGS1*/
set_item_val(item, val)
        Panel_item     item;
        int            val;
{
        if (val != (int) xv_get(item, PANEL_VALUE))
                xv_set(item, PANEL_VALUE, val, 0);
}

/* Actualiza la cadena de seleccion del panel de descripcion */
describe_delta_update()
{
        char   delta_string[(2 * AUDIO_MAX_TIMEVAL) + 8];
        char   tmp1[AUDIO_MAX_TIMEVAL];
        char   tmp2[AUDIO_MAX_TIMEVAL];
```

```
        SPRINTF(delta_string, "%s - %s",
          audio_secs_to_str(audio_bytes_to_secs(
          &Buffer.hdr, (unsigned) Buffer.display.start), tmp1, 2),
          audio_secs_to_str(audio_bytes_to_secs(
          &Buffer.hdr, (unsigned) Buffer.display.end), tmp2, 2));

        /*xv_set(Describe_delta_item, PANEL_VALUE, delta_string, 0);*/
}

/* Establece un nuevo buffer de sonido y lo despliega */
file_update()
{
        Buffer.display.position = 0;
        Buffer.play.io_position = Buffer.play.start;

        play_update_cursor();

        /* Actualiza la longitud de seleccion en el panel de Descripcion */
        describe_delta_update();
        describe_update_panel(TRUE);
}


/* Verifica la region seleccionada del buffer de forma de onda y regresa su
  tama&o */
int
buffer_selected_size()
{
        int    cnt;
        char   msg[256];

        if (Buffer.hdr.data_size == 0) {
                message_display("No hay datos en el buffer.");
                return (0);
        }

        cnt = Buffer.play.end - Buffer.play.start;

        if (cnt == 0) {
                message_display("No hay datos en la region seleccionada.");
                return (0);
        }

        /* XXX - chequeo de seguridad */
        if (Buffer.play.start >= Buffer.hdr.data_size) {
                SPRINTF(msg, "Start position (%d) beyond EOF (%d).",
                  Buffer.play.start, Buffer.hdr.data_size);
                message_display(msg);
                return (-1);
        }

        if (cnt < 0) {
                SPRINTF(msg, "Start position (%d) beyond End position (%d).",
                  Buffer.play.start, Buffer.play.end);
                message_display(msg);
                return (-1);
        }
        return (cnt);
}
```

```
/*
 * Si tratas de hacer uni ciclo en un buffer muy peque&o, la ventana del sis-
 * tema se colgara. Esta rutina filtra ese tipo de requisiciones. Regresa el
 * numero de muestras en el buffer, o -1 si el buffer es muy peque&o.
*/
selectcheck()
{
        int    samples;

        samples = buffer_selected_size();
        if (samples <= 0)
                return (samples);

        if ((samples < MIN_BUFSIZE) && Loop_flag) {
                Loop_flag = 0;
                set_item_val(Loop_item, 0);
                message_display("Looping disabled: buffer size too small.");
                return (-1);
        }
        return (samples);
}

/* Despliega una alerta ( con alerta de audio, si la hay). */
message_display(msg)
        char   *msg;
{
        int    playing;

        playing = audio_play_alert();

        notice_prompt(Base_frame, (Event *) 0,
          NOTICE_MESSAGE_STRINGS, msg, 0,
          NOTICE_BUTTON_YES, "Ok",
          NOTICE_NO_BEEPING, playing,
          0);

        if (playing)
                alert_close();
}

/*
 * Despliega una caja de confirmacion ( con alerta de audio, si la hay).
 * Regresa Veradero si confirma. Falso si cancela.
*/
message_confirm(msg)
        char   *msg;
{
        int    playing;
        int    result;

        playing = audio_play_alert();

        result = notice_prompt(Base_frame, (Event *) 0,
          NOTICE_MESSAGE_STRINGS, msg, 0,
          NOTICE_BUTTON_YES, "Confirm",
          NOTICE_BUTTON_NO,  "Cancel",
          NOTICE_NO_BEEPING, playing,
          0);
```

```
        if (playing)
                alert_close();

        return (result == NOTICE_YES);
}
```

```
* Esto puede ser usado para el estado obtener/establecer (ej. niveles de
* volumen) sin colgar el dispositivo de audio principal abierto (/dev/audio)
*/
audio_control_init()
{
        /* Abre el dispositivo de audio principal */
        if ((Audioctl_fd = open(AUDIO_CTLDEV, O_RDWR)) < 0) {
                PERROR(AUDIO_CTLDEV);
                Device_phdr.sample_rate = 8000;
                Device_phdr.channels = 1;
                Device_phdr.bytes_per_unit = 1;
                Device_phdr.samples_per_unit = 1;
                Device_phdr.encoding = AUDIO_ENCODING_ULAW;
                Device_rhdr = Device_phdr;
                Buffer.hdr = Device_phdr;
        } else {

                /* Indica al driver enviar SIGPOLL en los cambios de estado
                 * del dispositivo */
                if (ioctl(Audioctl_fd, I_SETSIG, S_MSG) < 0)
                        PERROR("Could not issue I_SETSIG ioctl");

                /* Obtiene la configuracion de play y record del dispositivo */
                if ((audio_get_play_config(Audioctl_fd, &Device_phdr) !=
                  AUDIO_SUCCESS) ||
                  (audio_get_record_config(Audioctl_fd, &Device_rhdr) !=
                  AUDIO_SUCCESS)) {
                        PERROR("No se puede obtener la configuracion codificada");
                }
        }
        AUDIO_INITINFO(&Audio_state);
}

/* Lee el estado del dispositivo de audio y traslada campos en valores comprensibles */
int
audio_readstate(ip)
        Audio_info      *ip;
{
        if ((Audioctl_fd > 0) &&
          (audio_getinfo(Audioctl_fd, ip) != AUDIO_SUCCESS)) {
                /* Si hay error, sale tratando de accesar el dispositivo de control */
                Audioctl_fd = -1;
                PERROR(AUDIO_CTLDEV);
        }
        if (Audioctl_fd < 0) {
                /* Establece valor de default */
                ip->play.gain = 35;
                ip->record.gain = 60;
                ip->play.port = 0;
                ip->play.eof = 0;
                ip->play.error = FALSE;
                ip->play.pause = FALSE;
                ip->record.error = FALSE;
                ip->record.pause = FALSE;
                return (FALSE);
        }

        /* Convierte a valores compresibles */
        ip->play.gain = unscale_gain((double)(ip->play.gain - AUDIO_MIN_GAIN) /
          (double)AUDIO_MAX_GAIN);
```

```
/* Funciones de audio */
/* Abre el dispositivo de audio, utilizando la ventana Activa para derivar
 * modo abiertos.
 * Regresa:
 *    0     Apertura exitosa
 *    1     El dispositivo de audio esta ocupado (tratar mas tarde)
 *    -1    Error durante la apertura
*/
int
audio_open(flag)
        int    flag;
{
        if (Audio_fd >= 0) {    /* ya estaba abierto */
                FPRINTF(stderr, "%s ya estaba abierto\n", AUDIO_DEV);
                return (-1);
        }

        /*
         * Acceso de solo lectura si se esta grabando.
         * Acceso de solo escritura si se esta tocando o Alerta.
         */
        flag = ((flag & RECORD) ? O_RDONLY : O_WRONLY) | O_NDELAY;

        if ((Audio_fd = open(AUDIO_DEV, flag)) < 0) {
                if ((errno == EINTR) || (errno == EBUSY))
                        return (1);
                PERROR(AUDIO_DEV);
                return (-1);
        }

        flag = fcntl(Audio_fd, F_GETFL, 0) | FNDELAY;
        if (fcntl(Audio_fd, F_SETFL, flag) < 0)
                PERROR("F_SETFL fcntl");
        if (ioctl(Audio_fd, I_SETSIG, S_INPUT|S_OUTPUT|S_MSG) < 0)
                PERROR("I_SETSIG ioctl");
        return (0);
}

/* Vacia la salida encolada y cierra el dispositivo de audio */
audio_flushclose()
{
        if (Audio_fd < 0)
                return;
        (void) audio_flush(Audio_fd);
        (void) close(Audio_fd);
        Audio_fd = -1;
}

/* Cierra el dispositivo de audio ( la salida encolada drenara) */
audio_close()
{
        if (Audio_fd < 0)
                return;             /* Ya estaba cerrado */
        (void) close(Audio_fd);
        Audio_fd = -1;
}

/*
 * Abre el dispositivo de control de audio (/dev/audioctl) y lee su estado.
```

```
        ip->record.gain = unscale_gain(
            (double)(ip->record.gain - AUDIO_MIN_GAIN) /
            (double)AUDIO_MAX_GAIN);
        ip->play.port = (unsigned) ((ip->play.port == AUDIO_SPEAKER) ? 0 : 1);
        return (TRUE);
}


/*
 * Encola el archivo de sonido completo (maximo 2 segundos de datos) hacia
 * /dev/audio. Abandona /dev/audio abierto...no espera que la salida drene.
 * Regresa verdadero si la salida de audio comenzo; sino regresa Falso.
 *
 * XXX - El buffer deberia ser alojado dinamicamente para acomodar otros formatos.
*/
audio_play_alert()
{
        int         fd;
        int         rtn;
        Audio_hdr       hdr;
        char        buffer[16000];

        /*
         * Si play esta activo...no lo interrumpe.
         * Si record esta activo, deberiamso ser capaces de ejecutar este alert.
         */
        if (Active_flag & PLAY)
                return (FALSE);

        /* Abre, lee, cierra el archivo de sonido */
        if ((fd = open(NOTICE_FILE, O_RDONLY)) < 0)
                return (FALSE);
        rtn = (AUDIO_SUCCESS == audio_read_filehdr(fd, &hdr, (char *)NULL, 0));
        if (rtn) {
                rtn = read(fd, buffer, sizeof (buffer));
        }
        (void) close(fd);
        if (rtn <= 0)
                return (FALSE);

        DEBUGF(("Ejecutando archivo de alerta >%s<\n", NOTICE_FILE));

        if ((Alert_fd = open(AUDIO_DEV, O_WRONLY)) < 0) {
                if ((errno != EINTR) && (errno != EBUSY))
                        PERROR(AUDIO_DEV);
                return (FALSE);
        }

        if (write(Alert_fd, buffer, rtn) < 0) {
                PERROR("Alerta de audio escrita");
                alert_close();
                return (FALSE);
        }
        return (TRUE);
}

/* Vacia la salida encolada y cierra el dispositivo de alerta de audio */
alert_close()
{
        if (Alert_fd < 0)
```

```
                return;
        if (ioctl(Alert_fd, I_FLUSH, FLUSHW) < 0)
                PERROR("alert I_FLUSH ioctl");
        (void) close(Alert_fd);
        Alert_fd = -1;
}

generic_notice(msg1)
        char    *msg1;
{
        notice_prompt(Base_frame, (Event *)NULL,
                NOTICE_MESSAGE_STRINGS, msg1, 0,
                NOTICE_BUTTON_YES, "Continua",
                0);
}
```

REFERENCIAS

[1] Ambron S. y Hooper K., *Interactive Multimedia*, Microsoft Press, CA,1988

[2] Congreso Metropolitano para Estudiantes de Ingeniería, *Ingeniería en Computación, Tomo II*, De. Trillas, D.F., México.1993

[3] Información extraída del host *trinity.edu* de la Universidad Trinity en San Antonio Texas.

[4] HP Computer/Instrument Systems, *Introduction to the X Window System*, HP, Palo Alto, CA, 1990

[5] SunSoft, *Writing Applications for the Solaris Environment*, Tomo II, Addison Wesley, Reading, MA, 1992

BIBLIOGRAFIA

Mink K., *The Process of Program Development*, CADIF, Cornell University, Ithaca, N.Y, 1987

HP Computer/Instrument Systems, *X Window System Programming: Xlib*, HP, Palo Alto ,CA, 1989

La información anterior sobre el consorcio X fue extraída del Host decuac.dec.com (192.5.214.1) del archivo /contrib/share/man/cat1/XConsortium.1 en la red de DIGITAL.

Tse-Hwa Shen, Libby D. H. y Abel J. F. , *User's Manual for GISMA*, Cornell University, Ithaca, N.Y, 1987

Tse-Hwa Shen, Libby D. H. y Abel J. F., *User's Manual for GISMO*, Cornell University, Ithaca, N.Y, 1987

SunSoft, *Multimedia Primer*, Sun Microsystems, Mountain View, CA, 1992

De Mendizabal Allende B., *Diccionario de Informática*, Diaz de Santos S.A., D.F., México, 1993

Johnson S y Reichard W, *Advanced X Window Applications programming*, Advanced computer books, MIS press, Portland, Oregon, 1990

Quercia V. y O'Reilly T, *X Window System User's Guide*, O'Reilly & Associates, CA, 1993

Young D. A. y Pew J. A., *The XWindow System Programming & Applications with Xt* , Prentice Hall, Reading, MA, 1992

McMinds D.L., *Mastering OSF/Motif Widgets*, Second Edition, Addison Wesley, Reading, MA,1993

Ayala G, García O y Contreras J J, *Interprete Gráfico de Operaciones Estructurales Matriciales*, Instítuto de Ingeniería, UNAM, México, 1993