PROYECTO DE GRADO

Arte y Tecnología.

Fabrizio Castro y Tomás Laurenzo
http://www.fing.edu.uy/~t5arte
t5arte@fing.edu.uy

mayo de 2003

Tutores
Dr. Etienne Delacroix. Instituto de Computación.
Dr. Gregory Randall. Facultad de Ingeniería.
Mag. Luis Sierra Universidad de la República.

"El cine no prosperó hasta que los ingenieros perdieron el control de los artistas.
Lo mismo está pasando con las computadoras."
Paul Heckel [1]

Queremos agradecer a las siguientes personas, sin las cuales hubiera sido imposible realizar el proyecto.

Etienne Delacroix
Gregory Randall
Javier Rodríguez
Hermanos Thevenet y varios alumnos de TAP
Magdalena Hourcade
Sebastián de los Campos
Joaquín Arambarri y Juan Ghigliazza
Gonzalo Tejera
Marcos Campal

Tabla de contenido

Tabla de contenido **4**

Prólogo. **5**
El Núcleo de Arte y Tecnología (NAT). 5

Introducción. **7**
Arte y tecnología. 7
Descripción general del proyecto. 11

Chatarra tecnológica **14**
Interacción con tecnologías de última generación. 23

Red. **25**
Servidor. 25
Protocolo. 26

Guionado. **30**
Lenguajes de programación. 30
Guiones 30

Imagen. **33**
Multimedia. 33
Interacción vía *webcams.* 35
Graficación en tiempo real. 41
MediaScripting y Visualización de vídeos grabados. 43

Música **44**
¿Qué es la música? 44
Síntesis de sonido 45
MIDI 47
Hiperinstrumentos [3]. 50
Prototipos sonoros 51
Integración de MIDI. 53

Arquitectura. **55**
Introducción. 55
Diseño. 56
Módulos implementados. 57

El Tecnocordio. **62**

Consideraciones finales. **66**

Referencias. **68**

Código del servidor. **70**

Datos de los estudiantes. **75**

Prólogo.

El Núcleo de Arte y Tecnología (NAT).

El NAT constituye una unidad académica, aún en gestación, de la cual formarán parte las Facultades de Arquitectura, Ciencias e Ingeniería, el Instituto Escuela Nacional de Bellas Artes y la Escuela Universitaria de Música.

Las actividades dentro del NAT operan en función de tres intereses básicos: la utilización de la tecnología como medio bruto de expresión artística, la apropiación de la tecnología y la puesta en escena de conocimientos.

Resulta claro que estos tres aspectos se encuentran íntimamente relacionados. La utilización de la tecnología como medio expresivo conlleva una contradicción frontal con la actividad habitual de mero consumidor de la misma.

Esta nueva actitud frente al medio tecnológico, sólo puede ser resultado de la puesta en escena de los conocimientos adquiridos y, sobre todo, de los mecanismos (tecnológicos, pero sobre todo sociales) que permiten esta reconversión fundamental del rol que se juega ante el panorama tecnológico y sus cambios.

A nuestro entender, este cambio de enfoque resulta impostergable; más aún cuando creemos que la postura tradicional de competencia a través de la especialización técnica en un aspecto concreto de la producción tecnológica, no permite sino, en el mejor de los casos, una inserción modesta dentro de un nicho comercial global.

En contraposición, nuestra idea de apropiación tecnológica pasa por la reutilización de tecnología rearticulándola en función de necesidades concretas, la cual no tiene por que ser de última generación, sino que muchas veces será tecnología de desecho.

Si bien es innegable que aún nos encontramos dando los primeros pasos (entre los cuales se encuentra nuestro Proyecto de Grado), los avances hasta ahora logrados brindan un promisorio panorama respecto a las posibilidades y al impacto que su aplicación sistemática conlleva.

Es por ello que uno de los objetivos del presente proyecto consistió en formalizar el marco conceptual presentado con el NAT. En ese sentido, creemos que el proyecto puede ser utilizado como ejemplo de alternativas, metodológicas pero también conceptuales, a la hora de afrontar el labor intelectual.

Estamos convencidos de que la única alternativa con posibilidades de éxito a la hora de afrontar la labor ingenieril (y la intelectual en general), es decir, la construcción de un mundo más humano, más justo, más feliz, pasa -desde este lado del mundo, o desde este lado de la brecha digital- por cambiar la forma de producción intelectual, apropiándose del conocimiento y construyendo nuevos caminos que apunten a la solución de los problemas de nuestra realidad.

Confiamos en que este cambio se producirá en forma masiva en un futuro no muy lejano, y para ello es fundamental que proyectos como el Núcleo de Arte y Tecnología cuenten con el apoyo activo de todos.

Así, nos resulta muy grato poder colaborar en el esfuerzo de que dentro de la Universidad de la República se realicen trabajos en este sentido, y creemos firmemente que la UDELAR es el marco natural para realizarlos, aunque consideramos también muy importante el involucrar e interactuar con otras Universidades de la región.

Etienne Delacroix
Fabrizio Castro
Tomás Laurenzo

capítulo 1

Introducción.

Este documento constituye el informe final del Proyecto de Grado “Arte y Tecnología”, proyecto de fin de carrera de Ingeniería en Computación de los estudiantes Fabrizio Castro y Tomás Laurenzo.

El proyecto se realizó en el Instituto de Ingeniería Eléctrica (IIE) de la Facultad de Ingeniería, y tuvo como responsables por el mismo al Dr. Etienne Delacroix (MIT) y al Dr. Gregory Randall, y como tutor responsable por el Instituto de Computación (INCO) al Mag. Luis Sierra.

A continuación se podrá encontrar una introducción al objeto de estudio, Arte y Tecnología, así como consideraciones acerca del marco ideológico que justifica nuestra visión del proyecto, seguida por la descripción general del mismo.

Arte y tecnología.

Con *Arte y Tecnología*, se denomina a un importante espacio de investigación, que en los últimos años ha concitado un interés creciente tanto desde el ambiente académico como desde el comercial. Ejemplo paradigmático de ello lo constituye el Laboratorio de Medios del Instituto Tecnológico de Massachusetts [2], dentro del cual se llevan a cabo distintas investigaciones (nuevos instrumentos [3], nuevas interfaces, medios sociales, estética y computación, etc.) mientras se obtienen financiaciones muy importantes por parte de la industria (como ejemplo sólo citaremos a la cátedra Toshiba).

Dentro de ese amplio y poliforme espacio de investigación, el Núcleo de Arte y Tecnología tiene como principal línea de investigación la utilización de la tecnología como *medio bruto* de expresión artística. Es decir, la utilización de elementos del ambiente tecnológico como herramientas de producción artística, pero también su utilización como parte constitutiva del fenómeno expresivo.

En pocas palabras, hacer arte utilizando tecnología y hacer de la tecnología un arte.

Este “hacer de la tecnología un arte” podría ser interpretado de distintas maneras (baste recordar a Donald Knuth y su famoso *“The art of computer programming”* [4]), sin embargo existe (como muchas veces) una interpretación literal que resulta útil. Hacer de la tecnología un arte implica, para nosotros, otorgar expresividad a la producción tecnológica, es decir, permitir y potenciar la transmisión de signos a través de la manipulación (no sólo la operación) del medio tecnológico.

Esta nueva dimensión de interacción con la tecnología, si bien requiere mayores y más finos conocimientos técnicos de los artistas, brinda una libertad de expresión más allá de los caminos y formas impuestos por la oferta comercial. Así, el artista puede buscar sus caminos expresivos, separándose del rol usuario que transita -con mejor o peor suerte- la estética cada vez más homogénea hacia la cual convergen los más hábiles en el manejo de interfaces de programas de diseño.

Hemos mencionado la transmisión de signos como sinónimo de expresión artística, definición que, si bien resulta reduccionista, reviste un interés inextricable dada la proliferación simbólica imperante. No resulta necesario un análisis semiótico muy fino para observar el bombardeo simbólico constante y ubicuo de la sociedad occidental contemporánea, y es en este marco en el cual la investigación de caminos alternativos de producción simbólica, de nuevos ámbitos de expresión, reviste un interés renovado.

Como primer ejemplo de expresividad asociada a la producción tecnológica genuina, mencionaremos el Tecnocordio[1], (limitado) instrumento musical fruto de este proyecto, el cual además de constituir como un instrumento musical, conlleva (a la manera del tradicional luthier) una intención estética, tanto en sus partes constitutivas como en los procesos que permiten su funcionamiento y operación.

Sin embargo, no sólo en la construcción de objetos de volumen puede darse la intención expresiva, la puesta en escena de los conocimientos se ve potenciada por una intencionalidad artística. La documentación, en este contexto, adquiere una nueva dimensión de interés: es pasible de ser convertida (en tanto parte insustituible e indispensable de la producción tecnológica) en medio expresivo, jugando además un rol fundamental en la transmisión de los conocimientos implicados.

Dado que consideramos que el único camino para lograr un cambio en la actitud que se toma frente a la tecnología pasa por la masificación de los conocimientos, este cambio dependerá en gran forma de la documentación asociada a (y formando parte de) los productos artístico-tecnológicos.

Virtualización.

La utilización de la tecnología como parte constitutiva de la expresión artística puede ser vista como uno más de los procesos de virtualización, fenómeno cada vez más común.

Esta virtualización es, en palabras del filósofo francés Pierre Lévy[2] [5] , *"un desplazamiento del centro de gravedad ontológico del objeto considerado. De esta forma, en lugar de definirse principalmente por su actualidad (una "solución"), la entidad encuentra así su consistencia esencial en un campo problemático. Virtualizar una entidad cualquiera consiste en descubrir la cuestión general a la que se refiere, en mutar la entidad en dirección a éste interrogante y en redefinir la actualidad de partida como respuesta particular"*.

En este sentido, la virtualización no se presenta como una desrealización -el ente virtualizado no pierde realidad- sino como una desactualización. Así, lo virtual se concibe como antónimo de lo actual y no de lo real. Esto permite una aproximación al objeto virtual, o virtualizado y al proceso mismo de virtualización que consideramos particularmente interesante.

La virtualización inmanente en la utilización de la tecnología en los ambientes de producción artística, resulta así enfocada no como una simplificación, sino como un desplazamiento ontológico hacia nuevos ámbitos de interacción.

[1] Ver el capítulo 8.

[2] Pierre Lévy (1956) Filósofo de la cultura virtual contemporánea. Nacido en Túnez, vive en París y enseña en el departamento de Hypermedia, Universidad de París-VIII. Estudió como investigador con Michel Serres y Cornelius Castoriadis y continuó en Montreal especializándose en acercamientos hypertextual. Con Michel Authier, desarrolló un concepto de la red conocido como "Arboles del Conocimiento". Lévy también está interesado en la inteligencia colectiva estudiada en un contexto antropológico y es uno de los principales filósofos trabajando en el área de los medios.

Esta virtualización, junto con la enorme proliferación de signos, han marcado la producción mediática contemporánea. Es nuestra intención brindar una alternativa de producción simbólica, a través de la apropiación tecnológica y la puesta en escena de conocimientos.

La frase de Panofsky[3] que proclama que "*el artista debe ser ojos, oídos y voz de su tiempo*", adquiere renovada vigencia; la virtualización de los distintos procesos comunicacionales del ser humano debe ser reflejada en la producción artística.

Apropiación y rearticulación de la tecnología.

"Si no hacemos nada, Internet y el cable estarán monopolizados dentro de diez o quince años por las megacorporaciones empresariales, la gente no conoce que en sus manos está la posibilidad de disponer de estos instrumentos tecnológicos en vez de dejárselos a las grandes compañías. Para ello, hace falta coordinación entre los grupos que se oponen a esa monopolización, utilizando la tecnología con creatividad, inteligencia e iniciativa para promocionar, por ejemplo, la educación."

Noam Chomsky.

La vorágine tecnológica, que muchos explican como una consecuencia natural de los avances que constantemente se realizan, sería incomprensible si no se tuvieran en cuenta las razones de índole puramente económica que la promueven y exacerban.

Las leyes de mercado, que algunos juzgan sagradas, obligan a esta mecánica en la cual el rol de consumidor tecnológico nunca puede ser pervertido, de forma tal que cuando resulta imposible la prosecución del ritmo de consumo impuesto, se activan procesos de exclusión que alejan a las personas de la posibilidad de utilizar la tecnología como herramienta real de resolución de sus problemas.

Nosotros consideramos que este fenómeno de exclusión social, junto con todos los demás fenómenos similares presentes en nuestra sociedad, es injustificable. Pero también creemos que es posible atacarlo si se logran mecanismos de producción que perviertan el rol de mero consumidor tecnológico impuesto por el sistema capitalista.

Una forma de lograrlo es a través de la apropiación de la tecnología.

Apropiarse de la tecnología implica conocerla, desmitificarla, deshacerla y reconstruirla desde nuestra realidad, para hacer frente a problemas de esta realidad, generando soluciones alternativas a las propuestas por el mercado.

Para ello, resulta necesaria una nueva actitud de la comunidad intelectual, según Eladio Dieste[4] [6], *"Cada problema [..] debería encararse con una especie de ingenuidad, no con ánimo de ser original, sino con una actitud humilde y vigilante. Pensarlo de nuevo, con el acervo básico que es ya patrimonio de todos los hombres, pero sin dejarnos enredar en los detalles resueltos por otros, y para situaciones que tienen muy poco que ver con las nuestras".*

[3] Erwin Panosfky (1892-1968), Historiador de Arte Americana, nacido en Hanover, Alemania. Ph.D. Univ. of Freiburg, 1914. Luego de enseñar en la Universidad de Hamburgo y actuar como Profesor de Bellas Artes de la Universidad de Nueva York, se unió en 1935 al Institute for Advanced Study, Princeton, N.J. Sus trabajos en Historia del Arte, están entre los más importantes del siglo XX. Panofsky contribuyó en el estudio (principalmente en el ára de la Iconografía), de los períodos Medieval, Renacentista, Manerista y Barroco. Entre sus trabajos destacamos *Studies in Iconology* (1939, 2d ed. 1962), *Albrecht Dürer* (1943, 4th ed. 1955), *Early Netherlandish Painting* (1953), y *Renaissance and Renascenses in Western Art* (2d ed. 1965).

[4] Eladio Dieste (1917-2000) nacido en Artigas, Uruguay, fue egresado de la Facultad de Ingeniería (UDELAR, 1943). Fue Miembro Correspondiente de la Academia de Ciencias de la República Argentina, Profesor Ad Honorem de las Facultades de Arquitectura de Montevideo y Buenos Aires, Miembro correspondiente de la Academia de Bellas Artes Argentina, Miembro de la Academia de Ingenieros del Uruguay y Doctor "Honoris Causa" de la UDELAR.

Donde -a nuestro entender- lo fundamental consiste en reconocer lo disímil de nuestra realidad con la del primer mundo y la necesaria diferencia entre las soluciones aplicables a nuestro contexto y las utilizadas bajo otros parámetros. Esta diferencia resulta aún más importante cuando *"como consecuencia de la equivocada actitud de imaginar una ciencia y una técnica ya hechas, que sólo esperan que las descubramos, se produce entre nosotros, por otro camino, una también lamentable ceguera frente a la realidad"*. [6].

Para lograr este cambio de actitud, es necesario un conocimiento fino y profundo de la tecnología, no sólo de su funcionamiento, sino de formas de obtención de los productos tecnológicos convertidos ahora en materia prima. Pero, por sobre todo, es necesaria la diseminación del conocimiento, su puesta en escena.

La puesta en escena de conocimientos, resulta así la piedra fundamental en la construcción de una inteligencia colectiva, social, que aporte soluciones y subvierta el rol jugado por la tecnología en la sociedad.

En el decir de Pierre Lévy [7] "*La prosperidad de una nación, región geográfica, negocio o individuo depende de su habilidad para navegar el espacio del conocimiento. Ahora el poder está conferido a través del manejo óptimo del conocimiento, o sea esté envuelto en la tecnología, la ciencia, la comunicación, o nuestra relación ética con el otro. Teniendo mayor habilidad para formar comunidades inteligentes, mentes abiertas, sujetos cognitivos capaces de iniciativa, imaginación y rápida respuesta, seremos capaces de asegurar nuestro éxito en un ambiente altamente competitivo".*

Si bien el *savoir faire* ha sido siempre una herramienta de poder en el orden económico, es en la actualidad cuando reviste una mayor importancia, dado que -como dice Manuel Castells[5]- *"Lo que caracteriza a la revolución tecnológica actual no es el carácter central del conocimiento y la información, sino la aplicación de ese conocimiento e información a aparatos de generación de conocimiento y procesamiento de la información y comunicación, en un círculo de retroalimentación acumulativo entre la innovación y sus usos. [...] La primera característica del nuevo paradigma es que la información es su materia prima: son tecnologías para actuar sobre la información, no sólo información para actuar sobre la tecnología, como era el caso en las revoluciones tecnológicas previas."* [8]

De esta manera, a través de la sistematización de procesos de rearticulación de la tecnología, es posible idear formas de producción de valor agregado que no dependan del capital involucrado, sino del saber hacer.

Citando otra vez a Lévy: *"Bajo el sistema asalariado el individuo vende su fuerza física o tiempo de trabajo dentro de una estructura cuantitativa y fácilmente medible. Tal sistema podría dar paso a una forma de autopromoción, involucrando habilidades diferenciadas cualitativamente, por productores independientes o equipos pequeños. Individuos y micro corporaciones son más capaces que compañías grandes de una reorganización continua y realzamiento óptimo de habilidades individuales que son los requerimientos actuales para el éxito."*

En función de la intención de minimizar la necesidad de inversión en términos de capital, aparece como camino natural la rearticulación de tecnología que no sea de última generación (lo impuesto por el mercado) sino que provenga del aprovechamiento del inmenso flujo de desechos tecnológicos.

[5] El profesor Manuel Castells (1942) nacido en Albacete, España, es catedrático de Sociología y Planificación Urbana y Regional en la Universidad de Berkeley (California). También es profesor (en excedencia) del Consejo Superior de Investigaciones Científicas. Ha sido catedrático y director del Instituto de Sociología de Nuevas Tecnologías de la Universidad Autónoma de Madrid y profesor de Sociología de la Escuela de Altos Estudios de París. Durante los últimos años ha publicado la trilogía "La Era de la Información", que consta de tres volúmenes "La Sociedad Red", "El Poder de la Identidad" y "Fin de Milenio".

Estos desechos, bautizados *e-waste*, implican además un riesgo ambiental importante. El último reporte de la SVTC (*Silicon Valley's Toxics Coalition*) informa que durante el año 2000 más de 4.6 millones de toneladas de desechos tecnológicos llegaron a los depósitos de basura de EE.UU. [9].

Si bien una parte de estos desechos es reciclada, por ejemplo aluminio, cobre y acero, frecuentemente son extraídos y reutilizados, entre el 50 y el 80 por ciento del *e-waste* producido en EE.UU. que es reciclado, es exportado a países como China, India o Pakistán, donde los trabajadores que separan los metales útiles, utilizan químicos tóxicos que generan serios problemas de salud [10].

En el marco de nuestro proyecto hemos adoptado y generado distintas técnicas que permiten utilizar elementos de desecho junto con tecnología marcadamente obsoleta para afrontar problemas concretos[6].

En este sentido, intentamos dar un paso más en el afán de generar caminos transitables, haciendo hincapié en la puesta en escena de los conocimientos.

Descripción general del proyecto.

El proyecto pretende, dentro del marco conceptual descrito, investigar los intereses del NAT a través del diseño de un marco de trabajo que, involucrando un sistema de computación en red, pueda ser utilizado como *medio* de expresión y como herramienta de transmisión de los conceptos involucrados en el mismo.

La definición del proyecto, sin embargo, proviene no sólo de los ejes temáticos mencionados, sino también de aquellas áreas específicas que resultan de interés para el NAT, ya sea por el potencial que revisten o por estarse realizando en el Núcleo desarrollos sobre esos temas. Como ejemplos podemos mencionar la utilización de sistemas de red sobre TCP/IP, la distribución de procesos, la programación con Java y el sistema operativo Linux.

Finalmente, se tuvo también en cuenta el interés personal de quienes lo realizamos, y en función de ello fue que se decidió concentrar los aspectos expresivos en áreas relacionadas con la música.

En resumen, se buscó la construcción de un sistema que permita interpretar una variedad de estímulos (imagen, archivos, guiones o *scripts*, etc.) para producir contenido expresivo en forma de sonido o imagen.

Si bien esta aproximación brinda un marco de trabajo, delimitando el área de estudio, al inicio del proyecto se consideró necesaria la definición de objetivos concretos que orientasen el desarrollo del mismo.

Dentro de estos objetivos destacamos:

- Investigar la posible reutilización de tecnologías de desecho con fines específicos, en particular computadores y accesorios informáticos obsoletos.
- Investigar la frontera entre lo digital y lo físico, estudiando las posibilidades de las construcciones físicas que, en nuestro caso, combinarán elementos de desecho con tecnología de última generación.

6 En nuestro caso, uno de los problemas concretos consiste en la utilización de la tecnología como medio de producción artística. La apropiación de la tecnología, en este marco, coadyuva para lograr lo que Eraso pide cuando dice "La creación contemporánea tiene que hacerse más laica" [13] (Santi Eraso es el director del proyecto de experimentación arquitectónica Arteleku en San Sebastián, España).

- Desarrollar un sistema informático que permita la interconexión y el control de dispositivos, constituyéndose tanto en un marco de trabajo artístico que permita crear instalaciones multimediales interactivas, como en una plataforma educativa utilizable para enseñar los distintos conceptos involucrados en el sistema.

- Desarrollar un pequeño lenguaje de guionado (*scripting*) que permita ahondar en las posibilidades interactivas del sistema.

- Como apoyo a la función educativa del sistema, se desarrollará un sitio Web que contenga la documentación del proyecto.

Además de éstos, se planteaban como objetivos opcionales la construcción de un instrumento musical en base a tecnología de desecho y de última generación, así como el desarrollo de interfaces hombre-máquina no tradicionales mediante la utilización de cámaras de video (*webcams*) y el análisis en tiempo real de las imágenes filmadas.

Dado su carácter fuertemente experimental e investigativo, el proyecto finalmente fue caracterizado por estas metas que, en lugar de definirse en función de resultados esperados específicos, en general consisten en la especificación de áreas de investigación.

De esta forma, el proyecto se reestructuró centrándonos en la investigación y construcción de prototipos funcionales (en general, en forma de instrumentos musicales rudimentarios[7]) que permitiesen abarcar los objetivos especificados al inicio del mismo.

Forma de trabajo.

Si bien se contó con una delimitación inicial como marco de trabajo, así como de objetivos específicos a cumplir, en un primer momento nos encontramos ante un enorme campo de estudio sin un camino claro a seguir.

Por ello, se optó por la construcción sistemática de pequeños prototipos que nos permitieron ganar experiencia mientras funcionan como aplicación y ejemplo de los conceptos de interés.

Luego, las actividades e intereses del NAT brindaron otro marco de referencia, en función de ello –por ejemplo- se trabajó con el sistema operativo Linux además de MS·Windows o se programó en Java.

La adopción de Java como principal lenguaje de desarrollo obedece a que ofrece un universo pedagógico abierto, totalmente orientado a objetos y multiplataforma, dentro del cual es posible adquirir los conocimientos necesarios con relativa facilidad, en función de la documentación y soporte existentes, así como de la activa comunidad de usuarios que ofrece acceso a código y documentos relacionados mediante foros de discusión y sitios web. [12]

Así, Java es uno de los lenguajes con la curva de aprendizaje más rápida y que al ser propiedad de una gran compañía (Sun Microsystems) que provee mucha documentación, tutoriales y ejemplos de código, es posible obtener rápidamente resultados satisfactorios tanto sea tratándose de software de red, sonido, imagen e incluso guionado, como se explicará más adelante.

Además, adoptando fundamentalmente un único lenguaje de desarrollo se facilitaría la integración e interacción de los distintos módulos desarrollados independientemente.

[7] Para una descripción más detallada de los prototipos realizados ver la sección Música.

Acerca de la documentación.

Ante la eventual utilización del producto de este proyecto con fines educativos, la documentación adquiere un nuevo rol, constituyendo no sólo la descripción del proyecto y sus resultados, sino también una valiosa herramienta de transferencia de los conocimientos y metodologías implicados.

En función de ello, a lo largo del presente documento se intentará brindar una visión lo más completa posible del proyecto, explicando no sólo los desarrollos realizados, sino la base conceptual que posibilita los mismos. En este mismo sentido, se incluirán introducciones a las áreas desarrolladas junto con, por ejemplo, porciones de seudo código que ofrezcan al lector un punto de entrada al área tratada.

Los extractos de código se incluirán (un poco a la manera de la *Literate Programming* de Donald Knuth [11]) para ilustrar el funcionamiento del software creado. Resulta necesario tener en cuenta que si bien es muy difícil separar la descripción de un módulo de otro, dividiremos la documentación según las grandes áreas abarcadas.

En las secciones siguientes se documentará el trabajo realizado en las principales áreas, ofreciéndose una introducción al tema y un detalle de nuestros desarrollos.

Así, el capítulo dos trata sobre la reutilización de chatarra tecnológica y la construcción de sistemas físicos a partir de ella. En él se describe cómo reciclar chips y otros elementos que habitualmente son desechados. Se hará énfasis en la utilización del puerto paralelo para controlar motores paso a paso reciclados y se comentará cómo puede esta chatarra reciclada interactuar con tecnologías de última generación.

En capítulo tres, Red, se verá el sistema implementado que permite controlar los motores en forma remota, a través de un sistema de red basado en TCP/IP. También se describirá el sistema cliente/servidor desarrollado.

Luego, en el capítulo cuatro se podrá encontrar una visión de lenguajes de programación con énfasis en la utilización de lenguajes de guionado, describiremos además la incorporación de esta capacidad a través de Beanshell.

El capítulo cinco dará una introducción a los principales conceptos de multimedia, para luego detallar las posibilidades actuales en el desarrollo de una interfaz gestual utilizando *webcams*. Además se describirá la implementación de software de graficación en tiempo real y visualización de videos grabados utilizando *scripts*.

Luego habrá un capítulo sobre la música en general (capítulo seis), y en particular la capacidad de generar sonido y su control a través del protocolo MIDI. Se describirá la inserción de esta capacidad en nuestros prototipos.

En el capítulo siete podrá encontrarse la especificación de la arquitectura propuesta junto con la explicación de su funcionamiento, que podrá ser utilizada como referencia para futuros desarrollos en el área.

Finalmente, el capítulo ocho documentará una aplicación de casi todos los componentes interactuando y el cual constituye el producto más tangible de nuestra labor, el Tecnocordio.

capítulo 2

Chatarra tecnológica

Como ya hemos mencionado, la velocidad de obsolescencia impuesta a los productos tecnológicos tiene como producto marginal una producción enorme de chatarra.

Por ello resulta relativamente sencillo obtener un volumen importante de desechos tecnológicos: fuentes, procesadores, placas, discos, chips, resistencias, y numerosos otros elementos. Dispositivos que ya nadie usa, pero que son reutilizables, rearticulando así la tecnología implicada.

Además, posiblemente nos encontremos en una "ventana de oportunidad" histórica que debe ser aprovechada. Esta ventana puede ser única en función de que probablemente sea insostenible el presente ritmo de actualización y de que los futuros avances en miniaturización dificulten enormemente el aprovechamiento de los distintos componentes de la tecnología desechada. Por lo que las formas de apropiación de la misma deberán evolucionar adaptando su habilidad para superar también las futuras dificultades. Reforzando la idea de que con apropiación de la tecnología nos referimos a una actitud más que a una metodología específica.

Describiremos ahora algunos ejemplos concretos de reutilización realizados.

Reciclaje de Chips.

Un chip puede definirse como un pequeño trozo de cristal semiconductor, en el que se han formado diodos, transistores u otros componentes que, interconectados, constituyen un circuito integrado funcional.

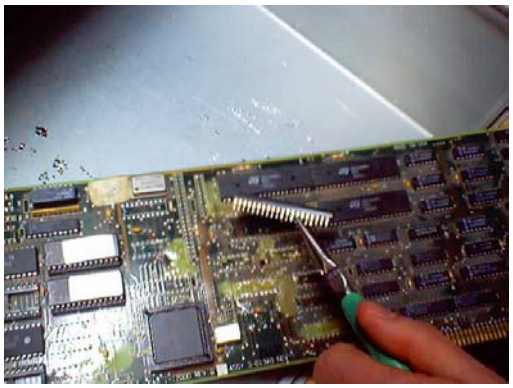
Este circuito integrado implementa cierta lógica o provee herramientas para lograrla. Interconectando chips con algunos circuitos elementales, es posible construir nuevos circuitos con un funcionamiento extremadamente sofisticado. De hecho, las tarjetas controladoras e incluso la propia tarjeta madre -que permite el funcionamiento y la comunicación entre todas las distintas unidades de la computadora-, siguen este esquema de chips interconectados.

Afortunada e inteligentemente, cada chip es reconocible por sus especificaciones marcadas físicamente, lo cual brinda la posibilidad de identificar en tarjetas viejas algunos chips que nos puedan resultar útiles. Luego es posible extraerlos y reutilizarlos para la implementación de los circuitos que nos interesen.

Se dispone de un método muy práctico para extraer chips de las placas desechadas. Utilizando una pistola de calor (la cual se utiliza habitualmente para remover pintura) se calienta la zona del chip o componente que se desea obtener para debilitar la soldadura de sus patas a la tarjeta. Es importante mantener en movimiento la pistola para evitar focalizar demasiado el calor hacia el chip ya que existe la posibilidad de quemarlo. Un rato después se podrá apreciar el estaño más brillante, entonces con un golpe se logra quitar el estaño, liberando prácticamente el chip. Es probable que haya que repetir estos pasos hasta quitar

todos los rastros de estaño antes de sacarlo. Suele ser útil ayudarse con un pequeño destornillador o pinza con suficiente delicadeza como para no dañarlo.





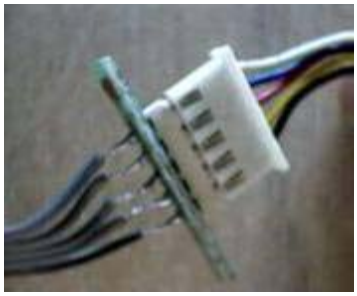

Con este método se obtienen chips perfectamente funcionales, PICs (microcontroladores), conectores, o lo que sea que pueda encontrar en las tarjetas chatarra y que pueda resultar útil para lo que se quiere construir.

Construcciones funcionales y modulares

El PCB (Printed Circuit Board, material plástico donde se imprimen los circuitos y se sueldan los componentes) de las tarjetas, una vez liberado de todos los chips, es muy útil para construcciones físicas. Se corta y se reutiliza para crear nuevos circuitos, pudiéndose crear fichas, conectores o circuitos completos.

Además hemos utilizado en varias ocasiones los hilos paralelos de los cables de los conectores IDE (*ribbon cables*), que han resultado excelentes para conectar dispositivos que intercambian unos pocos bits.

Finalmente, dada la necesidad de estabilidad en las conexiones entre los dispositivos, utilizando PCB e hileras de patas (*pines*) se han construido algunos conectores que las hacen más confiables.

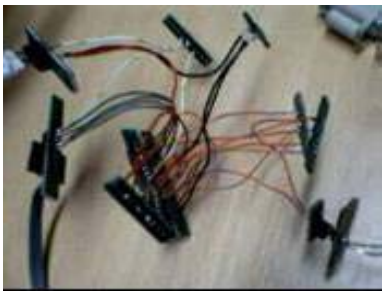



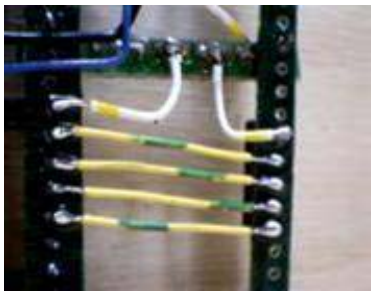
Uno de los aspectos que representan un desafío mayor es el de desarrollar formas de trabajo y construcciones físicas modulares, fácilmente reproducibles, interoperantes y confiables.

Existen, a priori, muchas posibilidades para armar un circuito lógico aprovechando estos desechos. Sin embargo, es necesario tener en cuenta que las construcciones que se realizan en el taller representan una labor muy artesanal, fina y frágil, por lo cual, la probabilidad de que el mismo finalmente no funcione correctamente es muy alta, pudiendo fallar por un mal contacto, un contacto inesperado entre elementos del PCB cortado, o por alguna otra misteriosa razón. El trabajo realizado apunta a encontrar métodos y diseños que permitan disminuir al mínimo esta fragilidad.

Muchas veces una idea surge como el camino más natural para encarar una problemática, pero la práctica se encarga de mostrar que no es la más conveniente por diversas dificultades inesperadas que surgen una tras otra en el camino.

No es extraño, aún brindando dedicación a la construcción de un circuito, encontrar que hay contactos inesperados o distintos valores de resistencia de los materiales que usamos que conducen a que una pequeña variación de voltaje provoque cambios relevantes en el comportamiento. De manera que debe priorizarse la sistematización del proceso de construcción para limitar estos imponderables que representan una de las mayores dificultades al trabajar con este tipo de construcciones.





La experimentación, a través de logros y fracasos va delineando poco a poco una estandarización en la manera de trabajar y en los diseños mismos de las construcciones.

Si bien, dada la característica artesanal del trabajo resulta difícil detallar la metodología, podemos citar, a modo de ejemplo, algunas pautas a seguir:

- Siempre verificar que el trozo de PCB que se va a utilizar no tenga pistas ocultas (existen PCBs de más de tres capas) que interfieran con el comportamiento esperado del circuito del que formará parte. Para esto debemos chequear con un *tester*, orificio por orificio, si existe algún contacto, agotando todas las combinaciones posibles.

- Pensar qué materiales y herramientas serán necesarios y verificar su disponibilidad, teniéndolos a mano antes de iniciar el trabajo. Aunque es prácticamente imposible cubrir mentalmente a priori todos los detalles, en un análisis previo habitualmente encontramos varios elementos requeridos cuya obtención requiere considerable cantidad de tiempo y trabajo.

- Pensar antes de cortar o romper: no cortar en forma sistemática, sino ponderar el beneficio resultante de la rotura del material. En caso de existir más de una opción para resolver algo, tener en cuenta que es preferible desarrollar una forma de construcción utilizando los elementos más comunes entre la chatarra. De hecho, muchas veces es preferible guardar los elementos más escasos para cuando su utilización es inevitable.

- Lamentablemente, existen ocasiones donde es imprescindible (en general, por falta de recursos) recurrir a soluciones premanufacturadas (cuando es *"absolutamente indispensable, por urgencias ilevantables"* según el texto de Dieste [6]). El reconocimiento de esos casos consiste una de las tareas más difíciles a la hora de diseñar la solución a un problema.

La misma política de rearticulación de los materiales se aplica también a las herramientas a utilizar. Por ejemplo, dada la necesidad de fijar los elementos a cortar o soldar, fue necesario encontrar una forma de fijar el PCB, construyéndose una morsa a partir de palillos de ropa.

Motores paso a paso.

El tiempo en que las disqueteras de 5 pulgadas y ¼ eran parte esencial de una computadora ya está muy lejos. Estos dispositivos poseen un elaborado sistema electromecánico que permite el manejo del disco flexible y de la información que allí se puede encontrar. Muchos de ellos disponen de un motor ya sea de pasos o de corriente continua.

Estos motores ofrecen una posibilidad muy interesante, en función de su relativamente fácil obtención (existen muchas disqueteras desechadas con motores y chips en buen estado), su buena respuesta y de que, en el caso de los motores de pasos, son fácilmente controlables a través de, por ejemplo, el puerto paralelo del PC.

Además, en parte gracias a su pequeño tamaño, pueden ser utilizados para fines totalmente distintos de su intención original. Se han utilizado como motor de pequeños autitos o –en nuestro caso- para tañer cuerdas de guitarra.

Los motores paso a paso transforman energía eléctrica en mecánica, convierten pulsos eléctricos en movimientos rotacionales discretos. Siempre necesitan de un circuito especial externo para controlarlo (*driver* o manejador). Son ideales para el posicionamiento, dado que lo único que se requiere es transmitir un número específico de pasos para llevarlo a una posición exacta y repetible. Alcanzan una gran precisión y pueden moverse en incrementos muy pequeños (son muy comunes los motores de 200 pasos por ciclo, es decir 1.8^{o} por paso, existiendo motores de hasta 400 pasos por ciclo), característica difícil de lograr en los motores de corriente continua, pues en estos últimos aunque se corte muy rápido la provisión de energía, la inercia del rotor continuará girando el eje hasta una posición casual.

Los motores paso a paso funcionan mediante un sistema de bobinas que se van excitando ordenadamente para producir el movimiento de cada paso (más adelante explicaremos tres formas distintas en que esto puede lograrse). Pueden ser bipolares o unipolares. Los bipolares son más difíciles de controlar y requieren una lógica más elaborada. Por lo general los motores con cuatro cables son bipolares.

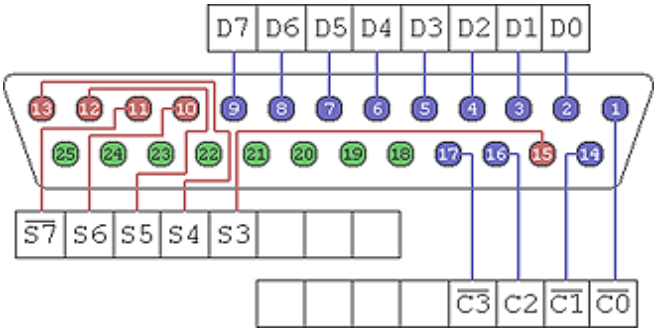
Los que revisten particular interés para nuestro trabajo son los unipolares que poseen cinco cables, cuatro correspondientes a una bobina cada uno y un cable común a todos. Algunos motores de seis cables también son unipolares de cuatro bobinas pero con dos cables comunes.

Para averiguar cuál es el cable común basta medir la resistencia entre cada cable. Entre el común y cualquiera de los otros habrá la mitad de resistencia que entre dos cables que correspondan ambos a una bobina. Poniéndolo en un ejemplo, si entre el cable rojo y el cable negro la resistencia es de aproximadamente 150 Ω, mientras que la resistencia entre el cable marrón y cualquier otro es de 75 Ω, entonces el cable marrón es el común de alimentación (12V) y los demás corresponden cada uno a una bobina.

Puerto paralelo

El puerto paralelo está formado por diecisiete líneas de señales y ocho líneas de tierra. Las líneas de señales están formadas por tres grupos, cuatro líneas de control, cinco líneas de estado y ocho líneas de datos.

El uso más común del puerto paralelo es para conectar una impresora al PC. Por lo general las líneas de control son usadas para la interfaz, control e intercambio de mensajes desde el PC a la impresora. Las líneas de estado son usadas para intercambio de mensajes, indicadores de estado desde la impresora al PC (falta papel, impresora ocupada, error en la impresora). Las líneas de datos suministran los datos de impresión del PC hacia la impresora y solamente en esa dirección, aunque nuevas implementaciones del puerto permiten una comunicación bidireccional mediante estas líneas.



Cada una de estas líneas (control, estado, datos) puede ser referenciada de modo independiente mediante un registro. Cada registro del puerto paralelo es accedido mediante una dirección.

Comúnmente la dirección base usada para el puerto paralelo es 378h o 278h, del espacio de direccionamiento de Entrada/Salida del procesador.

El BIOS (Basic Input Output System) crea en el momento de arranque una tabla en el espacio de la memoria principal (RAM) para cuatro direcciones base de puerto paralelo. Durante el arranque, el BIOS comprueba si hay puertos paralelos en las direcciones base y almacena la dirección en la tabla.

Las referencias a cada registro del puerto se realizan de la siguiente forma:

Base (datos) = base + 0
Estado = base + 1
Control = base + 2

Por ejemplo, si encontramos que la dirección base es 378h, entonces las direcciones del registro de datos, estado y control serán:

Base (datos) = 378h
Estado = 379h
Control = 37Ah

Cada una de ellas permite acceder a los siguientes bits:

Base (datos) = D0, D1, D2, D3, D4, D5, D6, D7
Estado = S3, S4, S5, S6, S7
Control = C0, C1, C2, C3

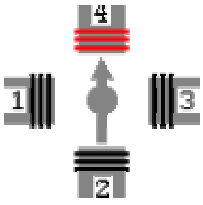
Un ejemplo de escritura al puerto paralelo, desde C (en Unix), sería:

```
#define base 0x378      // dirección del puerto
int dato;

if (ioperm(base,1,1))   // verificamos que el usuario disponga de permiso para escribir
                        // al paralelo, en caso contrario abortamos la ejecución
    fprintf(stderr, "Error: este programa debe ejecutarse con permisos de root;. %x\n", base), exit(1);

dato = XXXX;            // el valor que se quiere escribir al puerto

outb (aux, base);       // mediante outb realiza la escritura.
```

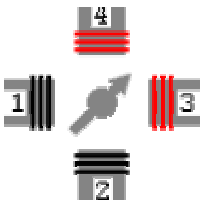
Control de motores paso a paso a través del puerto paralelo

El movimiento del motor se induce excitando ordenadamente las bobinas, generando ciclos rotacionales en la inducción magnética.

Activando sólo una bobina por vez generamos un movimiento de cuatro pasos por ciclo. En algunos motores esto brinda un funcionamiento más suave. La contrapartida es que al estar sólo una bobina activada, el torque de paso y retención es menor.



También podemos generar un movimiento de cuatro pasos por ciclo excitando de a dos bobinas. De esta forma debido a que siempre hay al menos dos bobinas activadas, se obtiene un alto torque de paso y de retención.



Una tercer manera de controlar el motor puede ser intercalando la activación de una y dos bobinas, logrando un movimiento de ocho pasos por ciclo.

Aquí explicaremos la manera en que lo hemos desarrollado, logrando controlar el movimiento de cada motor mediante un contador de dos bits. Cada unidad que se agrega al contador induce un paso del motor en una dirección, al restar unidades se inducen pasos en la dirección opuesta.

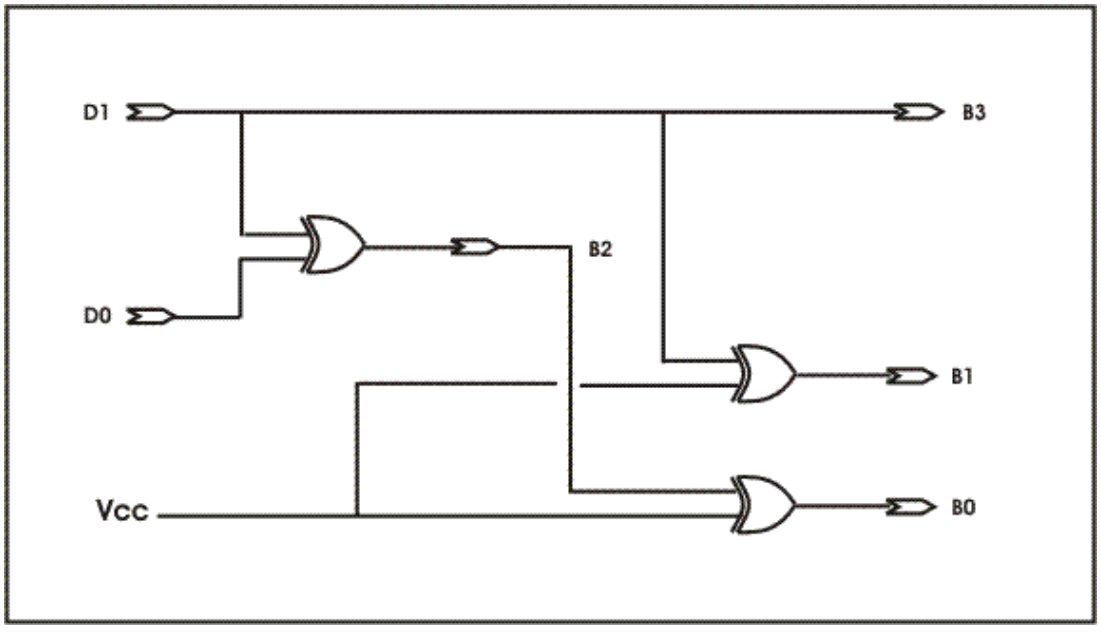
La siguiente tabla de verdad muestra la lógica del circuito a implementar, en el cual se obtienen los valores de control a partir de dos bits de datos.

D1	D0	B3	B2	B1	B0
0	0	0	0	1	1
0	1	0	1	1	0
1	0	1	1	0	0
1	1	1	0	0	1

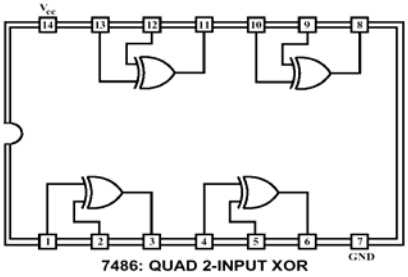
De esta forma vamos excitando las bobinas magnéticas del motor ordenadamente de a dos por paso. Los valores en 1 corresponden a voltaje alto, y, por lo tanto, la bobina correspondiente activada.

Una forma de implementar el circuito con compuertas lógicas XOR es combinando los datos de la siguiente manera:

$$B3 = D1$$
$$B2 = \overline{D1} \oplus D0$$
$$B1 = \overline{D1} = D1 \oplus Vcc$$
$$B0 = \overline{B2} = B2 \oplus Vcc$$

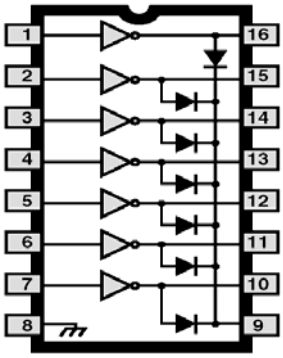


Los chips de la familia 7486 contienen 4 compuertas lógicas XOR. Por lo que con un chip de este tipo se puede implementar el circuito como se ve arriba en el dibujo.



7486: QUAD 2-INPUT XOR

Para lograr entregar el voltaje suficiente para mover el motor debemos agregar otro chip, cuya función será de levantar el voltaje de 5V que entrega el puerto paralelo hasta los 12V que necesitan las bobinas del motor para ser excitadas. Este otro chip es un 2003, que contiene 7 *Darlington Arrays*, que permiten esta adaptación.



El controlador consta entonces de dos partes con funciones bien diferenciadas. El circuito lógico que permite mapear una unidad en el contador de dos bits a un paso del motor, y el chip ULN2003 que levanta el voltaje de estos datos de 5 a 12 voltios.

De esta forma tenemos un controlador que nos permite manejar un motor utilizando sólo dos bits. Para ello usamos los bits de datos del puerto paralelo pudiendo controlar hasta cuatro motores. En la sección Red se puede encontrar una descripción detallada del servidor y de cómo se implementa este control. El código fuente del servidor se encuentra en el apéndice II.

La frontera físico-digital

Resulta interesante observar el bajo nivel de abstracción que se maneja en estas áreas: al diseñar y construir circuitos estamos también programando y de esta forma, se vuelve extremadamente difusa la frontera entre el software y el hardware, lo discreto y lo continuo, lo analógico y lo digital.

Estamos modelando un sistema lógico binario en el que un valor de 1 está representado por un determinado voltaje, mientras que otro menor determina un valor 0. Pero en la práctica apreciamos que rara vez obtenemos valores exactos de los voltajes que consideramos, de forma que una gama de voltajes corresponde a un 0 y otra lo hace a un 1, perdiéndose discreción y pasándose al terreno del continuo, comportamiento que se ve exacerbado al tratarse de sistemas construidos manualmente. Más aún, en el caso de los motores, estos mismos voltajes que representan los datos lógicos son los que poseen la capacidad de energizar las bobinas y provocar el movimiento. Por lo que su significado es ambiguo, y a la vez útil en ambas consideraciones.

Cabe destacar que debido a que los motores paso a paso son dispositivos mecánicos, deben vencer a la inercia, por lo que el tiempo de duración y la frecuencia de los pulsos aplicados es un punto muy importante a tener en cuenta. En tal sentido el motor debe terminar el paso antes que la próxima secuencia de pulsos comience. Si la frecuencia de pulsos es muy elevada, el motor puede no realizar ningún movimiento, puede comenzar a vibrar pero sin llegar a girar, puede girar erráticamente o, incluso, puede llegar a girar en sentido opuesto.

En general, la dificultad de acoplamiento de los tiempos mecánicos y los de procesamiento representa un interesante campo para la investigación. Es necesario medir los tiempos de respuesta, buscar las fronteras en los comportamientos e ir encontrando las posibilidades y limitaciones de los distintos dispositivos.

Esta problemática surge en distintos ámbitos en el proyecto, resulta relevante en el área de conectividad, en cómo escribe el servidor en el puerto paralelo y en los *steppers*. Resulta notable, también, lo útil que puede resultar la capacidad de *scripting* para medir las limitaciones de este tipo de interacción.

Utilizando guiones es posible medir en forma empírica la capacidad del sistema que se está controlando. Es posible, por ejemplo, controlar cuánto tiempo esperar para ordenar el siguiente paso a un motor o medir cuántos pasos se pierden si se envían demasiadas órdenes simultáneas al servidor.

Bit banging vs. lógica más elaborada.

La estrategia actualmente utilizada para controlar dispositivos a través del puerto paralelo tiene dos características que podrían ser mejoradas mediante la adición de lógica (circuitería) externa.

Primero, como el puerto paralelo brinda solamente ocho bits de datos, en el caso particular del control de motores paso a paso –para lo que se utilizan dos bits por motor- no es posible manejar más de cuatro motores por puerto. Sería deseable disponer de una cantidad mayor de motores.

Segundo, como el control de los motores se realiza enviando bits a una frecuencia específica, al seguirse una estrategia de *bit banging* (es decir, bombardeo de bits), enviándose los bits en forma directa a través del puerto paralelo, el procesador es utilizado en forma casi dedicada. Además, al ser necesario el envío de los bits con una determinada frecuencia (para evitar perder muchas instrucciones), la espera entre dos envíos sucesivos se implementa mediante *busy waiting*, aumentando aún más la carga del procesador[8].

En nuestras implementaciones, al utilizarse como servidores máquinas de bajo porte en forma dedicada, estas deficiencias no resultan importantes. Sin embargo, en otras implementaciones de la misma arquitectura, podría ser deseable utilizar lógica dedicada, liberando al procesador y permitiendo manejar más dispositivos.

En el marco de este proyecto inicialmente se trabajó en algunos diseños que proponían resolver estos problemas implementando la lógica necesaria en un dispositivo lógico programable o PLD. Sin embargo, dada la necesidad de obtener resultados funcionales lo antes posible y la dificultad de implementación que implica este enfoque, se optó por la solución finalmente implementada. Esperamos que futuros trabajos permitan la construcción de mejores soluciones.

La solución basada en PLDs consiste en la codificación en ellos de distintos comandos que pueden ser enviados a los motores, evitándose así la necesidad del *bit banging*. Con este diseño el procesador simplemente envía la orden de ejecución de un comando específico

[8] Claramente es necesario mejorarlo en futuras versiones, pudiéndose utilizar el comando *wait();*.

(por ejemplo una vuelta completa en sentido levógiro) y a partir de ahí es el PLD quien se encarga de enviar los bits a los motores.

Además, utilizando algún *bus* que permitiese conectar varios dispositivos con pocas líneas de datos podría desarrollarse un sistema para el manejo de múltiples motores. Una base para el desarrollo de esta estrategia podría constituirla el proyecto de Gustavo Brown [14], el cual utilizando el bus I2C, permitiría –utilizando sólo dos líneas de datos- controlar un número arbitrario de dispositivos. En este caso, en el PLD debería implementarse además la lógica correspondiente al protocolo de bus.

Interacción con tecnologías de última generación.

Si bien consideramos que la rearticulación tecnológica es importante *per se*, resultaría inútil si no se lograra encontrar mecanismos de interacción de la tecnología reciclada con los productos más novedosos y potentes.

Es por ello que hemos puesto especial énfasis en el diseño de mecanismos de interacción que permitan explotar las ventajas de los componentes más modernos, pero manteniendo la flexibilidad necesaria a la hora de utilizar elementos obsoletos.

Un mecanismo muy eficiente de interacción se obtuvo siguiendo la política de modularización en la arquitectura de los sistemas construidos. De esta forma resultó posible usar elementos obsoletos para brindar soluciones específicas, manteniendo la utilización de máquinas de mediano porte para las tareas de cómputo más pesado.

Ejemplo de ello resulta, como hemos comentado, la utilización de servidores Linux de muy poca capacidad (386 y 486 de 66 y 100 Mhz), los cuales controlan motores reciclados mediante lógica dedicada (compuesta por elementos reciclados), pero que se conectan con clientes mucho más poderosos a través de una red TCP/IP.

De esta forma, la conectividad a través de TCP/IP oficia como agente articulante de las distintas tecnologías. En nuestro entender, es factible una arquitectura distribuida, en la cual distintos componentes (de distintas procedencias y capacidades) interactúan a través de la red TCP. Es por ello que la utilización de PICs y pequeños PLDs o chips para implementar nodos de red TCP resulta una opción muy interesante.

Otra herramienta importante a la hora de vincular tecnologías tan disímiles es la posibilidad de construir guiones que se interpreten en tiempo real y así poder medir y ajustar el funcionamiento de los componentes rearticulados que, de por sí, presentan patrones de comportamientos muchas veces delicados.

La implementación del subsistema de *scripting*[9] otorga un enfoque flexible y adaptable, permitiendo obtener el mayor rendimiento de los componentes.

Como ejemplo podemos citar la adecuación de los tiempos de espera entre comandos sucesivos que se envían a los motores de disquetera reciclados, de acuerdo al poder de la máquina que oficia de servidor y la carga que en ese momento (cantidad de clientes, cantidad de motores) deba soportar.

[9] Ver capítulo 4.

Máquinas obsoletas y software libre.

Un elemento que ha formado parte de todos los prototipos que hemos implementado ha sido el *"servidor de motores"*, una pequeña computadora conectada a la red TCP/IP que se encarga del control de los motores paso a paso.

Este servidor hubiera sido impensable sin la utilización de una versión recortada del sistema operativo Linux, que permite un control muy fino tanto de los elementos de conectividad vía red, como del hardware de la máquina (específicamente el puerto paralelo).

El disponer de un sistema operativo tan robusto como Linux, *en formato de código abierto*, nos permitió recortar sus funcionalidades de forma tal de obtener el comportamiento y el rendimiento necesario, con un sistema que pueda instalarse cómodamente en pequeños discos duros. Todos los servidores que implementamos utilizan discos de 180 Mb o menos, tamaños irrisorios dados los parámetros actuales.

capítulo

3

Red.

Dada la necesidad de controlar elementos que se encuentren físicamente dispersos, surge como solución natural la utilización de algún tipo de red de datos.

La arquitectura elegida se basa en el enfoque cliente-servidor sobre TCP/IP, en la cual los servidores controlan los dispositivos físicos que forman parte del sistema. Como hemos dicho anteriormente, sólo hemos implementado el control de los motores paso a paso.

TCP/IP como tecnología de red, fue elegido en función del interés que reviste para el NAT, interés que surge de la ubicuidad de la misma, de su escalabilidad (permite crear sistemas de red con muy bajo costo) y su portabilidad (mediante la utilización de chips dedicados es posible dotar de conectividad a prácticamente cualquier dispositivo).

En función del planteo de nuestro proyecto y de la necesidad de abaratar costos, el hardware utilizado consiste en tarjetas de red obsoletas (de 10 Mbits), cables coaxiales (los cuales han caído en desuso, siendo suplantados por UTP o fibra óptica) y servidores de muy bajo porte.

Los servidores están constituidos por máquinas pequeñas (386 y 486) corriendo una versión muy recortada del sistema operativo Linux, recorte que nos permitió utilizar discos duros muy pequeños (de unos 200 MB).

Dado el poco poder de procesamiento de los servidores y la modularidad pretendida (que no permite caracterizar *a priori* los clientes y sus requerimientos), se optó por implementar la mayor parte del software en los clientes, dejando como único cometido de los servidores la atención de múltiples clientes y el control de los motores paso a paso.

Servidor.

El servidor se desarrolló en C[10], sobre el sistema operativo Linux. Se optó por este lenguaje, dada la alta velocidad de ejecución que ofrece, necesidad aumentada por el bajo poder del hardware utilizado.

Al ser implementado en C, y habiéndose decidido que la conexión cliente / servidor sería vía TCP/IP, se implementó utilizando Berkeley *sockets*[11] de forma tal de permitir múltiples clientes simultáneos.

Los *sockets* son una forma de comunicar programas entre sí utilizando descriptores de archivo estándar de Unix. Un descriptor de archivo no es más que un entero asociado a un archivo abierto, pero al tratarse del sistema operativo Unix, ese archivo puede ser una conexión de red, una cola FIFO, un tubo (*pipe*), una terminal, un archivo real de disco, o cualquier otra cosa. Por eso, para establecer una conexión entre programas (procesos) a

[10] El código del servidor utilizado en el Tecnocordio (recordar que se implementaron múltiples versiones, de acuerdo a los requerimientos de los distintos prototipos) se encuentra disponible en el Apéndice.
[11] Para un excelente tutorial sobre *sockets* y programación en red utilizando C, sugerimos consultar la guía Beej en http://www.ecst.csuchico.edu/~beej/guide/net/

través de una red TCP/IP (como Internet) se debe hacer utilizando descriptores de archivo (esto es, sockets).

Un esquema del funcionamiento del servidor sería:

```
Repetir
        Esperar conexión.
        Si hay conexión crear un proceso hijo  (fork)

        Si se está ejecutando el hijo
                Mientras el usuario no de un quit
                        Recibir comando
                        Ejecutar comando
                        si está en modo verbose (ver protocolo) devolver el OK
                Fin mientras
        Fin si.
Fin repetir
```

Gracias al fork() que se realiza con cada nueva conexión, es posible atender múltiples clientes con cada servidor.

Dentro de este esquema de funcionamiento, es necesario definir un protocolo de comunicación entre los clientes y el servidor.

Protocolo.

El protocolo especifica las reglas de comunicación entre el cliente y el servidor, especificando qué información se puede enviar o recibir y en qué forma se codifica la misma.

En nuestro sistema, la comunicación se reduce al envío de órdenes por parte del cliente, respecto al control de los motores paso a paso conectados al servidor y a las respuestas enviadas por el servidor.

Al realizarse la conexión sobre TCP/IP, no resultó necesario incluir en el protocolo elementos de la comunicación *per se* (como por ejemplo un saludo o *handshaking*), necesitándose definir sólo lo relativo al control de los motores.

De esta forma, se definieron comandos de control específico de los motores que permiten girarlos en una u otra dirección o ejecutar comandos un poco más complejos (como por ejemplo el tañido de un motor o el movimiento de más de uno simultáneamente).

Una vez establecida la conexión, los comandos y sus parámetros se envían en un único mensaje, donde el primer byte corresponde al comando y los siguientes a los parámetros.

Como muchas veces es necesario concatenar comandos (en particular cuando se controlan los motores a través de guiones), resulta importante disponer de un indicador que avise cuándo se terminó la ejecución del último comando. En vista de ello, se creó un comando que pone al servidor en modo *verbose*. Cuando el servidor se encuentra funcionando en este modo, devuelve un indicador de finalización luego de la ejecución de cada comando. Así, el cliente puede detener el envío del siguiente comando hasta no bien haya recibido la confirmación de ejecución del último enviado.

Además, dado que el control de estos motores se realiza a través del puerto paralelo, siguiendo una estrategia de *bit banging*[12], es necesario un tiempo de espera para disminuir la velocidad en que se envían los datos a los motores. Este tiempo es regulable por el cliente (se definen comandos que aumentan o disminuyen el tiempo de espera o *delay factor*), pero es necesario tener en cuenta que si no se espera lo suficiente, los motores no disponen de tiempo para ejecutar el paso y se pierden pasos generando una asincronía.

En caso de necesitarse otro comportamiento de los motores, resulta muy sencillo implementar nuevos comandos. Hasta el momento hemos implementado dos versiones, que detallamos en lo que resta de esta sección.

Primera versión[13].

Esta versión del protocolo, se especificó para ser utilizada en los primeros prototipos sonoros (como el modelo de una cuerda manipulada por dos motores), por lo cual necesita solamente controlar dos motores (además, sólo se utilizó un servidor para todo el prototipo).

Para esta versión se definieron comandos que permitían mover uno o dos motores una cantidad arbitraria de pasos y efectuar tañidos, así como comandos para manejar los parámetros de funcionamiento *verbose* y *delayFactor*.

Dada la similitud de esta versión con la siguiente, sólo daremos la especificación formal de los comandos para la segunda –y más completa- versión.

Segunda versión[14].

Esta versión fue implementada para el Tecnocordio, lo cual implica controlar la mayor cantidad posible de motores. Sin embargo, como sólo se utilizarían los bits de datos del puerto paralelo (sin utilizar más lógica externa que la multiplexación vista anteriormente), solamente es necesario el control de cuatro motores por servidor (siempre es posible controlar más motores agregando nuevos servidores, para lo cual no es necesario modificar la arquitectura de ninguna forma).

Los comandos disponibles son, entonces:

Comando	demToggleVerbose
Parámetros	Ninguno
Descripción	Permite el encendido o apagado del "modo *verbose*". Cuando el servidor se encuentra en modo *verbose*, luego de la ejecución de cualquier comando, devuelve un OK al cliente.

Comando	demWalk
Parámetros	Motor, dirección, pasos.
Descripción	Hace girar un motor una cantidad determinada de pasos en una dirección.

Comando	demTest
Parámetros	Ninguno
Descripción	Ejecuta el procedimiento test() que mueve ambos motores. Es utilizado para testear el funcioamiento del sistema.

[12] Ver la sección de motores paso a paso, página 17.
[13] Para ver los fuentes correspondientes al *daemon* o servidor, ver el Apéndice.
[14] Para ver los fuentes correspondientes al *daemon* o servidor, ver el Apéndice.

Comando	demIncDF
Parámetros	Ninguno
Descripción	Aumenta el *delayFactor*, es decir el tiempo que el servidor ocupa en *busy waiting*, cuando envía los datos al puerto paralelo (es decir, a los motores).

Comando	demDecDF
Parámetros	Ninguno
Descripción	Diminuye el *delayFactor*, es decir el tiempo que el servidor ocupa en *busy waiting*, cuando envía los datos al puerto paralelo (es decir, a los motores).

Comando	demPang
Parámetros	Motor
Descripción	Efectúa un movimiento de tañido en el motor seleccionado. El tañido son 20 pasos en una dirección opuesta a la última utilizada por este comando.

Comando	demPang2
Parámetros	Motor1 Motor2
Descripción	Efectúa un movimiento de tañido en los dos motores seleccionado.

Comando	demPang3
Parámetros	Motor1 Motor2 Motor3
Descripción	Efectúa un movimiento de tañido en los tres motores seleccionado.

Comando	demPang4
Parámetros	Ninguno
Descripción	Efectúa un movimiento de tañido en los cuatro motores seleccionados.

Comando	demQuit
Parámetros	Ninguno
Descripción	Termina la conexión con el servidor.

Las principales constantes utilizadas en esta versión del servidor son:

Constantes de conexión TCP/IP		
Nombre	Valor	Descripción
MYPORT	3490	Puerto (TCP) donde acepta nuevas conexiones.
BACKLOG	10	Tamaño de la cola de conexiones pendientes.
MAXDATASIZE	100	Máximo numero de bytes que se pueden obtener en un mensaje.

Constantes que representan los comandos del protocolo.		
Nombre	Valor	Descripción
demWalk	1	El nombre de todas las constantes de este tipo es igual al nombre del comando definido en el protocolo, para una descripción del funcionamiento, por favor ver la definición del mismo.
demWalk2[15]	2	
demToggleVerbose	3	
demPang	4	
demPang2	8	
demPang3	9	
demPang4	0	
demQuit	5	
demIncDF	6	
demDecDF	7	

Constantes utilizadas en el control de los motores paso a paso.		
Nombre	Valor	Descripción
IZQ	0	La dirección izquierda de rotación de los motores (levógira)
DER	1	La dirección derecha de rotación de los motores (dextrógira)
OK	"0"	El carácter devuelto como indicación de OK.

[15] Sólo implementado en la primera versión del servidor.

capítulo 4

Guionado.

Lenguajes de programación.

Un lenguaje de programación se define como un "conjunto de signos y reglas que permite la comunicación con un ordenador" [15]. Dentro de esta amplia definición, existen distintas taxonomías que agrupan los lenguajes de acuerdo a distintos criterios, como ser: nivel de abstracción, finalidad del lenguaje o alguna característica intrínseca.

Una de las clasificaciones más utilizadas es la introducida por Aho y Ullman [16], quienes clasifican los lenguajes según las herramientas utilizadas en el proceso de traducción, obteniendo de esta forma

Lenguajes Ensamblados: Aquellos que consisten en una representación simbólica de las instrucciones correspondientes al lenguaje ensamblador de alguna arquitectura específica con lo que, casi siempre, la correspondencia entre las instrucciones de este lenguaje, y las del lenguaje máquina son de 1 a 1.

Lenguajes Compilados: Son aquellos que son traducidos de un lenguaje de alto nivel (como FORTRAN o PASCAL) a lenguaje máquina o a lenguaje ensamblador, produciendo un programa objeto permanente. Una vez que el binario se generó, puede ejecutarse directamente sin volver a considerar el código fuente. La mayoría del software se distribuye como binarios compilados desde código que uno no puede ver. Estos lenguajes tienden a brindar una excelente performance y tienen acceso completo al sistema operativo, pero a la vez son bastante difíciles de programar. El ejemplo más extendido de este tipo es el lenguaje C.

Lenguajes Interpretados: Un lenguaje interpretado depende del programa intérprete que lee el código fuente y lo traduce al vuelo a llamadas al sistema y operaciones. Estos lenguajes tienden a ser más lentos que los compilados, y frecuentemente tienen acceso limitado al sistema operativo y al hardware. Pero por otro lado resultan más fáciles de programar brindando mayores posibilidades de interacción en detrimento de la performance.

Lenguajes de pseudo código. Desde 1990 una nueva clase de lenguajes ha cobrado cada vez más importancia. Son lenguajes híbridos que utilizan parte de compilación y de interpretación. El código escrito en estos lenguajes, llamados de pseudo código (P-code) o de Bytecode, se traduce a una forma binaria que es lo que en efecto se ejecuta. Sin embargo, no es código de máquina sino un pseudo código, que es independiente de una arquitectura de hardware específica. Al correr el programa p-code es interpretado. Algunos ejemplos de estos lenguajes son Python, Perl, Java y C#.

Guiones

Dentro de los lenguajes interpretados, encontramos los **lenguajes de guionado** o de ***scripting***. Son aquellos que se utilizan para extender la capacidad de una arquitectura preexistente. Por ejemplo, en las páginas web es posible incrustar código que ofrezca

nuevos niveles de interacción con el usuario. De igual forma, muchos programas comerciales, como ser AutoCad, o 3DS Max, implementan intérpretes de lenguajes de guionado propietarios, que permiten su extensión, pero constituyendo casos de muy difícil acceso dado que están dedicados a medios muy sofisticados.

El interés en el guionado, entonces, radica en su capacidad de extender las posibilidades de un sistema, en particular las capacidades expresivas o educativas del mismo (de ahí su éxito en programas de diseño como los mencionados). A través de los lenguajes de guionado resulta relativamente fácil crear ambientes expresivos a partir de las funcionalidades básicas implementadas en el sistema huésped.

Dada la finalidad del sistema a construirse en el marco de este proyecto, la posibilidad de incluir la potencialidad de los lenguajes de guionado ofrece un enorme atractivo.

Por ejemplo, al utilizarse el sistema con fines pedagógicos, es posible brindar un acercamiento novedoso a los conceptos básicos de programación. Es posible implementar un sistema de retroalimentación, mediante el cual el estudiante puede ver resultados tangibles de sus programas en tiempo real. Por ejemplo, al construir un pequeño guión que mueva un motor paso a paso, nada más gráfico para ejemplificar estructuras de repetición como el *for* o el *while* que ver girar un motor mientras se cumple la condición.

De igual forma, para poder utilizarse el sistema en función de los intereses expresivos concretos de un artista, se deberá brindar la suficiente flexibilidad para cubrir sus necesidades. Otra vez, el artista deberá manejar la tecnología implicada, en este caso deberá saber cómo programar el comportamiento deseado, reapareciendo la apropiación tecnológica como *el* medio de inserción sociocultural.

Beanshell.

Al comenzar a evaluar las distintas alternativas con la intención de incluir funcionalidades de guionado a nuestra arquitectura, surgió la posibilidad de desarrollar un lenguaje de guionado propio, a la manera de los grandes productos comerciales.

Si bien esta alternativa resultaba interesante de por sí, las posibilidades reales de construcción de un sistema sofisticado de *scripting*, dados los parámetros de disponibilidad de recursos humanos y de tiempo eran pocas.

Sin embargo, en el NAT se habían realizado alguna implementación de sistemas de guionado mediante Beanshell[17], un sistema de guionado liviano en Java, desarrollado por Pat Niemeyer que ahora es propiedad de SUN.

Tradicionalmente, la principal diferencia entre utilizar un lenguaje de guionado y un lenguaje de aplicación radica en el sistema de tipos. Si bien es verdad que la diferencia más obvia consiste en compilación vs. interpretación, la compilación o el interpretado en sí mismos no cambian de forma fundamental la manera de trabajar con un lenguaje.

Sin embargo, el sistema de tipos es el que permite a un compilador analizar la estructura de un código para verificar su corrección. Sin tipos, la compilación se reduce apenas a una verificación de la gramática y a una optimización para la velocidad. De la perspectiva del desarrollador, es también el sistema de tipos quien caracteriza la manera de la cual interactuamos con el código.

Tradicionalmente también, los lenguajes de guionado han limitado el sistema de tipos en función de la simplicidad. La mayoría de estos lenguajes disponen de un sistema de tipos con apenas un puñado de tipos tales como secuencias, números, o listas simples.

BeanShell, en cambio, es una nueva clase de lenguaje de guionado, ofreciendo el poderío de Java en un ambiente interpretado. De esta forma se obtiene todo el sistema de tipos de Java, que al ser orientado a objetos, dispone de un sistema de tipos muy poderoso, siendo

también posible escribir programas que mantienen la estructura de un guión tradicional mientras que todavía se mantiene el marco de sintaxis de Java.

De esta forma, a través de la emulación transparente de variables y parámetros tipados que realiza Beanshell, se obtiene un acceso en tiempo real a las funcionalidades implementadas en nuestro sistema, manteniéndose la flexibilidad necesaria para un uso expresivo o educativo del mismo.

Integración de Beanshell.

Beanshell ofrece un espacio de nombres, dentro del cual es posible instanciar variables, crear objetos y crear nuevas clases en línea.

Nuestra arquitectura vincula las distintas funcionalidades implementadas a través de un sistema de derivación de componentes de interfaz[16].

El desarrollador, al crear cada componente de interfaz debe proveer las funcionalidades necesarias para la integración con el sistema y la interacción con otros componentes, pero – al heredar de una clase plantilla común a todos los componentes- existen ciertas funcionalidades comunes a todos los componentes que ya se encuentran implementadas.

Entre estas funcionalidades, se encuentra la asociación de un sistema de guionado y la inclusión del objeto que implementa la funcionalidad al espacio de nombres.

De esta forma se obtiene, sin ningún costo, la posibilidad de controlar mediante guiones la nueva funcionalidad.

Veamos un ejemplo de control del cliente de los servidores de motores paso a paso. Al derivarse un componente se especifica el nombre del objeto que implementa la funcionalidad dentro del espacio de nombres, en este caso suponemos un nombre “c”. Suponemos también que el cliente ya estableció la conexión TCP/IP con el servidor, lo cual es razonable dada la funcionalidad implementada en el componente de interfaz.

```
If (!c.getVerbose()) c.enviarComando(c.demToggleVerbose);
for (int i = 0; i < 5; i++) {
        c.enviarComando (c.demWalk, c.IZQ, 1, 20);
        c.esperarOK();
        c.enviarComando (c.demWalk, c.DER, 1, 20);
}
```

Este guión, ordena al servidor mover 5 veces 20 pasos a la izquierda y luego 20 pasos a la derecha a su motor número 1.

Podemos ver que quien escriba un guión de este tipo, obtiene un *feedback* muy elocuente, además de poder verificar el funcionamiento del sistema en forma cabal.

Gracias a la posibilidad de almacenar y cargar guiones, es posible desarrollar comportamientos muy sofisticados con muy poco esfuerzo. Además, dentro de la arquitectura se deja el punto de entrada para la implementación del acceso a una base de datos de guiones, la que, al permitir la categorización de los guiones en forma eficiente, será una herramienta poderosísima a la hora de utilizar el sistema.

Para un ejemplo de utilización de bases de datos con este fin, aconsejamos ver el proyecto “Dibujar por nombres” del Dr. Delacroix, tutor del presente proyecto.

[16] Para una especificación más detallada de la arquitectura, ver el capítulo 7.

capítulo 5

Imagen.

Si bien el área relacionada con la imagen es muy amplia, por lo que *a priori* puede parecer inabarcable, resulta de interés para nuestro proyecto dadas las evidentes capacidades expresivas de la computación gráfica y las posibilidades de interacción que el análisis de imágenes captadas provee.

Sin embargo, teniendo en cuenta que esta área no hace a lo medular del proyecto, se decidió construir únicamente bocetos de estas posibilidades, desarrollándose módulos de control a través de la captura de imágenes, de graficación en tiempo real y de reproducción de vídeos previamente grabados, pero sin integrarlos aún a la arquitectura principal del sistema.

La integración, sin embargo, no es difícil, pero se dejó para una etapa futura en la cual el soporte de hardware implementado (es decir, el subsistema encargado de la producción de sonidos) permita una utilización realmente expresiva de la tipología de interfaz implementada o de los programas de graficación desarrollados.

Multimedia.

Cualquier conjunto de datos que cambie significativamente con el tiempo puede ser caracterizado como un "medio basado en el tiempo" (*"time-based media"* o TBM), ejemplos de ello son clips de audio o vídeo, secuencias MIDI, animaciones, etc. Estos datos pueden obtenerse desde varios tipos de fuentes, como ser archivos locales o de red, cámaras, micrófonos o transmisiones en vivo.

Una característica fundamental y común a todos los medios basados en el tiempo, es su necesidad de ser recibidos y procesados ajustándose estrictamente al tiempo requerido. Por ejemplo, si al reproducir una película los datos no llegan lo suficientemente rápido, aparecen pausas en la proyección. Pero, si se pretende mantener la velocidad de proyección puede suceder que la película aparezca con saltos si se pierden datos o si se saltean marcos en forma intencional para mantener el ritmo de reproducción.

En algunos casos, la presentación de los datos no puede comenzar de inmediato. El tiempo requerido para que la presentación efectivamente comience es conocido como tiempo de latencia, demora que es particularmente común cuando se solicita un flujo de datos a través de una red.

Las presentaciones multimedia combinan distintos flujos de información sincronizados. En este caso es esencial tener en cuenta los tiempos de latencia de los distintos flujos de información para evitar que la presentación de éstos comience en distintos momentos.

Tradicionalmente cuanto mayor calidad de presentación de los datos se requiere, mayor es el tamaño de los mismos y más poder de procesamiento y ancho de banda son necesarios.

Los requisitos de ancho de banda (esto es, la cantidad de información transmitida por unidad de tiempo) y de poder de procesamiento varían entre los distintos tipos de TBM, por ejemplo para una presentación de vídeo de calidad, es necesario presentar unos 30 cuadros por segundo, aunque una tasa de 15 marcos por segundo es suficiente para simular un movimiento continuo en la imagen.

Sin embargo, gracias a los avances en algoritmia, poder de procesamiento y vídeo digital, cada vez es más posible buscar e interpretar eventos con un alto contenido semántico dentro de los TBM (en particular audio y vídeo). Lo cual resulta muy importante para áreas como medicina, vigilancia e interacción hombre-computadora (HCI).

JMF.

Al igual que en la mayoría de nuestros desarrollos, se utilizaron las facilidades provistas por Java, en este caso el Java Media Framework [18] o JMF.

El Java Media Framework (JMF) es una interfaz de programación de aplicaciones (API) para incorporar TBM en aplicaciones Java. JMF permite fácilmente programar aplicaciones personalizadas basadas en módulos existentes e integrar las nuevas capacidades al marco de trabajo, pudiendo capturar *media data*, acceder a datos en bruto *(raw data)*, y desarrollar multiplexores, demultiplexores, *codecs*, efectos y *renderers* personalizados (a través de componentes enchufables o *plug-ins*).

Un demultiplexor extrae pistas individuales de un flujo de datos multiplexado, mientras que un multiplexor realiza la tarea opuesta, toma pistas individuales de datos y los combina en un flujo común.

Un códec (codificador-decodificador) comprime y descomprime datos, o los transforma de un formato a otro. Es un componente de proceso con solamente una entrada y una salida que lee datos de una pista individual, los procesa y entrega los resultados. Un códec trabaja con un tipo de datos a la vez, no con un flujo de distintas pistas combinadas (por ejemplo audio y video). Al codificar un flujo de datos, se lo convierte a un formato comprimido que permite describir la misma información utilizando menos espacio, disminuyendo los requerimientos de almacenamiento o ancho de banda. Al decodificarla se convierte a un formato bruto (sin compresión o *raw*), que permite su presentación.

Un filtro de efecto modifica los datos para crear algún efecto. Pueden ser aplicados antes del proceso efectuado por el códec, pero por lo general se aplican efectos a datos sin comprimir.

Un *renderer* (presentador) es un componente de presentación de los datos, siendo un ejemplo típico la tarjeta de video que luego envía los datos un dispositivo de presentación como un monitor.

Un dispositivo de captura (micrófono, cámara, etc.) podría entregar múltiples flujos de datos, estos flujos pueden manipularse por separado o combinarse en un flujo único multiplexado.

Un efecto enchufable es un tipo especializado de códec que realiza algún proceso adicional a la pista recibida. Puede ser usado tanto como un efecto de preproceso o posproceso. Por ejemplo, si el efecto hace a la presentación de los datos, seguramente se aplicará a los datos ya decodificados.

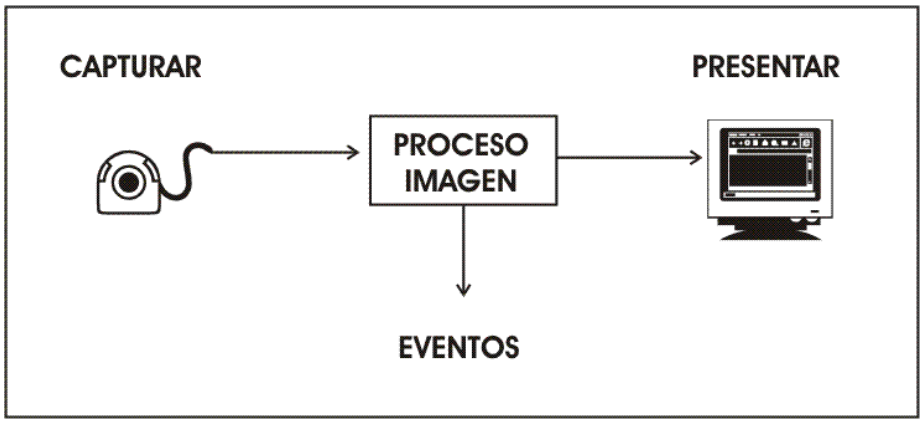
Interacción vía *webcams.*

Se implementó una aplicación que ofrece una interacción rudimentaria utilizando cámaras de vídeo del tipo de *web-cams* o cualquier otra fuente de vídeo en tiempo real soportada por JMF.

Fundamentalmente se trata del análisis de la imagen filmada y la generación de eventos a partir de ésta. Eventos que pueden representar desde una variación gráfica identificable en la pantalla hasta complicados comandos de interacción con sistemas externos.

El desarrollo de esta aplicación proyecta brindar una interfaz no tradicional de control de los distintos sistemas a los que pueda aplicarse.

A grandes rasgos la aplicación desarrollada se basa en el reconocimiento de movimiento en función de la comparación entre dos marcos (o *frames*) consecutivos. Se comparan, píxel por píxel, el último marco capturado y el inmediatamente anterior y se determina cuán diferentes son. De esta forma se obtiene una relación entre píxeles que permanecen incambiados y píxeles que cambiaron respecto al marco previo. Esta proporción brinda una medida de cuánto movimiento está sucediendo frente a la cámara, lo que permite generar eventos asociados a esa cantidad de movimiento.



La ejecución comienza averiguando los dispositivos de captura registrados en la máquina y presenta la posibilidad de seleccionar el dispositivo a utilizar (cuál cámara en caso de disponer de más de una). Luego se "enchufa" el componente de efecto, que pasamos a describir con más detalle.

Proceso imagen

Al comenzar el proceso se ejecuta la función *init*, la cual permite la inicialización de variables y la creación de objetos extra que puedan necesitarse para la ejecución de los propios eventos generados.

Se dispone de una variable *censor*, la cual cuenta los píxeles que han cambiado respecto a su valor previo. Al finalizar el conteo del marco, se realizan las acciones correspondientes.

La secuencia fundamental del proceso es la siguiente:

```
para cada píxel del marco {
        si activa entonces incrementar censor
                copiar píxel
                pintar píxel
}
considerar acciones
```

Para cada píxel se resuelven todas las acciones necesarias, así basta con recorrer solamente una vez el marco.

Primero se determina si el píxel ha variado suficientemente como para marcarlo, comparándolo con su correspondiente predecesor (activa), luego se guarda en la variable marco previo (copiar píxel), para que esté disponible luego al procesar el siguiente *frame*. Después se determina cómo pintar este píxel en la pantalla (pintar píxel). Los píxeles que determinan que ha habido movimiento por lo general se pintan con un color característico para una fácil apreciación visual del volumen de movimiento a la hora de ejecutar las pruebas.

Una vez procesados todos los píxeles del marco se deciden las acciones a tomar considerando la tasa de movimiento actual (píxeles que cambiaron sobre el total de píxeles disponibles). Si la variación entre ambos marcos se considera suficiente (si supera un umbral determinado), esto podría traducirse en la activación de una bandera, que se desactivaría después de cierto tiempo sin movimiento. O bien podría gatillar una acción específica cada vez que supera dicho umbral. O podría manejarse también como una relación más directa entre la cantidad de movimiento y el volumen de acciones. Permitiendo por ejemplo a más movimiento generar más sonido, como veremos más adelante.

Un marco está representado por un arreglo conteniendo los valores RGB de cada píxel. El color de un píxel en este esquema está dado por sus 3 componentes de color RGB (Red/Rojo, Green/Verde, Blue/Azul).

Consideremos la comparación entre un píxel y su predecesor:

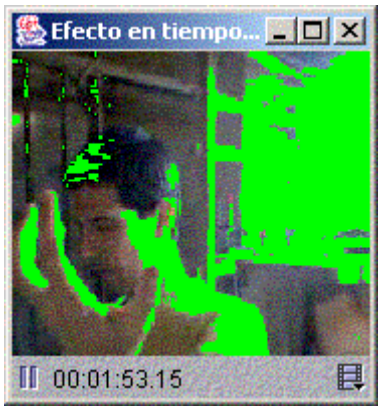
```
boolean activa( píxel ){
  return(
        | rojo (píxel) – rojo( píxel correspondiente en marco previo ) | > sensibilidad
        ||
        | verde (píxel) – verde( píxel correspondiente en marco previo ) | > sensibilidad
        ||
        | azul (píxel) – azul( píxel correspondiente en marco previo ) | > sensibilidad
  );
}
```

Aquí se inspecciona el píxel comparando la información del arreglo que contiene el marco que se está procesando con la del marco previo. Se pide que alguno de sus componentes haya variado más que el valor de *sensibilidad*, que es ajustable para hacer la detección más o menos sensible.

En la práctica no es posible tener una medida muy fina de la variación registrada dado que estas cámaras son muy sensibles a pequeñas variaciones de luz, y muchas veces ajustan automáticamente esta sensibilidad. Rara vez se encuentren dos *frames* consecutivos exactamente iguales, a pesar de no haber habido movimiento concreto frente a la cámara.

Otra posible estrategia de inspección de un píxel puede ser el reconocimiento de un color particular. Análogamente a contar los píxeles que han cambiado su valor, podría contarse cuántos píxeles se tienen de un color determinado. No se compararía un píxel contra su anterior correspondiente sino contra un valor concreto.

Aquí podemos ver una imagen donde los píxeles suficientemente blancos se pintan de verde.



Dividir en zonas

Este mismo esquema que aquí presentamos considerando todos los píxeles de un *frame*, puede llevarse a cabo determinando distintas zonas en la pantalla. Puede calcularse independientemente para cada zona el conteo de píxeles que han cambiado, y las distintas acciones a tomar. Así se tiene la capacidad de generar eventos asociados cada uno a su zona. Se debe realizar el conteo independientemente y así mismo el cálculo de las tasas de movimiento para cada zona. Las variables correspondientes ahora se convierten en arreglos, con una celda para cada zona.

Al iniciar el proceso (función *init*) se determinan las zonas a utilizar.

```
determinar Zonas

    cantZonas= 2;
    tamanio= new int[cantZonas+1];

    z1x1= 0;
    z1x2= ancho / 4;
    z1y1= 0;
    z1y2= alto;
    tamanio[1]= (z1x2-z1x1) * (z1y2-z1y1);

    z2x1= 3 * ancho / 4;
    z2x2= ancho;
    z2y1= 0;
    z2y2= alto;
    tamanio[2]= (z2x2-z2x1) * (z2y2-z2y1);

fin det Zonas
```

Para cada zona se guardan cuatro variables con sus coordenadas, y se guarda el total de píxeles de cada zona para los cálculos posteriores necesarios.

En el ejemplo se estarían determinando dos zonas que corresponden a dos franjas verticales de ancho igual a un cuarto de la imagen total cada una y situadas en ambos márgenes de la imagen.

Además de las zonas delimitadas aquí, se mantiene una zona por defecto (zona 0) que es todo lo que sobra de la pantalla, los píxeles que no han sido asignados explícitamente a una zona determinada.

Podría además cada zona considerar sus píxeles activados de distinta manera. Como mencionamos antes, una podría controlar la diferencia de píxeles entre marcos y otra medir el nivel de cierto color de sus píxeles correspondientes. Esto se lograría discerniendo en la función *activa* a qué zona pertenece el píxel y luego realizar la consideración correspondiente a su zona. Las distintas combinaciones de posibilidades abren una gran gama de estrategias a implementar.

Presentar

La función *pinta* determina el color con el que se presentará el píxel actual. Por lo general se diferencian los píxeles especiales (los que han cambiado), y se pinta cada zona con un efecto o con un color también característico para su individualización.
La variable *zona* se sobrescribe en el proceso de cada píxel dependiendo de sus coordenadas, así se dispone de esa información en el momento que se considera tal píxel.

```
pinta ( píxel )
        si es un píxel especial              // fue marcado en activa
                desmarcarlo
                aplicar efecto correspondiente y presentar
        si no
                switch ( zona )              // el píxel pertenece a esta zona
                        caso 1: aplicar efecto1 y presentar
                        caso 2: aplicar efecto2 y presentar
                        por defecto: aplicar efecto por defecto y presentar   // caso 0
                fin switch
        fin si
fin pinta
```

En esta imagen podemos ver cuatro zonas que abarcan todo el marco, aplicando un efecto distinto en cada una. Los píxeles en rojo son los que experimentaron movimiento.

Manipulando las tres componentes de color de un píxel puede muy fácilmente aplicarse distintos efectos a cada uno para su individualización visual.

La primer opción es copiar los datos como vienen para que se vea igual a como lo capta la cámara.

```
rojo a presentar ( píxel ) = rojo( píxel)
verde a presentar ( píxel ) = verde( píxel)
azul a presentar ( píxel ) = azul( píxel)
```

Es posible aplicar una máscara roja, verde, o azul maximizando la componente que corresponda y manteniendo el valor de las otras dos.

```
rojo a presentar ( píxel ) = rojo( píxel)
verde a presentar ( píxel ) = 255
azul a presentar ( píxel ) = azul( píxel)
```

Se puede directamente aplicar un color determinado al píxel, ignorando el valor que trae.

```
rojo a presentar ( píxel ) = 255
verde a presentar ( píxel ) = 0
azul a presentar ( píxel ) = 0
```

Aplicando a cada componente el promedio de las tres se obtiene la imagen en tonos de gris.

```
r = rojo( píxel )
g = verde( píxel )
b = azul( píxel )
gris = ( r + g + b ) / 3
rojo a presentar ( píxel ) = gris
verde a presentar ( píxel ) = gris
azul a presentar ( píxel ) = gris
```

Se puede generar una imagen invertida colocando en cada componente lo que falta para llegar al máximo.

```
r = rojo( píxel )
g = verde( píxel )
b = azul( píxel )
rojo a presentar ( píxel ) = 255 - r
verde a presentar ( píxel ) = 255 - g
azul a presentar ( píxel ) = 255 - b
```

Aquí vemos un ejemplo de lo descrito. Al generar suficiente movimiento en la zona verde, se genera un cambio en el proceso que se aplica a la imagen.

En la medida que el desarrollo lo permitió, se fue haciendo el sistema más configurable. Permitiendo manejar zonas, sensibilidad y acciones.

Eventos generados

Hasta ahora se ha trabajado fundamentalmente generando eventos de comunicación vía red con los prototipos sonoros implementados.

Se instancia un cliente y eventualmente se envían comandos para mover los motores de manera de tañer su correspondiente cuerda, así tenemos el ejemplo de asignar una zona a cada cuerda y brindar la capacidad de manejar o tocar un instrumento a través de la cámara.

Aquí vemos un ejemplo en donde cada zona blanca haría sonar un motor.

Otro prototipo implementado genera más sonido ante más movimiento, o los sonidos evolucionan a más agudos, más excitación ante más movimiento, y sonidos más graves, más descansados cuando hay poco movimiento.

No parece muy lejano pensar en una estrategia de análisis basada en el contraste entre píxeles vecinos (pudiéndose implementar detección de bordes o *edge detection*), lo que daría la posibilidad de reconocer cuerpos y figuras. Pero esa es una posibilidad más elaborada, a desarrollar en el futuro.

Teniendo un manejo más elaborado de la interfaz y pudiéndose extractar más datos de la imagen procesada podrían generarse eventos más elaborados y proveer un control más fino de los dispositivos.

Eventualmente, podría implementarse un sistema que –a partir de una interfaz gestual como la descrita- genere los mensajes MIDI, lo cual independiza la interfaz del instrumento que se controla.

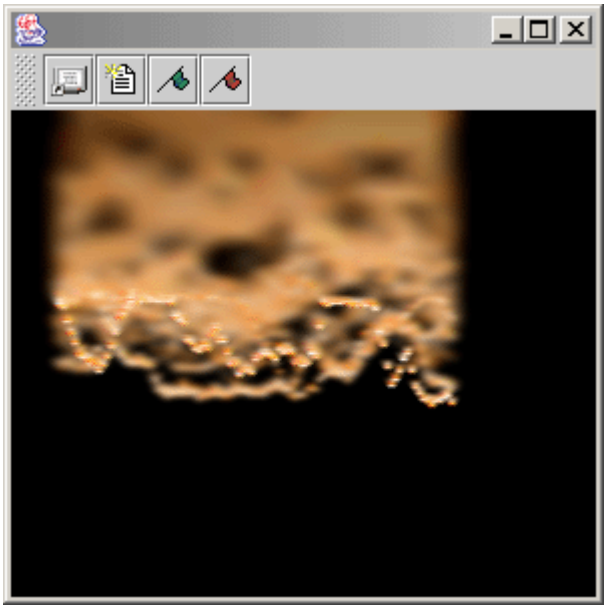
Graficación en tiempo real.

De igual forma, se bocetó la visualización de audio capturado, utilizando para la captura de audio, la clase SoundCapture, provista por la Universidad de California la que a su vez utiliza JMF [19].

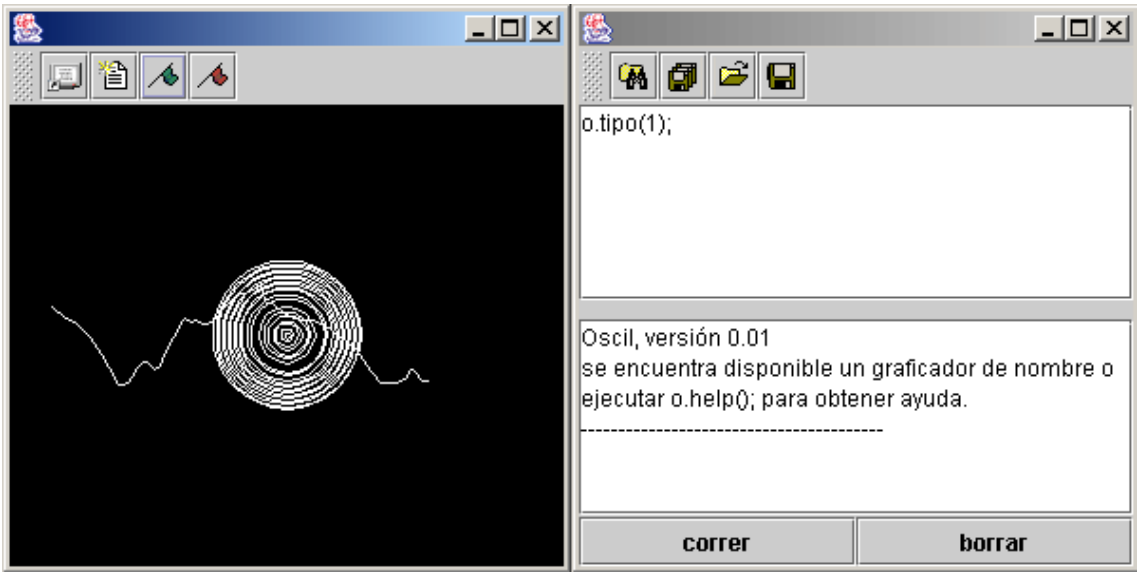
Luego, a partir de los datos capturados, es posible su graficación de distintas maneras, aplicándole efectos en tiempo real.

Las siguientes son capturas de pantalla que muestran algunas visualizaciones.

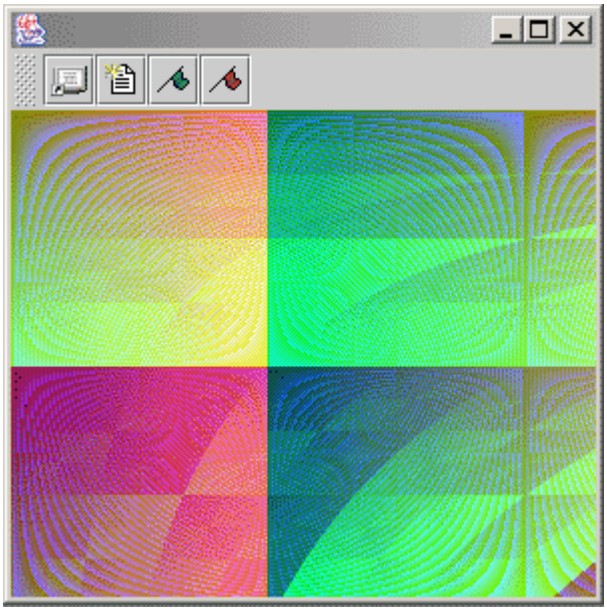
En esta imagen, vemos la onda graficada utilizando la visualización por defecto del componente, la cual aplica –en tiempo real- un efecto de “fuego”, difuminando el gráfico.



En la siguiente, vemos otro tipo de graficación, junto con la ventana de guión que seleccionó el tipo.



En esta última imagen vemos otro tipo, en el cual no se grafica la onda (es posible graficarla utilizando el comando o.setdw(true)), sino que se deforma un dibujo en función del sonido capturado.



MediaScripting y Visualización de vídeos grabados.

En función de la enorme potencialidad expresiva que conlleva, se implementó un componente que provee funcionalidades para reproducir un archivo de vídeo.

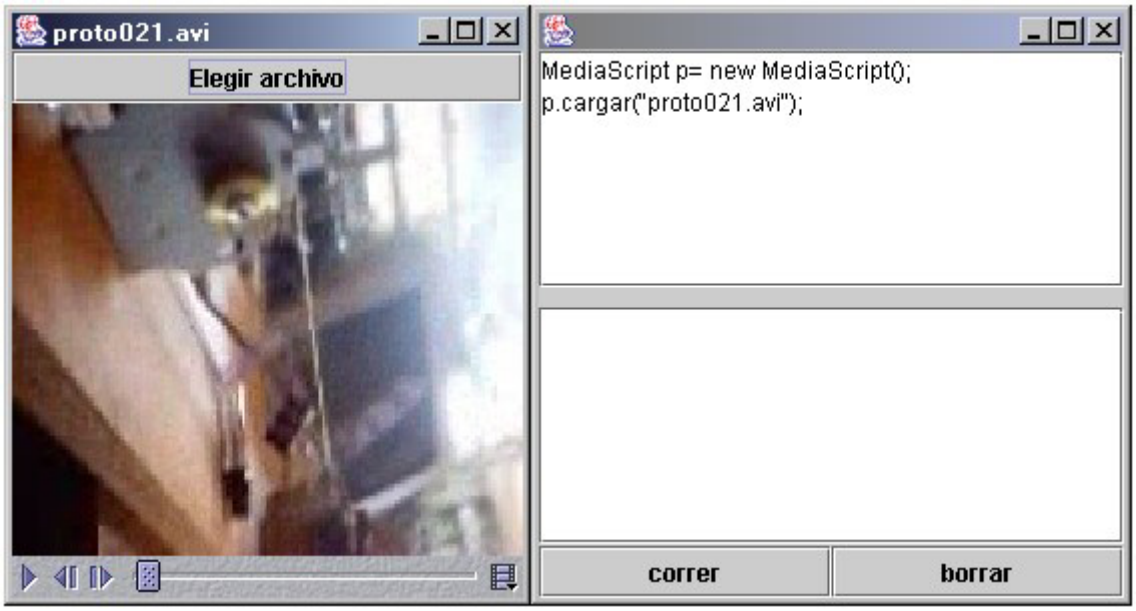
Al ser un componente de la arquitectura puede ser controlado a través de guiones, gracias a lo cual es posible alterar la reproducción de la secuencia, mostrar un *frame* determinado, iniciar o detener la reproducción, setear su velocidad o el volumen de reproducción e incluso saltar a un marco seleccionado en forma aleatoria.

De esta forma se logran explotar en otro contexto las posibilidades expresivas y educativas del guionado.

Más aún, podrían ejecutarse scripts que instancien otros componentes y sincronizar la reproducción de uno o más archivos de vídeo junto con, por ejemplo, un archivo MIDI interpretado en el Tecnocordio o la graficación de sonido capturado.

En la medida de que se implementen nuevos componentes dentro de la arquitectura, las posibilidades del sistema crecerán marcadamente. Si bien los resultados hasta ahora obtenidos son alentadores, las posibilidades de interacción y expresividad no han sido más que esbozadas.

Aquí vemos una imagen de la pantalla durante la reproducción de un vídeo. Previamente debió declararse un objeto de clase MediaScript de nombre "p", para disponer de las funcionalidades que la misma ofrece.



capítulo 6

Música

Si bien la intención inicial de generar un espacio expresivo dentro del marco científico tecnológico no indica *a priori* el camino a seguir, en función del interés personal de quienes realizamos el proyecto, se decidió focalizar los esfuerzos en las áreas relacionadas con la música.

De esta forma, gran parte de los esfuerzos se dedicaron estudiar formas de generar sonidos y modularlos para crear melodías.

¿Qué es la música?

La música soy yo
Ludwig Van Beethoven.

La siguiente recopilación de definiciones corresponde al profesor de la Universidad de Oviedo, Benjamín Dugnol Álvarez. Para más información sugerimos consultar su excelente sitio web "Matemáticas y Música" [24].

De acuerdo con Hilarión Eslava, sacerdote y músico español (1807-1878), música es el arte de bien combinar los sonidos y el tiempo. De acuerdo con la Enciclopedia Universal Espasa, es el arte de combinar los sonidos de la voz humana o de los instrumentos, o unos y otros a la vez, de suerte que produzca recreo al escucharlos, conmoviendo la sensibilidad, ya sea alegre, ya tristemente. Luis Villalba, compositor y crítico musical español (1873-1921) establece que música es la sucesión de una o varias series simultáneas de sonidos concertados, modulados y ritmados según el número, en orden a la expresión o emoción, así sentimental como estética.

Desde el punto de vista de los físicos, la música es una clase particular de sonido, el resultado de vibraciones transmitidas a través del aire, con frecuencias en el rango desde los 20 a los 20000 Hertzios. Para Tomás Vicente Tosca, arquitecto, matemático, filósofo y físico español (1651-1723), la música es una ciencia físico-matemática por participar su objeto la razón de los sentidos, propia del físico, y de la razón de la cantidad, propia del matemático.

En opinión del poeta alemán Frederick Schiller (1759-1805), es una forma de arte que corresponde a un logro exclusivo del *homo sapiens*. Sin embargo, la música es un medio de expresión patético con el que la Naturaleza dotó a muchas especies y su fuerza expresiva y emotiva tiene su raíz en la naturaleza psíquica y orgánica de muchos seres vivos. Así, el canto de los pájaros ¿no es música? Por otra parte, parece comprobado que la música ejerce sobre los organismos, en particular sobre el sistema nervioso, una influencia notable, hasta el punto de ser utilizada para favorecer la curación de determinadas dolencias.

Un concepto más actual, debido a los serialistas, sitúa la música como un arreglo ordenado de sonidos simples de distinta frecuencia en sucesión (melodía), sonidos en combinación (armonía) y sonidos (y silencios) en sucesión temporal (ritmo).

Por otra parte, el concepto debería integrar a las tradiciones y otros componentes culturales: sucesión de sonidos colocados ordenadamente, deliberadamente organizados de acuerdo con el patrón de otras sucesiones previas transmitidas por la tradición y aceptadas como una hipótesis. En este sentido, la Enciclopedia Británica se inclina por el arte relacionado con la combinación de sonidos vocales o instrumentales como una expresión de la belleza de la forma o de una emoción, usualmente de acuerdo con estándares culturales de ritmo, melodía, y, en la mayor parte de la música occidental, armonía.

Pero, por lo menos hasta el siglo XVII, la música era una de las disciplinas matemáticas que constituían el Quadrivium. Las otras tres eran la Aritmética, la Geometría y la Astronomía. En la hora presente, la música se analiza y se describe con Matemáticas.

Las Matemáticas ¿son una forma de música?

Síntesis de sonido

Sonido

El sonido puede definirse como la sensación producida en el órgano del oído por el movimiento vibratorio de los cuerpos, transmitido por un medio elástico, como el aire [13].

Esta vibración puede ser expresada como una onda que, gracias a la descomposición de Fourier, puede expresarse como la superposición de ondas simples.

Dada esta definición del sonido como la sensación producida, podemos caracterizarlo identificando:

- **Altura :** La percepción de tonos más agudos o más graves. Fundamentalmente está dada por la frecuencia.
- **Sonoridad**: La percepción de sonidos más fuertes o débiles. Está muy relacionada con la amplitud.
- **Duración**: El tiempo que se percibe el sonido. Depende directamente de la amplitud.

Estas relaciones (altura – frecuencia, sonoridad – amplitud y duración – amplitud) no son biunívocas[17]: la percepción se ve alterada por multiplicidad de factores, como ser grado de atención, otros sonidos simultáneos o anteriores, características del oyente, etc.

Para que una onda determine un tono puro, deberá ser sinusoidal, es decir al graficarla deberá obtenerse una curva cuya ordenada sea proporcional al seno de la abscisa correspondiente.

El tono o altura de un sonido depende de su frecuencia (medida en Hercios), en general, cuanto mayor sea la frecuencia, más agudo será el sonido. Cada nota musical, tiene un valor en Hercios, dado por su frecuencia fundamental, la predominante en una forma de onda compleja.

El rango de sonidos audible para las personas varía entre 20 Hz y 20 kHz, aunque para el común de las personas este rango es mucho más limitado. En sonidos mayores a 5 kHz ya es muy difícil distinguir altura.

[17] De hecho, gracias a las diferencias entre las características físicas y la percepción humana, fue posible el desarrollo de los modelos psicoacústicos utilizados para descartar información redundante en la compresión con pérdida de los archivos MP3, OGG y WMA (generalmente la redundancia obedece al efecto de enmascaramiento).

A diferencia del tono o altura, del nivel de sonoridad y de la duración de un sonido, el timbre es un atributo multidimensional y consiste en un conjunto de datos informativos que permiten la identificación por un individuo de la fuente de emisión de la señal sonora. En particular, en el contexto de la música, el timbre permite diferenciar entre los diferentes instrumentos musicales, incluso caracterizar al intérprete, pero también permite identificar a las personas por su voz o a un perro por su ladrido.

Subjetivamente, existe un abanico de adjetivos relacionados con el timbre de un sonido y que, en conjunto, contribuyen al diseño del atributo. Así, se pueden mencionar frío, caliente; duro, suave; brillante, apagado, etc.

Físicamente, el timbre está esencialmente determinado por el espectro de la señal y por su envolvente. La estructura del espectro incluye el número, magnitud y espaciado de las líneas espectrales; la fluctuación espectral; la presencia o ausencia de altas frecuencias; el ancho de banda de la señal; y la energía aportada a la misma por los armónicos en relación con la energía total.

Sintetizadores

Un sintetizador es un instrumento musical electrónico capaz de producir sonidos de casi cualquier frecuencia e intensidad, proporcionando así sonidos de cualquier instrumento conocido, o efectos sonoros que no corresponden a ningún instrumento convencional.

El sonido sintetizado es producido artificialmente, no se obtiene como tradicionalmente excitando elementos naturales, sino que es creado mediante distintas técnicas que han ido desarrollándose y evolucionando a lo largo del tiempo.

Originalmente los sintetizadores eran todos analógicos, se creaban sonidos que variaban en el tiempo utilizando formas de onda predefinidas. Se manipulaban la amplitud, la frecuencia y el contenido armónico de estas formas de onda para producir un gran número de resultados diferentes.

El sintetizador más antiguo que se conoce es el Telharmonium, creado por Thaddeus Cahill's por el año 1900. Luego, cerca de 1920, Leon Theremin crea el primero de los que se pueden llamar instrumentos musicales, el Theremin. Su eslogan es "El instrumento que se toca sin ser tocado", ya que el sonido se genera con unos osciladores cuya frecuencia varía al acercar o alejar una mano de una especie de antena. Una mano es usada para controlar el tono y la otra para controlar el volumen de sonido.

Leon Theremin

De a poco fueron creándose nuevos instrumentos sintetizadores que agregaron nuevas capacidades. El Martenot de 1928 era capaz de producir el glissando (deslizamiento continuo entre dos tonos) y el vibrato (ondulación del sonido producida por una vibración ligera del tono). En 1939 surge el Vocoder, diseñado para reproducir la palabra hablada.

En 1955, con los sintetizadores MARK I y MARK II aparece por primera vez la utilización del ruido blanco (sonido con todas las frecuencias audibles, que no transmite información y es parecido al de la radio sin sintonizar una estación determinada), utilizado para imitar los platillos o címbalos.

Mini Moog

Continúa el perfeccionamiento de los instrumentos generadores de sonido y promediando la década de 1960, Robert A. Moog, en colaboración con un músico llamado Herbert Deutsch desarrollan los filtros controlados de paso bajo y paso alto, los generadores de envolvente y los teclados con memoria, que representarían la base del famoso sintetizador Mini Moog. En 1968, Walter Carlos introduce el concepto de música electrónica de laboratorio, sorprendiendo con su álbum "*Switched on Bach"*, una prodigiosa muestra de virtuosismo electrónico interpretando música de J.S.Bach.

Una lista histórica muy completa e interesante de los instrumentos electrónicos puede encontrarse en el sitio web "*120 years of Electronic Music*" [22]

MIDI

La síntesis digital se logra utilizando números para crear sonidos y constituye el método más utilizado en los sintetizadores actuales. Algunas técnicas comúnmente usadas son la síntesis aditiva, en la que se construyen formas de onda complejas combinando ondas sinusoidales cuyas frecuencias y amplitudes varían de manera independiente; y la síntesis por modulación de frecuencia (FM) un método que implica la interacción de una señal (portadora) con otra (moduladora).

Al introducirse componentes digitales en los sintetizadores se generó un gran cambio. Esta tecnología demoró muy poco tiempo en afirmarse y extenderse, proveyendo un ambiente en el cual todas las funciones se llevan a cabo bajo forma de operaciones matemáticas realizadas por microprocesadores especialmente diseñados y programados a ese fin. Con el vertiginoso desarrollo de la tecnología informática -aumento de la velocidad de los microprocesadores y de la disponibilidad de memoria- las posibilidades de los instrumentos digitales se han ido ampliando sin cesar.

A principios de los años 80, la tecnología de los sintetizadores había conseguido avances importantes tanto en el afán de reproducir sonidos naturales como produciendo tonos irreales inventados por la imaginación de algunos músicos que encontraron en estos aparatos nuevas inspiraciones y medios para su creatividad. Sin embargo, continuaba sin solucionarse la incompatibilidad entre diferentes instrumentos, incluso de la misma marca. Por lo que las principales marcas de sintetizadores combinaron las tecnologías aplicándolas al lenguaje de computadoras y del sonido digital generando una propuesta de protocolo de comunicación universal, para poder crear sonidos u orquestaciones que pudieran oírse igual en distintos sintetizadores y teclados electrónicos.

En 1983 se dio a conocer el estándar finalmente diseñado, que fue denominado MIDI, por Musical Instrument Digital Interface. El protocolo fue rápidamente adoptado por todos los fabricantes, quienes deben respetar estrictamente todas las especificaciones, de modo de que está garantizada una total compatibilidad. Se puede decir que desde ese entonces todos los instrumentos electrónicos de aplicación musical, virtualmente sin excepción, incluyen una completa implementación MIDI. Esto hizo posible relacionar mediante una interfaz las computadoras con los instrumentos musicales, y acercó las PCs a los músicos, hasta entonces sumidos en aparatos con pocas capacidades, costosos y difíciles de manejar. Hoy en día un 70% de la música que escuchamos se encuentra hecha en parte con MIDI y las tecnologías digitales para la creación musical, y hace años que es prácticamente impensable prescindir del MIDI en ninguna aplicación musical que involucre instrumentos electrónicos.

Debemos diferenciar dos grandes partes de estos instrumentos: generador y controlador. El generador es la sección que produce el sonido propiamente, y que puede ser programada, asignándole valores determinados a sus diferentes parámetros, para variar sus características tímbricas. El conjunto de todos esos valores se denomina habitualmente *patch*, y es el que determina el tipo de sonido resultante. Un *patch* puede estar programado de manera de buscar reproducir o imitar el timbre de un instrumento acústico, o por el contrario producir un timbre totalmente sintético. El controlador por su parte es la sección sobre la que actúa el ejecutante en el momento de tocar, y que envía la información al generador, diciéndole por ejemplo que debe prender o apagar una determinada nota, aplicar cierta modulación, etcétera. Los más comunes son los controladores en forma de teclado del tipo piano u órgano, pero puede haber de otras clases, tanto con analogías a instrumentos tradicionales (guitarra, saxo, instrumentos de percusión) como de nuevos diseños. También integran el controlador otros dispositivos (como ser ruedas, pedales, etc.) que agregan expresión adicional en el momento de tocar, ya que, según cómo haya sido programado el generador, accionarlos producirá diferentes variaciones en el resultado sonoro.

En primera instancia entonces, el MIDI permitió tocar varios generadores a través de un solo controlador, obteniendo así sonidos con más cuerpo y timbres más complejos. Aparecieron también los módulos solamente generadores de sonido sin controlador propio, que podían ser tocados desde otro sintetizador, y controladores especializados que funcionaban solamente como tales, sin generador, y que ofrecían más posibilidades que los incluidos en los sintetizadores comunes. Pronto comenzaron a desarrollarse los secuenciadores, dispositivos que graban la información MIDI y que permiten además distintos tipos de edición, pudiendo luego reproducirse todas las veces que se desee, con las eventuales correcciones que fueran necesarias. Por medio de mensajes MIDI pueden alterarse también los parámetros de programación de un sintetizador o procesador.

Con el agregado de la interfaz correspondiente, una computadora puede comunicarse con cualquier instrumento MIDI, y surgieron así sofisticados programas de distintas aplicaciones, como ser secuenciadores, editores y almacenadores de patches para sintetizadores, generadores automáticos de acompañamientos, compositores algorítmicos, etcétera. También por medio del MIDI pueden sincronizarse secuenciadores entre sí, con grabadores de audio y con video, pueden controlarse mezcladoras, mesas de luces, etcétera.

Más adelante explicaremos de qué forma se implementó el manejo de los motores de pasos utilizando el MIDI.

Especificación

En la especificación MIDI están definidos dos tipos de byte, de estado y de datos, que están identificados por su primer bit o MSB, según sea 1 ó 0 respectivamente.

Tipos de byte:

Status byte (byte de estado): es el que determina de qué tipo de mensaje se trata, por ejemplo prender o apagar una nota en un determinado canal, iniciar o detener el secuenciador, seleccionar un *patch*, o enviar un pulso de reloj para sincronización, etcétera. Todos los bytes de estado comienzan con bit 1 (de 1000 0000 a 1111 1111): son los valores 128 a 255 (de 80 a FF en hexadecimal).

Data byte (byte de datos): según cuál sea el byte de estado, puede ser seguido de uno o más bytes de datos, con los valores correspondientes al tipo de mensaje. Por ejemplo si el byte de estado es de prender una nota en cierto canal, deben seguir dos bytes de datos indicando respectivamente qué nota hay que prender y con qué velocidad. Si el byte de estado es seleccionar un *patch* en cierto canal, le sigue un byte de datos con el número de *patch*. Los bytes de datos son los que comienzan con bit 0 (de 0000 0000 a 0111 1111), de 0 a 127 en decimal, de 00 a 7F en hexadecimal. Es por esto que varios parámetros (como las velocidades, las modulaciones, volumen, etcétera) tienen 128 valores posibles. También es la cantidad total de notas que hay en MIDI, diez octavas y 8 semitonos, de C-1 a G9, siendo A4 el la central de 440 Hz. Otros parámetros, como el *pitch bending* y el posicionador dentro de una canción, precisan mayor definición; en ese caso son necesarios dos bytes consecutivos de datos, lo que da una definición de 14 bits (16.384 valores).

Running status: cuando hay una sucesión de mensajes del mismo tipo, no es necesario repetir todas las veces el status byte; solamente se envía el primero de ellos, y luego los bytes de datos respectivos. Por ejemplo si se trata de prender una serie de notas en un canal, se envía el status byte correspondiente, y luego los dos bytes de datos para cada una de las notas.

Canales MIDI:

La línea MIDI transmite información simultáneamente en 16 canales. Si se están controlando varios instrumentos a la vez, cada uno de ellos puede ser programado para responder a un canal específico, de modo de que queda conformada una "orquesta" en la que se pueden tocar notas independientemente en cada instrumento. La mayoría de los instrumentos musicales digitales actualmente tienen capacidad multitímbrica, es decir que funcionan como varios instrumentos virtuales en uno: reciben en más de un canal simultáneamente (usualmente 8 o los 16), pudiendo asignar un *patch* o timbre diferente a cada canal. Algunos mensajes se dirigen a un canal específico; en ellos el primer *nibble* del status byte indica de qué comando se trata, y el segundo a qué canal se aplica (recordar que un *nibble* puede representar precisamente hasta 16 valores).

Tipos de mensaje:

Hay dos tipos de mensaje: de canal (*channel message*) y de sistema (*system message*).

channel messages: afectan solamente a uno de los 16 canales MIDI, y sólo responden a este tipo de mensajes los instrumentos sintonizados para recibir en ese canal. Los mensajes de canal se subdividen a su vez en: **channel voice** tienen que ver con la

producción de sonido (como por ejemplo prender y apagar notas) y **channel mode** determinan la forma en que el dispositivo responde a los mensajes recibidos .

system messages: éstos son recepcionados por todos los instrumentos conectados a la red. Hay tres tipos de mensajes de sistema: **system exclusive** (sistema exclusivo) estos mensajes comienzan con un encabezamiento que identifica la marca y modelo determinado del instrumento al cual están dirigidos, siendo ignorados por todos los demás dispositivos; **system common** (sistema común) afectan a todos los instrumentos conectados al sistema; y **system real time** (sistema tiempo real) están relacionados con el funcionamiento de secuenciadores, su *timing* y sincronización.

Para una especificación comentada del protocolo MIDI se sugerimos consultar la referencia publicada por Luis Jure en el sitio web de la Escuela Universitaria de Música [23]

Hiperinstrumentos [3].

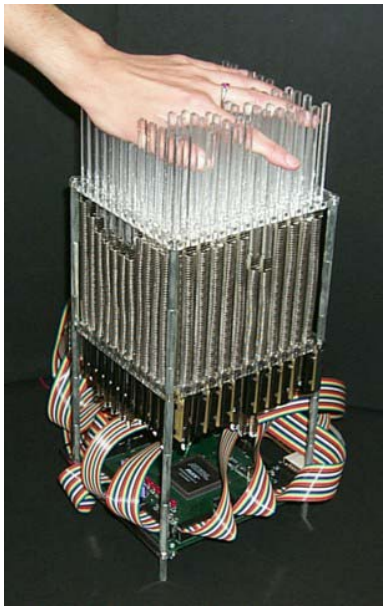
En 1986, el concepto de los hiperinstrumentos fue introducido en el *Media Lab* del MIT por Tod Machover, con el fin de diseñar instrumentos musicales expandidos usando tecnología para brindarles poder y delicadeza extra a los intérpretes. Tales hiperinstrumentos fueron diseñados para extender guitarras y teclados, percusión y cuerdas. Han sido usados en vivo por algunos músicos famosos tales como Yo-Yo Ma, la filarmónica de Los Ángeles, o Peter Gabriel.

Desde 1992, el foco del *Hyperinstruments Group* se ha ampliado intentando construir sofisticados e interactivos instrumentos musicales para músicos no profesionales, estudiantes, y amantes de la música en general. Parte de este trabajo se enfoca en el principio de comprometer a los participantes en una relación activa y cinética con el proceso musical – sea mediante composición, interpretación, improvisación o alguna nueva experiencia que puede únicamente existir con la ayuda de una computadora.

Se pretende diseñar sistemas computacionales (sensores, proceso de señales, software) capaces de medir e interpretar la expresión y sentimientos humanos. Aunque últimamente han habido grandes avances tecnológicos en música para ayuda a la producción, proceso y composición, estas herramientas han limitado la habilidad de controlarlas en formas musicalmente interesantes. Las interfaces musicales, tanto para expertos como novatos, deben poseer un control físico refinado y una relación intuitiva con los sonidos que producen. Una tecla que produce un gong electrónico gigante no se tocará tan bien como un instrumento que requiere un movimiento del brazo y un golpe para producir tal sonido. Más aun, mientras más abstractos los distintos elementos elegidos para ofrecer a los usuarios control en tiempo real (tiempo, ritmo, color), más desafiante se torna el desarrollo de la interfaz.

Así se han construido juguetes musicales (Music Toys), que proveen a los niños una forma divertida de interactuar con la música, interpretar, e incluso componer ciertas piezas. Se han diseñado distintos tipos de *music shapers,* que permiten al usuario darle forma a un fragmento musical original provisto por el sistema.

La siguiente fotografía corresponde a la *Matrix*, interfaz de hardware del proyecto *Emonator,* el cual genera audio mapeando los gestos expresivos del intérprete a diferentes parámetros musicales. Su superficie esculpible puede controlar una variedad de parámetros sonoros, desde armónicos audibles de un motor de síntesis aditiva hasta el nivel de actividad de música generada algorítmicamente.



Por supuesto que sobran los escépticos y los elitistas que insisten en que esto no es música y que ni siquiera se le puede parecer. La cuestión es una antigua puja entre la creatividad y la ejecución. Si vale la ayuda masiva a poblaciones que hoy son impotentes para generar sonido alguno o si más vale dejar la ejecución y la creación sólo en manos de los expertos. En el caso de los niños, por ejemplo, lo interesante no radica en la técnica sino en la espontaneidad y la creatividad. Con estos instrumentos, se puede componer desde un principio en lugar de tener que pasar años preparándose para la posibilidad de expresarse creativamente.

Prototipos sonoros

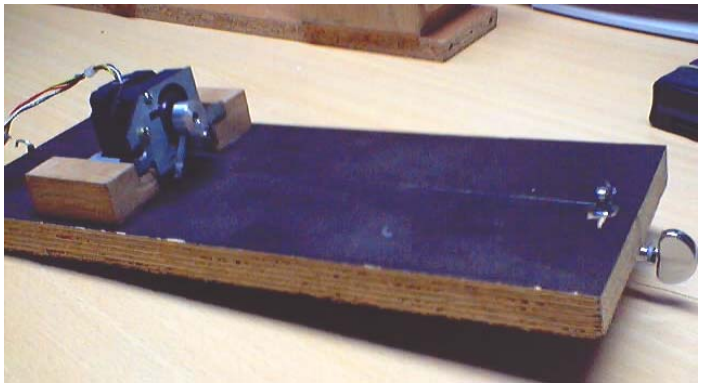
Describiremos ahora nuestro trabajo en relación a la música, utilizando la chatarra y otros elementos de desecho a nuestro alcance.

Estudiando la manera de reproducir una melodía surgió la idea (luego de idear muchos sistemas irrealizables, en general por no disponer de los materiales necesarios) de utilizar los motores paso a paso de las disqueteras de 5 y ¼ para tañer una cuerda de guitarra. Así se podría producir un sonido puramente sinusoidal con elementos y materiales de los que podíamos disponer.

En esa dirección se enfocaron los prototipos. Las alternativas que surgieron inmediatamente fueron tañer con un motor una cuerda ya afinada, la que produciría una nota predefinida, por lo que necesitaríamos tantas cuerdas y tantos motores como tonos quisiéramos que sonaran; también pensamos que tal vez podríamos lograr ejecutar más de un tono por cuerda combinando el movimiento de dos motores. Las posibilidades consistían en utilizar un segundo motor para modificar la tensión de la cuerda (al aumentar la tensión aumenta la altura del tono producido al tañer la cuerda) o modificar el tono implementado un sistema análogo al de las guitarras o violines, donde se obtienen nuevos tonos disminuyendo el largo de la cuerda que se tañe.

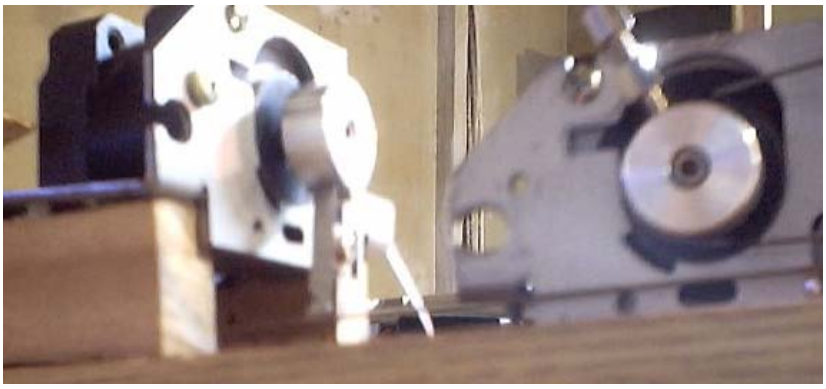
Inicialmente alimentábamos la ilusión de que el modelo con dos motores resultara muy útil, dado que en teoría permitiría múltiples tonos por cuerda. Sin embargo, las limitaciones de los motores impedirían que este enfoque prosperase.

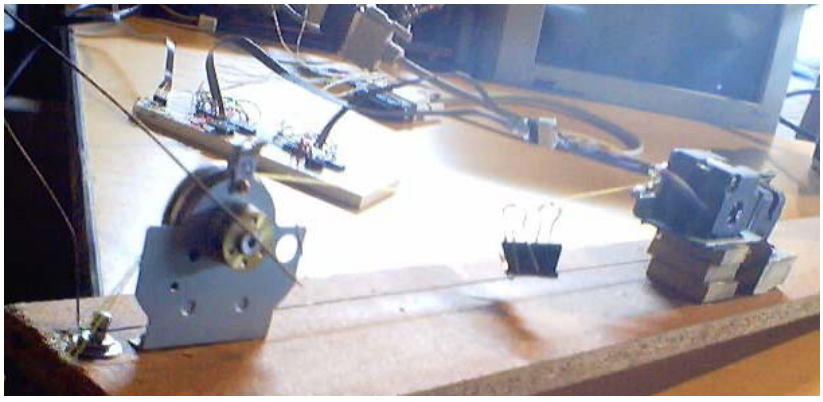
El sistema básico (que constituyó el primer prototipo construido) consiste en una cuerda de guitarra dispuesta sobre una tabla y un motor de pasos que actúa sobre ella para producir el sonido. Es posible afinar la cuerda a través de una clavija de guitarra colocada en la tabla prototipo a tales efectos. Al motor asociamos un adminículo que hace las veces de púa y tañe la cuerda. El movimiento habitual de este motor es de unos pocos pasos hacia un lado y hacia el otro cada vez que se quiere hacer sonar la cuerda.



Luego de construir este primer acercamiento, se atacó el problema de generar más de un tono por cuerda, retomando los modelos de dos motores manejados al inicio. Lamentablemente comprobamos que los motores carecen de la fuerza (torque) necesaria para la construcción del sistema basado en alterar la tensión de la cuerda, por lo que el siguiente prototipo construido se basó en el diseño de funcionamiento análogo a las guitarras.

De esta forma, buscando variar el tono en la cuerda agregamos al primer prototipo un sistema de rondanas que, utilizando el segundo motor de pasos, moviera transversalmente un tubito de vidrio que marcara la cuerda en distintas longitudes y así controlar el tono del sonido al ser tañida la cuerda.





Trabajamos un tiempo importante en esta dirección pero finalmente se decidió que no era un acercamiento viable, esta vez no sólo por las limitaciones de los motores sino por problemas en la construcción física.

Los principales problemas de este modelo fueron:

- El sonido logrado rara vez mostraba la fidelidad esperada, obteniendo un sonido sucio. Esto se debía a la falta de peso del tubo utilizado para marcar la longitud de la cuerda, peso que estaba limitado por la fuerza del motor (si incrementábamos el peso, los motores dejaban de responder en forma mínimamente razonable, siendo imposible encontrar un termino medio satisfactorio).

- Falta de fuerza (torque) de los motores, lo cual resultaba en que no obteníamos el mismo comportamiento al enviar dos veces el mismo comando. El motor, al levantar el peso utilizado para mantener la tensión, perdía pasos en forma aleatoria, por lo que el desplazamiento del tubo variaba en forma perceptible.

Las dificultades de la construcción mecánica también fueron relevantes, si consideramos que se esperaba que el sistema creciera en tamaño (cantidad de cuerdas/tonos), este esquema se volvería muy difícil de construir y manejar. Más aún si tenemos en cuenta que las variaciones en el largo de la cuerda deberían ser extremadamente precisas, dada la percepción logarítmica de variación de tono.

Sin embargo no descartamos continuar el desarrollo de cualquiera de los prototipos presentados y no se debe desestimar la utilidad de estos enfoques en otros proyectos.

Integración de MIDI.

Dada la estandarización de MIDI y su sencilla utilización en los PCs, resultó evidente desde un principio, que los prototipos construidos deberían ser controlados utilizando este protocolo para comunicar las interfaces de usuario con las construcciones físicas.

Todos los prototipos construidos, son controlados enviando órdenes (utilizando distintas versiones del protocolo especificado en la sección Red) a un servidor que controla los motores de acuerdo a las necesidades del prototipo. Estos comandos pueden enviarse directamente, pueden obtenerse a partir de mensajes MIDI o pueden enviarse utilizando alguna de las interfaces gestuales construidas.

La importancia de MIDI radica, no sólo en que se dispone de un medio de enviar órdenes todo lo complejas que se quiera (dentro del protocolo se implementan mecanismos de control extremadamente fino), lo cual resuelve el problema de controlar un eventual sistema que involucre muchos motores y cuerdas, sino también en que resuelve el problema de interoperar con software y hardware preexistente. Es posible utilizar teclados o guitarras MIDI[18], secuenciadores (programas donde se escribe música que se traduce en órdenes MIDI) y archivos donde se encuentren almacenadas las melodías a interpretar.

Dado que el único prototipo capaz de ser utilizado para interpretar melodías de más de un tono es el Tecnocordio, solamente para el mismo se implementó el sistema de control vía MIDI.

[18] Para utilizar dispositivos MIDI en las PCs, se utiliza el puerto que habitualmente –en su tarjeta de sonido- disponen (también conocido como Joystick port)

MIDI y JMF.

La implementación de MIDI en JMF se basa en una arquitectura productor / consumidor, en la cual existen interfaces que permiten conectar dos clases, una que produce un flujo (*stream*) de datos y otra que lo consume.

La clase productora ralentiza el flujo de datos, operando como un secuenciador que marca los tiempos en los cuales se realiza el consumo de los datos.

Un ejemplo sencillo de funcionamiento bajo esta arquitectura consiste en un productor que lee un archivo MIDI, y lo envía a un consumidor que reproduce los datos utilizando un sintetizador.

Dado un error en la implementación del secuenciador incluido en JMF en todas las versiones hasta la fecha, se optó por utilizar el secuenciador de Tritonus [21], una implementación GPL del JMF para Linux.

A continuación incluimos una porción de código Java que reproduce –en el secuenciador por defecto del sistema- un archivo MIDI utilizando el secuenciador Tritonus (que se supone instalado).

```
File midiFile = new File(archivoMidi);
InputStream sequenceStream = new FileInputStream(midiFile);
sequenceStream = new BufferedInputStream(sequenceStream, 1024);

// instancio el tritonus pure java sequencer
String strSequencerName = "Tritonus Java sequencer";
MidiDevice.Info seqInfo = getMidiDeviceInfo(strSequencerName, true);
if (seqInfo == null) {
   throw (new Exception("No se encontró el secuenciador " + strSequencerName));
}
sequencer = (Sequencer)MidiSystem.getMidiDevice(seqInfo);

// abro el archivo
sequencer.open();
sequencer.setSequence(sequenceStream);

// instancio el default sinte -- eventualmente se puede usar otro sinte
Synthesizer synth = MidiSystem.getSynthesizer();
synth.open();

// conecto el seq con el default sinte
Receiver synthReceiver = synth.getReceiver();
Transmitter seqTransmitter = sequencer.getTransmitter();
seqTransmitter.setReceiver(synthReceiver);
```

capítulo

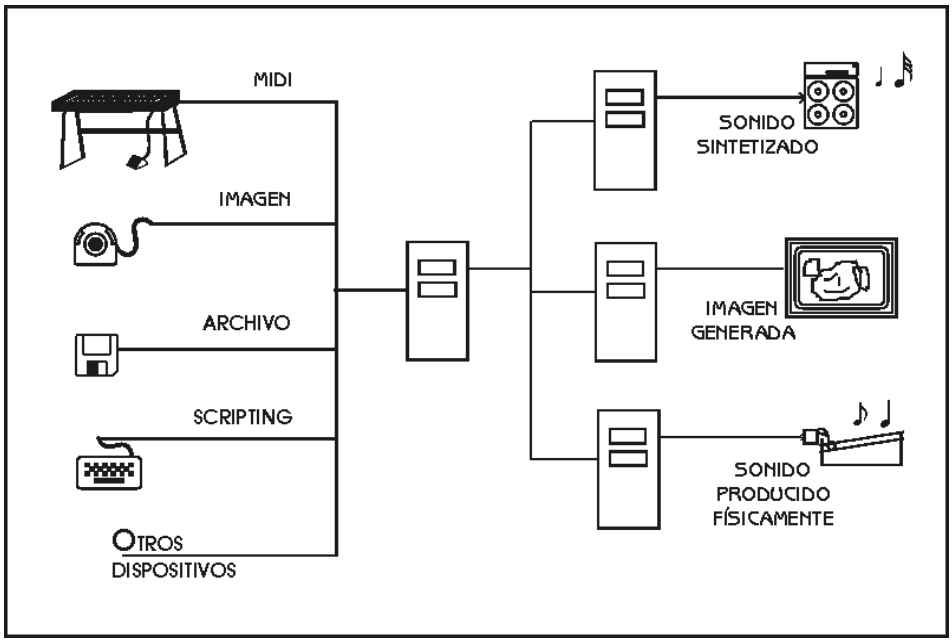
7

Arquitectura.

Introducción.

Al pretender construir herramientas que permitieran la construcción de instalaciones multimediales, resultó deseable poder interpretar distintos estímulos para controlar dispositivos y producir sonido e imágenes.

La figura muestra nuestro primer acercamiento, en el cual se explicitan algunas entradas y algunas posibles salidas del sistema.



El sistema que utilizase esta arquitectura, podría ser controlado mediante comandos MIDI, imágenes capturadas e interpretadas en tiempo real, archivos de comandos o guiones, para producir sonido, imágenes y controlar dispositivos (por ejemplo, motores).

El diseño de tal sistema estuvo pautado, desde su concepción, por requerimientos específicos surgidos tanto de la necesidad de hacer un uso expresivo del sistema, como de nuestra intención de utilizar tecnología de bajo porte e investigar aspectos específicos que resultan de interés para el NAT.

Dentro de estos aspectos destacamos la utilización de redes TCP/IP, computadoras y componentes electrónicos obsoletos, la utilización de software libre, la implementación mediante el lenguaje de programación Java y la implementación en el sistema de algún lenguaje de guionado.

A continuación se describirán someramente los aspectos de diseño más relevantes. Para una referencia más detallada sugerimos utilizar el CD que acompaña esta documentación, donde se dispone de un diagrama UML completo, de la documentación JavaDoc y de los códigos fuente Java.

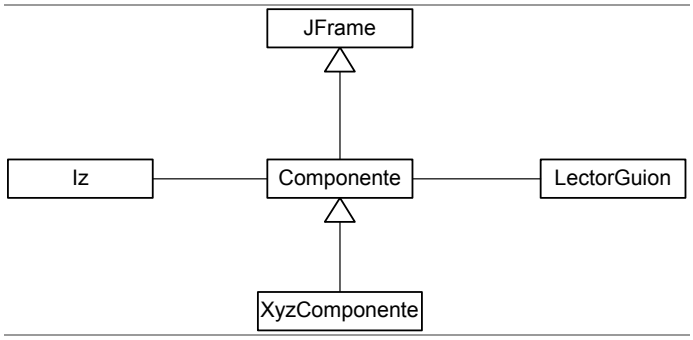
Diseño.

La arquitectura desarrollada ofrece un diseño modular, en el cual las distintas funcionalidades, se agrupan e interactúan mediante la utilización de *componentes de interfaz*, es decir, clases que implementan la funcionalidad necesaria para la integración.

Estos componentes de interfaz, se derivan de una clase plantilla la cual implementa la comunicación con una interfaz de usuario (GUI) sencilla, permitiendo además la utilización de guiones para controlar el módulo implementado. Los guiones se manejan a través de Beanshell y los componentes de interfaz se encargan de introducir la funcionalidad dentro del espacio de nombres.

De esta manera, cuando un desarrollador desea agregar una funcionalidad específica al sistema, deberá crear las clases que la realizan y luego derivar un nuevo componente de interfaz para así integrar su desarrollo con el sistema.

El siguiente diagrama[19], muestra este diseño básico:



Aquí vemos las siguientes clases:

- **Iz:** Implementa la interfaz gráfica del sistema, por lo que tiene (ve) una colección de componentes de interfaz (se deriva de JFrame, clase provista por el paquete Swing de Java).
- **Componente:** Es la clase plantilla de los componente de interfaz, tiene asociado un LectorGuión y es derivada para la implementación de un componente específico.
- **LectorGuion:** Es la clase que ofrece la funcionalidad de guionado, mediante la utilización de Beanshell. Cada componente tiene asociado un LectorGuion.
- **XyzComponente:** Esta clase ejemplifica la adición de un nuevo módulo al sistema. Derivando un componte, el desarrollador vincula su módulo con el sistema de interfaz gráfica y guionado. El XyzComponente oficia, entonces, como interfaz entre el sistema y el módulo Xyz.

[19] La arquitectura del software diseñado se documentará utilizando diagramas UML (Unified Modeling Language). Para más información ver [20]

Derivación de un nuevo componente.

Para implementar un nuevo componente, deberán seguirse los siguientes criterios.

- Heredar de la clase Componente.
- El constructor tiene que recibir un parámetro del tipo Iz (donde se implementa la interfaz) y debe llamar a **super(padre)** (donde padre es el parámetro de tipo Iz).
- Se deberá implementar los métodos getObject y getObjectName. El primero devuelve el objeto que será visible desde el guión, mientras que el segundo devuelve el nombre que se le dará dentro del guión.
- En caso de necesitarse, dentro del guión, más de un objeto, se puede sobrescribir el método setObject para que realice las operaciones necesarias (se sugiere utilizar el setObject implementado en Componente como guía).
- Se puede definir el mensaje de bienvenida al guión (el mensaje que recibe el usuario cuando abre la ventana de guionado) sobrescribiendo el método mensajeBienvenidaGuion (que devuelve un *string*).

Módulos implementados.

Dentro de esta arquitectura, se han implementado distintas funcionalidades, los cuales se integran como se ha descrito. Estos módulos han formado parte de los distintos prototipos implementados.

En lo que resta de esta sección mostraremos algunos de los módulos implementados, los cuales –en su mayoría- son utilizados por el Tecnocordio.

Interfaz (Iz).

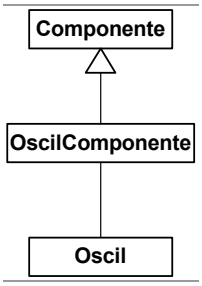
Esta pequeña clase, simplemente agrupa (como se mostró en el diagrama anterior) los distintos componentes de interfaz.

Luego, brinda una rudimentaria interfaz gráfica con un botón para cada módulo

OscilComponente.

El componente de interfaz OscilComponente, integra el módulo de graficación de sonidos capturados vía JMF, descrito en el capítulo de imagen.

Su diseño es el más sencillo de los módulos integrados:

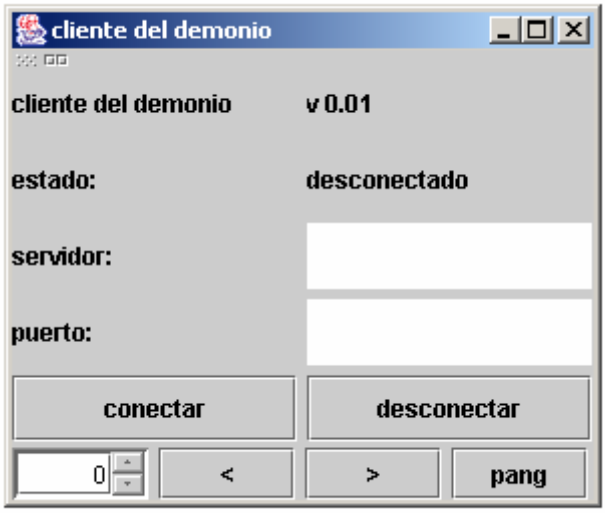


En el diagrama podemos ver el esquema básico de integración de un módulo a través de la derivación de un componente de interfaz. Dado que todas las funcionalidades se implementan en la clase Oscil (amén de la utilización de JMF, que no se especifica en el diagrama), el diagrama resultante sólo involucra estas clases.

ClienteComponente.

Este componente integra al sistema la funcionalidad de cliente de los servidores de motores paso a paso.

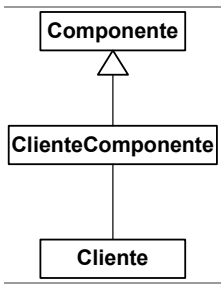
El componente de interfaz, presenta la siguiente pantalla



mediante la cual deberá especificarse la dirección IP a la que deberá conectarse (recordar que una conexión TCP se identifica mediante el par (dirección de máquina, puerto)).

Luego de conectarse se podrán enviar dos comandos básicos al servidor, utilizando los botones de la parte de más abajo: demWalk (con los botones etiquetados **<** y **>**) para hacer girar el motor seleccionado (se selecciona un motor con el cuadro de texto inferior izquierdo) o demPang para ordenar un tañido.

El diagrama de clases, para este módulo es el siguiente:



La comunicación TCP/IP se realiza utilizando la clase provista por Java (paquete java.net) Socket, donde se implementa toda la funcionalidad especificada en los Berkeley Sockets.

TocaMidiComponente.

Este módulo, es el encargado de tocar un archivo MIDI en el Tecnocordio, por lo cual se basa en la estrategia productor consumidor implementada en JMF.

Como fue explicado en el capítulo 8, JMF estructura la reproducción de archivos MIDI utilizando dos tipologías de clases, un secuenciador que estructura la lectura del archivo y oficia de productor para uno o más consumidores que se encargan de la reproducción real del archivo.

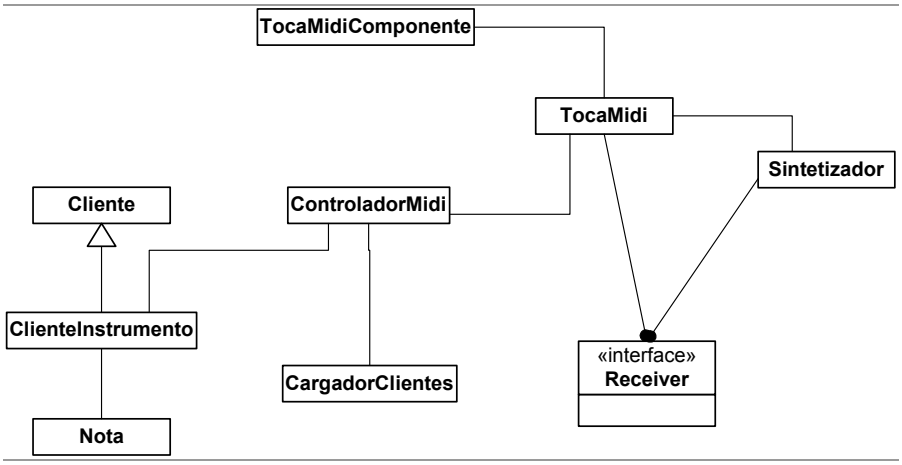
El módulo TocaMidi, sin embargo, utiliza el secuenciador provisto por Tritonus [21], dado que el implementado en JMF contiene errores que no permiten la redirección del flujo hacia otro consumidor (si bien implementa los métodos para ello, los mismos no influyen en su comportamiento), además de carecer de una secuenciación correcta.

TocaMidi, entonces, utilizando el secuenciador de Tritonus, dirige el flujo de datos hacia dos consumidores: el sintetizador por defecto y uno implementado por nosotros que se encarga de decodificar los datos y enviar las órdenes correspondientes al Tecnocordio.

El componente de interfaz presentado es el siguiente

En la cual vemos que se deberá seleccionar el archivo a reproducir, el canal que se utilizará para extraer los comandos a enviar al tecnocordio y el mapa de instrumento a utilizar.

La siguiente figura esquematiza la estructura del módulo.



Como se puede ver, se dispone de un componente de interfaz, de nombre TocaMidiComponente, el cual vincula el módulo TocaMidi con el sistema.

Este módulo oficia de contenedor de los consumidores y el productor (en el diagrama sólo se muestran los consumidores, mostrándose el sintetizador del sistema bajo la clase Sintetizador).

TocaMidi, entonces dirige el flujo de datos hacia ControladorMidi, quien dispone de un mapeo entre los distintos tonos y la terna (servidor, puerto, número de motor). De esta forma, al recibir un mensaje MIDI de ejecución de un sonido (NoteOn), puede utilizarlo para enviar la orden correspondiente vía la red.

El sistema de servidores y motores paso a paso se modela con una lista de servidores que contienen listas de motores, obteniéndose así la terna antes mencionada.

Mapeo.

Para saber cuál motor deberá accionarse para tocar determinado tono, es necesario disponer de un mapeo de la topología del instrumento, asociando tonos con motores.

Este mapeo se especifica mediante un archivo de texto, formado por líneas que obedezcan el siguiente formato:

DirecciónIP, Puerto, Nota, Número, Nota, Número, Nota, Número...

Es decir que la línea
10.100.3.3, 3490, A4, 1, B4, 2, C4, 3, D4, 4, A5, 1, A6, 1

Especifica que en el servidor de dirección IP 10.100.3.3, puerto 3490, se disponen de los tonos A4, B4, C4, D4, A5 y A6 (notación norteamericana, donde A4 indica La de la 4º octava, es decir 440Hz) y que están asociados a los motores 1, 2, 3, 4, 1 y 1 (vemos que es posible asociar muchos tonos a una cuerda, de igual forma pueden asociarse muchas cuerdas a un tono).

El cargado del archivo que contiene el mapeo, se efectúa mediante la clase CargadorClientes.

TocaMidiRTComponente.

Este componente es similar al anterior (TocaMidiComponente), con la diferencia que en lugar de utilizar un secuenciador para obtener el flujo de datos MIDI, utiliza un dispositivo MIDI IN, esto es, un dispositivo que pueda enviar comandos MIDI en tiempo real.

El funcionamiento es idéntico, vinculando la misma clase TocaMidi con el dispositivo MIDI IN en lugar de con el secuenciador.

El componente de interfaz presentado es

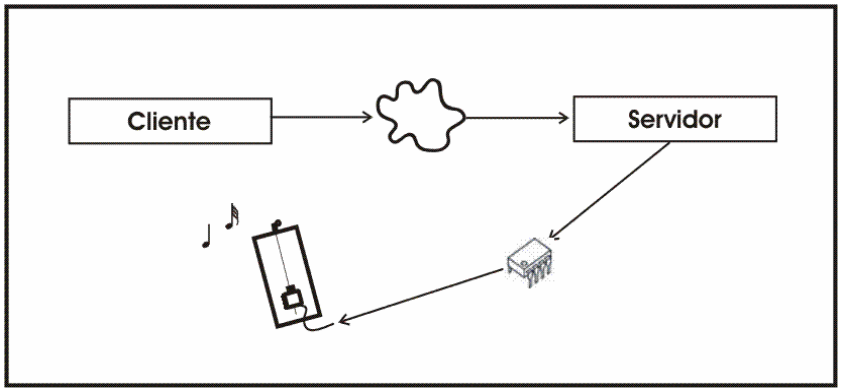
donde vemos que, además, es necesario seleccionar el dispositivo MIDI IN a utilizar.

capítulo 8

El Tecnocordio.

El Tecnocordio es el último prototipo implementado, el cual involucra todos los conceptos que hemos documentado. Instancia la arquitectura planteada en el capítulo anterior en un aparato concebido como instrumento musical. El mismo, como tal, al momento es bastante limitado, pero vemos fácil su extensión para darle más capacidad expresiva.

Consiste en un conjunto de cuerdas de guitarra que son tañidas por motores paso a paso, los cuales son controlados (en grupos de cuatro) por servidores dedicados (implementados usando PCs de muy bajo porte), los cuales son controlados por un cliente desarrollado en Java.



Esta arquitectura es abierta, en el sentido de que cualquiera de sus módulos puede ser reemplazado. Es posible utilizar las interfaces desarrolladas para controlar otros instrumentos, así como es posible también comunicarse con los servidores utilizando otros mecanismos.

Además es expansible. La cantidad de servidores y clientes es –en principio- ilimitada, por lo que puede aumentarse la cantidad de tonos disponibles agregando servidores y motores.

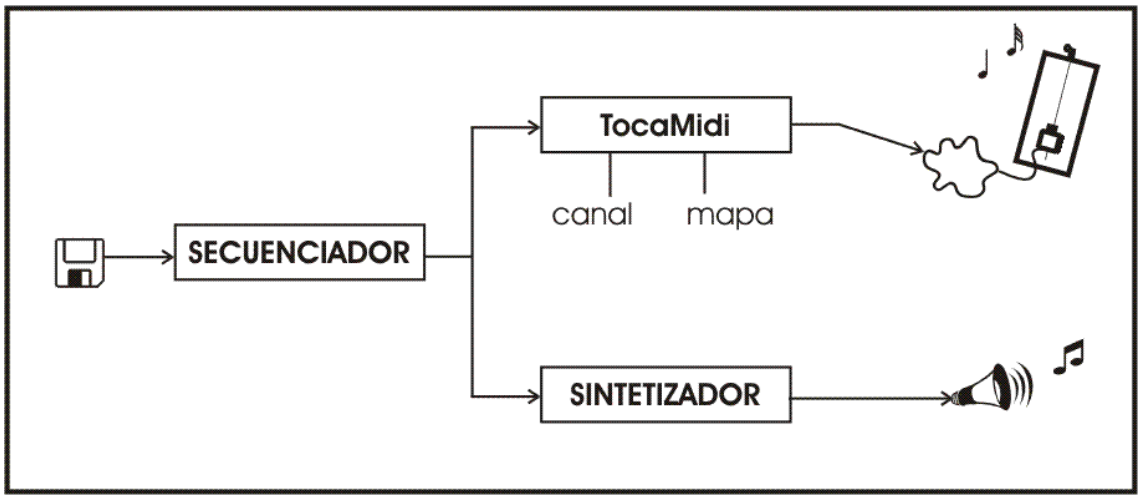
El control del Tecnocordio se realiza vía MIDI, pudiéndose controlar mediante un archivo MIDI que contiene la secuencia a ejecutar o en tiempo real con cualquier dispositivo MIDI que se conecte a la máquina cliente.

Control mediante un archivo

En la reproducción de una secuencia MIDI almacenada en un archivo se pueden destacar tres actores importantes, la secuencia *per se* (esto es las instrucciones que gatillan las notas y efectos como glissando o vibrato, es decir, los propios datos MIDI), el secuenciador[20] utilizado para controlar el *tempo* y el sintetizador usado para ejecutar la melodía (software y hardware para generar los sonidos requeridos).

En el control del Tecnocordio se sustituye el sintetizador por un controlador que traduce los comandos *Note On* MIDI, en órdenes de tañido e ignora los demás. La selección del motor a mover (o sea de la cuerda a tañer) se realiza en función de un archivo de mapeo como el especificado en la sección de Arquitectura.

Su funcionamiento se puede esquematizar de la siguiente forma:



Como podemos ver, al utilizarse el secuenciador Tritonus, es posible enviar los comandos a un sintetizador además de al Tecnocordio, por lo cual es posible ejecutar melodías en el Tecnocordio y en un sintetizador en forma simultánea.

Es preciso notar que no todos los comandos *Note On* son enviados al Tecnocordio, sino que se envían sólo aquellos que se encuentren en un canal especificado previamente. De esta forma es posible aislar dentro de la secuencia MIDI los tonos que se ejecutarán en el Tecnocordio y separarlos del resto de los instrumentos.

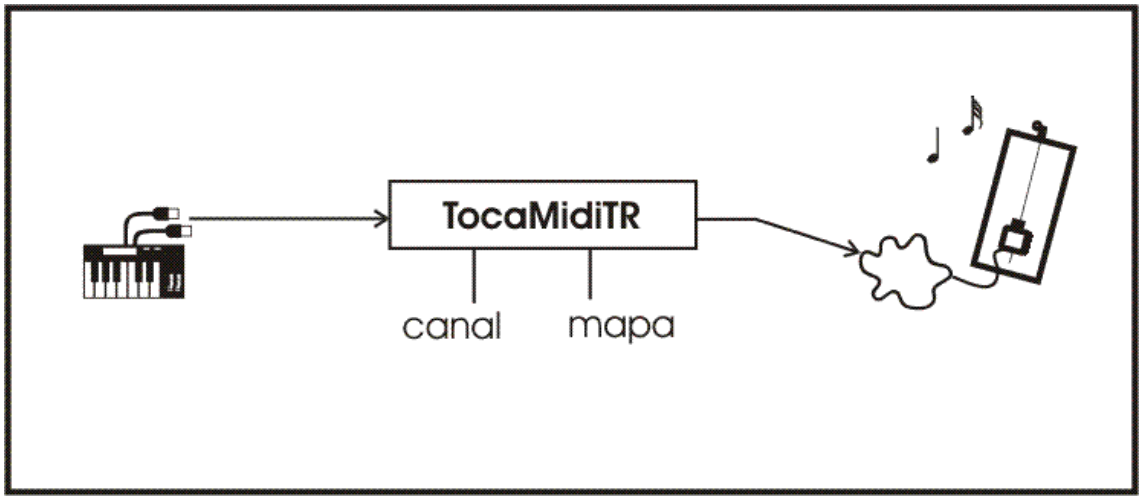
Otro enfoque posible que manejamos, consiste en la sustitución de un instrumento por el Tecnocordio, lo cual nos independiza del canal (pudiéndose así enviar más de una orden en forma simultánea) pero perdiéndose la posibilidad de interpretar ese instrumento en el sintetizador. Si bien no implementamos este modelo, sólo se precisa modificar la selección de los comandos que se pasan efectivamente al dispositivo físico, por lo cual implementarlo a partir del desarrollo actual no requiere mucho esfuerzo.

[20] La palabra secuenciador se utiliza para denominar dos cosas radicalmente distintas. Por un lado (como en este caso), designa el componente de hardware o software que administra la ejecución desde el punto de vista temporal, es decir que secuencia un flujo de datos MIDI. Sin embargo es utilizada habitualmente para denominar al software o hardware utilizado para *crear* la secuencia, es decir que se secuencia una melodía en un archivo.

Ejecución en tiempo real

Además de la ejecución de archivos MIDI, resulta deseable utilizar el Tecnocordio como un instrumento musical en el sentido tradicional. Este caso es más simple que el anterior: existe un único instrumento mapeado que responde a los mensajes MIDI recibidos del controlador generando los comandos correspondientes para el servidor según el mapa indicado.

Previamente a la ejecución es necesario seleccionar el dispositivo MIDI IN a utilizar (puede existir más de uno, e incluso es posible utilizar un secuenciador emulando un dispositivo mediante una técnica conocida como *loopback*).



Control mediante interfaz gestual.

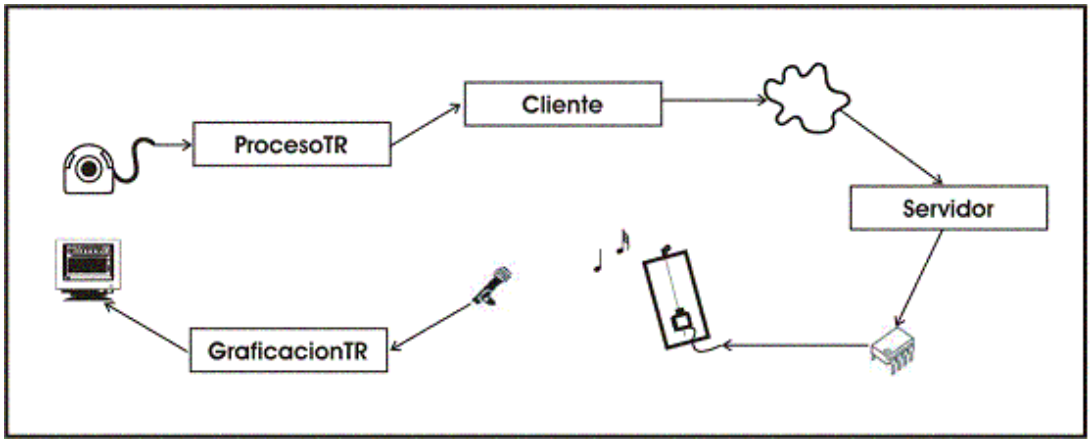
Si bien esta interfaz no ha sido integrada con el resto del sistema, es posible correrla en forma independiente y controlar el Tecnocordio a través de ella.

De esta forma se pueden utilizar las técnicas descritas en la sección de imagen para tañer las distintas cuerdas.

Instanciación de un ciclo completo.

Resulta particularmente interesante la posibilidad de utilizar los módulos desarrollados para construir un sistema que se retroalimente, reinterpretando los estímulos para generar nuevo contenido.

Para ello, es necesario diseñar formas de interpretar los estímulos de manera de lograr el comportamiento necesario (con las interpretaciones implementadas, en particular en la captura de imágenes en tiempo real que ha sido diseñada para una interfaz gestual, no funcionaría).



Así, la cámara digitaliza la señal analógica emitida por el monitor, interpretándola en tiempo real, para controlar (mediante el sistema cliente servidor) el Tecnocordio, quien vuelve los datos a formato analógico (mediante el tañido), los cuales son capturados (y digitalizados) a través de un micrófono, para finalmente ser devueltos al territorio analógico vía los módulos de graficación en tiempo real.

capítulo

9

Consideraciones finales.

Si bien en la propuesta inicial del proyecto se tenía en cuenta que dado el carácter fuertemente experimental e investigativo del mismo, era muy difícil definir en forma concreta los resultados esperados, se fijaron determinados objetivos (detallados en el capítulo dos) que consideramos que fueron ampliamente cumplidos.

Debemos mencionar que el sistema aún no ofrece alternativas suficientes para utilizarse como plataforma educativa, fundamentalmente por lo difícil que puede resultar el acceso para personas carentes de conocimientos técnicos.

Por otro lado, aunque el Tecnocordio representa un ejemplo bastante completo de los desarrollos realizados, nuestra intención inicial era construir un instrumento musical más acabado.

Sin embargo decidimos focalizar nuestro esfuerzo en avanzar de forma más o menos equitativa en todas las áreas, buscando producir resultados que aumentasen las capacidades del sistema en lugar de dedicar tiempo y trabajo en reproducir modelos ya concretados. De hecho, es posible aumentar arbitrariamente el rango tonal del Tecnocordio mediante la sola adición de cuerdas (junto con sus respectivos *drivers* y motores), con lo que se obtendría un instrumento mucho más capaz.

De todas formas, estamos muy conformes con la especificación lograda, la cual permite expandir el sistema para ofrecer la expresividad deseada o utilizar el marco de trabajo construido para nuevos proyectos que respondan a las necesidades expresivas (o educativas) de otros.

Por ejemplo, una leve adaptación de lo que hoy es el Tecnocordio permitiría realizar una instalación en la cual sería posible controlar el tañido de las cuerdas en función de lo captado por la cámara. Luego, podría instalarse en un lugar e interactuar con los espectadores generando atmósferas sonoras que respondan a la actividad circundante.

Mirando hacia el futuro, aparecen muchos desarrollos o combinaciones de lo actualmente disponible que podrían constituir opciones muy interesantes. Resulta natural pensar en combinar los distintos medios e interfaces mediante guiones o también podría avanzarse en una dirección similar a la de los hiperinstrumentos anteriormente descritos. Por ejemplo podría controlarse mediante gestos varias características del sonido, aplicar efectos o manejar el volumen, brindando una expresividad extra a la ejecución de una pieza musical.

Es preciso tener en cuenta, además, que nuestro proyecto se inscribe en un proceso que se inicia hace dos años con el primer curso *Taller de Arte y Programación* (al cual asistimos) y que continúa evolucionando en el marco del NAT. Actualmente el Núcleo de Arte y Tecnología vive en el Instituto de Ingeniería Eléctrica y se encuentra abierto para todos aquellos interesados en colaborar y aprender.

Finalmente, volveremos a citar a Dieste, quien dice al explicar su posición frente a la apropiación tecnológica y a la postura que debe tomarse frente a la tecnología y al primer mundo *"No se trata, pues, de apego reaccionario y sentimental a técnicas superadas, no; se trata de no caer en la otra actitud, aún más sentimental, de la adoración de la riqueza y la eficacia mecánica de los países desarrollados. Porque es muy fácil caer en estas admiraciones sin sentido: recuerdo la fruición casi religiosa con que un amigo arquitecto me describía una fachada que había visto en Alemania, de 100 m de largo, con cristales ahumados, con el perfil de sostén detrás, en los que no se veía "ni una junta, de modo que parecía un solo cristal sin herrería". Esto lo producía una embriaguez casi religiosa, por un hecho de escaso valor, sin dificultad técnica verdadera y sin significación como hecho artístico.*

Contra esta seducción del poder, la riqueza y la eficiencia sin contenido, debemos reaccionar. Creo que ese riesgo ya es menos grave, en este momento de la historia de nuestros países, que han adquirido una conciencia política que no tenían, de lo que significa la explotación del mundo que hacen los poderosos. Debemos salir del subdesarrollo, pero de una manera humana y nuestra, sin copiar ni los procesos, ni las técnicas, mas que cuando nos sea absolutamente indispensable, por urgencias ilevantables. La actitud primaria debe ser repensarlo todo. Por eso creo que, contrariamente a lo que se suele creer, es fundamental la investigación y la meditación personal sobre los fundamentos de todas las cosas. Si queremos formar ingenieros, no debemos formar jóvenes que manejen tablas y manuales de los que desconocen el fundamento, sino al contrario, darles una base sólida en esos fundamentos científicos de su carrera, y, además, mediante cursos de proyecto o taller, enseñarles a usar esos fundamentos.

Incluso políticamente, es ésta la única manera de asegurar esa independencia de las naciones pobres, que es quizá el más gigantesco hecho político de nuestro tiempo" [6]

No podríamos estar más de acuerdo.

Referencias[21].

[1] Heckel, Paul. *The Elements of Friendly Software Design*, Nueva York, EE.UU, 1984, p. 5.

[2] Massachusetts Institute of Technology Media Lab
http://www.media.mit.edu

[3] Hyperinstruments
http://brainop.media.mit.edu/Archive/Hyperinstruments/

[4] Knuth, Donald. *The art of computer programming*
http://www-cs-faculty.stanford.edu/~knuth/taocp.html

[5] Lévy, Pierre, *¿Qué es lo virtual?*, Barcelona, España, Paidós, 1999.

[6] Dieste, Eladio. *Introducción a la teoría*, Montevideo, Uruguay, Facultad de Arquitectura, 1998.

[7] Lévy, Pierre, *L'intelligence collective, pour une anthropologie du cyberspace*, Paris, Francia, La Découverte, 1994.

[8] Castells, Manuel *La era de la información. Economía, sociedad y cultura. Volumen I. La sociedad red*, Madrid, España, Alianza, 1996.

[9] SVTC's (Silicon Valleys Toxics Coalition) Fourth Annual Computer Report Card
http://www.svtc.org/cleancc/pubs/2002report.pdf

[10] E-Waste: Dark Side of Digital Age, 2003, Revista Wired
http://www.wired.com/news/technology/0,1282,57151,00.html

[11] Knuth, Donald, *Literate Programming*,
http://www-cs-staff.stanford.edu/~knuth/lp.html

[12] The Source for Java Technology,
http://java.sun.com/

[13] Del Arenal, Mónica. *El arte como síntesis del caos, entrevista a Santi Eraso,* Guadalajara, México, 2003, Revista PuntoG.
http://www.puntog.com.mx/2001/20010223/ENA230201.htm.

[14] Brown, Gustavo. *Diseño Lógico II - Proyecto I2C*
http://iie.fing.edu.uy/ense/asign/dlp/proyectos/2002/i2c/ProyectoII2C.htm

[15] Diccionario de la Real Academia Española,
http://www.rae.es/

[16] Aho, Alfred y Ullman, Jeffrey; *Principles Of Compiler Design*, Addison-Wesley, 1977

[17] Beanshell - Lightweight Scripting for Java,
http://www.beanshell.org

[21] Los sitios web fueron visitados por última vez en abril de 2003.

[18] JMF Home Page
http://java.sun.com/products/java-media/jmf/

[19] Ptolemy,
http://ptolemy.eecs.berkeley.edu/

[20] Rational, *Unified Modeling Language Resource Center*
http://www.rational.com/uml/index.jsp

[21] Tritonus: Open Source Java Sound,
http://www.tritonus.org/

[22] 120 years of Electronic Music
http://www.obsolete.com/120_years/].

[23] Jure, Luis, *Implementación MIDI 1.0 - Referencia comentada,* Escuela Universitaria de Música, UDELAR,
http://www.eumus.edu.uy/docentes/jure/midi/midi.html

[24] Alvarez, Benjamín, *Música y Matemáticas,* Universidad de Oviedo.
http://campus-llamaquique.uniovi.es/virtual/docencia/musica/]

apéndice i

Código del servidor.

```
/*
server.c
 ---------
 demonio que recibe conecciones TCP/IP en el puerto MYPORT y controla motores paso a paso

 El protocolo consiste en el envío de un comando y cero o más parámetros
 El primer byte es el comando y los últimos parámetros del mismo
 El demonio devuelve OK (definido con un #define) en caso de estar en modo "verbose"
 El modo verbose por defecto es verdadero, pudiéndose desactivar con el comando correspondiente

*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>
#include <asm/io.h>
#include <math.h>


#define DEBUG 0   // si debug == true, envía printf's de status.
#define MYPORT 3490
#define GREETING "Demonio"
#define VERSION "0.1.2 (d13)"

#define PANGLENGTH 20;  // la cantidad de pasos de un tañido

// Comandos

#define demWalk 1                               // equivale a enviar un 1
#define demWalk2 2

#define demToggleVerbose 3

#define demPang 4
#define demPang2 8
#define demPang3 9
#define demPang4 0

#define demQuit 5

#define demIncDF 6
#define demDecDF 7

#define IZQ 0
#define DER 1

// fin comandos

#define OK "0"                                  // lo que devuelve el demonio en modo verbose
#define ERR "1"
```

```
#define BACKLOG 10                      / tamaño de la cola de conexiones pendientes
#define MAXDATASIZE 100                 / máximo numero de bytes que podemos obtener de una
#define base 0x378                      // dirección del parport

#define cantMotores 4;                  / la cantidad de motores que tenemos

int motores[4];                         // el contador del motor
int ultimoPang [4];                      la dir del último pang del motor

int verbose = 0;                         si vale 1 devuelve un ok luego de cada comando, si vale 0 no
int delayFactor = 45000;


void sigchld_handler(int s) {
        while(wait(NULL) > 0);
}

void walk (int motor, int dir, int n, int df) {   // da n pasos en la dir dir al motor motor con un delay df
        int aux, paso, foo;
        foo = 0;

        for (paso = 0; paso < n; paso++){
                for (aux = 0; aux < df; aux++) {  // delay...
                        foo++;
                }
                if (dir == IZQ)
                        motores[motor] = (motores[motor] < 3 ? motores[motor] + 1 : 0);
                if (dir == DER)
                        motores[motor] = (motores[motor] > 0 ? motores[motor] - 1 : 3);

                // al multiplicar por 4 a la motor-1, elijo los bits de salida que uso
                aux = motores[motor] * pow(4, (motor-1));
                outb (aux, base);

                if (DEBUG) printf ("out: %d\n", aux);
        }
}


// hace un pang de hasta 4 motores, los motores no usados deben ser pasados como parámetro 0.

void pang (int m1, int m2, int m3, int m4) {
        int aux, paso, foo;
        int largo = PANGLENGTH;

        foo = 0;


        for (paso = 0; paso < largo; paso++){

                for (aux = 0; aux < delayFactor; aux++) {  // delay...
                        foo++;
                }

                if (m1 != 0) {
                        if (ultimoPang[m1] == IZQ) motores[m1] = (motores[m1] < 3 ? motores[m1] + 1 : 0);
                        else motores[m1] = (motores[m1] > 0 ? motores[m1] - 1 : 3);
                }

                if (m2 != 0) {
                        if (ultimoPang[m2] == IZQ) motores[m2] = (motores[m2] < 3 ? motores[m2] + 1 : 0);
                        else motores[m2] = (motores[m2] > 0 ? motores[m2] - 1 : 3);
                }

                if (m3 != 0) {
                        if (ultimoPang[m3] == IZQ) motores[m3] = (motores[m3] < 3 ? motores[m3] + 1 : 0);
                        else motores[m3] = (motores[m3] > 0 ? motores[m3] - 1 : 3);
                }

                if (m4 != 0) {
                        if (ultimoPang[m4] == IZQ) motores[m4] = (motores[m4] < 3 ? motores[m4] + 1 : 0);
                        else motores[m4] = (motores[m4] > 0 ? motores[m4] - 1 : 3);
                }
```

```
                aux = 0;

                if (m1 != 0) aux += motores[m1] * pow(4, m1-1);
                if (m2 != 0) aux += motores[m2] * pow(4, m2-1);
                if (m3 != 0) aux += motores[m3] * pow(4, m3-1);
                if (m4 != 0) aux += motores[m4] * pow(4, m4-1);

                outb (aux, base);

        }

        if (m1 != 0){
                if (ultimoPang[m1] == IZQ) ultimoPang [m1] = DER;
                else ultimoPang [m1] = IZQ;
        }
        if (m2 != 0){
                if (ultimoPang[m2] == IZQ) ultimoPang [m2] = DER;
                else ultimoPang [m2] = IZQ;
        }
        if (m3 != 0){
                if (ultimoPang[m3] == IZQ) ultimoPang [m3] = DER;
                else ultimoPang [m3] = IZQ;
        }
        if (m4 != 0){
                if (ultimoPang[m4] == IZQ) ultimoPang [m4] = DER;
                else ultimoPang [m4] = IZQ;
        }
}


int main(void) {
        char buf[MAXDATASIZE];
        int sockfd, new_fd, numbytes;  // escucho en sock_fd, nueva conexión en new_fd
        struct sockaddr_in my_addr;    // la info de mi dirección
        struct sockaddr_in their_addr; // la indo de la dirección de quien se conecta
        int sin_size;
        int i;
        int termino = 0;
        struct sigaction sa;
        int yes=1;

        printf (GREETING);
        printf ("\nversion ");
        printf (VERSION);
        printf ("\nAceptando conexiones en el puerto %d.\n",MYPORT);
        if (DEBUG) printf ("[modo debug]\n");


        if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                perror("socket");
                exit(1);
        }

        if (setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1) {
                perror("setsockopt");
                exit(1);
        }

        my_addr.sin_family = AF_INET;         // host byte order
        my_addr.sin_port = htons(MYPORT);     // short, network byte order
        my_addr.sin_addr.s_addr = INADDR_ANY; // lo lleno con mi IP
        memset(&(my_addr.sin_zero), '\0', 8); // pongo el resto del struct en cero

        if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1) {
                perror("bind");
                exit(1);
        }

        if (listen(sockfd, BACKLOG) == -1) {
                perror("listen");
                exit(1);
        }
```

```
sa.sa_handler = sigchld_handler; // destruir todos los procesos muertos
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;
if (sigaction(SIGCHLD, &sa, NULL) == -1) {
        perror("sigaction");
        exit(1);
}

while(!termino) {  // bucle principal del accept()


        sin_size = sizeof(struct sockaddr_in);

        if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -1) {
                perror("accept");
                continue;
        }

        printf("Conexion desde %s\n", inet_ntoa(their_addr.sin_addr));

        if (!fork()) { // es el hijo


                ose(sockfd); // que no necesita el listener

                // no envío nada, no está verbose por default
                // if (send(new_fd, OK, 2, 0) == -1) perror("send OK inicial");


                // si está en modo DEBUG y saco el comentario envía un saludo al cliente

                // if (DEBUG) if (send(new_fd, "Hello, world!\n", 14, 0) == -1) perror("send");


                // inicializo los arrays

                for (i = 0; i < 5; i++) {

                        ultimoPang[i] = DER;
                        motores[i] = 0;
                }


                // verifico que se pueda escribir al parport, si no puedo termino la ejecución.
                if (ioperm(base,1,1))
                        fprintf(stderr, "Error: este programa debe ejecutarse con permisos de root;
no pude obtener el puerto paralelo. %x\n", base), exit(1);

                // Ejecuto los comandos, los cuales vienen en buf[0]
                // En los comentarios, con Pi me refiero al parámetro número i

                if (DEBUG) printf ("Esperando comandos...\n");


                while (buf[0] != demQuit) {



                        // recibo el mensaje en buf
                        if ((numbytes = recv(new_fd, buf, MAXDATASIZE-1, 0)) == -1) {
                                perror("recv");
                                exit(1);
                        }


                        if (buf[0] == demWalk) { // P1 = motor; P2 = dir; p3 = steps

                                walk (buf[1], buf[2], buf[3], delayFactor);

                                if (DEBUG) {
                                        if (buf[2] == IZQ) printf ("caminando: motor=%d;
dir=IZQ (%d); pasos=%d; delayFactor=%d\n",buf[1],buf[2],buf[3],delayFactor);
```

```
                                                    else if (buf[2] == DER) printf ("caminado: motor=%d;
dir=DER (%d); pasos=%d; delayFactor=%d\n",buf[1],buf[2],buf[3],delayFactor);
                                                    else printf ("caminand: motor=%d; dir=???
(%d); pasos=%d; delayFactor=%d\n",buf[1],buf[2],buf[3],delayFactor);
                                                    }
                                            }


                                            if (buf[0] == demPang) { // P1 = motor
                                                    pang (buf[1],0 ,0, 0);
                                                    if (DEBUG) printf ("demPang %d\n", buf[1]);

                                            }

                                            if (buf[0] == demPang2) { // P1 = motor1; P2 = motor2
                                                    pang (buf[1], buf[2], 0, 0);
                                                    if (DEBUG) printf ("demPang2 %d %d\n", buf[1], buf[2]);
                                            }

                                            if (buf[0] == demPang3) { // Pi = motor i
                                                    pang (buf[1], buf[2], buf[3], 0);
                                                    if (DEBUG) printf ("demPang3 %d %d %d\n", buf[1], buf[2],
buf[3]);
                                            }

                                            if (buf[0] == demPang4) { // Pi = motor i
                                                    pang (buf[1], buf[2], buf[3], buf[4]);
                                                    if (DEBUG) printf ("demPang4 %d %d %d %d\n", buf[1], buf[2],
buf[3], buf[4]);
                                            }


                                            if (buf[0] == demIncDF) { // aumento delayFactor
                                                    delayFactor += 1000;
                                                    printf ("delayFactor: %d\n", delayFactor);
                                            }

                                            if (buf[0] == demDecDF) { // disminuyo delayFactor
                                                    delayFactor -= 1000;
                                                    printf ("delayFactor: %d\n", delayFactor);

                                            }

                                            if (buf[0] == demToggleVerbose) {
                                                    verbose = ! verbose;
                                                    if (verbose) printf ("verbose ON\n");
                                                    else printf ("verbose OFF\n");
                                            }

                                            if (verbose) {
                                                    if (send(new_fd, OK, 1, 0) == -1) perror("send OK verbose");


                                            }

                                            buf[numbytes] = '\0';
                                            if (DEBUG) printf ("String recibido: %s\n",buf);
                                    } // while !quit
                                    printf ("\nQuit recibido... terminando la conexión...\n");
                                    termino = 1;
                            } // if ! fork()
                            close(new_fd);  // parent doesn't need this
                    } // while !termino);
                    return 0;
}
```

apéndice

ii

Datos de los estudiantes.

Este proyecto fue realizado por

- Fabrizio Castro.
  C.I. 3812435-7
  jfcastro@montevideo.com.uy

- Tomás Laurenzo.
  C.I. 2836617-1
  laurenzo@fing.edu.uy