UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

"DESARROLLO DE SOFTWARE PARA LA CAPTURA, DESPLIEGUE, ALMACENAMIENTO, PROCESAMIENTO Y PUBLICACION DE DATOS EN TIEMPO REAL OBTENIDOS DEL RADIO INTERFEROMETRO SOLAR RIS"

T E S I S

QUE PARA OBTENER EL TITULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACION

P R E S E N T A :

VICTOR HUGO DE LA LUZ RODRIGUEZ

DIRECTOR DE TESIS: DR. ALEJANDRO LARA SANCHEZ

ASESOR: M. EN C. JORGE LUIS ORTEGA ARJONA

FACULTAD DE CIENCIAS
UNAM

2005

ACT. MAURICIO AGUILAR GONZÁLEZ
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito:
"Desarrollo de Software para la Captura, Despliegue, Almacenamiento, Procesamiento y Publicacion de Datos en Tiempo Real obtenidos del Radio Interferómetro Solar RIS".

realizado por De la Luz Rodríguez Víctor Hugo

con número de cuenta 09625939-5 **, quien cubrió los créditos de la carrera de:** Ciencias de la Computación

Dicho trabajo cuenta con nuestro voto aprobatorio.

A t e n t a m e n t e

Director
Propietario Dr. Alejandro Lara Sánchez
Asesor
Propietario M en C. Jorge Luis Ortega Arjona

Propietario Dr. Octavio Páez Osuna

Suplente Dr. Víctor Manuel Velasco Herrera

Suplente M en C. Enrique Cruz Martínez

Consejo Departamental de Matemáticas

Dr. Francisco Hernández Quiroz
Coordinador de la Carrera
de Ciencias de la Computación

A mis padres:

Jovita Rodríguez Vazquez

José Gabriel De la Luz De Lázaro

Agradecimientos

Deseo agradecer al Dr. Alejandro Lara Sánchez, mi director de tesis; por la confianza en encomendarme este proyecto y por las oportunidades que me a dado y que han tenido un gran impacto en mi formación profesional y humana.

Al técnico Filiberto Matías Dominguez y al Físico Samuel Torres Mendoza por su apoyo en el Laboratorio de Interferometría y principalmente por su sincera amistad.

A los miles de Hackers que hacen la comunidad Open Source, que jamas tuve el gusto de conocer y que me ayudaron directa o indirectamente sin ningún fin de lucro.

Y sobre todo a mi mujer Noemi Jasmín por todo lo que en realidad importa.

Índice general

Índice de figuras

Índice de Tablas

Resumen

El desarrollo de software es un proceso largo y costoso. Involucra tanto el diseño como la planeación, y por lo tanto, tener acotados los posibles riesgos resulta casi imposible, pues los constantes cambios tecnológicos hacen difícil el conocimiento total de un sistema. Hay diferentes riesgos, que abarcan desde la estructura intrínseca de los sistemas operativos hasta una mala planeación en los modelos creados.

La finalidad de este proyecto es modernizar la última etapa del Radio Interferómetro Solar RIS del Instituto de Geofísica de la UNAM. El proyecto consiste principalmente en la adquisición y despliegue de datos en tiempo real mediante un conjunto de programas que llámamos Xquetzal/Xitris. Esta construido en mayor parte con Software Libre y por consecuencia hereda la licencia GPL[15].

Usamos dos tipos de metodologias de desarrollo: una formal, la ESA PSS-05-0 de la Agencia Espacial Europea y por otro lado eXtreme Programming, el cual, mas que una metodología, son una serie de recomendaciones para la produccion de software mas dinamicamente y menos rigido en su estructura de desarrollo. El uso de estas dos metodologias se debió a que pasamos por un problema técnico que exigió replantear un nuevo modelo para el sistema. Un error que costo varios meses de trabajo, y demostró que es de vital importancia elegir un modelo adecuado de desarrollo de software y que hay modelos que no se ajustan a los requerimientos que demanda la producción de Software Científico de alta especialización y de pequeños equipos de trabajo, los cuales poco tienen que ver con el paradigma de la POO[1] y el desarrollo modular que varios enfoques como el TSP[2] pretenden darle a un proyecto, de tal forma que es necesario usar otras estrategias.

Los documentos que aquí se encuentran tratan de apegarse lo más posible a los estándares de la industria, no son exactos porque algunos están pensados en la distribución de la carga de trabajo, es decir, hay varias personas con características o roles específicos que juegan papeles determinados en cada etapa del desarrollo, en otro caso, dictan reglas que no se pueden cumplir, como es el caso de la programación en pares de XP[3] .

En la primera parte de este trabajo se presenta una introducción sobre el Radio Interferómetro Solar y el papel que juega dentro del Instituto de Geofísica de la UNAM y la importancia de su modernización. En la segunda parte se presenta el proceso de desarrollo de software bajo el modelo de cascada propuesto por el estándar ESA PSS-05-0. En la tercera parte se plantea un nuevo modelo tanto conceptual como de desarrollo. El modelo esta basado en el esquema cliente-servidor con una metodología de eXtreme Programming modificado. La cuarta parte se presentan las conclusiones generales del trabajo. Existen una serie de apéndices, donde se encuentran los documentos y el código generado en cada una de las fases de desarrollo.

[1] Programación Orientada a Objetos.
[2] Team Software Process (Proceso de Software en Equipo).
[3] Abreviación de eXtreme Programming (Programación Extrema).

Capítulo 1

Introducción

1.1. Laboratorio de Interferometría Solar

1.1.1. Historia

El laboratorio de Interferometría Solar está ubicado en el segundo piso del Instituto de Geofísica de la UNAM. Su principal actividad está basada en el monitoreo del sol, aproximadamente de las 8:00 am a las 6:00 pm. El laboratorio está a cargo del Dr. Alejandro Lara Sánchez. Investigador de tiempo completo en el área de Física Espacial. El Técnico Laboratorista es Filiberto Matías, encargado del mantenimiento del Radio Interferómetro Solar RIS y de la supervisión de los datos generados por el RIS.

El laboratorio cuenta con equipo que fue donado por la Unión Soviética en la década de los 70s [9] el cual a funcionado intermitentemente debido a que las piezas se desgastan o sufren daños y es difícil encontrar substitutos. El Radio Interferómetro Solar (RIS) continua funcionando hasta la fecha y a sido objeto de diversas modificaciones desde su llegada a México [45].

El RIS como cualquier otro radiotelescopio se compone esencialmente de una antena receptora, un generador de ruido, un receptor y finalmente un dispositivo de salida (figura 1.1).

El dispositivo de salida en este caso es un graficador, también de origen Ruso y en la cual, solo se ven curvas que representan el flujo de energía del sol captada y procesada a través de las diferentes etapas del RIS.

Estas graficas de flujo contra tiempo (imagen 1.2) son el principal material de trabajo para los investigadores que están interesados en eventos solares. Debido a que el graficador es demasiado viejo, los datos obtenidos no se pueden utilizar, ya que para el análisis moderno se necesitan datos digitales para ser procesados en una computadora. Este hecho nos demuestra la importancia del proyecto.

La posición geográfica del RIS es Latitud: $19^0 19'$N, Longitud: $99^0 10'$O, Altitud: $2,240$ m sobre el nivel del mar. Estos datos son aproximados y se basan en el mapa topográfico de la zona emitido por el INEGI.

1.1.2. Etapas del RIS

El RIS trabaja de forma modular, la señal proveniente del sol (en este caso una onda electromagnética con una frecuencia media de 7.5 GHz) pasa por las siguientes etapas [45]:

1. Espejos primarios parabólicos.

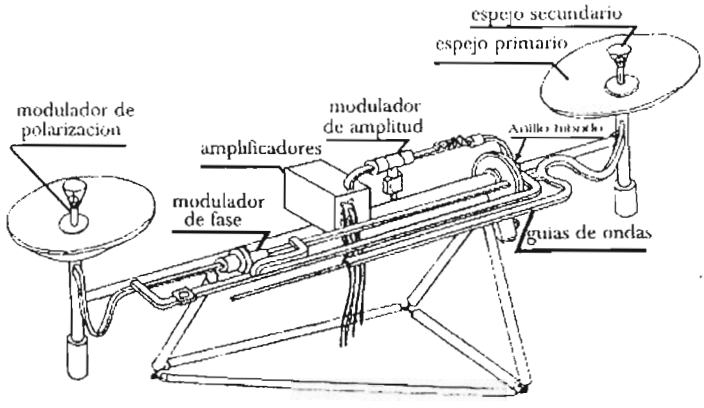

Figura 1.1: Radio Interferómetro Solar RIS.

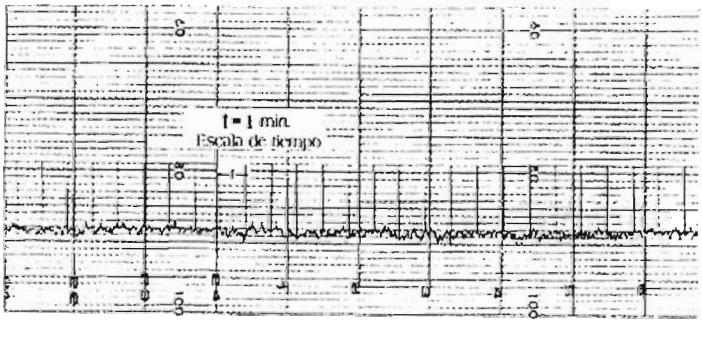

Figura 1.2: Registro típico en el canal de intensidad durante un día sin actividad solar.

2. Espejos secundarios.
3. Bocina de entrada.
4. Modulador de polarización.
5. Guía de ondas.
6. Modulador de fase.
7. Anillo híbrido.
8. Moduladores de amplitud o intensidad.
9. Amplificadores.
10. Grabadores.

La señal finalmente se ve como una curva a lo largo del papel graficador que indica la variación de flujo proveniente del sol a una frecuencia de 7.5GHz.

1.1.3. Infraestructura de Computo

Al iniciar el proyecto el laboratorio contaba con el siguiente equipo:

1. Red:
   a) 2 Direcciones IP Homologadas (132.248.6.147 y 132.248.182.55).
2. Hardware:
   a) Computadora Compaq Deskpro (Cintli).
      1) Procesador Pentium MMX a 166MHz
      2) 46Mb en RAM
      3) 12Gb Hdd.
      4) Bus Master a 66MHz.
      5) Tarjeta de Vídeo Cirrus Logic GD 5430.
   b) Computadora Armada con procesador 486 (Ris).
   c) Impresora LaserJet 5L
   d) Tarjeta Convertora Análogo/Digital LabPC+ Board con interfase ISA.
3. Software:
   a) Debían GNU/Linux potatoe
   b) Windows 95.
4. Documentación:
   a) Manuales de Usuario Compaq
   b) Lab-PC+ User Manual

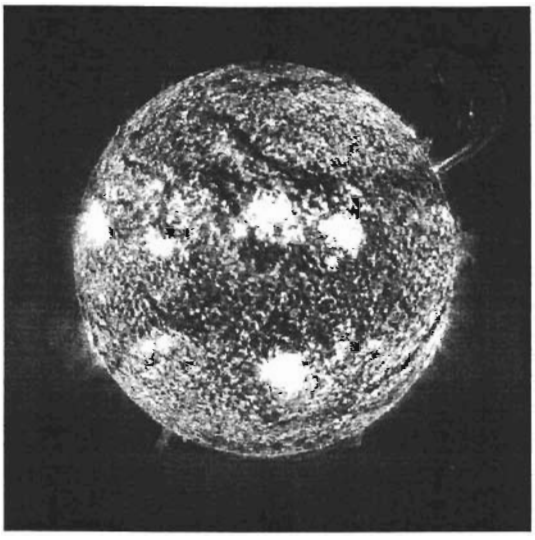

Figura 1.3: Imagen del sol, capturada por el EIT (Extreme ultraviolet Imaging Telescope) del satelite SOHO (Solar and Heliospheric Observatory) el 14 de Septiembre de 1999.

1.2. El sol, el objeto de estudio del RIS

1.2.1. Propiedades

Los astrónomos suelen clasificar a las estrellas, pues existen diferentes tipos de ellas. El Sol, con base en su temperatura y su tamaño se conoce como una estrella enana del tipo G2(V) (imagen 1.3); este tipo de estrellas es de color amarillo, con temperatura superficial del orden de 6000^0C, más calientes que las estrellas rojas pero más frías que las azules, y son moderadamente brillantes. Aunque para nosotros resulta deslumbrante debido a su cercanía, existen estrellas que son decenas de miles de veces más brillantes que él, pero también hay otras decenas de miles de veces más tenues. No es una estrella grande (tiene sólo alrededor de un millón 400000 kilómetros de diámetro), las hay 30 millones de veces más grandes. Es una estrella de mediana edad (aproximadamente de 5000 millones de años) y con una masa de dos quintillones de kilogramos.

El Sol es una de los cientos de miles de millones de estrellas que forman nuestra galaxia, la cual convive con alrededor de otras 20 galaxias en el llamado grupo local, compuesto por al menos 10 000 millones de ellas. No es un cuerpo sólido, sino gaseoso, como todas las estrellas, con una densidad media de 1.4 veces la densidad del agua. Como todas las estrellas, el Sol gira, completando una vuelta en aproximadamente 27 días, pero como no es sólido, sus regiones ecuatoriales giran más rápido que las polares. Algunas observaciones han sugerido que su diámetro polar es 70 kilómetros menor que el diámetro ecuatorial, pero prácticamente puede considerarse esférico.

1.2.2. Estructura

El Sol es una esfera de gas caliente, pero no una esfera homogénea; tiene una estructura diferenciada en capas concéntricas de diferentes propiedades. La superficie visible del Sol es la fotosfera, cuyo nombre quiere decir "esfera de luz" y es una capa muy delgada, de aproximadamente 300 kilómetros de espesor (0,05 % del radio del Sol). Aunque parece que ésta es la capa más externa, en realidad no es así. Cuando la brillante luz de la fotosfera es cubierta por el disco de la Luna durante un eclipse total de Sol, es posible distinguir dos capas superiores de tenue brillo pero claramente diferentes. La primera de ellas es una capa de luz rojiza llamada cromosfera, de aproximadamente 8 000 kilómetros de espesor. Por encima de ella se encuentra la corona, de tenue luz aperlada que se extiende hasta más allá de la Tierra. En realidad, el Sol no tiene una "superficie"bien definida, sino que su densidad disminuye continuamente desde su centro hacia afuera a través de todo el sistema planetario y se mezcla, más allá de él, con el material interestelar. Lo que llamamos "el radio del Sol"es la distancia del centro al borde superior de la fotosfera, pero el Sol se extiende en realidad por muchísimos millones de kilómetros y hablando de manera propia deberíamos decir que la Tierra y todos los planetas se encuentran inmersos en él.

La estructura del interior del Sol no puede observarse en forma directa y sólo puede deducirse mediante consideraciones teóricas a partir de sus características superficiales. De esta manera se ha estimado que su interior está diferenciado en tres zonas. La más interna, que va desde el centro hasta una distancia de aproximadamente dos décimas del radio del Sol, es el núcleo, donde se produce de forma constante una enorme cantidad de energía. Esta energía es transportada hacia la superficie del Sol, primero en forma de radiación (por absorción y emisión de rayos X) y posteriormente en forma convectiva (por medio de burbujas de gas caliente que suben hasta la superficie). La primera región es la llamada zona radiativa, que se extiende desde dos décimas hasta seis u ocho décimas del radio del Sol, y la segunda es la zona convectiva, que va desde seis u ocho décimas del radio del Sol hasta la superficie [3].

1.2.3. Espectro Electromagnético

Cuando hablamos de la energía emitida por el Sol nos referimos a la luz; más específicamente a ondas electromagnéticas. Es en esta forma como el Sol envía la mayor parte de la energía que recibe la Tierra.

Las ondas electromagnéticas se distinguen unas de otras por su frecuencia (ciclos por segundo) o su longitud de onda. Las ondas más largas (de menor frecuencia) son las ondas de radio, cuya longitud de onda puede ser desde más de mil kilómetros hasta unos cuantos metros. Las ondas electromagnéticas de longitudes entre un metro y un milímetro se llaman microondas y tienen frecuencias mayores que las ondas de radio. Siguen después los rayos infrarrojos, que son las ondas electromagnéticas que se encuentran entre microondas y el rojo, que es el primer color, o la frecuencia más baja que el ojo humano puede detectar. Entre 700 y 400 milimicras de longitud de onda se encuentran las ondas electromagnéticas visibles, que es lo que propiamente llamamos luz, y va desde el rojo hasta el violeta. Solamente en este rango de longitudes de onda es sensible el ojo humano; las frecuencias correspondientes para el intervalo visible son de cientos de billones de ciclos por segundo. Las ondas electromagnéticas de frecuencias más altas que las visibles (longitudes de onda más cortas) son: la luz ultravioleta, los rayos X y los rayos γ; estos últimos comprenden hasta longitudes de onda menores que una billonésima de metro y hasta frecuencias superiores a miles de trillones de ciclos por segundo. Todas estas ondas constituyen el espectro electromagnético [3].

El Sol emite energía en todas las longitudes de onda: desde los ultra cortos rayos γ hasta

las gigantescas ondas de radio; sin embargo, no emite la misma cantidad de energía en todas ellas. Aproximadamente el 40% de la energía emitida por el Sol está en la porción visible del espectro y 50% en el infrarrojo; casi todo el resto está en el ultravioleta. La emisión continua de rayos X y de ondas de radio del Sol es sumamente baja y sólo aumenta esporádicamente debido a la ocurrencia de ciertos eventos solares explosivos. También en estos eventos suelen emitirse rayos γ, pero no parece haber una emisión continua de ellos [3].

1.2.4. Emisiones Solares

El Sol emite tanto ondas electromagnéticas (en todas las longitudes de onda) como partículas (en su mayor parte protones y electrones de muy diversas energías). Algunas de estas emisiones son continuas, mientras que otras son esporádicas, originadas sobre todo en las grandes explosiones que ocurren en el Sol, llamadas ráfagas.

En forma continua, el Sol emite desde la fotosfera principalmente luz visible y algo en el ultravioleta y el infrarrojo; la mayoría de la emisión ultravioleta se origina en las capas superiores, la cromosfera y la corona, y casi toda la emisión continua en rayos X proviene de esta última. También hay una emisión continua (muy débil) de ondas de radio provenientes principalmente de la alta cromosfera y baja corona. Cuando ocurren ráfagas solares, la emisión de rayos X y de ondas de radio aumenta en forma considerable (además, por supuesto, de un abrillantamiento en la región del visible) y pueden eventualmente emitirse rayos γ.

Respecto a la emisión de partículas, existe un flujo continuo de plasma solar, constituido en su mayor parte por electrones y protones, que barre todo el sistema planetario. Asociados con la ocurrencia de una ráfaga, suelen también detectarse protones y partículas alfa (núcleos de helio) muy energéticos, aunque éstos se registran a veces sin que se haya observado una ráfaga. A las más energéticas de estas partículas se les llama rayos cósmicos solares [3] .

1.3. El RIS, un Monitor Solar

Como hemos visto, el Sol emite energía en una gran variedad de frecuencias, esta emisión es variable y en ciertas longitudes de onda la emisión no es continua, como en el caso del radio. Esta es una de las características en las que se basan la mayoría de los monitores solares, los cuales miden la cantidad de flujo energético del Sol y reportan sus variaciones en una longitud de onda específica.

Como ejemplos tenemos:

1. Mees Solar Observatory [36].
2. The Solar and Heliospheric Observatory [35].
3. Nobeyama Radio Observatory: Solar [34].
4. Kanzelhohe Solar Observatory [12]

El Radio Interferómetro Solar RIS es parte de esta comunidad, lleva a cabo observaciones metódicas sobre la región de Radio a 7.5GHz y se encarga de registrar los eventos ocurridos en la atmosfera baja del Sol.

1.4. Objetivos del Proyecto

Como primera aproximación podemos decir que el objetivo principal es: "Modernizar la ultima etapa del RIS", las metas son:

1. Desarrollo de Software para manejar la tarjeta convertidora Análogo/Digital(DAC).
2. Desarrollo de Software para monitorear la señal proveniente del RIS en tiempo real.
3. Diseño e implementación de una Base de Datos para almacenar las observaciones del RIS y los Eventos Solares.
4. Diseño e implementación del Sitio Web para publicar los Eventos Solares en tiempo real.

En el siguiente capítulo usaremos el estándar ESA PSS-05-0 como metodología y modelo de desarrollo, definiremos técnicamente tanto los requerimientos del usuario, los requerimientos del software, su modelo arquitectónico y su codificación para llevar a buen termino los objetivos del proyecto.

Capítulo 2

ESA PSS-05-0

2.1. Introducción

El estándar ESA PSS-05-0[28], de la European Space Agency, corresponde a un estándar que especifica como un equipo de desarrollo de software debe realizar los procesos para producir software de calidad. Este estándar define como hacer cosas para el caso de sistemas de información de tamaño pequeño, en vez de definir que cosas hacer [20], es decir, explica los mecanismos de control y el ciclo de vida de un proyecto.

El ciclo de vida del software se inicia cuando el software se concibe, y termina cuando este ya no es usado. Un modelo de ciclo de vida estructura las actividades del proyecto en fases, y define su actividades.

En este capítulo abordaremos los rasgos mas característicos del Estándar y se definirán los documentos técnicos que conforman el proyecto.

El estándar ESA PSS-05-0 divide el ciclo de vida del software en seis fases principales y cuatro fases complementarias [24], las cuales son:

1. RU: Definición de requisitos de usuarios [21]. Corresponde a la especificación de los requisitos del sistema de software, usando el lenguaje del usuario y sin usar términos técnicos. Dentro de las actividades principales de esta fase, se encuentra la determinación del ambiente operacional y la especificación de los requisitos de usuarios. El documento generado en esta fase es el DRU o Documento de Requisitos de Usuarios, el cual detalla las necesidades específicas del usuario, así como lo que espera que el software haga por él. También se genera la primera versión del PAPS o Plan de Administración del Proyecto de Software, en este plan se estiman los costos del proyecto.

2. RU/R: Como un anexo a la fase anterior, se incluye un documento en donde el cliente aprueba el DRU, con la finalidad de asegurar la calidad.

3. RS: Definición de requisitos de software [26]. Este corresponde a la especificación de los requisitos del sistema de software, vistos desde la perspectiva del equipo de desarrollo, los cuales definen el sistema de software a construir utilizando terminología y vocabulario técnico, es en esta fase donde se debe especificar que es lo que debe y no debe hacer el software, mas aun no se especifica como hacerlo, es decir, aun no se especifica la programación ni la arquitectura del software. Las principales actividades en esta fase son:

   a) Construcción del modelo lógico del sistema.

b) Especificación de los requisitos del software.

El documento que se desprende de esta fase es el DRS o Documento de Requisito de Software, en este documento debe omitirse terminología de implementación como los requisitos de software u otros, esto ultimo tiene como finalidad ligar el trabajo de análisis con el de diseño. Es aquí donde se debe revisar el PAPS o Plan de Administración del Proyecto de Software preparado en la fase RU, en dicho plan se debe presentar un plan detallado para la fase de Diseño Arquitectónico.

4. RS/S Como un anexo se incluye la aprobación del DRS por parte del cliente.

5. DA: Definición del Diseño Arquitectónico [22]: El propósito de esta fase es definir la estructura del software tomando como base el modelo construido en la fase RS, este modelo es transformado en diseño arquitectónico asignando funciones a componentes de software y definiendo el control y el flujo de los datos entre ellos. Es aquí donde se deben identificar las partes criticas del diseño, puede ser necesario realizar prototipos de software para confirmar las suposiciones básicas del diseño. El documento generado es el DDA o Documento de Diseño Arquitectónico.

6. DA/R: Como un anexo se incluye la aprobación del DDA por parte del cliente y de la gente que este involucrada en el proyecto.

7. DD: Diseño Detallado y Producción del Código [23]: El propósito de esta fase es el de detallar el diseño del software, codificarlo, documentarlo y probarlo. Es así como se forman el DDD y el MUS (Documento de Diseño Detallado y Manual de Software de Usuario). En esta fase hay que entregar el DDD, el MUS y el código generado. Al final de esta fase el software puede considerarse listo para pruebas del personal.

8. DD/R: Como un anexo se incluye la aprobación del DDD. MUS por parte del cliente.

9. TR: Transferencia de Software a Operaciones [27]: En esta fase se instala la aplicación en el lugar donde será usado. Es aquí donde se encuentra un gran número de problemas técnicos si no se tiene una estrategia previa, pues pueden aparecer una gran cantidad de errores, desde compatibilidades en versiones de herramientas y sistemas operativos como en capacidad de procesamiento en las PCs. En esta etapa se genera el DTS o Documento de Transferencia de Sistema en el cual se especifica los problemas que surgieron en la transferencia de plataformas así como la aceptación parcial del cliente.

10. OM: Operación y Manutención[25]: Es la última fase y puede ser muy larga ya que involucra el constante mantenimiento del software para corregir errores que se puedan presentar en este transcurso de tiempo. Es aquí donde se genera el último documento denominado DHP o Documento de Historia del Proyecto. Este documento es de suma importancia pues aquí se resume toda la historia del proyecto y servirá como base para el siguiente desarrollo.

A continuación se describen las primeras cuatro fases de desarrollo, las cuales contienen la información fundamental para la conformación del proyecto y los documentos técnicos generados según el estándar. Observaremos que las primeras fases de desarrollo son las mas demandantes en diseño, planeación e investigación y son de vital importancia para la conformación solida y el éxito del proyecto.

Una vez planteado el problema se procede a su implementación y es en la sección 2.5 donde se lleva a cabo dicha tarea. Desafortunadamente una serie de eventos (Adquisición de nuevo equipo de computo, problemas en tiempos de ejecución, etc.) hacen que se tome una segunda alternativa de desarrollo, planteada en este mismo capítulo en el apéndice C.5.

2.2. Definición de Requisitos de Usuarios

2.2.1. Determinación del Entorno Operacional

En el Instituto de Geofísica de la UNAM se encuentra el Radio Interferómetro Solar (RIS), este Interferómetro tiene como salida de datos un graficador de origen Ruso. Este graficador dibuja una línea quebrada y contínua sobre papel graduado a un intervalo de aproximadamente 0.5 seg. Esta línea representa los datos que registra el RIS. Posteriormente estos datos son almacenados y catalogados para su análisis.

Es de vital importancia para este laboratorio tener los datos de una manera más práctica, es decir tenerlos de manera digital para analizarlos con paquetes de alto nivel como los son IDL o GNUplot. Para ese propósito el Laboratorio cuenta con una interfaz ISA de origen Estadounidense de la marca National Instruments llamada LabPC+. Esta tarjeta es capaz de convertir datos análogos a digitales y además puede estar monitoreando 4 canales a la vez[6].

El RIS cuenta con 4 canales de salida de datos : Intensidad, Polarización, Seno y Coseno, 3 de los cuales han estado en desuso por falta de graficadores (Polarización, Seno y Coseno).

Al inicio del proyecto la infraestructura del Laboratorio era algo pobre, contaba con 2 computadoras (486 y Pentium) con poca memoria y lentas (Ver apéndice B.4.1).

En el laboratorio hay 2 trabajadores. El Doctor Alejandro Lara Sánchez que se encuentra a cargo y el Laboratorista Filiberto Matías, es este ultimo el que monitorea constantemente el papel graduado en busca de cambios de intensidad a la que llaman ráfagas solares.

2.2.2. Identificación de los requisitos del usuario

En la computadora llamada *Cintli* se encuentra un programa que esta funcionando en fase de prueba desde hace 6 meses.

El Dr. Alejandro Lara a estado trabajando algunos meses con el y encontró una gran cantidad de deficiencias en el programa, entre las cuales mencionaremos las siguientes:

1. Perdida de datos con respecto al tiempo.
2. Dificultad para modificar las escalas de tiempo en la visualización.
3. No hay pruebas confiables para saber que los datos obtenidos mediante el programa sean verídicos.
4. Imposibilidad de crear una aplicación más ambiciosa como lo es un detector de eventos, como las ráfagas solares.
5. No hay un esquema fijo en el desarrollo del software y mas bien es una caja negra.

Por estos puntos el Dr. Alejandro Lara tuvo la necesidad de modificar el proyecto, teniendo como base la creación de un software que fuera reutilizable, fácil de usar, que tuviera esquemas de captura de datos como se llevan a cabo diariamente en el laboratorio, con la posibilidad de almacenar los datos en formatos estándares como el ASCII, crear herramientas de búsqueda y al mismo tiempo publicar los datos en tiempo real para el mundo a través de Internet.

Para la realización del proyecto, el Dr. Alejandro Lara instalo Debían Linux en la máquina llamada Cintli y construyo una pequeña aplicación en la que grafica datos en una ventana en el entorno X, pero no se tiene ningún tipo de control, además de una serie de inconvenientes. Con este antecedente el Dr. Alejandro Lara se dio cuenta de la necesidad de crear un proyecto que se enfocara 100 % a la solución de este problema, pero con una serie de restricciones que enumeramos a continuación:

1. El software debe estar construido en una plataforma GNU/Linux por lo siguiente:
   a) Bajo Costo en términos de Licencias.
   b) Malas experiencias con el S.O. Microsoft Windows 95.
   c) Para contribuir al desarrollo de Tecnología Mexicana que fuera reutilizable sin ningún problema de licencias.
   d) Estabilidad, facilidad y experiencia en el manejo de sistemas tipo Unix.
   e) Las herramientas gráficas que usan para el análisis de datos corren bajo sistemas tipo Linux/Unix.
2. Los requerimientos del sistema son:
   a) Capturar los datos en tiempo real con intervalos de tiempo que van desde los 0.001 seg. hasta 30 seg.
   b) Poder visualizar los datos en tiempo real en alguna aplicación gráfica.
   c) Controlar la tarjeta (Cambios de amplificación y frecuencia de muestreo) en tiempo real.
   d) Que todos estos cambios queden registrados en un log del Sistema.
   e) Almacenar los datos de una manera estándar y ordenada.
   f) Que esos datos estén disponibles en por lo menos dos formatos: ASCII para que cualquier aplicación los pueda entender y en algún formato comprimido para ahorrar espacio en el servidor.
   g) Todos los datos que estemos capturando se encuentren a disposición para cualquier gente que tenga conexión a Internet.
   h) Que haya un monitor de eventos solares en tiempo real.
3. Que cuente con una interfaz amigable.
4. Que este totalmente documentado.
5. Sea modular y reutilizable pues posteriormente se pretende comprar otra tarjeta convertora.

2.2.3. Resultados

El documento generado en esta fase del desarrollo se encuentra al final de esta publicación con el titulo de DRU (Ver Apéndice A).

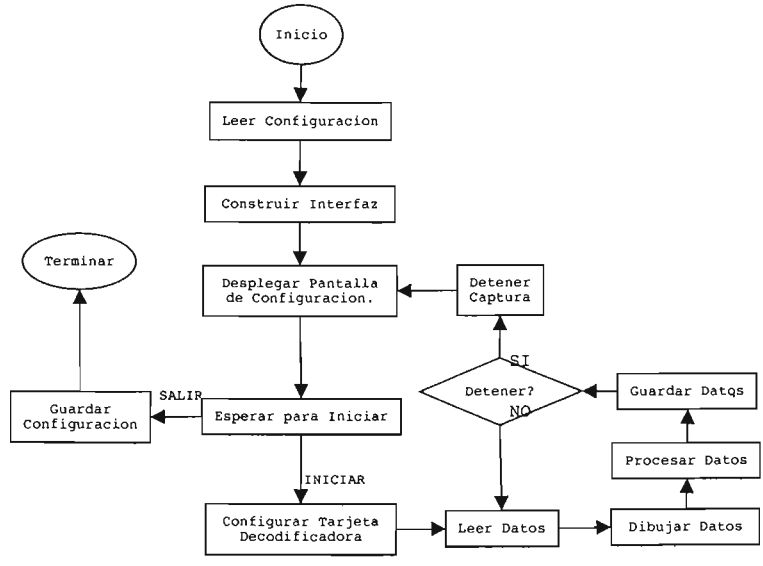

Figura 2.1: Modelo Lógico de xquetzal

2.3. Definición de los Requisitos de Software

2.3.1. Construcción del modelo lógico

Analizando el DRU (Apéndice A) en su ultima versión, podemos hacer una primera aproximación al modelo lógico representado en la figura 2.1 que comprende la captura de datos en tiempo real, el despliegue en pantalla, el control de la tarjeta en tiempo real y el almacenamiento ordenado de la información.

Este modelo funcionaría de la siguiente manera:

1. El programa lee la configuración de un archivo.
2. De acuerdo con esta configuración se construye y despliega la interfaz.
3. El usuario tiene a su disposición una ventana con las posibles configuraciones que estén disponibles como lo es la amplificación y la frecuencia de muestreo.
4. Una vez que el usuario haya configurado la interfaz procede a iniciar la captura o en su defecto terminar la aplicación.
5. Si desea iniciar la captura, xquetzal se comunica a la tarjeta y la configura de acuerdo a los parametros establecidos en la interfaz.
6. Una vez configurada la tarjeta se solicitan datos a ésta, cada vez que un dato esté listo, se manda graficar a la pantalla y a procesar, posteriormente se guarda en el disco.
7. El procedimiento lo repetimos con cada dato, teniendo en cuenta que después de cada ciclo, xquetzal tiene que verificar si se debe detener o continuar con la captura.

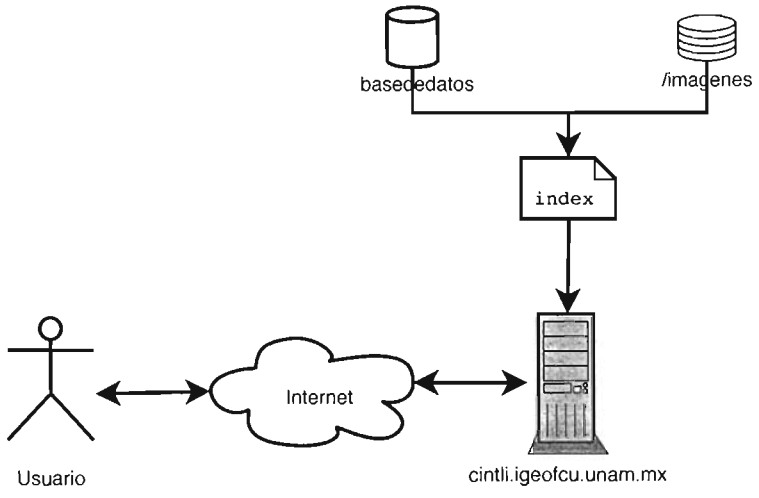

Figura 2.2: Modelo Lógico para la Publicación Web.

8. Si se continua se repite el ciclo.

9. Si se detiene, avisa a la tarjeta que hay que detenerse y regresa a la pantalla de configuración.

10. Si se desea terminar, en la pantalla de configuración puede salir. Esta orden hace que xquetzal guarde su configuración y termine correctamente.

En el modulo de procesamiento de datos es en donde la señal puede ser monitoreada para los efectos del proyecto. Aquí pueden incluirse los módulos de aviso por evento solar, calculo de media o alguna transformación que se le quiera dar a la señal.

Para el caso de publicación en tiempo real la figura 2.2 explica gráficamente como funciona:

1. Un usuario accede por medio de Internet a la URL http://cintli.igeofcu.unam.mx.

2. Un servidor web responde con una pagina html en la cual mediante algún CGI[1] , da servicios de búsqueda en nuestra base de datos y también da acceso a imágenes y archivos que se estén generando en ese momento.

Estas dos aplicaciones trabajando sincronizadamente producen los resultados deseados. Es decir, uno de ellos captura y despliega datos en tiempo real, produciendo los archivos y las imágenes necesarias. Mientras que el otro atiende peticiones de usuarios externos y da acceso a imágenes y archivos.

[1] Common Gateway Interface (Pasarela de Interfaz Común), es una tecnología de Internet que permite a un cliente (explorador web) solicitar datos de un programa ejecutado en un servidor web para transferir datos entre el cliente y el programa [53].

2.3.2. Identificación de los requisitos de software

Con el modelo lógico podemos identificar los Requisitos del Software:

1. El proyecto Xquetzal esta constituido por dos programas interdependientes.

   Uno de ellos es nombrado xquetzal-caption, el cual tiene la responsabilidad de configurar la tarjeta; capturar, desplegar, guardar y procesar la información obtenida, atender los cambios de configuración y reflejarlos en los archivos.

   El otro proyecto es xquetzal-publish, que consiste en el diseño de un sitio web mediante algún CGI que sea capaz de dar servicios web a bases de datos y tendrá como responsabilidad atender peticiones http para búsqueda y acceso a la base de datos interna.

2. Xquetzal-caption debe estar construido bajo la premisa de eficiencia ya que los tiempos de respuesta esperados son mínimos (en milisegundos) y debe tener un sistema robusto para tratamiento de errores, además debe ser extremadamente confiable en termino de frecuencia de captura y representación de datos de acuerdo a los datos verídicos.

3. Debe tener una interfaz amigable e intuitiva, haciéndolo parecer lo mas posible a la manera en que trabajan en el laboratorio, es decir, tener la capacidad de decidir si una alteración en la gráfica se debe a un evento solar o algún desperfecto o calibración del equipo.

4. Xquetzal-caption debe estar construido de forma modular, para que cualquier parte pueda ser reescrita sin modificar a las demás, siendo el de mayor prioridad el modulo para acceder a la tarjeta decodificadora.

5. Xquetzal-caption debe tener la capacidad de manejar eventos y señales, pues la orden de salir puede darse a través de eventos o el uso de memoria compartida.

6. Por otra parte xquetzal-publish debe tener una presentación sencilla con controles fáciles de usar y que sea totalmente visible en cualquier navegador Web.

7. El modelo debe estar construido con licencia GPL para satisfacer los Requisitos del Usuario definidos en la sección 2.2.2.

Debido a los puntos anteriores, el ambiente de desarrollo elegido es GNU/Linux con la distribución Debían versión potatoe. Este ambiente de desarrollo fue elegido de acuerdo a los siguientes criterios:

1. El personal del laboratorio ya tiene experiencia con GNU/Linux.

2. En el laboratorio se encuentra instalado el Sistema Operativo GNU/Linux Debían Potatoe.

3. Experiencia de programación en entornos GNU/Linux.

4. Satisface las necesidades del usuario al tener una licencia tipo Open Source.

2.3.3. Desarrollos anteriores

Anteriormente este proyecto fue desarrollado bajo ambiente Microsoft Windows 95 y con la herramienta de desarrollo LabView [7], el cual es un lenguaje visual que tiene integradas las funciones esenciales de la tarjeta LabPC, ya que ambos (Software y Hardware) fueron desarrollados por la misma empresa (National Instruments). En este punto nos dimos cuenta de una cosa, cuando una empresa vende la tarjeta entregan un paquete (El hardware), el cual puede ser instalado en la PC, pero a la hora de trabajar es necesario tener el control de la tarjeta, y hasta ahora solo había una posibilidad y era comprar el LabView y comenzar a programar en él. Las grandes desventajas del LabView como lo son todos los lenguajes visuales, es su falta de control. Al tratar de modificar el comportamíento de un módulo se vuelve muy difícil, además de que su desempeño no es el que nosotros esperariamos. Esto se reflejó claramente en el proyecto anterior,

Otro punto importante es el costo de las herramientas de Desarrollo, pongamos unos datos:

1. Licencia de Windows 2000 Server ($ 1,199 US [30]).
2. Licencia del LabView ($ 4,370 US [18]).
3. Licencia de SQL server 2000 ($ 4,999 US [31]).
4. Licencia de Visual Estudio ($ 799.0 US [32]).

Estamos hablando de $ 11,367 US.

Por otro lado, pudimos analizar la manera en que trabajaba el proyecto anterior, el cual dejaba mucho que desear, su principal problema era que se iba atrasando a través del tiempo y para cuando pasaban algunas horas, el sistema entero estaba casi pasmado. Mediante un estudio detallado de este defecto, se encontró que la causa de este desperfecto se debe a 2 registros en la Tarjeta LabPC+: Overflow y UnderFlow [6], estos registros son revisados antes de recuperar el dato de la conversión. Si alguno de estos registros esta activado, quiere decir que la tarjeta a producido un error por falta de atención ya que la cola de datos se a desbordado y es posible que algunos datos se hayan perdido. Lo que hace LabView en estos casos, es tirar toda la secuencia de datos (limpiar la cola) y volver a iniciar la tarjeta. Con el tiempo esto se hace tan común que el sistema completo tira una gran cantidad de datos.

Analizando estos problemas la otra alternativa era utilizar un sistema UNIX, como es el caso de Linux. Linux es una implementación OpenSource de un sistema tipo UNIX que fue liberado por SUN en la década de los 80s, su crecimiento a sido exponencial y hay una gran cantidad de personas trabajando en este proyecto. Para cuando se inicio xquetzal Linux era ya bastante robusto y contaba con un gran cantidad de programas OpenSource y bibliotecas de los lenguajes más populares.

2.3.4. Herramientas de Desarrollo

Cuando realizamos un desarrollo en ambiente gráfico (en este caso X), nos encontramos con una gran cantidad de herramientas como Xlib [46], XForms[55], OpenGL[19], GTK[17][39], AWT[16][33], Swing[11], OpenDX[37], etc. Algunas son bibliotecas de un lenguaje y en otros casos son aplicaciones que implementan su propio lenguaje. Decidí usar las siguientes herramientas y expongo los motivos:

1. Lenguaje de Programación: C.
   a) Experiencia para programar en C.

b) Existen programas escritos para comunicarse con la tarjeta LabPC+.

c) Facilidad para comunicarse con el Hardware a bajo nivel.

d) Por ser el lenguaje base de GNU/Linux.

2. Bibliotecas de Desarrollo Gráfico: GTK
GTK+[2] es un grupo importante de bibliotecas o rutinas para desarrollar interfaces gráficas de usuario (GUI) principalmente los entornos gráficos GNOME[3] , del sistemas Linux.

GTK+ se basa en bibliotecas del equipo de GTK+:

a) GLib es una biblioteca de bajo nivel estructura básica de GTK+ y GNOME. Proporciona manejo de estructura de datos para C, portabilidad, interfaces para funcionalidades de tiempo de ejecución (runtime) como ciclos, hilos, carga dinámica o un sistema de objetos.

b) Pango es una biblioteca para el diseño y renderizado de texto, hace hincapié especialmente en la internacionalización. Es el núcleo para manejar las fuentes y el texto de GTK+ 2.

c) ATK es una biblioteca para crear interfaces con características de una gran accesibilidad muy importante para personas discapacitadas o minusválidos. Pueden usarse útilerias como lupas de aumento, lectores de pantalla, o entradas de datos alternativas al clásico teclado o ratón.

d) GDK es básicamente una envoltura alrededor de las funciones de Xlib. Dibuja puntos, lineas, colores en widgets del tipo de GTK[14]. También proporciona rutinas para manejar: listas doblemente enlazadas, listas con enlace sencillo, cronómetros, manipulación de cadenas, un analizador léxico y funciones de errores.[8].

GTK cuenta con documentación incompleta [49] y con manuales en linea también incompletos [50], aun así hay muchos ejemplos y una gran cantidad de foros en Internet.

3. Diseño de Interfaz Gráfico: Glade.
Glade es una herramienta de desarrollo visual de aplicaciones mediante GTK/GNOME. Puede crear un aplicación tipo Visual Estudio de MS para poder crear interfaces de manera dinámica y transparente al programador usando GTK[5].

Soporta la mayoría de los widgets de GTK/GNOME. Genera el código de la interfase gráfica en C y en otros lenguajes [5]. Nos permite concentrarnos en nuestro programa, no en los detalles del código de las interfaces [44].

4. Diseño de Paginas Web: BlueFish.
BlueFish es un proyecto OpenSource que permite desarrollar paginas btml de una manera fácil y ordenada [48], se diferencia de otras aplicaciones por su orientación NO WYSIWYG[4] que lo hace a mi parecer más versátil y fácil de usar. Anteriormente había tenido experiencia con paquetes como Dreamweaver, el resultado de haber usado este software fue el código generado es redundante en extremo, haciendolo demasiado complejo para usos sencillos. El aumento de código hace que las paginas tarden mas

[2] GIMP toolkit (Conjunto de rutinas para GIMP). GIMP es un programa de manipulación de imágenes del proyecto GNU (GNU Image Manipulation Program).

[3] GNOME es un entorno de escritorio para sistemas operativos de tipo Unix bajo tecnología X Window.

[4] Acrónimo de What You See Is What You Get (lo que ves es lo es lo que obtienes).

en cargar y si no se tiene cuidado, cambiar el estilo a un sitio completo puede ser un trabajo agotador.

5. Lenguaje para Servicios Web: JSP.
   Con Java Server Pages (JSP), podemos crear aplicaciones web que se ejecuten en varios servidores web, de múltiples plataformas, ya que Java es en esencia un lenguaje multiplataforma. Las páginas JSP están compuestas de código HTML mezclado con etiquetas especiales para programar scripts de servidor en sintaxis Java [38]. Preferí este lenguaje por tener experiencia programando aplicaciones web con JSPs.

6. Lenguaje para Paginas Web: XHTML 1.0.
   XHTML es una familia de módulos y tipos de documentos que reproduce, engloba y extiende HTML 4.0 [51]. Los tipos de documentos de la familia XHTML están basados en XML, y diseñados fundamentalmente para trabajar en conjunto con agentes de usuario basados en XML [52].

7. Lenguaje para Base de Datos: SQL El Lenguaje de Consulta Estructurado (Structured Query Language) es un lenguaje diseñado especialmente para acceso a base de datos relacionales. Actualmente es el lenguaje para base de datos relacionales mas importante. Actualmente hay dos especificaciones para SQL: ANSI (X3.135-1992[1]) e ISO (INCITS/ISO/IEC 9075-1). Desafortunadamente estos estándares son comerciales.

8. Servidor JSP/HTML: Jakarta Tomcat 5.x
   Tomcat es una implementación de la referencia oficial para la tecnología Java Servlet y Java Server Pages[41]. La especificación de Java Servlet y JavaServer Pages son desarrolladas por SUN. Tomcat tiene una licencia del tipo Apache Software License. Tomcat es desarrollado por una gran cantidad de programadores de todo el mundo[42].

9. Servidor de Base de Datos: Postgresql.
   El Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos conocido como PostgreSQL (y brevemente llamado Postgres95) está derivado del paquete Postgres escrito en Berkeley. Con cerca de una década de desarrollo tras él, PostgreSQL es el gestor de bases de datos de código abierto más avanzado hoy en día, ofreciendo control de concurrencia multi-versión, soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones, y tipos y funciones definidas por el usuario), contando también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, perl, tcl y python)[40].

2.3.5. Resultados

Publicamos nuestra primera versión del DRS (Documento de Requisitos del Software). Este documento se encuentra al final de la Tesis con el titulo de DRS (Apéndice B) con sus respectivas modificaciones.

2.4. Diseño Arquitectónico

2.4.1. Construcción del modelo físico

La construcción del modelo físico implico úna ardua tarea de investigación y recopilación de información. Anteriormente había tenido algunas entrevistas con gente que trabaja en el

desarrollo de software para X. Una de estas personas fue Antonio Tellez, que tiene experiencia programando en C a bajo nivel y fue él quien mostró en una de sus clases, que Glade y GTK+ son herramientas de desarrollo de alto nivel.

Pero necesitaba mas información, la bibliografía es casi nula ([17]) y realmente hay muy pocos ediciones que hagan referencias a Desarrollo de aplicaciones en Linux.

La documentación de GTK está parcialmente construida [49] y la verdadera ayuda fueron las listas de correo y los tutoriales en línea [50].

Varios sitios de Internet fueron consultados para recopilar información y hacer algunas preguntas en sus foros como en la Asociación BULMA [4] y la Cofradía Digital [10].

Finalmente, pude dar una primera aproximación del diseño arquitectónico de *xquetzal-caption* no sin antes hacer una serie de pruebas que enumero a continuación. El código generado se encuentra en los apéndices.

1. Hola Mundo. Este ejemplo implementa una ventana con un botón, al apretar el botón se escribe un hola mundo en la terminal.
   Este ejemplo sirvió para familiarizarme con Glade, la forma en que se compila un programa y el funcionamiento de los signal connects (Ver apéndice G).

2. Iniciar Detener. Este ejemplo implementa una ventana con 2 botones, Un botón inicia la impresión de un numero que se va incrementando y el otro botón detiene la impresión (Ver apéndice H).
   Al correr esta programa de prueba paso algo que no esperaba. Al apretar el botón de iniciar, la aplicación dejaba de funcionar. Seguía imprimiendo los datos en la consola pero los botones se quedaban "congelados". A falta de documentación o experiencia previa sobre este problema sugerí algunas teorías: Yo suponía que al lanzar la señal de click en el botón iniciar, la función on_button1_clicked (Ver H) se apropiaba del foco de la aplicación y como esta función entraba a un ciclo que nunca terminaba (while(1)) jamas atendía a las demás señales. Entonces supuse que había una función en GTK que al invocarla, mandaba a un segundo plano la actual función, o en su defecto hubiera algún tipo de gtk_signal_connect_tread que al ejecutarse lanzara un hilo. Desafortunadamente había muy poca documentación Por ese entonces aun no tomaba materias como Sistemas Operativos o Redes. Estuve varios días tratando de resolver este problema, comenze a saber de la función fork() pero aun no entendía su funcionamiento, además de que no sabía como usarlo dentro de la aplicación y más en específico, hacerlo compatible con GTK pues en algunas listas de discusión donde se hacia alusión a los problemas que había con fork() y GTK. Decidí continuar con el desarrollo del proyecto.

2.4.2. Tarjeta LabPC+

Con la tarjeta convertora Análogo Digital LabPC [6] lo único con lo que contaba era con el manual de la tarjeta y algunos ejemplos de como iniciar la tarjeta en lenguaje C.

Para la tarjeta LabPC+ se comenzó estudiando el manual de usuario [6]. Hay dos capítulos importantes: El Capítulo 4 ("Register Map and Description") y el Capítulo 5 ("Programming"). En el primero se da una descripción detallada de los 28 registros de 8 bits con los que cuenta la tarjeta. En el segundo capítulo se dan ejemplos "hablados" o pseudocódigos sobre como se puede programar la tarjeta.

En el apéndice D se encuentra un estudio detallado de la tarjeta LabPC y en el Apéndice I se encuentra un programa que configura e imprime datos de la tarjeta LabPC+.

2.4.3. Definición de los componentes principales

Para xquetzal-caption (imagen C.1):

main.c Se encarga de llamar al constructor de la Interfaz (createWindow() en interface.h); desplegar la interfaz en pantalla (gtk_widget_show()); leer las configúraciones del disco y a partir de estas configuraciones crear una estructura lógica en la memoria; colocar las configuraciones para que cualquier función, a través de la aplicación, pueda consultar el estado actual del sistema. La última función del main es llamar a gtk_main() para atender los eventos.

interface.c Se encarga de construir la interfaz gráfica y conectar los widgets con las funciones a partir de las señales (gtk_signal_connect).

1. support.c: Se encarga de leer los paths de configuración para las imágenes que se vayan a desplegar en la aplicación.

callbacks.c Aquí están las instrucciones que debe seguir el programa una vez que hay algún evento. En general la manera en que funcionan es alterando la estructura lógica donde se encuentra la configuración actual del programa.

Objeto.h Define un elemento de un buffer.

Buffer.c En esta biblioteca se encuentra la implementación de un buffer en donde se pueden guardar los datos de una manera estructurada. El buffer esta estructurado en bloques, cada bloque contiene 2 elementos importantes: la cabecera y el área de datos. En la cabecera se encuentran una secuencia de códigos que definirán el área de datos. En esencia hay 3 tipos de códigos: 0 para datos, 1 para media de datos y 2 para cambio de estado.

Guardar.c Guarda en el Disco Duro las estructuras contenidas en un Buffer. Puede guardar los elementos de 2 maneras: De forma binaria (Exactamente como vienen en la estructura Buffer) o en formato ASCII (Convierte la forma binaria en elementos legibles para el humano).

LabPC.c Funciones de bajo nivel para comunicarse con la tarjeta LabPC+.

procesar.c Procesa la señal obtenida de la tarjeta convertora análogo digital y se la entrega a graficos.c.

graficos.c Gráfica los datos obtenidos después de procesar.c. Análogo/Digital.

Para xquetzal-publish (Figura C.2):

index.html Documento html estático para la publicación web. En esta pagina damos una introducción de lo que es el interferómetro, sus características físicas y su localización geográfica. En esta pagina se encuentra un menú con 2 opciones: Tiempo Real y Eventos Solares.

tiemporeal.html Pagina html estática en donde se muestra la imagen que se esta obteniendo en estos momentos. Esta imagen es generada por xquetzal-caption. La pagina se actualiza automáticamente cada 5 minutos.

EventosSolares.jsp Pagina dinámica que despliega un calendario en donde se muestran los días con mediciones y los días con eventos solares.

Build.jsp Pagina dinámica que muestra los datos de un día en particular.

2.4.4. Resultados

El documento generado en esta parte del proyecto se encuentra en el apéndice C.

2.5. Diseño Detallado y Producción del Código

2.5.1. Introducción

Las siguientes bibliotecas no fueron implementadas en un solo ciclo de desarrollo, algunas tomaron varios años en volverse estables. Cada biblioteca esta pensada de manera modular y por si solas son proyectos importantes. A continuación damos una explicación de las funciones mas importantes y representativas de cada módulo, una breve historia de su construcción y los problemas que se encontraron así como su solución e implementación. Todo el código generado se encuentra en el apéndice J.

2.5.2. LabPC.c

El módulo LabPC puede utilizarse para construir aplicaciones rápidamente, ya que abstraen las partes mas importantes del funcionamiento de la tarjeta. La biblioteca de interfaz para la tarjeta convertora Análogo Digital fue la primera en diseñarse. Esta sirve de interfaz entre la aplicación y el hardware. Hay 3 funciones de especial interés:

1. void iniciaTarjeta(void): Ejecuta la secuencia de pasos descritos en D para iniciar la tarjeta.

2. int ADFIFOregister(void): Regresa el Dato que convirtió la tarjeta Análogo Digital en Complemento A2. Este método tiene una larga historia, comenzó como una lectura sucesiva del ADFIFOregister (Ver D) con un corrimiento y una unión binaria. Desafortunadamente esto no funcionaba, ya que algunos datos eran convertidos de manera equivocada. Así que tuve que analizar mas detenidamente los datos obtenidos.

   Una primera aproximación fue detectar cuando un elemento de la conversión era negativo. Como sabemos, para obtener un dato en la tarjeta LabPC+, tenemos que leer 2 veces el ADFIFO, la primera lectura nos da los primeros 8 bits (Byte Bajo), una segunda lectura nos da los siguientes 8 bits (Byte Alto), de los cuales solo necesitamos los últimos 4, teniendo en cuenta el modelo del Bit mas significativo a la izquierda. La segunda lectura es la importante. Según el modelo de representación de datos por complemento A2, lo primero que tenemos que especificar es el conjunto de datos de representación, en este caso son 12 bits. Si el primer Bit es 0 entonces el numero sera positivo:

   $$(0100\ 00000001)_2 = 2049_{10}$$

   Si por el contrario, el primer bit de nuestro conjunto es 1, entonces el numero es negativo:

   $$(1011\ 11111111)_2 = (0100\ 00000000)_2 + 1_2 = (0100\ 00000001)_2 = -2049_{10}$$

   La tarjeta LabPC+ encapsula este procedimiento de entrega de datos, realmente no se tiene que hacer la conversión, el punto aquí es tener en cuenta la representación de enteros en C.

   C representa un entero generalmente con 4 bytes, tenemos 12 bits de la conversión, pero nos hace falta "rellenar" los otros 20 bits, es ahí cuando se hace necesario saber si el

numero en cuestión es positivo o negativo. Con solo observar el bit numero 12 de nuestro dato podemos saber si los primeros 20 bits estarán en 1 o en 0. Si el numero es negativo (El Bit doce es 1) entonces los primeros 20 bits del entero serán 1, en otro caso son 0.

Al hacer este análisis, los datos obtenidos mejoraron, pero ahora parecía haber un overflow. Después de cierto límite (Aplicando un voltaje negativo muy grande) los datos comenzaban a aparecer positivos. Después de un análisis, el error fue encontrado en el momento de la lectura del Byte menos significativo. El problema: Una Mascara que no se hacia.

Hay una gran complejidad a la hora de manejar bits en el código, ya que para traerlos, la estructura mínima en C son los Char, pero con estos no se puede hacer corrimientos ni comparaciones binarias, es por eso que se usan los Enteros (int), pero al hacer una asignación del tipo int = char, C auto completa los valores a complemento a2, es decir, si el bit mas significativo de la variable tipo char es 1, entonces C "completa" sus bits a 1. Ejemplo:

$$x = c$$

Donde x es declarada int y c es char. En asignación binaria tendríamos:

$$x = (10000010)_2$$

Pero x tendría el siguiente valor:

$$(11111111\ 11111111\ 11111111\ 10000010)_2$$

Es por eso que se hace necesario un cast con una Mascara. Esta Mascara es una variable entera del tipo:

$$MASK = (00000000\ 00000000\ 00000000\ 11111111)_2$$

Para que al hacer un & el dato no se corrompa:

$$x = c\&MASK$$

Esto deja a x "limpio" para continuar las operaciones.

El siguiente paso es atender los posibles errores que pudiera haber en la captura, así que antes de que leamos el ADFIFO tenemos que revisar los registros de posibles errores. Solo tenemos que checar los bits 3 y 2 del Status Register. Si alguno de ellos esta encendido (1), avisamos con un código especial: 5000 Si es overflow y -5000 si es underflow, estos datos jamas son alcanzados por la tarjeta en forma bipolar.

Las pruebas realizadas con sleeps y sobre lectura arrojaron resultados exitosos y las pruebas realizadas con simuladores de señales tambien obtuvieron resultados exitosos.

3. void datoListo(): Esta función es un tope. Mientras no exista un dato, hay que esperar, el bucle termina cuando se reporta que hay un dato. Esta función fue parte medular de diseño del software, ya que para cualquier modelo esta función tendría que mandarse a ejecutar en un segundo plano.

2.5.3. Buffer.c

La biblioteca Buffer esta pensada para guardar cronológicamente todos los eventos que se vayan suscitando y fue implementada de acuerdo a la establecido en C.3.4.

El buffer fue diseñado como un objeto. Para crear un buffer mandamos llamar a la función *nuevoBuffer* indicándole el tamaño del buffer y regresa una estructura tipo Buffer, la cual esta especificada en Objeto.h, con esta estructura-objeto podemos insertar datos, verificar si esta lleno, borrarla o simplemente pasar esta estructura a otra biblioteca para que pueda manejarla.

La biblioteca Buffer tiene 2 formas de agregar datos a su estructura, de manera estructurada y de manera secuencial. La manera estructurada va indicándole al Buffer que tipo de datos son, con que valores, le pasa la hora, etc. La manera secuencial simplemente va insertando los bytes que le vayan mandando. La forma en que se asigna fue pensado en que esta estructura puede ser guardada de manera binaria, por lo tanto al tratar de recuperarse se necesita ir leyendo secuencialmente los bytes del disco. Así tenemos una manera fácil de recuperar el estado de la captura.

2.5.4. Guardar.c

Guardar.c está diseñada para trabajar con una estructura tipo Buffer. La idea es que si tenemos una estructura, la podamos guardar en el formato que nosotros escojamos. En un principio solo guardaba en forma Binaria, es decir, guardar el Buffer tal y como está. Pero una segunda implementación permitió guardar los datos en formato ASCII. Para un trabajo posterior se pretende guardar los datos en formato FITS. Esta biblioteca lo que hace es recibir la estructura, decodificarla y guardarla de acuerdo a los parámetros que se le indiquen (Binario o ASCII).

2.5.5. interface.c

interface.c fue generada automáticamente por Glade[5], con pequeñas modificaciones hechas a mano. Las principales modificaciones fueron implementar la lectura de configuraciones para poder restablecer el estado anterior del sistema.

2.5.6. main.c

main.c también fue creado automáticamente por Glade.

2.5.7. callbacks.c

Las cabeceras de callbacks.c fueron generadas automáticamente por Glade. Esta clase tiene las funciones que se activan con los diferentes eventos que se forman. La idea principal es que tenemos un arreglo de enteros que sirve para establecer el estado del sistema (tabla C.1), cada vez que se presiona un botón, checkbox o radiobutton, el estado del arreglo también se modifica. El botón de inicio de captura lee el estado del sistema mediante el arreglo y se configura para desplegar los datos que así se han seleccionado, pasándole el control de la graficación a Graficos.c, la forma en que le pasa el control es llamando una función *graficador* de esta biblioteca con el DrawingArea a dibujar. Esta biblioteca es la que contiene los botones de inicio y paro de la captura. La forma es que funciona es con el mismo arreglo de enteros. Cuando la captura inicia el índice 1 del arreglo se pone en 1, cuando se detiene, se pone en 0.

2.5.8. Graficos.c

Se encarga de graficar los datos que vienen de la tarjeta LabPC+. Esta biblioteca establece comunicación con la tarjeta, la configura (mediante las interfaces), y comienza la captura. Mientras el índice 1 de la memoria (arreglo de enteros) este en 1, sigue capturando; cuando este en 0, se detiene.

La memoria en todo momento es accedida mediante la declaración *extern* para todas las bibliotecas.

La técnica que se usa para la graficación es la de doble buffer: Pintar en segundo plano, una vez que se haya terminado se hace visible (Haciendo una copia al primer plano), esto reduce el parpadeo de la pantalla.

2.6. Conclusiones

Cuando se termino de programar estas clases y se comenzaron a hacer pruebas ocurrió el problema que habíamos detectado hacia tiempo: Al comenzar la captura no había modo de detenerla. Abstrayendo la manera en que funciona el algoritmo de captura de datos tenemos lo siguiente:

```
    while (Memoria[1]== 1)
CAPTURA;
```

Por otro lado tenemos los botones de inicia/para la captura. Estos botones lo único que hacen es Memoria[1]=1; cuando inicia y Memoria[1]=0; cuando la detienen. Los botones y el while están en dos bibliotecas diferentes, pero se comunican entre ellas mediante la sentencia extern gint *Memoria;

Llegue a una primera conclusión: Cuando GTK ejecuta una función, no genera un tread, al contrario, ejecuta la función y espera a que termine el proceso, una vez terminado, regresa el control al programa principal y continua con su ejecución.

Esperaba encontrar una forma de invocar un proceso en GTK pero en segundo plano. Cuando conecto un evento con una función utilizo *gtk_signal_connect*, yo esperaba encontrar un *gtk_signal_connect_thread* que lanzara el proceso en segundo plano, esta función no existe. Al momento de estar construyendo la aplicación no había mucha documentación sobre este caso. Al parecer este problema era común, ya que en las listas aparecieron soluciones a este problema, la respuesta era poner en el código las siguientes lineas, antes de la función que se apropia de los recursos:

```
while (gtk_events_pending())
gtk_main_iteration_do(0);
```

El problema real es que el proceso de pintado es una función perezosa de alta prioridad, así que cualquier otro evento entra a una cola de espera, mientras el pintado no se termine los demás eventos no serán atendidos. Así que las lineas anteriores lo único que hacen es vaciar la cola de espera, ejecutando todos los procesos que no sean de prioridad, cuando termina le vuelve a pasar el control al hilo principal de GTK.

Otro problema importante era la cantidad de datos que podíamos procesar en nuestra PC. Como habíamos pronosticado, la captura de datos se veía afectada por la velocidad del equipo. Haciendo una serie de pruebas los datos desplegados en la pantalla sufrían un retraso de 3 a

5 segundos. Cuando cambiábamos la intensidad de voltaje, la gráfica tardaba en mostrar los cambios. Otro problema serio que había era que cuando queríamos hacer una captura a muy alta frecuencia de muestreo, llegaba un momento en donde la gráfica arrojaba datos erróneos, posiblemente causados por errores de overflow.

Si continuábamos desarrollando el código, el problema persistiría, pues cada vez se cargaban mas las funciones de graficación, causando mas problemas de desfasamiento.

Así llegamos a un acuerdo con el Dr. Alejandro Lara, haciéndole notar la necesidad de comprar un equipo mas poderoso, con la posibilidad de hacer una implementación mas ambiciosa y con mejor calidad.

Cerramos entonces este ciclo de desarrollo con una serie de experiencias y software reutilizable. Tenemos las bibliotecas principales que son las de almacenamiento, comunicación con la tarjeta LabPC+ y las funciones de graficación. Estas bibliotecas serán reutilizadas en el siguiente proyecto. Y aun mas, tenemos una clara idea de lo que necesitamos y queremos, para la siguiente fase de desarrollo utilizaremos otro esquema, el XProgramming.

Capítulo 3

XP programming

3.1. Introducción

En este capítulo retomamos toda la información contenida en los documentos: DRU (Apéndice A), DRS (Apéndice B) y los módulos programados. Con esta información damos inicio a una nueva fase de desarrollo con un nuevo esquema llamado eXtreme Programming.

Este término es la forma de designar un nuevo ciclo de vida que es diferente al presentado en el capítulo anterior. En las siguientes secciones describiremos que es el eXtreme Programming (también se le conoce como XP programmig), posteriormente se encuentran los 14 ciclos de desarrollo generados por XP para el proyecto Xquetzal/Xitris[1] .

Encontraremos que el desarrollo en XP es más dinámico, produce resultados que inducen a seguir trabajando arduamente y son más fáciles de manejar para equipos de desarrollo pequeños.

3.1.1. La programación XP

Podríamos decir que XP nace oficialmente hace cinco años en un proyecto desarrollado por Kent Beck en Daimler Chrysler, después de haber trabajado varios años con Ward Cunningham en busca de una nueva aproximación al problema del desarrollo de software que hiciera las cosas más simples de lo que nos tenían acostumbrados los métodos existentes [2].

Esta nueva metodología tiene como principales características:

1. El código será revisado continuamente, mediante la programación en parejas (dos personas por máquina).
2. Se harán pruebas todo el tiempo, no sólo de cada nueva clase (pruebas unitarias) sino que también los clientes comprobarán que el proyecto va satisfaciendo los requisitos (pruebas funcionales).
3. Las pruebas de integración se efectuarán siempre, antes de añadir cualquier clase nueva al proyecto, o después de modificar cualquiera existente (integración continua).
4. Se (re)diseñará todo el tiempo (refactoring), dejando el código siempre en el estado más simple posible.

[1] El proyecto xquetzal es renombrado como Xitris en la sección 3.16.1

5. Las iteraciones serán radicalmente más cortas de lo que es usual en otros métodos, de manera que nos podamos beneficiar de la retroalimentación tan a menudo como sea posible.

3.1.2. Las 4 variables

XP define cuatro variables para cualquier proyecto de software: coste, tiempo, calidad y alcance. Además, especifica que, de estas cuatro variables, sólo tres de ellas podrán ser fijadas por las fuerzas externas al proyecto (clientes y jefes de proyecto), mientras que el valor de la variable libre será establecido por el equipo de desarrollo en función de los valores de las otras tres. Lo novedoso es que normalmente los clientes y jefes de proyecto se creen capaces de fijar de antemano el valor de todas las variables.

Frecuentemente, aumentar la calidad conduce a que el proyecto pueda realizarse en menos tiempo. En efecto, en cuanto el equipo de desarrollo se habitúe a realizar pruebas intensivas (pues es el pilar básico de XP) y se sigan estándares de codificación, poco a poco comenzará a avanzar mucho más rápido de lo que lo hacía antes, mientras la calidad del proyecto se mantiene asegurada (por las pruebas) al 100 %, lo que conlleva mayor confianza en el código y, por tanto, mayor facilidad para adaptarse al cambio, sin estrés, lo que hace que se programe más rápido ... y así sucesivamente.

En cuanto al alcance del proyecto, es una buena idea dejar que sea ésta la variable libre, de manera que, una vez fijadas las otras tres, el equipo de desarrollo determinaría el alcance mediante:

1. La estimación de las tareas a realizar para satisfacer los requisitos del cliente.

2. La implementación de los requisitos más importantes primero, de manera que el proyecto tenga en cada instante tanta funcionalidad como sea posible.

3.1.3. El coste del cambio

La idea fundamental aquí es que, en vez de diseñar para el cambio, diseñaremos tan sencillo como sea posible, para hacer sólo lo que sea imprescindible en un momento dado, pues la propia simplicidad del código, junto con nuestros conocimientos de factorización [13] y, sobre todo, las pruebas y la integración continua, hacen posible que los cambios puedan ser llevados a cabo tan a menudo como sea necesario.

3.1.4. Las prácticas

¿En qué consiste realmente XP? ¿Cuáles son esas prácticas a las que ya se ha hecho mención y que hacen posible ese cambio de mentalidad a la hora de desarrollar software? Estas se enumeran a continuación:

3.1.5. La planificación

XP plantea la planificación como un permanente diálogo entre la parte empresarial y técnica del proyecto, en la que los primeros decidirán el alcance (¿qué es lo realmente necesario del proyecto?), la prioridad (qué debe ser hecho en primer lugar), la composición de las versiones (qué debería incluir cada una de ellas) y la fecha de las mismas. En cuanto a los técnicos, son los responsables de estimar la duración requerida para implementar las funcionalidades deseadas por el cliente, de informar sobre las consecuencias de determinadas decisiones, de

organizar la cultura de trabajo y, finalmente, de realizar la planificación detallada dentro de cada versión.

3.1.6. Versiones pequeñas

El sistema se pone por primera vez en producción en, a lo sumo, unos pocos meses, antes de estar completamente terminado. Las sucesivas versiones serán más frecuentes (entre un día y un mes). El cliente y el equipo de desarrollo se beneficiarán de la retroalimentación que produce un sistema funcionando, y esto se reflejará en las versiones sucesivas.

3.1.7. Diseño simple

En vez de perseguir a toda costa un diseño en el que hemos de desarrollar dotes adivinatorias para tratar de anticiparnos al futuro, lo que XP preconiza es diseñar en cada momento para las necesidades presentes.

3.1.8. Pruebas

Cualquier característica de un programa para la que no haya un test automatizado, simplemente no existe. Y es que éste es sin duda el pilar básico sobre el que se sustenta XP. Otros principios son susceptibles de ser adaptados a las características del proyecto, de la organización, del equipo de desarrollo. Pero aquí no hay discusión posible: si no hacemos tests, no estaremos haciendo XP.

3.1.9. Refactorizar

Responde al principio de simplicidad, y muy escuetamente, consiste en dejar el código existente en el estado más simple posible, de manera que no pierda (ni gane) funcionalidad y que se sigan ejecutando correctamente todas las pruebas. Esto nos hará sentirnos más cómodos con el código ya escrito y, por tanto, menos reacios a modificarlo cuando haya que añadir o cambiar alguna característica. En el caso de sistemas heredados, o de proyectos que se tomen ya empezados, seguramente habrá que dedicar varias semanas sólo a factorizar el código.

3.1.10. Programación en Parejas (Pair programming)

Todo el código será desarrollado en parejas, dos personas compartiendo un solo monitor y teclado. Quien codifica estará pensando en el mejor modo de implementar un determinado método, mientras que su compañero lo hará de una manera más estratégica:

1. ¿Estamos siguiendo el enfoque apropiado?
2. ¿Qué es lo que podría fallar aquí? ¿Qué deberíamos comprobar en las pruebas?
3. ¿Hay alguna manera de simplificar el sistema?

Por supuesto, los roles son intercambiables, de manera que en cualquier momento quien observaba puede tomar el teclado para ejemplificar alguna idea o, simplemente, para turnar a su compañero. Igualmente, la composición de las parejas cambiará siempre que uno de los dos sea requerido por algún otro miembro del equipo para que le ayude con su código.

3.1.11. Propiedad colectiva del código

Cualquiera puede modificar cualquier porción del código, en cualquier momento. En efecto, cualquiera que vea una posibilidad de simplificar, mediante refactoring, cualquier clase o cualquier método, hayan sido o no escritos por él, deberá hacerlo sin más dilación. El uso de estándares de codificación y la confianza que nos dan los tests de que todo va a seguir funcionando bien tras una modificación, hacen que esto no sea ningún problema en XP.

3.1.12. Integración continua

Cada pocas horas, o al cabo de un día de programación, como mucho, se integra el sistema completo Para ello existirá un máquina así llamada, de integración, a la que se acercará una pareja de programadores cada vez que tengan una clase que haya sido probada unitariamente. Si al añadir la nueva clase junto con sus tests unitarios, el sistema completo sigue funcionando correctamente (pasa todos los tests), los programadores darán por finalizada esa tarea. Si no, serán los responsables de dejar el sistema de nuevo con los tests funcionando al 100 %. Si después de un cierto tiempo no son capaces de descubrir qué es lo que falla, tirarán el código a la basura y volverán a comenzar de nuevo.

3.1.13. 40 horas semanales

Si realmente queremos ofrecer calidad, y no meramente un sistema que funcione (lo cual, en computación, ya sabemos que es trivial) queremos que cada miembro de nuestro equipo se levante cada mañana descansado y se vaya a las 6 de la tarde a su casa cansado y con la satisfacción del trabajo bien hecho, y que al llegar el viernes sepa que cuenta con dos días para descansar y dedicarse a cosas que nada tengan que ver con el trabajo. Naturalmente, las 40 horas no es una regla fija (pueden ir de 35 a 45), pero lo que es seguro es que nadie es capaz de trabajar 60 horas a la semana y hacerlo con calidad.

3.1.14. Cliente en el sitio

Otra controvertida regla de XP: al menos un cliente real debe estar permanentemente junto al equipo de desarrollo, para responder cualquier consulta que los programadores le planteen, para establecer prioridades. Si el cliente arguye que su tiempo es demasiado valioso, deberemos entender que realmente el proyecto que nos ha encomendado es lo suficientemente trivial como para que no merezca su atención, y que no importa que esté construido a partir de suposiciones hechas por programadores que nada, o muy poco, saben del negocio real.

3.1.15. Estándares de codificación

Es decisiva para poder plantear con éxito la propiedad colectiva del código. Esta sería impensable sin una codificación basada en estándares que haga que todo el mundo se sienta cómodo con el código escrito por cualquier otro miembro del equipo.

3.1.16. Planificación

XP no es sólo un método centrado en el código, sino que sobre todo es un método de gestión de proyectos software y la planificación es una parte fundamental de XP. El proceso de desarrollo de software es un proceso caótico, XP busca manejar esta complejidad aceptándola

y sabiéndola manejar sin caer en las manos de los métodos pesados y burocráticos. XP es un método ligero, adaptativo mas que predictivo.

3.1.17. El ciclo de vida

Si, como se ha demostrado, los largos ciclos de desarrollo de los métodos tradicionales son incapaces de adaptarse al cambio, tal vez lo que haya que hacer sea ciclos de desarrollo más cortos. Esta es una de las ideas centrales de XP. Los ciclos son: Análisis, Diseño, Implementación y Pruebas. Cada ciclo es muy corto, en esencia para dejar trabajando al proyecto con un paso mas adelante. La principal característica de estos ciclos de vida es fijar el alcance inmediato, una vez fijado este alcance en la fase de Análisis, se pasa al diseño coordinado, siempre pensando en el siguiente ciclo de desarrollo. Se podría hacer una analogía con la construcción de un Puente que vaya a cruzar una basta cantidad de agua. El Puente debe de tener un alcance global, en este caso XP lo define como una variable (alcance) y es mas o menos pensar de donde partimos y a donde queremos llegar, la siguiente son los alcances inmediatos en cada ciclo de desarrollo y es como pensar que una vez proyectado un punto de arribo comenzamos a construir el primer peldaño dejando siempre el camino abierto para construir el siguiente ciclo. Una vez construido el diseño conceptual de un ciclo viene la codificación, y finalmente las pruebas.

3.1.18. Conclusiones

Como pudimos ver XP es una nueva manera de desarrollar software de forma mas adaptativa, reduciendo los riesgos a los cambios que puede sufrir el proyecto debido a adaptaciones a nuevos modelos o nuevos requerimientos. Uno de las principales modificaciones que se le hizo al modelo de desarrollo de Extreme Programming y que es importante mencionar, es la programación en parejas. Debido a las características de este trabajo, toda la programación la hice yo solo, tomando en cuenta siempre la opinión del Dr. Alejandro Lara y la experiencia en los anteriores proyectos la programación en parejas fue suplantada por la revisión detallada del código y el fuerte énfasis en las pruebas. Las cuatro variables que pide Extreme Programming las tenemos bien identificadas, debido al trabajo de investigación realizado en la primera parte de este trabajo, quedando cada una de ellas como se muestra a continuación:

1. Coste: El costo del proyecto tiene que ser el menor posible, por esas razones solo contamos con software libre para poder ahorrar al máximo en recursos. En el laboratorio se hizo una inversión para comprar una computadora extra y esta sera toda la inversión.

2. Tiempo: El tiempo que se tomo para este proyecto es a mas tardar 1 año, trabajando de 4 a 5 horas diarias en el laboratorio de Interferometría Solar de lunes a jueves, se tiene proyectado tener el programa funcionando a mas tardar 6 meses después de comenzada esta etapa y posteriormente se ira adaptando a las necesidades del investigador y el laboratorista.

3. Calidad: Se pretende que la calidad no se vea mermada por las cuestiones de tiempo y coste, teniendo como base las pruebas de testing y el visto bueno del investigador.

4. Alcance: El alcance se deja como variable libre siempre y cuando respetemos una cota inferior, el alcance se deja como en el documento enunciado en el Apéndice A bajo el titulo de Especificación de Requisitos A.3.

Enunciadas las 4 variables nos enfocamos al Desarrollo del Software, pero antes de eso, hacemos una recapitulación de la infraestructura del Laboratorio de Interferometria Solar, ya que para principios del 2004 el laboratorio adquirió nuevo equipo, además de que personalmente cuento con mas herramientas técnicas y mas experiencia profesional.

Para finales del 2003, hice una estancia corta en el Goddard Space Flight Center, donde conocí el CDAW Data Center, este grupo de trabajo utiliza un catalogo llamado SOHO LASCO CME Catalog [54].

Este catalogo tiene características similares a nuestro sistema, la diferencia es que sus observaciones las toman del satélite espacial SOHO y los datos que recolecta son imágenes en 2 dimensiones y en varias longitudes de onda. El trabajo de catalogar esas imágenes como posibles eventos solares corren a cargo de Seiji Yashiro. La forma en que se catalogan las imágenes aun se hace de manera manual, hay esfuerzos considerables que buscan automatizar este proceso. La base de eventos solares se encuentra disponible en Internet, pero no en tiempo real.

Con esta experiencia, con el nuevo equipo y con las materias cursadas (Sistemas Operativos, Análisis de Algoritmos II, Procesos Paralelos, Supercomputo, Principios de Computo Distribuido y Redes), pude dar un nuevo enfoque al proyecto, mas ambicioso que en un principio y teniendo una idea mas clara del problema a resolver.

3.2. Infraestructura del Laboratorio de Interferometria Solar

Para principios del 2004, el Laboratorio adquirió 2 nuevas computadoras, un switch y la infraestructura para construir una red local.

Con este material se construyo con ayuda de Carlos Sierra, estudiante de Ciencias de la Computación y becario del Instituto, una red interna con una puerta de enlace hacia el Instituto de Geofísica (Figura 3.1), administrando mejor los recursos y la seguridad del laboratorio. Las máquinas fueron rebautizadas como se muestra en la figura 3.2 teniendo en cuenta nombres que hicieran alusión al lenguaje Mexica, como quetzal, tlaplalli, tlaloc y cintli. Tenemos en el laboratorio la siguiente tabla DNS y su principal carga de trabajo:

1. cintli: IP /dev/eth0: 132.248.6.147, /dev/eth1: 192.168.1.2, Servidor WEB, puerta de enlace y FireWall.
2. tlalpalli: IP /dev/eth0: 192.168.1.1, Servidor de Impresión.
3. tlaloc: IP /dev/eth0: 192.168.1.3, Cliente xquetzal.
4. quetzal: IP /dev/eth0: 192.168.1.4, Servidor xquetzal, interfaz con la tarjeta LabPC+.

Cintli fue escogida como servidor web y es nuestra puerta de enlace a Internet, configuramos a Cintli mediante NAT, el archivo de configuración lo puede observar en el apéndice E.

Quetzal es nuestra conexión con el RIS. Escogimos a quetzal porque tiene una slot ISA.

Con estas tres máquinas se pudo distribuir la carga.

En el transcurso del trabajo, explicaré qué hace cada programa y cómo llegamos a su concepción y su construcción.

El código de xquetzal esta presentado en su versión final al concluir los ciclos de desarrollo y no en cada uno.

El código generado en cada ciclo se puede encontrar en:

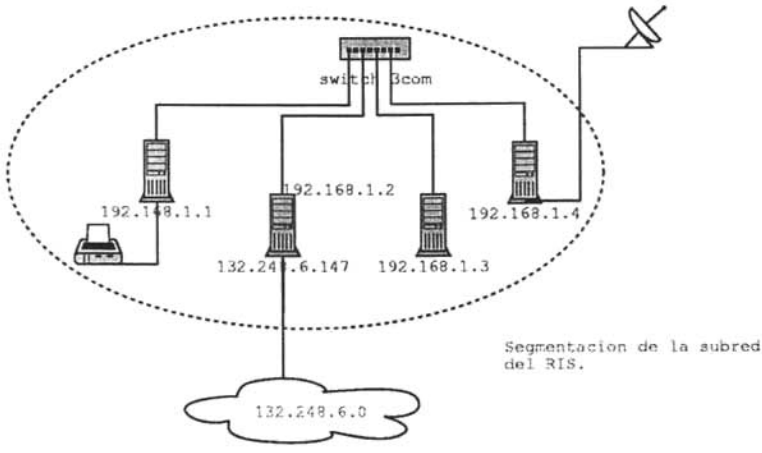

Figura 3.1: Topología de Red del laboratorio de Interferometria Solar

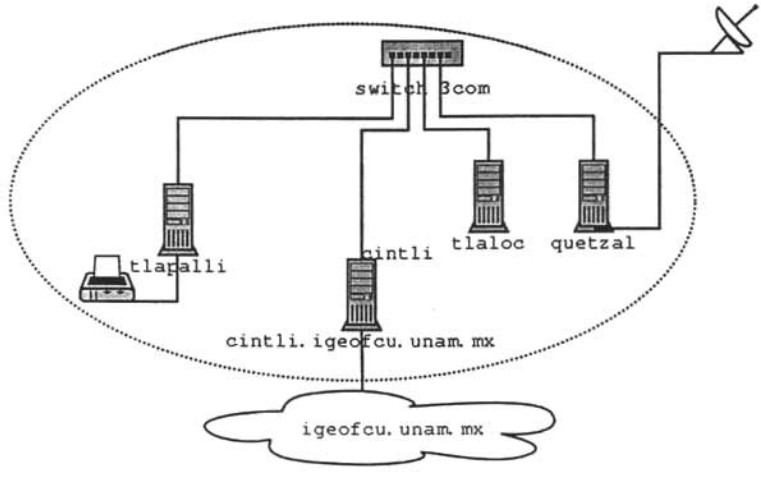

Figura 3.2: Nombres de las máquinas del laboratorio de Interferometria Solar

```
Inicio;
PublicaPuerto();
While(true){
  EsperaPeticion();
  Si (Peticion()){
    EstableceConexion();
    RecibeConfiguracion();
    While(puedoMandar()){
      Envia(0,1,2,3);
    }
    TerminarConexion();
  }
}
Fin;
```

Figura 3.3: Pseudocódigo para xquetzal-server en el Ciclo 1.

http://cintli.igeofcu.unam.mx/xitris.

3.3. Ciclo I (Desarrollo del Cliente-Servidor)

3.3.1. Análisis

Por ser este el primer ciclo de desarrollo debemos de tener muy claro que es lo que queremos construir, teniendo como base los documentos A,B y C podemos construir la base de nuestro proyecto. El esquema que pretendo utilizar para esta etapa de desarrollo es el modelo Cliente-Servidor. El modelo Cliente-Servidor es un esquema en donde se manejan esencialmente dos actores, por un lado un programa que necesita de algún tipo de servicio y por el otro un programa que puede ofrecer este servicio. Esta idea nació por la necesidad de distribuir la carga de trabajo y de hacer la interacción entre varios componentes más sencilla. El modelo cliente servidor no solo trabaja en una red, también funciona en una sola máquina. Como actualmente tenemos 3 pcs en una red local, podemos poner en practica este modelo. Lo que necesitamos en primera instancia es un servidor que esté ofreciendo el servicio de: "Datos del RIS" y un cliente que pueda graficar y procesar estos datos. Pero como en este momento no tenemos mucha experiencia programando en red comenzaremos con algo mas sencillo: construir un simple servidor que envíe datos, en especifico 4 enteros, y otro programa, el cliente, que los reciba y los imprima en la consola.

3.3.2. Diseño

Tenemos una máquina Pentium I con la tarjeta decodificadora LabPC+ conectada a un Puerto ISA, en esta computadora no tenemos aun las pruebas de flujo de datos máximo, es decir, cuantos enteros puede transmitir sin perder la información. Esta máquina la bautizamos como quetzal y tiene la dirección IP 192.168.1.4 en nuestra red local. Sera la que hospedara al servidor que bautizamos como xquetzal-server. Este primer servidor publicara un Puerto y solo aceptara por el momento un solo cliente, también negociara la transmisión de datos, es decir, primero recibirá una cadena de 20 enteros y después transmitirá ininterrumpidamente a través de un soquet bloques de 4 enteros a la vez. El pseudo código de xquetzal-server queda como en la figura 3.3.

```
Inicio;
RealizarPeticion();
EstablecerConexion();
Si (HayConexion()){
  EnviaConfiguracion();
  while(HayDatos()){
    RecibeDatos();
    ImprimeDatos();
  }
}
Fin;
```

Figura 3.4: Pseudocódigo para xquetzal-cliente en el Ciclo 1.

Por otro lado tenemos una máquina Athlon XP+ a 2 GHz, esta máquina es de reciente adquisición y es la que hemos elegido como cliente, esta máquina la bautizamos como tlaloc y tiene la dirección IP 192.168.1.3. El programa cliente tiene el nombre de xquetzal y para este ciclo lo único que necesitamos es que establezca una conexión con quetzal, mande 20 enteros y después comience a recibir los datos e imprimirlos en la consola (figura 3.4).

3.3.3. Implementación

Para implementar la conexión en red, se usaron sockets. Se tomaron como base los ejemplos de sockets publicados en [29], en particular en el capítulo nombrado "Mecanismos IPC del UNIX" se cubren los aspectos principales de interconexión entre sistemas tipo UNIX, en este caso Linux.

La forma en que funcionan es haciendo una llamada al sistema para que al llamar a un puerto, redirija la información a un proceso. Para establecer este tipo de mecanismos necesitamos publicar nuestro puerto, en este caso escogimos 7000, y mediante esta especie de conexión virtual podemos intercambiar bytes a bajo nivel.

Por el lado del cliente es el mismo asunto, decimos al sistema que necesitamos una conexión a una dirección IP especifica y a un puerto en especial, si del otro lado hay un servidor abre una conexión donde nos podemos comunicar.

La comunicación es bidireccional, pero el control de flujo esta a nuestro cargo. El sistema operativo se encarga de la sincronización (HandShake Protocol) y envío de bytes pero somos nosotros los que establecemos cuando enviamos y cuando recibimos.

Para hacerlo transparente al programador, se establece un esquema de lectura/escritura. Si formalizamos un protocolo en donde el cliente es el primero que se comunica entonces el programa cliente utiliza la función write(Bytes) y el servidor la función read(&Bytes), el S.O. sincroniza estas tareas.

3.3.4. Pruebas

1. Mandar una serie alternada de 0,1,2,3 por parte del servidor en cada paquete y recibirla del lado del cliente.

2. Mandar 0,-1,-2,-3 en cada paquete del servidor e imprimirlo correctamente en la parte del cliente.

3. Mandar 0..19 por la parte del cliente e imprimirlo temporalmente en el lado del servidor.

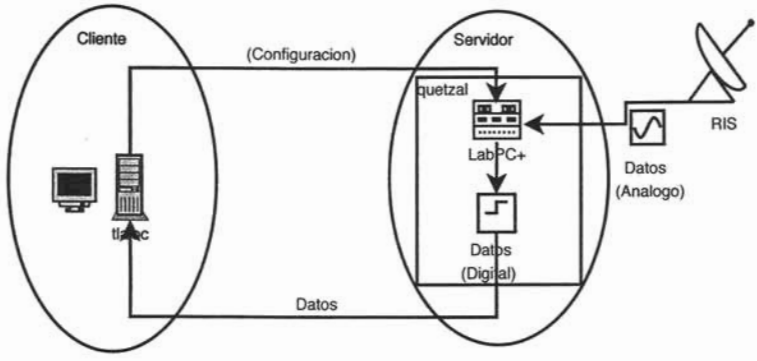

Figura 3.5: Modelo Cliente-Servidor en el Ciclo 2.

4. Instalar EtherRape para revisar las conexiones entre el cliente y el servidor.

Estas pruebas tienen como propósito saber que se están mandando correctamente los datos por parte del servidor y que se están recibiendo por parte del cliente y viceversa.

Las pruebas se realizaron satisfactoriamente, teniendo como resultado la liberación del primer ciclo de desarrollo. Estas pruebas se realizaran cada vez que un nuevo ciclo de desarrollo sea realizado.

3.4. Ciclo 2 (Integración con LabPC+)

3.4.1. Análisis

El ciclo 2 esta compuesto de la integración del modulo LabPC.h construido en el proyecto anterior. Esta biblioteca nos provee de una interfaz para poder comunicarnos con la tarjeta convertora Análogo Digital LabPC+ de National Instruments.

3.4.2. Diseño

El programa xquetzal-server hasta este momento solo puede mandar datos a través de un puerto lógico y una sesión pre-establecida, el único cambio substancial es que nos comunicaremos con la tarjeta LabPC+ y obtenemos los datos (Figura 3.5)

Para este ciclo es conveniente tener una configuración fija para la tarjeta y hemos fijado la configuración estándar, de acuerdo con lo señalado en el apéndice D y como lo programado en el apéndice I. Hay una modificación, el multiescaneo, ya que la configuración estandard escucha un solo canal.

El pseudocódigo se muestra en la figura 3.6.

Las funciones que se encuentran en el pseudocódigo tienen el mismo nombre que la funciones implementadas en LabPC.c.

3.4.3. Implementación

La integración de la biblioteca LabPC.c en xquetzal-server llevo aproximadamente una semana, fue relativamente sencillo, solo sustituir la generación aleatoria de datos por la ad-

```
Inicio;

PublicaPuerto();
While(true){
  EsperaPeticion();
  Si (Peticion()){
    EstableceConexion();
    RecibeConfiguracion();
    ConfigurarLabPC();
    IniciarCaptura();                                          10
    While(puedomandar()){
      datoListo();
      x0 = leeFIFO();
      datoListo();
      x1 = leeFIFO();
      datoListo();
      x2 = leeFIFO();
      datoListo();
      x3 = leeFIFO();
      Enviar(x0,x1,x2,x3);                                     20
    }
    detenerCaptura();
    TerminarConexion();
  }
}

Fin;
```

Figura 3.6: Pseudocódigo para xquetzal-server en el ciclo 2.

quisición de datos de la tarjeta LabPC+.

En tan solo 3 semanas, ya teníamos un programa que esta adquiriendo datos del RIS en tiempo real.

3.4.4. Pruebas

Se realizaron las siguientes pruebas:

A partir de un generador de señales, se pudo constatar que de a cuerdo a la variación de voltaje, los datos obtenidos en la terminar variaban de acuerdo al voltaje especifico, tuvimos como ayuda al laboratorista Filiberto Matias para corroborar la información. Aunque dejamos para posteriores ciclos el análisis detallado, pues aun no tenemos un método para el almacenamiento y el análisis gráfico de los datos.

3.5. Ciclo III (Interfaz Gráfica)

3.5.1. Análisis

Contamos con un servidor que prácticamente funciona y podríamos empezar a guardar los datos y analizarlos, pero por ahora es mas importante la visualización, ya que el Dr. Alejandro Lara le interesa sobremanera ver como está reaccionando el equipo y también comparar las gráficas que resulten con las obtenidas del graficador ruso. Por estas circunstancias decidimos empezar a construir la interfaz gráfica del cliente xquetzal. Para esto tenemos que tomar en cuenta los posibles problemas en cuanto al proceso goloso. Ya había implementado el problema clásico de programación paralela conocido como el problema del barbero. La manera en que lo resolví fue usando memoria compartida. Es decir dos procesos independientes generan un espacio de memoria que los dos pueden leer y escribir. La otra idea fue probar de nuevo el proyecto anterior y corroborar que cualquier otro proceso puede ejecutarse a la par sin verse afectado el primero (Corrimos xquetzal en segundo plano con emacs y no hubo problema).

Así decidí hacer dos programas para xquetzal, uno que modificara las variables de la memoria compartida y otro que solo las estuviera leyendo. Claramente se ve que el programa que modifica las variables es el que configura todo el entorno y el que las lee es el graficador, este graficador solo estará atento a las variables para poder así cambiar su comportamiento.

Nació el proyecto cuyo nombre es *tochtli*[2] , el programa que crea la memoria compartida y esta atento a las señales producidas por el usuario y las refleja en la memoria compartida, por otro lado tenemos al proyecto llamado *ollin*[3] . Es el programa encargado de revisar periódicamente la memoria compartida y a partir de esta poder cambiar su comportamiento en tiempo real. Pero por ahora el alcance de este ciclo esta reducido a la construcción de estas dos interfaces gráficas y que se ejecuten correctamente al invocarlas con el comando xquetzal.

3.5.2. Diseño

Tochtli Es una ventana de configuración extremadamente sencilla (Imagen 3.7), contara con una barra de menú con las siguientes opciones:

a) Archivo>Salir: Sale de xquetzal.

b) Capturar>Iniciar: Inicia Captura con la configuración actual.

[2] Conejo en Nahuatl

[3] Movimiento en Nahuatl.

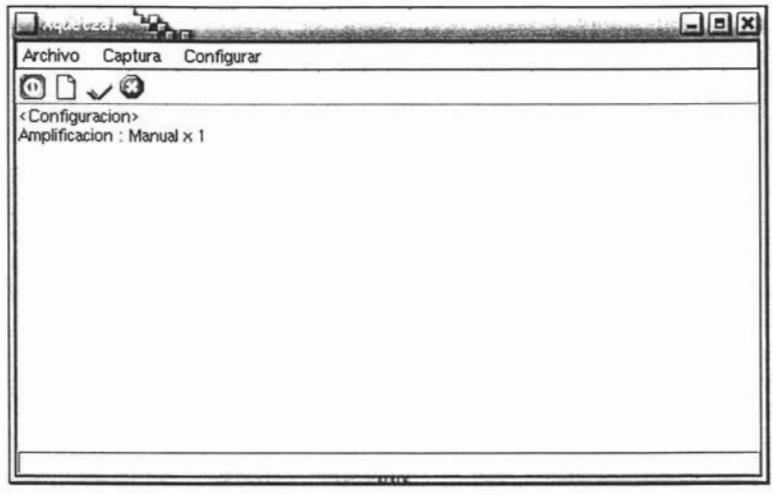

Figura 3.7: Diseño gráfico de Tochtli.

c) Capturar>Detener: Detiene/Pausa La captura, no la finaliza.

d) Configuración>LabPC: Abre una ventana de configuración que describiré posteriormente.

e) Configuración>Pantalla: Abre una ventana donde podremos configurar los gráficos que se están produciendo en tiempo real, esta ventana se describirá en ciclos posteriores.

f) Configuración>Eventos: Abre una ventana para configurar las variables necesarias para detectar un evento, esta ventana se describirá en un ciclo posterior porque por ahora no esta dentro de nuestras posibilidades diseñarla.

g) La pantalla de Configuración>LabPC contara con 3 configuraciones principales, estas opciones fueron analizadas en el capítulo 2 de este trabajo y es por eso que podemos enunciarlas en este ciclo, y las cuales son 2 para la amplificación y 1 para la frecuencia.

h) La amplificación puede ser manual o automática, si es manual nunca cambia, si es automática, una vez que el canal se sature procederá a una amplificación mas baja. Si por otro lado no rebasa un umbral preestablecido, la amplificación aumentaría. La frecuencia máxima de muestreo es de 1MHz. Esta pantalla tiene 2 botones, aceptar y cancelar.

i) Una barra de botones: Salir, Iniciar y Detener.

j) Área de texto para poder escribir las variables internas o describir el sistema en tiempo real.

Ollin En el diseño exterior es mas fácil, es una ventana con solo una área de dibujo.

Una vez construidas estas dos interfaces procedemos a compilarlas y a construir un tercer programa al que ya hemos bautizado como xquetzal.

```
Inicio;

Si ((x=fork())==0){
  Tochtli();
}else{
  Ollin();
}

End;
```

Figura 3.8: Pseudocódigo para xquetzal-server en el ciclo 3.

Xquetzal es nuestro nombre clave para el programa cliente, así que xquetzal lanza a tochtli y a ollin mediante un fork() o creador de procesos hijos.

El pseudocódigo de xquetzal quedaría como en la imagen 3.8.

3.5.3. Implementación

El código fue generado por Glade, tiene varios cientos de líneas, xquetzal está implementado en C y ejecuta mediante la sentencia fork(); y system(); los dos programas.

3.5.4. Pruebas

Se ejecuto xquetzal en una terminal y ambos programas (ollin y tochtli) se ejecutaron correctamente, para terminar los programas se tuvo que cerrar cada uno independientemente.

3.6. Ciclo IV (Implementación de Iniciar/Detener Captura)

3.6.1. Análisis

Una vez construida la interfaz gráfica para xquetzal llegamos al paso de la integración. Tenemos un servidor que está recibiendo datos y por otro lado tenemos al subprograma ollin que será el encargado de graficarlos en tiempo real, la integración será realizada en ollin. Ahora bien, en el proyecto anterior una serie de factores técnicos y de diseño hicieron que el desarrollo fuera interrumpido. Por estas circunstancias debemos tener especial cuidado en esta parte del desarrollo, pues es la fase más crítica del proyecto. Ollin y Tochtli trabajarán bajo un esquema de semáforos. Ollin solo estará mirando verde o rojo, si es verde empieza la captura, si es rojo la detiene. Tochtli es el policía que pone la señal en verde o en rojo. Es el usuario el que decide cuando va a cambiar de verde a rojo. En este ciclo integraremos el cliente de xquetzal con el modelo del semáforo a través de memoria compartida, seguiremos imprimiendo los resultados en una terminal. Dejaremos para el siguiente ciclo graficar los datos.

3.6.2. Diseño

Ollin Revisará cada cierto tiempo la variable Memoria[0], esta variable será la variable de "cambio", cada vez que haya un cambio de estado, se procederá a modificar esta variable, mientras esté en 0 no hay ningún cambio. En el momento que la variable cambie su valor

a 1 indicará que a habido un cambio de estado en el sistema y procederemos a verificar cada una de las variables de estado en el sistema.

Pero ¿porque este esquema y no solo revisar si inició la captura o no?.

El problema principal es que la revisión de la memoria compartida se lleva a cabo cada 0.5 segundos, si tenemos que revisar cada una de las variables de estado estaríamos trabajando 20 veces más. En lugar de eso, solo revisamos la variable de "cambio" y si está activada ahora pasamos a revisar cada una de las variables de cambio. Es importante hacer notar todos estos puntos, pues el equipo con el que se cuenta estará encargado de realizar muchos servicios, tiene que graficar los datos, analizarlos, guardarlos, modificarlos y cada línea depurada o de diseño tiene un peso específico muy alto. Una vez que se haya revisado la variable de "cambio", procedemos a revisar la variable "semáforo" o Memoria[1], Si la variable está en 1 indica que la captura a sido iniciada por lo que se establece una conexión a xquetzal-server, se le transmiten las 20 posibles variables de configuración que aun no hemos definido y comenzamos a adquirir los datos y los imprimimos en una terminal. Para poder realizar la captura de datos, una vez que se haya confirmado que el cambio es para empezar la captura, se invocara a la función "gráfica" que estará en un nuevo modulo llamado ImageFactory.h, a esta función le pasamos el área de dibujo y le dejamos la responsabilidad de que ella grafique y determine cuando la captura sera detenida.

He delegado la responsabilidad de graficar los datos a la función *gráfica* porque una vez que se haya iniciado la captura, puede haber cambios en tiempo real, como el cambio de amplificación que tiene que ser tratados por el mismo modulo de ollin, y es solo cuestión de ImageFactory determinar que la captura a sido detenida cuando la variable "semáforo" cambie de nuevo a 0 (rojo). Por el momento ImagenFactory desplegara los datos en la terminal.

Tochtli En este ciclo, insertamos la creación de memoria compartida y toda la infraestructura para modificar los estados de memoria de acuerdo a los cambios producidos por el usuario.

3.6.3. Implementación

Algunas veces cuando se genera la memoria compartida el programa generaba un segmentation default. El problema fue resuelto insertando un sleep en ollin, que es el proceso que pide memoria compartida. La integración se realizó en el callbacks.c de ollin, especificamente en ImagenFactory.c, es aquí donde se insertó el código del ciclo 2.

3.6.4. Pruebas

Se ejecuto xquetzal y xquetzal-server. Se inicio la captura en xquetzal. La comunicación fue exitosa y los datos empezaron a fluir. Se detuvo la captura y respondió aceptablemente. Este proceso se repitió varias veces sin que se encontrara algún problema. Al terminar el programa se cerraron independientemente cada subprograma (ollin y tochtli). Los datos mostraron fluctuaciones en funcion del voltaje de entrada de acuerdo a las pruebas preestablecidas en los primeros ciclos.

3.7. Ciclo V (Graficación de Datos)

3.7.1. Análisis

Xquetzal fue integrado satisfactoriamente, ahora solo es cuestión de sumarle funcionalidad pues la parte central y de mayor riesgo quedo atrás. En este ciclo, implementaremos la graficación en ollin. En el primer proyecto de esta tesis había construido un modulo independiente para graficar los datos.

Ahora es solo cuestión de integrar este modulo al actual modelo, y en teoría es fácil pues solo hay que invocar a la función contenida anteriormente.

Esta función emula al graficador de origen ruso, generando una secuencia finita de lineas quebradas que con el transcurso del tiempo va desplegando la señal.

El canal 0 se tomo como la intensidad, el canal 1 la polarización, el canal 2 seno y el canal 3 coseno.

3.7.2. Diseño

Prácticamente no habrá ningún cambio, solo se modificara la parte de la obtención de datos, en el anterior proyecto se tomaban directamente de la tarjeta LabPC+ a partir de la biblioteca LabPC.h, ahora los datos los tomamos de la interfaz de red, que a manera de diseño es exactamente lo mismo, solo que en lugar de un canal, ahora tenemos los 4, pero solo es repetir el proceso en diferentes niveles de la gráfica.

No se necesitó de algún temporalizador que sincronizara la señal con el despliegue, en el mejor de los casos el despliegue se detiene hasta que el siguiente dato este listo.

3.7.3. Implementación

Se sustituyo a graficar.c por ImageFactory.c, pero casi todo el código fue insertado sin modificación. La técnica de doble buffering se conservó. El código fue depurado para que fuera más legible.

3.7.4. Pruebas

Las gráficas generadas en esta parte del proyecto respondieron de acuerdo al generador de señales.

La figura 3.9 muestra un snapshot de ollin trabajando en 4 canales.

3.8. Ciclo VI (Guardando los datos)

3.8.1. Análisis

Es sorprendente que para este ciclo ya tengamos un sistema funcionando y en el cual se puedan empezar a realizar pruebas de calibración del sistema. En este ciclo integraremos el modulo de Guardar.h construido en el proyecto anterior. Este módulo fue pensado para que fuera independiente y ésta será una prueba de ello. El módulo se describe en el apéndice C, ahora aprovecharemos para depurar el código y también para volverlo a probar. También crearemos un programa que llamaremos *xdat2txt*, se encargara de descomprimir los datos en formato xdat a código ASCII.

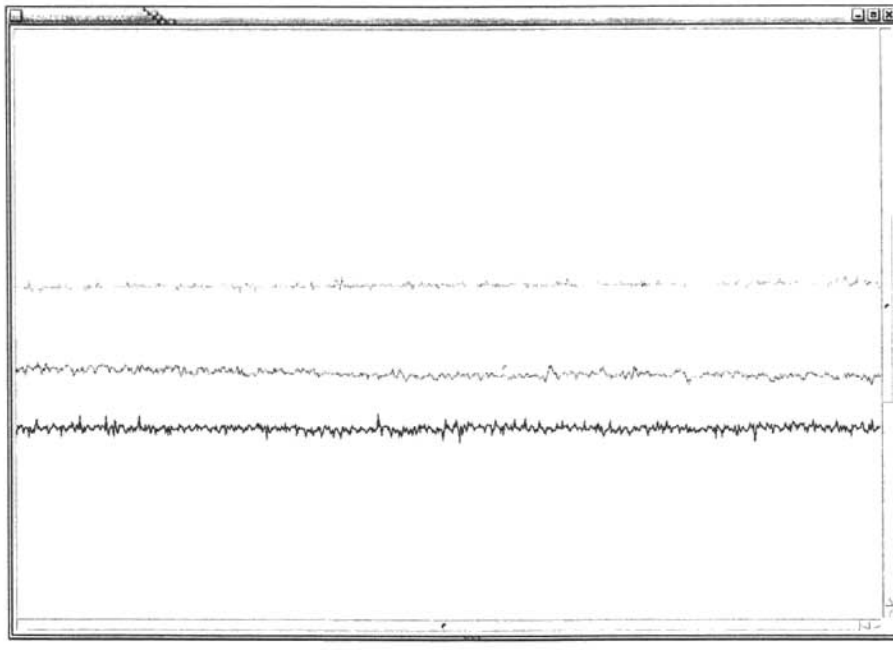

Figura 3.9: Ollin graficando en Tiempo Real.

3.8.2. Diseño

En la biblioteca ImageFactory.c, en la parte donde se reciben los datos de la interfaz de red, introduciremos el módulo para guardar los datos. Una vez que los hayamos graficando, no hay mas que pasárselos al buffer encargado de almacenarlo y posteriormente revisar si el buffer está lleno, si el buffer está lleno, entonces lo guardamos y continuamos con el siguiente ciclo. Ahora bien, este modulo solo quedó construido para guardar datos crudos y medias, no para guardar modificaciones, estas partes del código quedaran para siguientes ciclos, ahora lo que nos interesa es poder guardar los datos para hacer pruebas mas confiables acerca de lo que estamos capturando.

Xdat2txt utilizará las bibliotecas Guardar.c y Buffer.c, la forma en que funcionará es leer secuencialmente un archivo e ir creando una estructura tipo Buffer, cuando este buffer se llene, lo guardamos al disco pero con formato ASCII, así hasta terminar con el archivo de origen.

3.8.3. Implementación

Cuando integramos esta parte del proyecto me dí a la tarea de depurar el código. Cuando construí este modulo no sabía muchas funciones de C, pero ahora pude reducir de casi 500 lineas a solo unas decenas. El código quedó más limpio y decidimos probarlo con situaciones reales.

La implementación de xdat2txt llevo unas cuantas lineas. es muy similar al programa cp de linux. El código se encuentra en el apéndice J.

3.8.4. Pruebas

Probamos el modulo independientemente con una función que generaba todos los número posibles que soporta la tarjeta LabPC, lo cual nos dio malos resultados. Pues llegando al 1234 los datos comenzaban a descomponerse. Me dí a la tarea de investigar la posible causa y después de varias pruebas pude solucionar el problema, la causa se debía a un error al comprimir los datos, en un corrimiento que estaba mal hecho, así después de varias pruebas pudimos liberar parcialmente el código de Buffer.h

Ahora bien, al probar xquetzal completamente, se generaban satisfactoriamente archivos con los datos obtenidos de la tarjeta, con estos datos pudimos hacer un análisis más confiable de los datos obtenidos, entonces nos dimos cuenta de que existia un error.

Los datos reportados en los archivos generados por xquetzal no concordaban con lo obtenido en el graficador ruso después de cierto umbral, en especial cuando el voltaje era negativo, esto no lo habíamos tomado en cuenta porque pensábamos que el canal se saturaba llegando a este umbral, pero esto era falso, había algo mal, podía estar en el servidor, en el graficador o en el buffer para guardar los datos.

Comenzamos por lo más probable y era revisar si se estaban haciendo correctamente las transformaciones de la tarjeta, haciendo un análisis en la forma en que se estaban transformando los datos me pude dar cuenta de que podía haber un error a la hora de unir los dos bytes obtenidos de los registros de la tarjeta y hacerlos un entero en C. El problema era una mascara que yo presuponía había de hacerse automáticamente. Así añadi la siguiente orden al modulo de LabPC.h en la función de DameDato(); Si el segundo byte que leo de la tarjeta, el 3er bit es un uno, entonces es un negativo, por lo que en el resultado le añadimos 1s en las primeras 20 bits. Este problema fue descrito en la bitácora de Buffer.c.

Al solucionar estos problemas decidimos cerrar el Ciclo VI. Para esta fecha (Agosto 2004) el Laboratorio de Interferometria Solar comenzó a archivar datos del sol en forma digital.

3.9. Ciclo VII (Estructura de Base de Datos)

3.9.1. Análisis

Comenzaremos a crear la estructura de nuestra base de datos. Para este momento tenemos el programa capturando datos y guardando los resultados en el disco, ahora les pondremos orden, para eso tenemos que sincronizar el reloj de la computadora con algún reloj preciso. Usaremos el protocolo NTP, sincronizaremos el reloj, una vez sincronizado procedemos a crear una estructura de directorio donde guardaremos los datos capturados.

3.9.2. Diseño

El protocolo NTP es un protocolo para la sincronización de relojes a través de una red. En México tenemos el servidor CRONOS que lo mantiene el gobierno federal, este servidor ofrece el tiempo en México y nos asegura que el tiempo estará sincronizado con respecto a los relojes de otros observatorios solares. Es importante tener la hora lo mas sincronizada posible pues en el momento de un evento podemos estar seguros y comparar los datos obtenidos de otros telescopios. Así que cuando iniciemos a xquetzal, lo primero que hará es sincronizar la computadora con cronos.

Después crearemos una estructura de directorio de la siguiente manera:

/database/aaaa/mm/dd/

Donde aaaa es el año, mm el mes y dd el día. El tiempo que se estará usando sera el UTM o tiempo universal para no tener confusiones entre horario local o de verano u otros.

Una vez que tengamos esta estructura tendremos 5 archivos: canal0.xdat, canal1.xdat, canal2.xdat, canal3.xdat y finalmente README. En el README estarán registrados todos los cambios de estado en xquetzal así como sus configuraciones iniciales, pero por el momento solo escribiremos la hora de inicio de la captura y la hora final.

3.9.3. Implementación

Usamos las características de Linux para poder implementar esta parte, por un lado SUDO fue usado para no comprometer al sistema junto con ntpdate y hwupdate para la actualización del reloj de la computadora.

3.9.4. Pruebas

Se procedió a todas las pruebas anteriores (Ciclo I al VI), resultando satisfactoriamente. Se cambio la fecha de la computadora y al momento de ejecutar xquetzal se actualizo satisfactoriamente la fecha y también se crearon los directorios correspondientes. La hora de Inicio y de Fin del README coincidieron con las horas de Inicio y Fin de las Pruebas.

3.10. Ciclo VIII (Creación del README)

3.10.1. Análisis

Hasta este momento, los datos obtenidos de la tarjeta se guardan en .xdat, pero no tienen ninguna estampa de tiempo. En este ciclo se implementara que cualquier cambio de estado en xquetzal se vea reflejado en el README y en los archivos .xdat. Es decir que guardaremos en forma binaria los cambios hechos en el sistema en tiempo real.

Es ollin el encargado de generar la estructura donde se guardan los datos y de crear el README, ya que en ollin se encuentra la función que está revisando periódicamente la memoria compartida y tomando las decisiones correctas. Cuando tochtli cambia una variable, ollin lo detecta y guarda en el README ese cambio de estado.

3.10.2. Diseño

En la biblioteca Buffer.c sera añadida la funcionalidad para guardar estampas de tiempo y los cambios de estado del sistema.

Ollin estará a cargo de administrar al Buffer. Especificamente, serán las bibliotecas callbacks.c e ImageFactory.c, estas bibliotecas son las encargadas de estar revisando los cambios de estado en tiempo real. callbacks.c tendrá a su cargo el registrar en el archivo README cualquier cambio de estado. Se eligió a Ollin porque es el encargado de revisar los cambios que se ejecutan, en otro caso tendríamos un pequeño desfasamiento. Hay que recordar que la función que revisa los cambios periódicamente se ejecuta cada 0.5 segundos.

3.10.3. Implementación

Se tuvo que revisar la implementación del Buffer.c, ya que no estaba implementada la asignación de fecha. La forma en que se asignan las fechas esta implementada de acuerdo a lo establecido en el Diseño Arquitectónico del Capítulo 2 de este trabajo.

Por otra parte se añadio al callbacks las instrucciones correspondientes al cambio de estado.

3.10.4. Pruebas

Se ejecutaron cambios de estado en el sistema: iniciar, detener, cambiar amplificación (Aunque aun no este implementado), cambiar frecuencia.

Los cambios se reflejaron correctamente tanto en el README como en los datos grabados.

3.11. Ciclo IX (Cambio de Amplificación y Frecuencia)

3.11.1. Análisis

En ese ciclo implementaremos el cambio de Amplificación y el cambio de frecuencia en tiempo real. Se modificaran xquetzal-server y xquetzal-cliente. El Dr. Alejandro Lara sugirió la creación de frecuencias de captura especificas: 0.001 seg, 0.01 seg, 0.5 seg, 1 seg, 2 seg, 5 seg, 10 seg y 30 seg. Así que el servidor xquetzal-server tendrá 8 posibles configuraciones de frecuencia (0..7). Estas configuraciones tiene que estar en xquetzal-client, por un lado las configuraciones gráficas en tochtli y la ejecución de esas ordenes en ollin.

3.11.2. Diseño

La interfaz se muestra en la imagen 3.10.

Cuando el usuario presione un cambio de amplificación, tochtli pondrá a Memoria[0]=1 que significa un cambio de estado y Memoria[27]=2 cambio de frecuencia; también pondra Memoria[4]=X donde X es la configuración de frecuencia.

Por otro lado ollin estará leyendo a Memoria[0] para un posible cambio de estado en el sistema, cuando este cambio ocurra, leerá Memoria[27] para saber que está pasando, cuando lea que Memoria[27]=2 sabrá que es un cambio de frecuencia, entonces ira a Memoria[4] y leerá la configuración que esta recibiendo. Con esta información Manda un Memoria[0]=0, lo

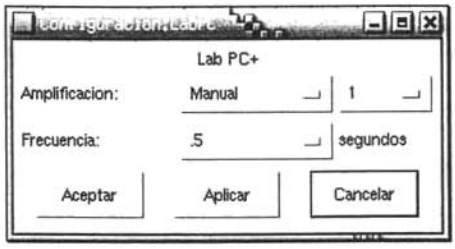

Figura 3.10: Interfaz para cambiar la amplificación y la frecuencia.

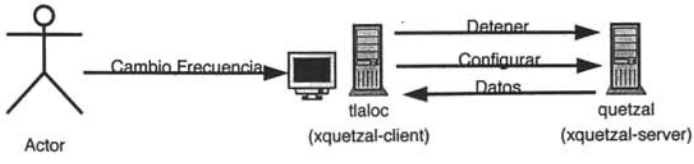

Figura 3.11: Modelo Cliente-Servidor en el Ciclo 9.

que hace que ImagenFactory se entere si está graficando, se detenga porque hay un cambio entrante, en ese caso, Ollin se detiene con un sleep para garantizar que ImagenFactory se detenga correctamente y después manda una nueva configuración a través del socket para que el servidor reconfigure la tarjeta, una vez que la configuración haya tenido efecto, Ollin (por medio de callbacks.c) guardara todos los cambios tanto en el README como en el buffer, y mandara de nuevo a graficar.

El servidor no le interesan todas estas cosas, únicamente esta esperando una petición de algún cliente. Cuando haya una petición, lee la configuración, la ejecuta y comienza a mandar datos hasta que le avisen que se detenga, entonces vuelve a su estado de inactividad, esperando una nueva petición.

Para el cambio de Amplificación es exactamente el mismo proceso solo que Memoria[27]=1 y la configuración se guarda en Memoria[3].

3.11.3. Implementación

La tarjeta tiene limites de tiempo para la conversión, la forma en que se implemento esta parte fue que mientras el intervalo entre capturas fuese menor al máximo soportado por la tarjeta, configuramos con ese intervalo de tiempo. Si el intervalo de captura es mayor al máximo soportado entonces tomamos una cantidad fija entre capturas y esperamos hasta que se tenga el tiempo de captura especificado.

Para configuraciones de 0.001 seg y 0.01 seg. no hay ningún problema, configuramos la tarjeta con esa frecuencia. Sin embargo hay que tener cuidado cuando configuramos la frecuencia de la tarjeta. Esta frecuencia es entre conversiones, es decir, si tomamos una frecuencia de muestreo de .25 seg, entonces el tiempo entre el canal 0 y el canal 1 es de .25 seg, pero la frecuencia de muestreo por canal es de 1 seg. Por esta razón debemos dividir entre 4 la frecuencia de muestreo, o mejor aun, tomar una frecuencia de muestreo alta (0.01 seg) y después

esperar el tiempo necesario para volver a realizar una captura.

La tarjeta LabPC+ cuenta con un reloj interno a 1 MHz, si queremos que entre cada intervalo haya X segundos para 4 canales, tenemos que:

1. Para 0.001 segundos:

$$1x10^6 * 0{,}001/4 = 250_{10} = 11111010_2 = FA_{16}$$

2. Para 0.01 segundos:

$$1x10^6 * 0{,}01/4 = 2500_{10} = 100111000100_2 = 9C4_{16}$$

3. Para 0.5 segundos: Necesitamos un tiempo base pequeño:

$$X * f = A$$

Donde X es la configuración de la tarjeta, f la frecuencia del reloj interno de la tarjeta LabPC (1MHz) y A la frecuencia entre canales.

Ahora, la frecuencia A debe ser congruente con 4:

$$4A = B$$

La frecuencia B multiplicada por "N" pasos tiene que ser igual a 0.5 seg:

$$NB = 0{,}5 = 1/2$$

Sustituyendo:

$$\frac{4NX}{1000000} = \frac{1}{2}$$

Despejando a N:

$$N = \frac{1000000}{8X} = \frac{125000}{X}$$

Es decir X divide a 125000. Si escogemos a $X = 5000$:

$$N = \frac{125000}{5000} = 25$$
$$A = \frac{5000}{1000000} = 0{,}005$$

Es decir, para 0.5 seg, al configurar la tarjeta con 5000, hay un intervalo de muestreo entre canales de 0.005 seg, dejando 25 iteraciones sin reportar datos.

Cabe mencionar que utilizamos este método porque no queremos usar el reloj de la PC, es más coherente utilizar los relojes internos de la tarjeta convertora.

4. Para 1 segundo:

$$\frac{4NX}{1000000} = 1$$
$$N = \frac{1000000}{4X} = \frac{250000}{X}$$

Si escogemos a $X = 5000$:

$$N = \frac{250000}{5000} = 50$$
$$A = \frac{5000}{1000000} = 0{,}005$$

5. Para 2 segundos:

$$\frac{4NX}{1000000} = 2$$

$$N = \frac{1000000}{4X} = \frac{500000}{X}$$

Si escogemos a $X = 5000$:

$$N = \frac{500000}{5000} = 100$$

$$A = \frac{5000}{1000000} = 0{,}005$$

6. Para 5 segundos:

$$\frac{4NX}{1000000} = 5$$

$$N = \frac{5000000}{4X} = \frac{1250000}{X}$$

Si escogemos a $X = 5000$:

$$N = \frac{500000}{5000} = 250$$

$$A = \frac{5000}{1000000} = 0{,}005$$

7. Para 10 segundos:

$$\frac{4NX}{1000000} = 10$$

$$N = \frac{10000000}{4X} = \frac{2500000}{X}$$

Si escogemos a $X = 5000$:

$$N = \frac{2500000}{5000} = 500$$

$$A = \frac{5000}{1000000} = 0{,}005$$

8. Para 30 segundos:

$$\frac{4NX}{1000000} = 30$$

$$N = \frac{30000000}{4X} = \frac{7500000}{X}$$

Si escogemos a $X = 5000$:

$$N = \frac{7500000}{5000} = 1500$$

$$A = \frac{5000}{1000000} = 0{,}005$$

3.11.4. Pruebas

Capturamos para una hora a 0.001 seg, la siguiente hora a 0.01 seg y así sucesivamente para cada frecuencia.

La captura coincidió con la hora, falta comparar un evento solar real con nuestro sistema de captura.

3.12. Ciclo X (Publicación de Datos por Internet)

3.12.1. Análisis

La siguiente parte del proyecto es publicar los datos por Internet. Xquetzal-client guarda los datos en el directorio /database, la idea es montar esta partición mediante NFS en el servidor web y que por otro lado un JSP genere una pagina dinámica que muestre de una manera ordenada los datos obtenidos.

El almacenamiento de datos a frecuencias alta puede saturar rápidamente el disco.

16 bits por dato a una frecuencia de 0.01 seg por 4 canales en 8 horas de captura nos da: 23040000 Bytes = 22500 KBytes = $22MBytes$ diarios. Si pensamos capturar 20 días al mes o aproximadamente 240 días al año:

$$22 * 240 = 5280MBytes = 5GBytes$$

Aproximadamente al año.

Por lo tanto debemos de cuidar el espacio en el disco, por eso solo vamos a tener accesible los datos en formato xdat, en otro caso el crecimiento de la base de datos se puede ver multiplicado hasta 10 veces.

3.12.2. Diseño

El diseño general de xquetzal-publish ya está definido en el Apéndice C (imagen C.2).

Entonces tenemos un programa en Cintli que copia los archivos de captura a /database/actual y los transforma en .txt, también copia el README. Independientemente, un JSP crea una pagina web con un registro para acceder a las observaciones anteriores y una sección que sirve de interfaz para obtener los datos en tiempo real.

Creamos una cuenta especial en el servidor para montar los archivos a publicar

3.12.3. Implementación

Se instalo el Servidor Tomcat 5.0 en Cintli. Se modificó el archivo /etc/inittab para montar el directorio /database mediante NFS. El desarrollo web se llevo a cabo sin contratiempos. Para crear la pagina web de observaciones anteriores se desarrollo un algoritmo que hace lo siguiente: Busca en los directorios de /database si existe el archivo README, si existe, crea una liga con una serie de opciones para redirigirlo a otro jsp. Si no existe el README quiere decir que no hubo medición ese día. Este segundo JSP llamado giveME.jsp, el que captura las opciones, que son año, mes y día; con estos datos crea a su vez una serie de ligas hacia los archivos de esa fecha. La sección de tiempo real es una pagina web estática que siempre apunta a /database/actual. El programa que copia los datos del día actual a /database/actual se llama cpXoc y usa xdat2txt para convertir los archivos.

Solo en /database/actual se encuentran los datos disponibles en .txt. También se hizo una liga para poder bajar el programa xdat2txt.

3.12.4. Pruebas

Al principio costó un poco de trabajo poner a funcionar en su conjunto a todo el proyecto, por lo que se construyeron scripts que ejecutan los procesos correspondientes, uno para el servidor, otro para el cliente y uno finalmente para el publicador.

Se tuvieron que modificar los scripts de configuración de Tomcat y algunos del sistema en tlaloc y en cintli.

Para el Ciclo X del proyecto (Mayo 2004) tenemos datos en tiempo real del RIS publicados en Internet.

3.13. Ciclo XI (Imágenes en tiempo real)

3.13.1. Análisis

Teniendo los datos en formato xdat, ahora queremos mostrar las imágenes que obtenemos del xquetzal-client al publico en general. Se llegó a la conclusión de que la imagen creada debería ser una grafica representativa de la captura diaria. Se creó una imagen que empieza a las 14 horas y termina a las 24 horas tiempo UTC y que va promediando los datos de acuerdo a la frecuencia de muestreo, el programa que genera estas imágenes lleva el nombre de *Xoc*.

3.13.2. Diseño

Para generar esta imagen, leemos los 4 canales que se encuentren en /database/actual, mediante la biblioteca Buffer.c reconstruimos la captura y conforme vayamos leyendo el archivo, vamos promediando los datos y los desplegamos. Cada 0.5 seg revisamos de nuevo los archivos, si hay algún cambio, desplegamos una nueva imagen. Esta imagen es guardada en /database/actual y en /database/AAAA/MM/DD, para que tengamos un registro histórico en imágenes. El tamaño de la imagen es de 800x600 px y lleva el nombre de /database/actual/flujo.png. Las paginas web que despliegan los datos tanto en tiempo real como de eventos anteriores se modificaran para que desplieguen esta imagen.

3.13.3. Implementación

Se uso GTK 2.0, es la primera vez que se usa esta versión, la razón fue que implementaron la función gdk_pixbuf_save, la cual guarda en formato png un pixbuffer. Se creó una función que haciendo uso del gdk_pixbuf_save puede guardar un DrawingArea. También se crearon funciones genéricas para graficar señales de manera genérica.

Como resultado tenemos la imagen 3.12 , la cual nos muestra los 4 canales en diferentes colores, solo en el horario de observaciones del laboratorio.

3.13.4. Pruebas

Cuando ejecutamos el xoc funcionó correctamente después de algunas modificaciones a los permisos de los directorios (Tuvimos que usar sudo para conservar la seguridad). Para cerrar este ciclo se modificó el script para ejecutar publicator, este script ahora lanza a xoc. Algunas veces no se muestra la imagen por Internet, lo resolvimos poniendo un aviso de "Imagen generandose...".

3.14. Ciclo XII (Creación de Visor)

3.14.1. Análisis

Xoc daba una visión global de la captura y Xquetzal una visión instantánea, faltaba una herramienta que diera los detalles de la captura pero con una visión global. Entonces se planteo modificar a Xoc, naciendo así un nuevo programa: Visor.

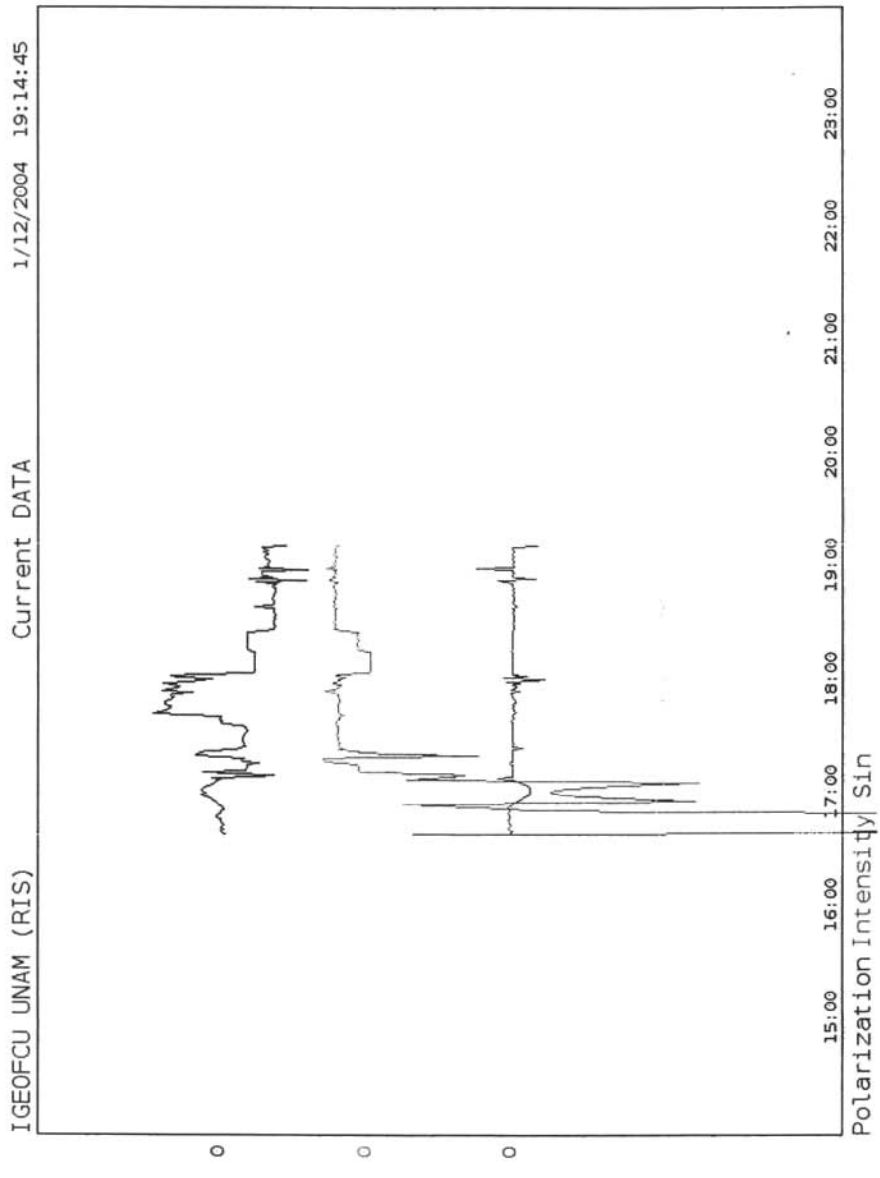

Figura 3.12: Imagen que se publica vía web generada dinámicamente por xoc.

3.14.2. Diseño

Visor nace de Xoc, las modificaciones que se le hacen:

1. El horario de captura se extiende de las 0 horas hasta las 24 horas.
2. La imagen crece a 4800 x 2000 px.
3. No se guarda la imagen.
4. Posibilidad de leer desde cualquier Path.

3.14.3. Implementación

Solo copiamos a Xoc y modificamos los parámetros de acuerdo al diseño.

3.14.4. Pruebas

Xoc se ejecuto satisfactoriamente, se planeo montar a Xoc en otra computadora ya que el servidor web (cintli) podría tener sobrecarga. La señal registrada en Xquetzal, en Xoc y en Visor coinciden.

3.15. Ciclo XIII (Capturar Eventos Solares)

3.15.1. Análisis

En general, toda la infraestructura que hemos creado es con la finalidad de capturar eventos solares de manera automática. Se desarrollaron una serie de herramientas que ahora hacen posible la siguiente fase del proyecto.

El flujo solar cuando no hay eventos no excede cierto límite. Este límite lo marca el estado del equipo, la época del año y las condiciones atmosféricas, es por lo tanto imposible tener margenes fijos para saber si el flujo energético es normal o hay un evento solar.

De acuerdo a la experiencia obtenida en el laboratorio un evento solar se ve reflejado como un aumento significativo en el canal de Intensidad y que se puede ver o no reflejado en el canal de Seno y Coseno. Los tres canales puede variar de forma positiva o negativa. Cuando el evento termina, los 4 canales vuelven a un estado de equilibrio anterior.

En teoría, el flujo promediado se aplana en una linea con pendiente 0, pero muchas veces por problemas electrónicos o mecánicos hacen que la señal oscile a lo largo del día. Una ultima aclaración, hay veces que en el transcurso del día hace falta hacer pruebas para enfocar la antena o hacer pruebas de amplificación, estos movimientos en el equipo pueden confundirse con eventos solares. Otros cambios que se confunden con eventos solares son transmisiones por antenas cercanas, llamadas de celular, nubes, relámpagos o algún movimiento en los cables del equipo.

Este último hecho, hace que esta fase de desarrollo crezca en complejidad, quedando fuera el modelo básico de XProgramming. Es por ésta razon que se desarrollo un generador de eventos, pero a cargo del Laboratorista, es decir, el tendrá la tarea de catalogar los diferentes fenómenos qué se estén presentando.

Como propuesta, este ciclo de desarrollo se puede fraccionar en una gran cantidad de ciclos de vida, dejando abierta la posibilidad de otro trabajo de tesis, el cual llevaria por nombre: "Detección de Eventos Solares en Tiempo Real".

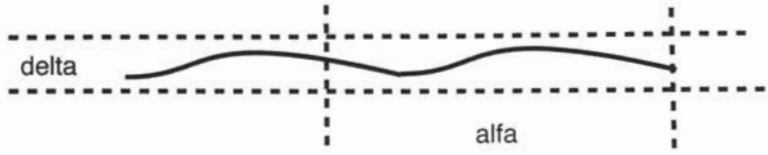

Figura 3.13: Una señal estable en xquetzal.

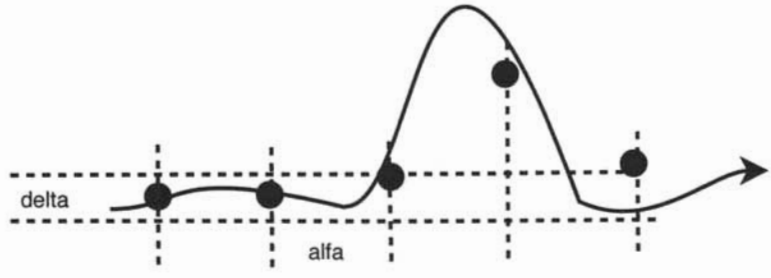

Figura 3.14: Una señal inestable en xquetzal.

3.15.2. Diseño

La señal se dividió en dos estados: Estable e Inestable. La señal estable es una secuencia de lineas quebradas que promediadas no exceden un umbral δ. El promedio se da a partir de una región α (Imagen 3.13). Una señal inestable es aquella que promediada excede un umbral δ. El promedio se da a partir de la misma región α (Imagen 3.14). Como nos podemos dar cuenta, tenemos que definir experimentalmente los parámetros δ y α. Un candidato a evento solar es el intervalo de tiempo en el que transcurre la señal inestable. Una vez que detectamos una señal inestable, tenemos que saber si es un error del equipo o un posible evento solar. Por ahora, la única forma de saber esto, es con ayuda del laboratorista. Una interfaz creará una lista con la hora de inicio y la hora final del posible evento, e indicará en una ventana las siguientes posibilidades:

1. Eventos Solar Confirmado.
2. Posible Evento Solar.
3. Calibración del equipo.
4. Fenómeno Atmosférico.
5. Error del Equipo.

La captura de eventos solares se deja a cargo de Visor, sera renombrado "analyze". Analyze, aparte de desplegar los datos, buscará en el flujo variaciones que indiquen una señal inestable, cuando el técnico resuelva su situación (Evento, calibración, etc), guardará: Hora de Inicio, Hora Final, Origen de la Inestabilidad y una nota de texto en nuestra base de datos

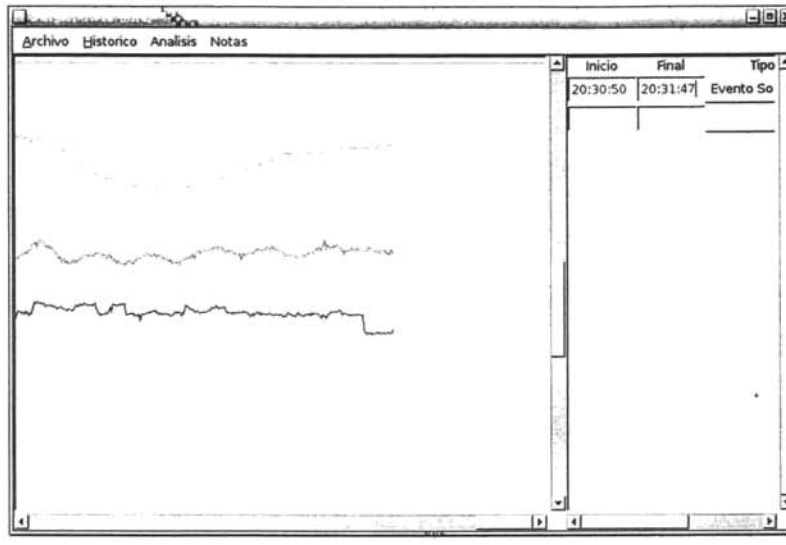

Figura 3.15: Interfaz Gráfica de Analyze.

(postgresql y en el archivo EVENTOS de cada día de observación), la base de datos se llama eventossolares, la tabla es eventos y tienen los datos anteriormente citados, más el día de la observación. Se necesita modificar el JSP para eventos solares para que muestre los días de captura y los días con eventos solares.

Analyze tendrá dos pantallas principales, el área de despliegue de datos y, a un costado, se encontrara una lista con los posibles eventos.

También contará con un área de configuración para indicar la hora real de captura y la hora final, ya que los primeros intervalos son de calibración.

En el menú se le añadira una opción para Guardar Imagen, Realizar de Nuevo el Análisis y refrescar las Notas.

La opción de Guardar Imagen guarda la imagen en formato png en el lugar que se le indique. Realizar de Nuevo el Análisis, limpiará todo posible análisis anterior y generará un nuevo análisis. Por ahora Refrescar Notas, ordenara los posibles eventos de la lista y generará de nuevo el archivo EVENTOS.

3.15.3. Implementación

La imagen 3.15 es un screenshot de la interfaz para analizar eventos solares. La interfaz fue modificada por Glade-2, generalmente Glade sobreescribe el archivo interface.c, por lo que es necesario volver a escribir las modificaciones externas a este archivo. Cuando hacemos un "refresh" a las Notas, lo primero que hacemos es ordenar las notas por hora (Desde la interfaz gráfica) después sobreescribimos el archivo EVENTOS y finalmente borramos los datos que haya ese día en la base de datos y volvemos a meter todas las notas.

Cuando hacemos un "Redo", borramos todas las notas y volvemos a hacer el análisis desde la hora de inicio de análisis.

3.15.4. Pruebas

Se ejecuto analyze satisfactoriamente. Analyze genera el archivo EVENTOS y también guarda las imagen del flujo analizado en formato PNG.

Se liberó Analyze para pruebas más extensas en el laboratorio y para familiarizar al personal que hará uso de este.

3.16. Ciclo XIV (Paquetes Instalables)

3.16.1. Análisis

El nombre clave del proyecto es xquetzal, pero para fines de producción sera llamado Xitris, "X Interface To RIS".

Xitris esta conformado por 4 módulos principales:

1. Xitris-www: Publicación de datos en Tiempo Real vía Web.
2. Xitris-analyze: Analizador para Eventos Solares en Tiempo Real.
3. Xitris-client: Graficador de Datos en Tiempo Real.
4. Xitris-server: Servidor de Datos.

3.16.2. Diseño

Estructura de Directorio:

1. Xitris-www:

   GNUmakefile: MakeFile Principal para el proyecto xitris-www.

   README: Archivo README.

   INSTALL: Instrucciones de Instalación.

   autogen.sh: Script que verifica si existen las condiciones para poder instalar el proyecto.

   /cpXoc (Gestor de Archivos que usa Xoc).

   - GNUmakefile
   - cpXoc.c: Código fuente.

   /filessystem (Ejemplos de configuración para archivos del sistema).

   - fstab: /etc/fstab.
   - sudouser: /etc/sudouser (Editar con visudo).
   - tunneling.sh: Tablas de ruteo para NAT.

   /public_html (Página web de publicación de datos en Tiempo Real).

   - GNUmakefile
   - index.html
   - RIS.jpg
   - gnu.png
   - /WEB-INF

/classes
xitris.properties: Archivo de configuración para el sitio de publicación.
/xitris
Config.class: Clase para leer la configuración del sitio.
Config.java: Código fuente.
/caracteristicas (Pa'gina de características del RIS)
index.html
escala.jpg
esquemaRIS.jpg
evento.jpg
/contactos (Página de contacto con el staff).
index.html
geofisica.jpg
/eventossolares (Página que despliega nuestra base de datos de eventos solares).
index.jsp: JSP que genera dinámicamente una tabla con todas las mediciones hechas desde el 29 de marzo del 2004.
giveMe.jsp: Interfaz para poder obtener los datos de nuestro servidor.
/grupodetrabajo (Página del grupo de trabajo).
index.html
/historia (Página de la historia del RIS).
index.html
/ligas (Página con ligas a otros sitios de interés).
index.html
/publicaciones (Página con publicaciones del grupo de trabajo).
index.html
/tiemporeal (Página que muestra los datos obtenidos del RIS en tiempo real).
index.jsp: Página de Inicio.
README: Instrucciones para instalar el xdat2txt.
xdat2txt: Binario del xdat2txt.
xdat2txt.tar.gz: Paquete al publico del xdat2txt.
/scripts (Scripts que ejecutan a xitris-www).
GNUmakefile
script.sh
/Xdat2txt (Descompresor del formato xdat).
GNUmakefile
Buffer.h Buffer.c: Biblioteca Buffer.
Guardarh Guardar.c: Biblioteca Guardar.
Objeto.h: Header de Objeto.
xdat2txt.c: Código fuente.
/Xoc (Generador de Imágenes y Archivos para publicar en Tiempo Real).
xoc.glade: Proyecto xoc.
autogen.sh

Makefile
/src (Código Fuente de Xoc)
Makefile
callbacks.h callbacks.c
main.c
suport.h support.c
interface.h interface.c

2. xitris-analyze

GNUmakefile: Make file principal.
INSTALL: Instrucciones de Instalación.
README: Archivo README.
autogen.sh: Script que verifica si existen las condiciones para la instalación.
script.sh: Script para ejecutar xitris-analyze
/visor (Analizador de datos en tiempo real)
visor.glade: Proyecto visor.
autogen.sh
Makefile
/src (Código Fuente de Visor)
Makefile
callbacks.h callbacks.c
main.c
suport.h support.c
interface.h interface.c

3. xitris-client

GNUmakefile: Make file pincipal.
autogen.sh: Script que verifica si existen las condiciones para la instalación.
INSTALL: Instrucciones de Instalación.
README: Archivo README.
/scripts (Scripts de Instalación).
GNUmakefile
xquetzal.sh: Scripts que ejecuta xitris-client.
/xcopy (Copia archivos a /database/actual/)
GNUmakefile
xcopy.c: Código fuente.
/xquetzal (Cliente para graficar y almacenar datos de Xitris).
GNUmakefile
autogen.sh
xquetzal.c : Código fuente.
/ollin (Graficador de Datos en Tiempo Real)

ollin.glade: Proyecto Ollin.
autogen.sh
Makefile
/src (Código Fuente de Ollin)
Makefile
callbacks.h callbacks.c
main.c
suport.h support.c
Buffer.h Buffer.c
Guardar.h Guardar.c
Objeto.h
ImageEngine.h ImageEngine.c: Código fuente del motor de gráficas.
i_addr.h: Cabeceras de configuración para la Red.
/tochtli (Configurador de la Tarjeta ADC en Tiempo Real)
tochtli.glade: Proyecto tochtli.
autogen.sh
Makefile
/src (Código Fuente de Tochtli)
Makefile
callbacks.h callbacks.c
main.c
suport.h support.c
utils.h utils.c: Herramientas de uso para tochtli.

4. xitris-server

GNUmakefile: Make file principal.
INSTALL: Instrucciones de Instalación.
README: Archivo README.
/xitris-simulator (Simulador de LabPC+)
GNUmakefile
i_addr.h: Cabeceras de configuración de Red.
simulator.c: Código Fuente.
/xitris-server (Servidor de datos para LabPC+)
GNUmakefile
i_addr.h: Cabeceras de configuración de Red.
LabPC.h LabPC.c: Biblioteca para usar la tarjeta ADC LabPC+.
Servidor.c: Código fuente del servidor de datos.

3.16.3. Implementación

Se crearon las estructuras que se mencionan en el área de diseño, generándose 4 paquetes independientes con el programa tar y gunzip:

1. xitris-analyze-1p0.tar.gz
2. xitris-www-1p0.tar.gz
3. xitris-client-1p0.tar.gz
4. xitris-server-1p0.tar.gz

también se generó un paquete completo:

1. xitris-1p0.tar.gz

y se pusieron a disposición del publico en 2 repositorios:

http://sourceforge.net/projects/xquetzal
http://cintli.igeofcu.unam.mx/xitris

3.16.4. Pruebas

Se re-instalaron los paquetes en el sitio de producción de manera satisfactoria de acuerdo a las instrucciones de cada paquete.

Se instalaron los 4 paquetes en una misma máquina con diferentes usuarios y funciono correctamente.

3.17. Conclusiones

Finalmente se necesitaron 14 ciclos de producción. Mas de un año de Análisis, Diseño, Construcción y Pruebas para llegar a un producto robusto. Xquetzal/Xitris comenzó a capturar a manera de prueba el Lunes 29 de Marzo del 2004 y actualmente se sigue calibrando el equipo.

Es con este ciclo 14 que se cierra la fase de producción de Xquetzal/Xitris, teniendo como base los mas de 400 días en modo de capturas de prueba y teniendo una base de datos de observaciones coherente a partir de Mayo del 2005.

Es posible visitar el histórico en la pagina web del proyecto (http://cintli.igeofcu.unam.mx). Lo que sigue es plantear nuevos ciclos de desarrollo bajo el esquema de XProgramming para satisfacer las nuevas necesidades del Laboratorio de Interferometria Solar. Pero dejemos las conclusiones finales para el ultimo apartado de esta tesis.

Capítulo 4

Conclusiones Generales

4.1. Resultados

1. Actualmente podemos encontrar el servicio de monitoreo solar en tiempo real en la siguiente página web:

   http://cintli.igeofcu.unam.mx

2. Se crearon 4 meta-programas principales xitris-server, xitris-client, xitris-www y xitris-analyze, que forman parte de la versión final "Xitris Ver 1.0".

3. Este paquete y los anteriores se encuentran en SourceForge.net, disponible al publico bajo licencia GPL en:

   http://sourceforge.net/projects/xquetzal

4. xitris-server, xitris-client, xitris-www y xitris-analyze están instalados y funcionando de lunes a viernes de 9:00 a.m. a 18:00 p.m. en el Laboratorio de Interferometría Solar, controlados por el Técnico Laboratorista Filiberto Matías.

5. Para Junio del 2005, el laboratorio comenzó a funcionar de manera ordenada de acuerdo al esquema de trabajo propuesto por el propio Xitris teniendo como resultados los primeros registros digitales de eventos solares.

4.2. Pruebas

1. De manera independiente hemos bajado los paquetes y han sido instalados en 1 sola máquina produciendo resultados aceptables, lo que demuestra que el proyecto realmente distribuye la carga de trabajo y es altamente configurable.

2. Como ejemplo presentamos 2 eventos capturados por Xitris el 1 de Junio del 2005, marcados con una cruz en la figura 4.1 y corroborado por el Mees Solar Observatory [36] (ver figura 4.2). La gráfica que presenta el Mees Solar Observatory esta compuesta por sus instrumentos (MWLT, COR e IVM) y por el registro del satélite espacial GOES [47]. Se puede observar que el evento registrado por el GOES ocurre simultáneamente en el RIS. Recordemos que la longitud de onda a la que capturan son diferentes, por un lado el RIS captura a una longitud de onda aproximada de 4 cm y el GOES en rayos X (Aproximadamente 2 Amstrongs).

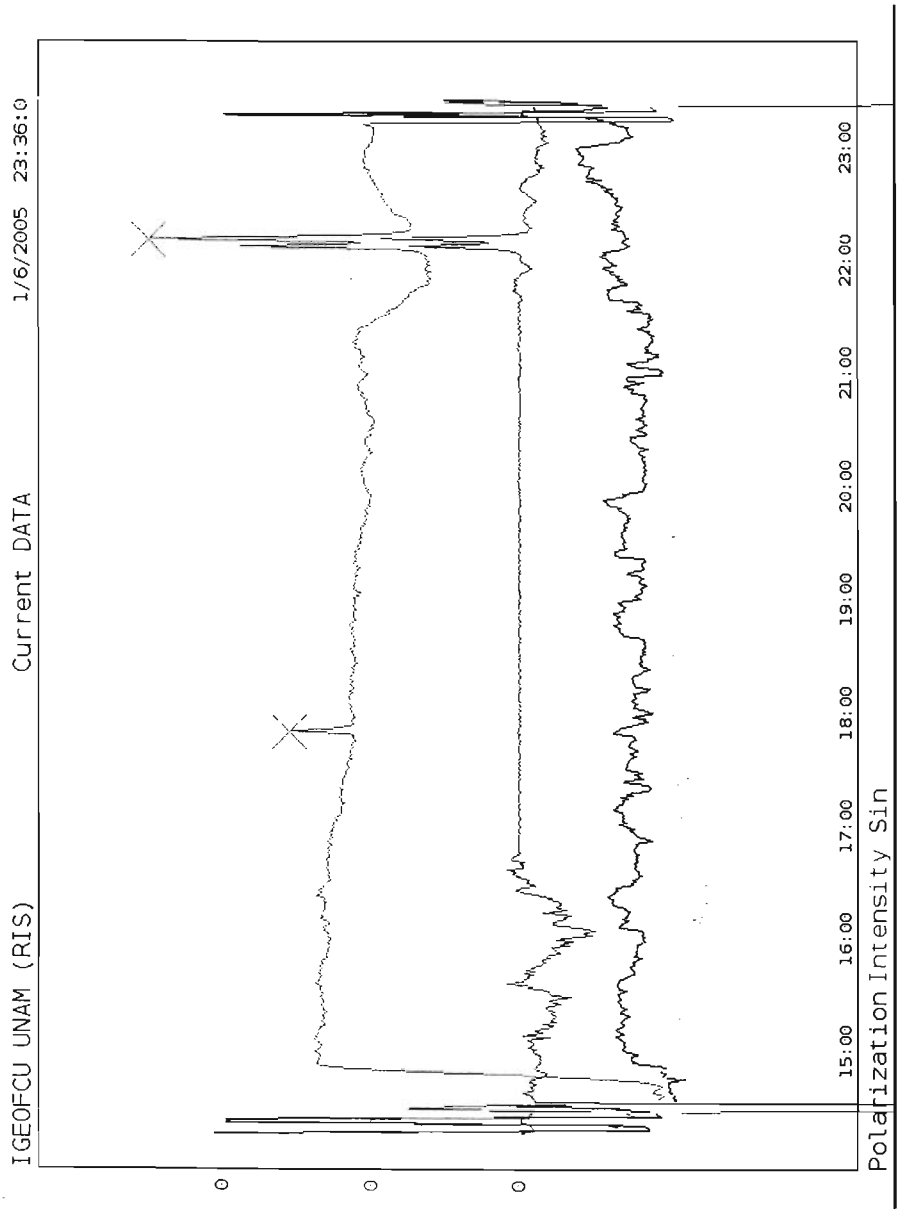

Figura 4.1: Flujo solar capturado por Xitris el 1 de Junio del 2005.

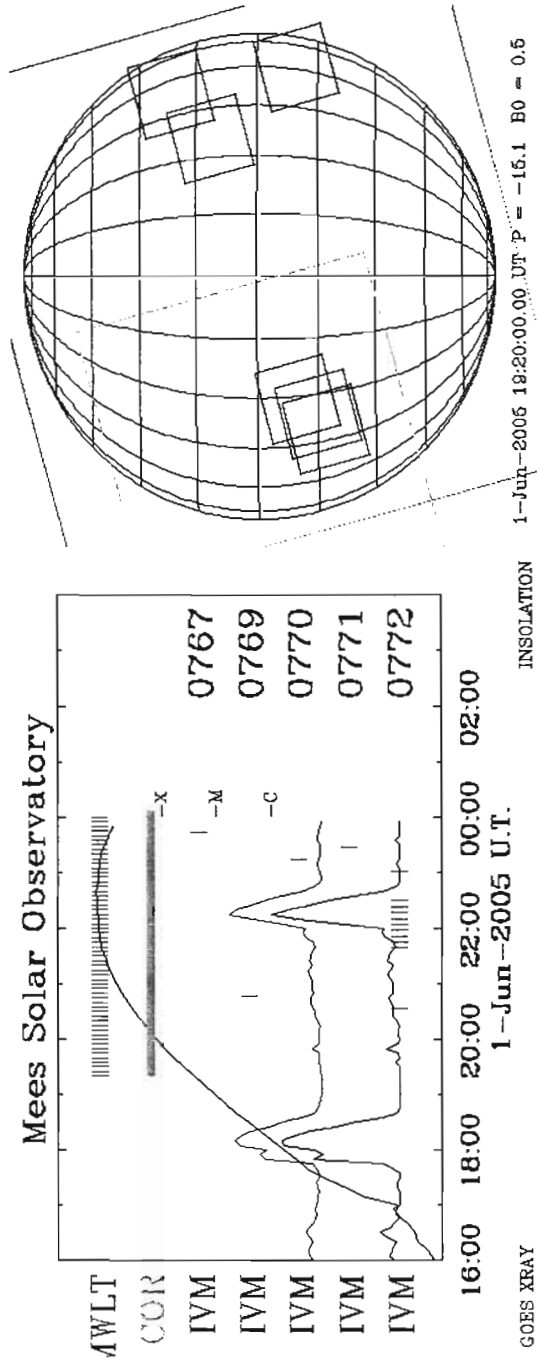

Figura 4.2: Flujo solar capturado por el Mees Solar Observatory el 1 de Junio del 2005.

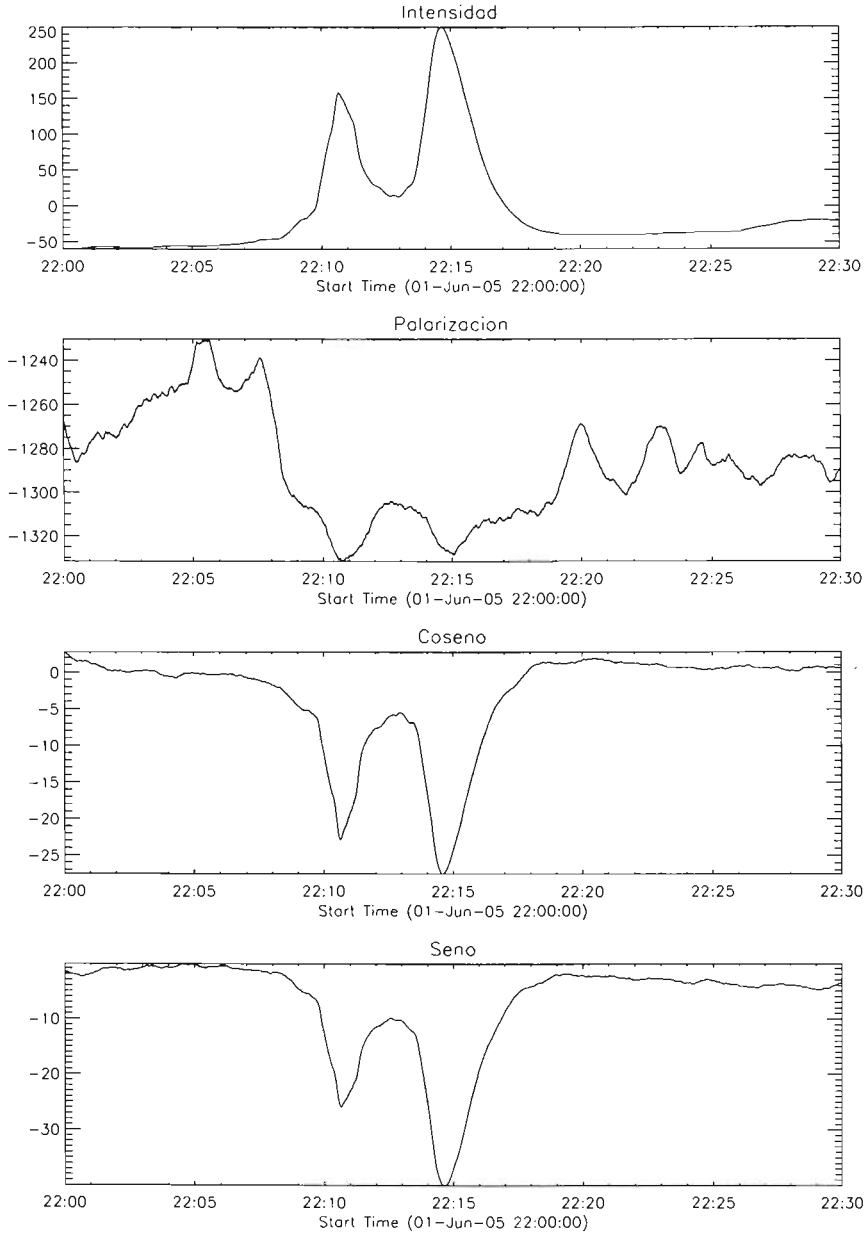

Figura 4.3: Evento del evento solar del 1 de Junio del 2005.

3. Estos registros fueron procesados y analizados por el Dr. Alejandro Lara y dieron como resultado una serie de gráficas que demuestran el buen funcionamiento tanto del RIS como de Xitris. La figura 4.3 muestra un acercamiento del segundo evento, de las 22:00 a las 22:30 UT. De la parte superior a la inferior se encuentran los flujos registrados en los canales (a) intensidad o flujo total, (b) polarización, (c) seno y (d) coseno.

   Una vez calibrados los datos, se puede obtener el flujo y la posición de la región en la que tuvo lugar el evento (dentro del disco solar). La figura 4.4 muestra el flujo de fuentes puntuales (panel superior)[1] y su posición (panel inferior) en segundos de arco con respecto al ecuador solar, en función del tiempo. Estas gráficas fueron generadas por IDL y demuestran que los datos obtenidos de Xitris pueden ser procesados fácilmente con las herramientas de trabajo con las que cuenta el Laboratorio.

4.3. Conclusiones y Consideraciones Finales

1. Un punto importante que queremos destacar son los ciclos de producción del software. Tardamos aproximadamente 6 meses en definir y llegar a una versión que enfocaba las necesidades del laboratorio pero que por motivos de diseño se tuvo que abandonar. Sin embargo como el código fue pensado de manera modular y reutilizable, fue relativamente fácil reagrupar esos componentes y generar un nuevo proyecto. Por esa razón, después de 3 ciclos de desarrollo en XP programming ya teníamos un producto usable. El primer proyecto no fue un trabajo infructuoso, al contrario, sentó las bases para el análisis y el diseño del nuevo concepto de Xquetzal/Xitris, como se puede apreciar a lo largo del trabajo, es común encontrar referencias hacia el proyecto anterior.

2. Xquetzal/Xitris no solo se enfoca a la adquisición de datos provenientes del Sol, también puede ser usado como graficador para cualquier fuente de energía que tenga como interfaz una tarjeta Convertora Análogo Digital.

3. Xquetzal/Xitris se tiene contemplado como modelo base para el radiotelescopio RT5, un proyecto conjunto del INAOE y del Instituto de Geofísica de la UNAM, su papel sería el mismo que para el RIS (Imagen 4.5), un programa para "Adquirir, Procesar, Almacenar, Desplegar y Publicar Datos en Tiempo Real".

4. Dejamos abierta la posibilidad para expandir a Xitris tanto como sea posible: Soporte para nuevas tarjetas convertoras, desarrollo de módulos de análisis, soporte para exportar a formatos como el FITS, calibración automática del RIS, automatización de la maquinaria del RIS, etc...

5. Finalmente Xquetzal/Xitris es una alternativa de bajo costo para el manejo de una tarjeta comercial, que de otra forma, estaría sujeta al precio de lista injustificado del Software Propietario.

[1] En unidades arbitrarias

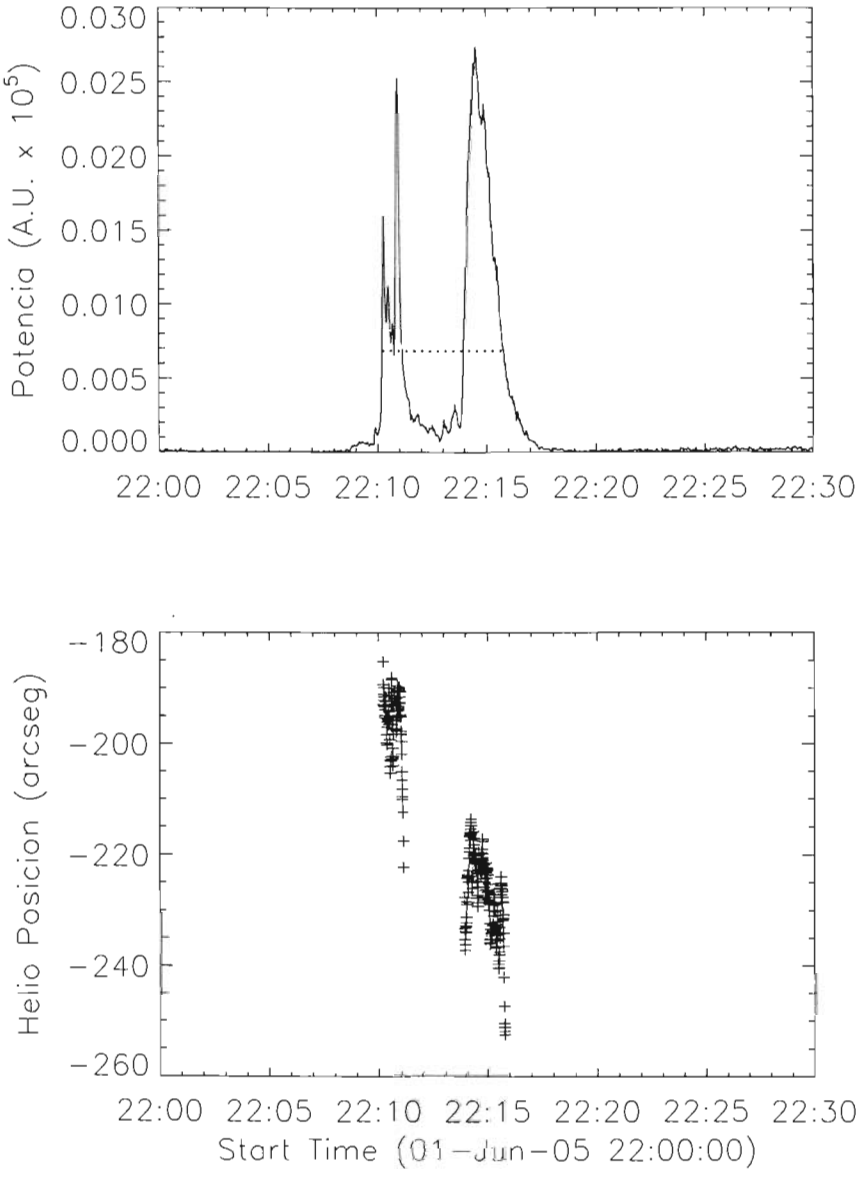

Figura 4.4: Analisis del evento solar del 1 de Junio del 2005.

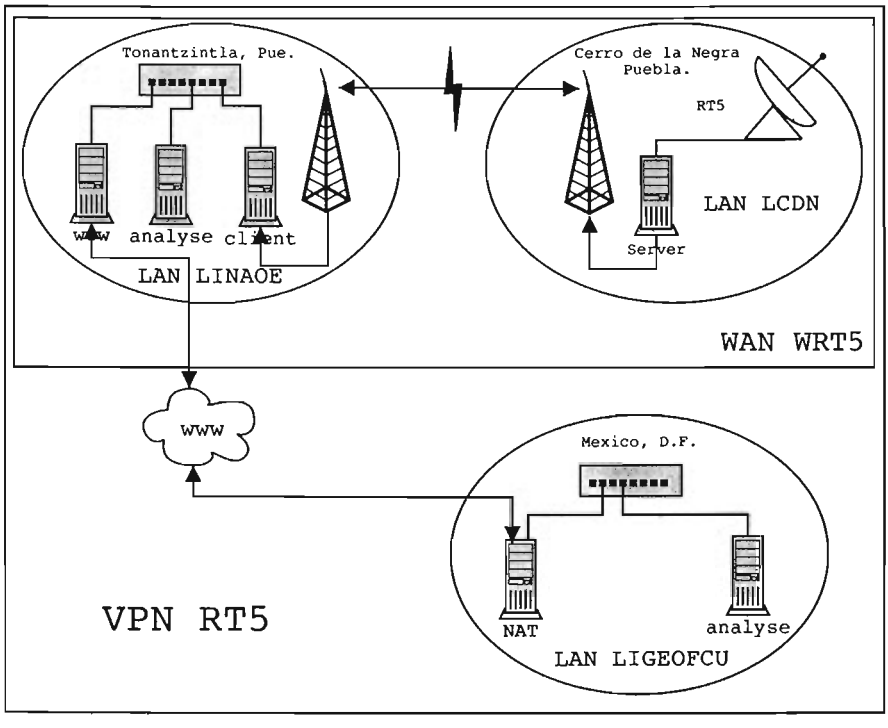

Figura 4.5: Diagrama general para el RT5.

Apéndice A

DRU

Información:	Documento de Requisitos de Usuario		
Proyecto:	Xquetzal	Versión:	01022003
Preparador por:	Víctor De la Luz	Fecha:	1 / 02 / 2003
Recibido por :	A. Lara	Status:	Aprobado

A.1. Introducción

A.1.1. Propósito

Este documento tiene por propósito describir los requisitos de usuario para el sistema *xquetzal*. Este documento se basa en el estándar ESA PSS-05-0, que define un método para determinar y especificar los requerimientos de usuario de un proyecto.

A.1.2. Alcance

Captura de la tarjeta decodificadora LabPC+ de National Instruments por 4 canales diferentes. Los datos obtenidos por el Radio Interferómetro Solar RIS se grafican en pantalla y se guardan de acuerdo a el año/mes/dia de su captura, estos datos también se podrán ver a través de Internet y serán analizados para saber si en el curso de la captura a ocurrido un "Evento Solar", todo en tiempo real.

La figura A.1 representa el ámbito general del sistema.

A.1.3. Definiciones

1. ESA: European Space Agency.
2. RIS: Radio Interferómetro Solar.
3. Plataforma: Sistema Operativo, ejemplo: GNU/Linux, Microsoft Windows, Sun Solaris, etc.

A.1.4. Visión General

La segunda parte de este documento presenta una descripción general del proyecto, que incluye el prospecto del producto, las características del usuario, restricciones, supuestos, dependencias y ambiente operacional. La tercera parte describe con mayor detalle los requisitos del usuario.

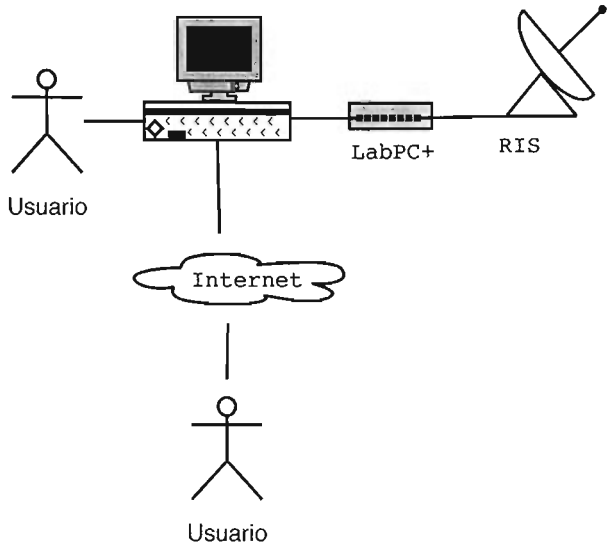

Figura A.1: Ámbito general del sistema.

A.2. Descripción General

A.2.1. Prospecto del producto:

Software que se encargara de graficar, almacenar, procesar y publicar datos procedentes del RIS en tiempo real.

A.2.2. Características del usuario:

Lista que describe el tipo de usuarios al que va dirigido el software:

1. Técnico de Laboratorio: Usualmente es un solo técnico el encargado de monitorear la señal. Nivel de conocimiento en el ámbito computacional: bajo. Este usuario observara la señal en el monitor y podrá distinguir entre una ráfaga solar y una calibración o error en el equipo. Esta persona sera la encargada de iniciar y detener el programa.

2. Investigador: Son varios investigadores que pueden acceder a los datos, ellos se interesan principalmente en los datos crudos. En particular es importante la frecuencia de muestreo, el estado de calibración del equipo y la hora exacta en que se esta desarrollando la captura. Niveles de conocimiento en el ámbito computacional: en general alto.

3. Publico en General: A este tipo de usuarios les interesa ver los datos de una manera gráfica y concentrada, en ella se puede apreciar el cambio en el flujo de energía del sol. Niveles desconocimiento en el ámbito de la computación: de medio a bajo.

A.2.3. Restricciones Generales.

1. Disponibilidad para plataformas GNU/Linux.
2. El tiempo de respuesta para la conversión de datos Análogo Digitales estará en función de la capacidad de respuesta del equipo de computo con el que contamos, siendo este un equipo viejo, no esperamos una gran velocidad en la frecuencia de conversión.
3. El software funciona con la tarjeta LabPC+, pero estará construido de forma modular para alguna futura implementación con hardware distinto.

A.2.4. Ambiente Operacional.

En el Instituto de Geofísica de la UNAM se encuentra el Radio Interferómetro Solar (RIS), este Interferómetro tiene como salida de datos un graficador de origen Ruso. Este graficador dibuja una linea quebrada y continua sobre papel graduado a un intervalo de aproximadamente 0.5 seg. Esta linea representa la intensidad con que registra los datos el RIS.

Posteriormente estos datos son almacenados y catalogados para su Análisis.

Es de vital importancia para este laboratorio tener los datos de una manera mas práctica, es decir tenerlos de manera digital para analizarlos con paquetes de alto nivel como los son IDL o GNUplot.

Para ese propósito el Laboratorio cuenta con una interfaz ISA de origen Estadounidense de la marca National Instruments llamada LabPC+.

Esta tarjeta es capaz de convertir datos análogos a digitales y además puede estar monitoreando 4 canales a la vez.

El Laboratorio cuenta con otros 3 canales de comunicación (a parte del mencionado canal de intensidad) los cuales han estado en desuso por falta de graficadores.

La infraestructura del Laboratorio es algo pobre, cuenta con 2 computadoras (486 y Pentium) con poca memoria y además lentas.

Este software implementara la ultima fase del RIS que es el de almacenamiento y despliegue de datos.

A.3. Especificación de requisitos

A.3.1. Requisitos de Capacidad

1. El software es modular y flexible para que pueda trabajar sin mucha dificultad con diferentes tipos de tarjetas decodificadoras.
2. Captura de Datos en tiempo real en los 4 canales con un intervalo máximo de 0.001 segundos por dato.
3. Visualizar los datos de manera gráfica en tiempo real.
4. Controlar la tarjeta de decodificación LabPC+ en tiempo real.
5. Guardar los datos y las modificaciones en archivos que sean fácilmente utilizables así como en formatos comprimidos para su eficaz procesamiento.
6. Crear un esquema para guardar la captura en forma estándar y crear una base de datos para su rápido acceso.

7. Crear una pagina de Internet para publicar los datos que se vayan generando y un sistema de búsqueda y envió.
8. Crear una interfaz para monitorear los cambios en la señal y así poder actuar de manera correcta ante un evento solar o ante una simple calibración del equipo.
9. El desarrollo debe estar bajo los términos de GPL [15] (General Public License) para su uso sin restricciones y sin problemas de Licenciamiento.
10. Usar herramientas GNU/Linux. para el desarrollo de la aplicación.

A.3.2. Restricciones

1. Los datos almacenados deben de estar en formatos que sean legibles para cualquier investigador sin importar la plataforma que use.
2. Tener un ambiente de seguridad confiable y realizar esquemas para copias de seguridad.
3. La sincronización con la hora real debe ser prioritario para su buen funcionamiento.

Apéndice B

DRS

Información:	Documento de Requisitos de Software		
Proyecto:	Xquetzal	Versión:	03022003
Preparador por:	Víctor De la Luz	Fecha:	3 / 02 / 2003
Recibido por :	A. Lara	Status:	Aprobado

B.1. Introducción

B.1.1. Propósito

Este documento tiene por propósito describir los requisitos de software para el sistema *xquetzal*. Este documento se basa en el estándar ESA PSS-05-0, que define un método para determinar y especificar los requerimientos de software de un proyecto.

B.1.2. Alcance

El sistema *xquetzal* leerá de la tarjeta convertora Análogo Digital LabPC+ de National Instruments 4 canales (Intensidad, Polarización, Seno y Coseno) de acuerdo a una configuración pre-establecida que podrá ser modificada en cualquier momento de la captura, cada canal será graficado en la pantalla, al mismo tiempo, el sistema guardará los datos y analizará la señal de cada canal para ver si hay algún evento solar, si existe un evento, lo reportará a una base de datos y cambiará la frecuencia de muestreo para su posterior análisis. La parte de la publicación la hará un sitio web.

B.1.3. Definiciones

1. ESA: European Space Agency.
2. SR: Software Requeriment.
3. UR: User Requeriment.
4. RIS: Radio Interferómetro Solar.
5. IGEOF: Instituto de Geofísica.
6. UNAM: Universidad Nacional Autónoma de México.

B.1.4. Visión General

El proyecto xquetzal fue dividido en dos componentes principales: *xquetzal-caption* y *xquetzal-publish*, uno encargado de la captura, almacenamiento y procesamiento de la información y el otro encargado de la publicación.

La segunda parte de este documento presenta una descripción general del proyecto, que incluye el prospecto del producto, las características del usuario, restricciones, supuestos, dependencias y ambiente operacional. La tercera parte describe con mayor detalle los requisitos del software.

B.2. Descripción General

B.2.1. Relación con proyectos actuales

En este momento no se están desarrollando proyectos dentro del laboratorio que tengan que ver con xquetzal. Solo hay algunas aproximaciones por crear ventanas en donde se intenta graficar la señal proveniente de la tarjeta pero sin mucho éxito. Lo mas importante de estas pruebas es que nos dan un esquema de como podría funcionar el sistema.

B.2.2. Función y Propósito

Software que se encargara de graficar, almacenar, procesar y publicar datos procedentes del RIS en tiempo real.
Su propósito final sera establecer un Monitor Solar moderno dentro del Laboratorio de Interferometria Solar.

B.2.3. Consideraciones Ambientales

El lugar en donde se encuentran las computadoras es estrecho, con poca ventilación. En los días de verano alcanza una temperatura de hasta 30 grados centígrados y en Invierno casi nunca baja de los 5 grados centígrados, no cuenta con calefacción ni aire acondicionado pero si cuenta con nobreaks.

B.2.4. Relación con otros sistemas

Este sistema se relaciona principalmente con:

1. Sistemas de Análisis Numéricos: IDL es uno de ellos, tiene potentes filtros para convertir diferentes tipos de formatos, en esencia los datos obtenidos del RIS deben ser formateados para que IDL pueda leerlos fácilmente.

B.2.5. Restricciones Generales

1. Navegadores Web: Es importante concientizarnos de usar estándares como lo son XHTML para que cualquier navegador pueda visualizar correctamente la información y pueda acceder fácilmente a ella, ya que los investigadores que realizaran las búsquedas en nuestro sistema usan todo tipo de plataformas: VAX, Solaris, Linux, MacOS X, Windows; lo que significa una gran cantidad de navegadores usados.

2. Datos: Debe haber un formato para los datos que sea estándar para cualquier sistema (ASCII).

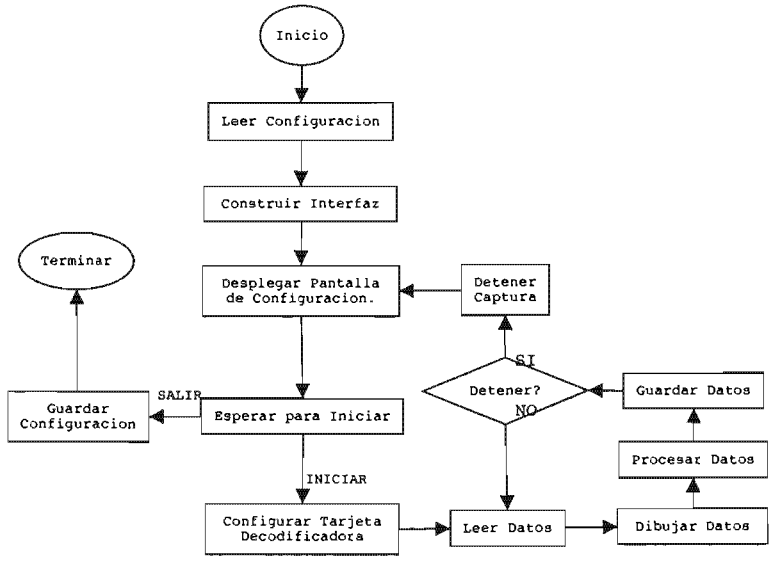

Figura B.1: Modelo Lógico de xquetzal-caption

B.2.6. Descripción del Modelo

Este modelo funcionaria de la siguiente manera (Ver imagen B.1):

1. El programa inicia leyendo las configuraciones de un archivo.
2. De acuerdo con estas configuraciones se construye y despliega la interfaz.
3. El usuario tiene a su disposición una ventana con las posibles configuraciones que estén disponibles como lo es la amplificación y la frecuencia de muestreo.
4. Una vez que el usuario haya configurado la interfaz puede proceder a iniciar la captura o en su defecto terminar la aplicación.
5. Si desea iniciar la captura, xquetzal-caption se comunica con la tarjeta y la configura de acuerdo a lo establecido en la interfaz.
6. Una vez configurada la tarjeta se le solicitan datos, cada vez que un dato este listo, se manda graficar y procesar a la pantalla y posteriormente se guarda en el disco.
7. Este proceso se repite con cada dato, teniendo en cuenta que después de cada ciclo, xquetzal tiene que verificar si se debe detener o continuar.
8. Si se continua se repite el ciclo.
9. Si se detiene, avisa a la tarjeta que hay que detenerse y regresa a la pantalla de configuración.

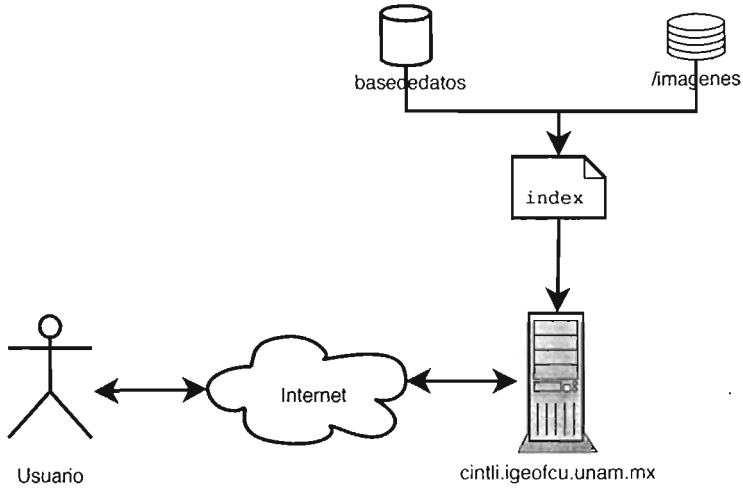

Figura B.2: Modelo Lógico para la Publicación Web.

10. Si se desea terminar, en la pantalla de configuración puede salir. Esta orden hace que xquetzal guarde sus actuales configuraciones y termine correctamente.

En el modulo de procesamiento de datos es en donde la señal puede ser monitoreada para los efectos del proyecto. Aquí pueden incluirse los módulos de aviso por evento solar, calculo de media o alguna transformación que se le quiera dar a la señal.

Para el caso de publicación en tiempo real la figura B.2 explica gráficamente como funcionaria:

1. Un usuario accede por medio de Internet a la dirección http://cintli.igeofcu.unam.mx.
2. Un servidor web responde con una pagina html en la cual mediante algún CGI da servicios de búsqueda en nuestra base de datos y también da acceso a imágenes y archivos que se estén generando en este momento.

Estas dos aplicaciones trabajando sincronizadamente producen los resultados deseados. Es decir, uno de ellos captura y despliega datos en tiempo real, produciendo los archivos y las imágenes necesarias. Mientras que el otro atiende peticiones de usuarios externos y les da acceso a estas imágenes y archivos.

B.3. Especificación de requisitos

B.3.1. Requisitos Funcionales

Para xquetzal-caption:

1. RS0: Leer Configuración: El sistema debe ser capaz de reconstruir los estados anteriores para poder configurarse.

Nivel de Necesidad: Negociable.
Prioridad: Media
Estabilidad en tiempo de desarrollo: Estable

2. RS1: Construir Interfaz: El sistema debe ser capaz de construir la interfaz y poner como valores iniciales lo obtenido en la fase de configuración o en su defecto, iniciar valores predeterminados.
Nivel de Necesidad: No Negociable.
Prioridad: Alta
Estabilidad en tiempo de desarrollo: Estable

3. RS2: Configurar LabPC+ en el entorno gráfico: Una vez desplegada la pantalla de configuración, el usuario puede modificar la configuración de xquetzal-caption, pudiendo modificar los siguientes aspectos:

   a) La frecuencia de muestreo que va de 0.001 a .5 segundos
   b) La amplificación por hardware.
   c) La escala de la graficación.
   d) Un valor Delta que dirá si hay un evento solar o no.
   e) El numero de datos para la media, esto servirá para saber si hay un evento solar o no.
   f) Desplegar la media.

   Nivel de Necesidad: No Negociable.
   Prioridad: Alta
   Estabilidad en tiempo de desarrollo: Estable

4. RS3: Iniciar la Captura: Cuando se inicia la captura, el software deberá:

   a) Configurar la tarjeta LabPC+.
   b) Desplegar pantalla de Graficación.
   c) Revisar si en la tarjeta hay datos disponibles.
   d) Si los hay, graficarlos.
   e) Procesar los datos de acuerdo a las configuraciones (En este momento solo se puede asegurar que el procesamiento es revisar si hay un evento solar o no).
   f) Guardar los Datos.
   g) Revisar si terminamos.

   Nivel de Necesidad: No Negociable.
   Prioridad: Alta.
   Estabilidad en tiempo de desarrollo: Estable

5. RS4: Terminamos captura: Cuando el botón de parada sea clickeado se detendrá la captura, esto lo hace, sealando en una variable (semáforo) que el loop de captura a terminado. Nivel de Necesidad: No Negociable.
   Prioridad: Alta
   Estabilidad en tiempo de desarrollo: Estable

6. RS5: Terminar xquetzal-caption: Al terminar, guardamos las configuraciones y terminamos correctamente. Nivel de Necesidad: Negociable.
   Prioridad: Media
   Estabilidad en tiempo de desarrollo: Estable

Para xquetzal-publish:

1. RS5: Petición Web: Cuando hay una petición web (e.d. http://cintli.igeofcu.unam.mx) el servidor responde con la pagina de inicio, la cual contendrá una liga al buscador de eventos solares y una imagen con los datos capturados en tiempo real. Nivel de Necesidad: No Negociable.
   Prioridad: Alta
   Estabilidad en tiempo de desarrollo: Estable

2. RS6: Imagen en Tiempo Real: La imagen debe estar en un formato estándar (png o jpg) y debe contener información necesaria para saber que se esta capturando.
   Nivel de Necesidad: No Negociable.
   Prioridad: Alta
   Estabilidad en tiempo de desarrollo: Estable

3. RS7: Buscar Eventos: Cuando una búsqueda sea solicitada, buscaremos en nuestra base de datos de acuerdo a los posibles opciones:

   a) Fecha: Periodo de fechas.

   b) Intensidad: Intensidad del evento solar.

   c) Duración: Por la duración del evento.

   d) Otras: Por otras características del evento, estas características serán definidas posteriormente, pues aun esta en fase de desarrollo el sistema de detección de eventos solares.

   Nivel de Necesidad: Negociable.
   Prioridad: Media
   Estabilidad en tiempo de desarrollo: Estable

4. RS8: Solicitud de Datos: Una vez encontrado los datos, el Usuario podrá guardar los datos en formato ASCII o con alguna aplicación que los convierta a imágenes tipo png.
   Nivel de Necesidad: Negociable.
   Prioridad: Media
   Estabilidad en tiempo de desarrollo: Estable

B.3.2. Requisitos de Desempeño

1. RS3: Alto desempeño, por ser el requisito que mas recursos podría utilizar.

B.4. Requisitos de Interfaz

B.4.1. Interfaces de Hardware

1. Computadora Compaq Deskpro (Cintli).
   a) Procesador Pentium MMX a 166MHz
   b) 46Mb en RAM
   c) 12Gb Hdd.
   d) Bus Master a 66MHz.
   e) Tarjeta de Vídeo Cirrus Logic GD 5430.
2. Computadora Armada con procesador 486 (Ris).
3. Impresora LaserJet 5L
4. LabPC+ Board:
   a) Aproximación sucesiva de 12 bits para conversiones Análogo/Digitales.
   b) 8 Canales de Entrada análogos.
   c) 2 Convertidores Digitales/Análogos de 12 bits.
   d) 24 Lineas TTL.
   e) Interfaz ISA (IBM Personal Computer Compatible).
   f) 3 contadores (Timers) internos de 16 bits.

B.4.2. Interfaces de Software

1. SO: Debían GNU/Linux potatoe.
2. Lenguajes de desarrollo:
   a) Lenguaje de programación para xquetzal-caption: C.
   b) Lenguajes de programación para xquetzal-publish: JSP, XHTML 1.0.
   c) Lenguaje para base de datos: SQL.
3. Ambientes de desarrollo:
   a) Para xquetzal-caption: Glade, GTK 1.2, GDK 1.2, gcc, emacs.
   b) Para xquetzal-publish: BlueFish, JDK 1.4.
4. Software Involucrado:
   a) Jakarta-Tomcat 4 como servidor web.
   b) Postgresql como servidor de Base de Datos.
5. Formatos Involucrados:

a) PNG: Portable Network Graphics: Las imágenes producidas tendrán este formato.

b) ASCII: Los datos públicos estarán en este formato por ser el formato universal mas usado (Aun después de UTF-8).

c) XML: Cuando se trate de guardar los logs del sistema, utilizaremos esta convención.

B.4.3. Interfaces de Comunicación

1. Peticiones IP para postgresql.
2. Para la interacción con la tarjeta LabPC+ nos comunicaremos directamente por los puertos de la tarjeta.

B.5. Requisitos Operacionales

El programa esta planteado para emular el ambiente de trabajo que se lleva a cabo en el laboratorio. Para empezar, el programa solo estará activo en el día, es decir, de las 8 a.m. a las 6 p.m. normalmente estará monitoreado por el laboratorista técnico. El programa iniciará, preguntará al sistema la fecha y la hora y lo guardará en el log, una vez hecho esto, se procederá a configurar de acuerdo a protocolos que se vayan generando internamente. Una vez configurado el sistema, este procederá a su ejecución. El laboratorista presionará el botón de iniciar y xquetzal-caption comenzará a trabajar. Cuando el flujo cambie de estado, es decir, baje o suba de acuerdo a ciertas cotas, emitirá una seal visual indicando que un posible evento está ocurriendo, cuando esto pase, en teoría el laboratorista podrá distinguir gracias a su experiencia si se trata de un evento solar o simplemente hubo un desperfecto en el RIS. Una vez señalado el origen del posible cambio de estado, el sistema guarda esto en el log y prosigue de manera normal la captura, teniendo en cuenta que la captura se seguirá en segundo plano. Una vez terminado el día laboral, el laboratorista detendrá xquetzal-caption y se saldrá del programa. Mientras que xquetzal-publish estará atento todo el tiempo, sin tener que contar con asistencia, pues esta diseñado para que funcione de manera automática.

B.6. Requisitos de Recursos

De acuerdo con experiencias previas, podemos asegurar que con el hardware con el que contamos no se pueden tener muchas expectativas de éxito, pues la carga de trabajo es demasiada para una máquina Pentium con menos de 100 Mb en RAM, yo esperaría tener un servidor o una máquina de por lo menos 1GHz para poder soportar el hosteo y el despliegue de gráficas en tiempo real.

B.7. Requisitos de Verificación

Cada vez que un nuevo modulo sea implementado o modificado, todo el sistema debe ser probado de la siguiente manera:

1. Reproducir una señal de por lo menos 1 Volt con un generador de Onda.
2. Verificar en la pantalla, los datos generados y tienen que coincidir con el voltaje generado.
3. Verificar estos datos con los datos guardados.

4. Probar los tiempos de respuesta con timers externos, e internos, ya sea con los timers de la tarjeta LabPC+ así como los que cuenta el procesador.
5. Hacer estas pruebas tanto para Voltajes positivos como negativos.
6. Hacer las pruebas por lo menos 2 veces, estando seguro de compilar y verificar.

B.8. Requisitos de Pruebas de Aceptación

Una vez que las pruebas de verificación hayan concluido, tiene que escribirse correctamente la documentación tanto técnica como operacional de la parte que fue construida o modificada. Una vez que la documentación este lista, el software tiene que estar en funcionamiento y no presentar ningún tipo de error para el ciclo en el que se este trabajando y que el cliente, en este caso el Dr. Alejandro Lara Sánchez de el visto bueno.

B.9. Requisitos de Documentación

La documentación esta constituida por:

1. Documentación en el Código: El código tiene que estar documentado en los aspectos más importantes y críticos de la aplicación, así como llevar una lista de códigos para el caso de memoria compartida y el formato de los datos.
2. API: La api del desarrollo, que es la documentación para los programadores.
3. Manual de Usuario: Un manual que detalla el funcionamiento del Software.
4. Manual de Procedimientos: Un manual para el laboratorista o el que este a cargo para enumerar los pasos que sean necesarios para llevar a cabo la captura diaria.

B.10. Requisitos de Seguridad

1. Como la máquina va a servir de host, necesitamos políticas de seguridad para estar monitoreando el trafico en nuestro servidor.
2. Los datos serán públicos para cualquier persona que lo desee.
3. El uso de xquetzal sera restringido solo al personal autorizado del Laboratorio, creando para este fin, cuentas especiales en las máquinas para que solo un usuario pueda acceder a estos servicios.
4. Crear una lista de usuario restringidos que puedan modificar o actualizar la base de eventos.

B.11. Requisitos de Transportabilidad

El proceso de desarrollo, creara un modelo transportable dentro de entornos GNU/Linux. Las promesas que hace GKT para su portabilidad pueden hacer factible la migración entre sistemas Linux-¿Windows, aunque el principal enfoque del proyecto es respetar los estándares y las licencias publicadas por la FSF (Free Software Fundation) o por lo menos de Open Source. Sin embargo se hace factible el hecho de transportar el software a otra máquina con sistemas tipo UNIX.

B.12. Requisitos de calidad

El software es de calidad por:

1. Su Ingeniería de Software.
2. Porque sera implementado por una sola persona que estará al tanto de todo el proceso de desarrollo.
3. Porque habrá testers (El laboratorista y las personas que estén visitando el sitio así como el Investigador Dr. A. Lara) que estarán probando y criticando cada parte del software y no permitirán que algo falle.

B.13. Requisitos de confiabilidad

El software es confiable si no presenta ninguna falla o muera misteriosamente en el transcurso de su funcionamiento que esta proyectado de 9:00 a.m. a 6:00 p.m. En cualquier otro caso, el programa puede tener fallas por cuestiones horarias. Pero solo por este motivo y por ningún otro.

B.14. Requisitos de mantenibilidad

Se plantea un proyecto modular que aunque cada modulo sea interdependiente, cada uno puede ser modificado sin afectar a los demás. La idea principal e importante es que vamos a tener módulos para guardar los datos, para graficarlos, uno mas para procesarlos, otro para crear las interfaces gráficas y uno mas para interactuar con la tarjeta LabPC+. Para interactuar con la tarjeta decodificadora abstraemos a la tarjeta en pocos objetos:

1. Configuración.
2. Inicia Captura.
3. Dato Listo.
4. Dame Dato.
5. Hay Error
6. Termina Captura

Con esta abstracción, la tarjeta decodificadora finalmente es un elemento reemplazable.

B.15. Requisitos de Seguridad

En caso de falla, el software siempre estará guardando la información cada determinado tiempo (Cosa de algunos segundos), por lo que si el software fallara, la perdida podría ser mínima, tal vez de algunos minutos o menos. Lo importante es que si el software falla, tenga la posibilidad de reconstruir su ultimo estado (Ayudado por algún usuario) y continuar capturando. Reduciendo al mínimo el daño.

Apéndice C

DDA

Información:	Documento de Diseño Arquitectónico		
Proyecto:	Xquetzal	Versión:	03032003
Preparador por:	Víctor De la Luz	Fecha:	3 / 03 / 2003
Recibido por :	A. Lara	Status:	Aprobado

C.1. Introducción

C.1.1. Propósito

Este documento tiene por propósito describir los requisitos de software para el sistema *xquetzal*. Este documento se basa en el estándar ESA PSS-05-0 Issue 2, de febrero de 1991, que define un método para determinar y especificar los requisitos de usuario de un proyecto.

C.1.2. Alcance

El sistema *xquetzal* leerá de la tarjeta convertora Análogo Digital LabPC+ de National Instruments 4 canales (Intensidad, Polarización, Seno y Coseno) de acuerdo a una configuración pre-establecida que podrá ser modificada en cualquier momento de la captura, cada canal sera graficado en la pantalla, al mismo tiempo, el sistema guardara los datos y analizara la señal de cada canal para prever si hay algún evento solar, si existe un evento, lo reportara a una base de datos y cambiara la frecuencia de muestreo para su posterior análisis. La parte de la publicación la hará un sitio web.

C.1.3. Visión General

El proyecto xquetzal esta en dos componentes principales: *xquetzal-caption* (Imagen C.1) y *xquetzal-publish*(Imagen C.2), uno encargado de la captura, almacenamiento y procesamiento de la información y el otro encargado de la publicación.

C.2. Contexto del Sistema

C.2.1. Definición de Interfaces Externas

Los datos obtenidos en la captura serán procesados por IDL[43].

Existen 2 políticas de seguridad, una publica y otra privada. La política de seguridad publica permitirá observar los datos e imágenes de nuestra base de datos, sin posibilidad de

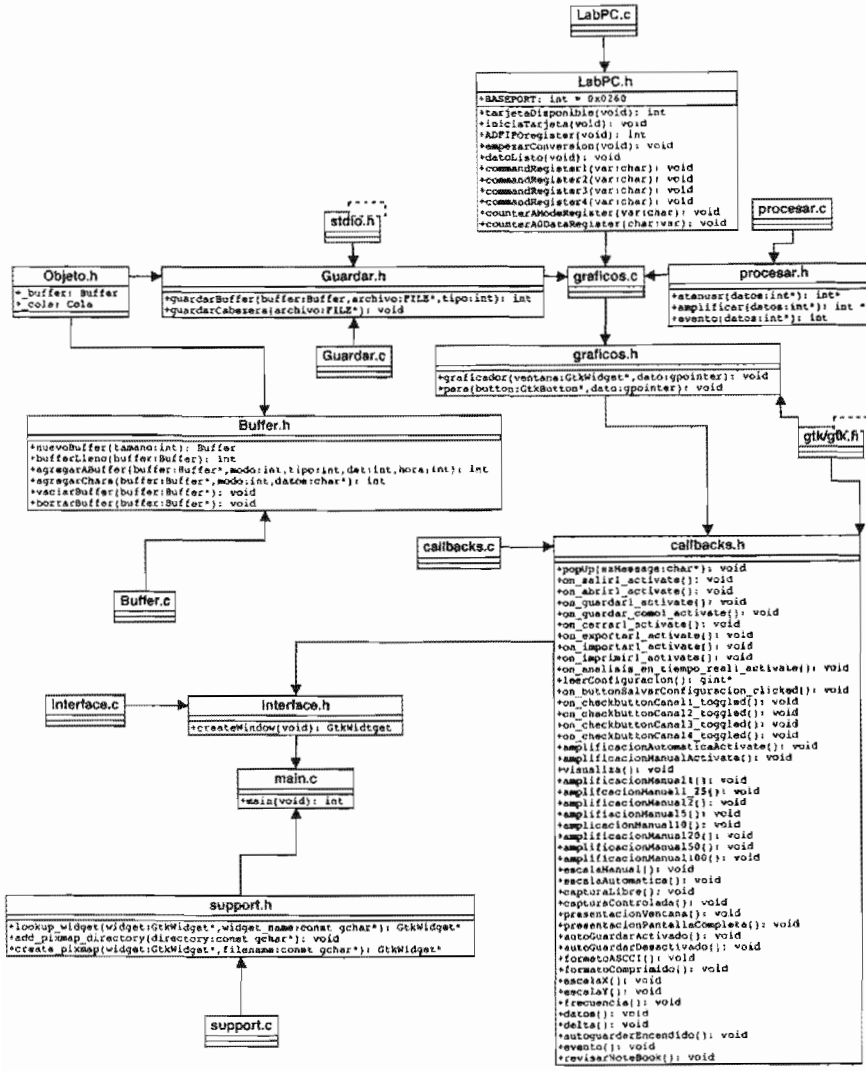

Figura C.1: Diseño Arquitectónico de xquetzal-caption

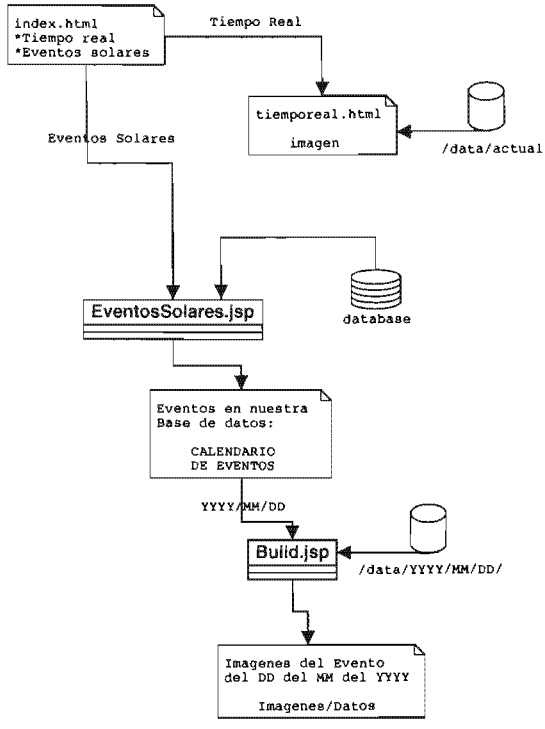

Figura C.2: Diseño Arquitectónico de xquetzal-publish

modificarlos, solo tendrán acceso mediante una interfaz externa (Pagina web). Las políticas Internas de seguridad son tener acceso al software mediante el uso de cuentas sin privilegios de root.

C.3. Diseño del Sistema

C.3.1. Método del Diseño

Se espera que el software pueda ser modificado fácilmente en caso de que se usa una tarjeta convertora más moderna, a su vez se espera que la aplicación sea transportable a otros instrumentos de medición análogos.

C.3.2. Modelo Físico

Para xquetzal-caption (imagen C.1):

main.c Se encarga de llamar al constructor de la Interfaz (createWindow() en interface.h); desplegar la interfaz en pantalla (gtk_widget_show()); leer las configuraciones del disco y a partir de estas configuraciones crear una estructura lógica en la memoria; colocar las configuraciones para que cualquier función a través de la aplicación pueda consultar el estado actual del sistema. La ultima función del main es llamar a gtk_main() para atender los eventos.

interface.c Se encarga de construir la interfaz gráfica y conectar los widgets con las funciones a partir de las señales (gtk_signal_connect).

1. support.c: Se encarga de leer los paths de configuración para las imágenes que se vayan a desplegar en la aplicación.

callbacks.c Aquí están las instrucciones que debe seguir el programa una vez que hay algún evento. En general la manera en que funcionan es alterando la estructura lógica donde se encuentra la configuración actual del programa.

Objeto.h Define un elemento de un buffer.

Buffer.c En esta biblioteca se encuentra la implementación de un buffer en donde se pueden guardar los datos de una manera estructurada. El buffer esta estructurado en bloques, cada bloque contiene 2 elementos importantes: la cabecera y el área de datos. En la cabecera se encuentran una secuencia de códigos que definirán el área de datos. En esencia hay 3 tipos de códigos: 0 para datos, 1 para media de datos y 2 para cambio de estado.

Guardar.c Guarda en el Disco Duro las estructuras contenidas en un Buffer. Puede guardar los elementos de 2 maneras: De forma binaria (Exactamente como vienen en la estructura Buffer) o en formato ASCII (Convierte la forma binaria en elementos legibles para el humano).

LabPC.c Funciones de bajo nivel para comunicarse con la tarjeta LabPC+.

procesar.c Procesa la señal obtenida de la tarjeta convertora análogo digital y se la entrega a graficos.c.

graficos.c Gráfica los datos obtenidos después de procesa.c. Analogo/Digital.

Para xquetzal-publish (Figura C.2):

index.html Documento html estático para la publicación web. En esta página damos una introducción de lo que es el interferómetro, sus características físicas y su localización geográfica. En esta página se encuentra un menú con 2 opciones: Tiempo Real y Eventos Solares.

mporeal.html Página html estática en donde se muestra la imagen que se esta obteniendo en estos momentos. Esta imagen es generada por xquetzal-caption. La página se actualiza automáticamente cada 5 minutos.

tosSolares.jsp Página dinámica que despliega un calendario en donde se muestran los días con mediciones y los días con eventos solares.

Build.jsp Página dinámica que muestra los datos de un día.

C.3.3. Modelo Funcional

El modelo funcional esta dividido en 3 partes principales:

1. Configuración (Figura C.3): Es esta ventana se pueden configurar los siguientes aspectos:
   a) Canal: Los canales que se podrán visualizar (De 1 a 4).
   b) Amplificación: Puede ser Manual (Que se fija de 1 a 100 amplificaciones) o Automática (La Amplificación varia acomodándose a una mejor escala).
   c) Escala: La escala de la gráfica tanto en eje x como en eje y.
   d) Frecuencia: La frecuencia de Muestreo que va desde 0.001 segundos hasta 10 segundos.
   e) Modo de Captura: Libre o ControladoD
   f) Presentación: Ventana o Pantalla Completa.
   g) Autoguardar: Activado o Desactivado.
   h) Formato: Formato en que se guardan los datos, ASCII o Comprimido.
   i) Evento: Limite para que se active la señal de Inicio de Evento.
   j) Programar: Programar el Inicio de Captura y su terminación.
   k) Salvar: Salvar las configuraciones.
2. Recibir (Figura C.4):
   a) Área de Despliegue: Aquí se grafican los datos en tiempo real de acuerdo a las configuraciones de la pestaña de Configuración
   b) Botón Iniciar: Inicia la Captura de Datos.
   c) Botón Detener: Detiene la Captura de Datos.
   d) Botón Calibrar: Inicia la Captura de Datos en modo de Calibración.
3. Análisis (Figura C.5):: En esta área podremos Analizar los datos obtenidos mediante funciones que podemos programar.

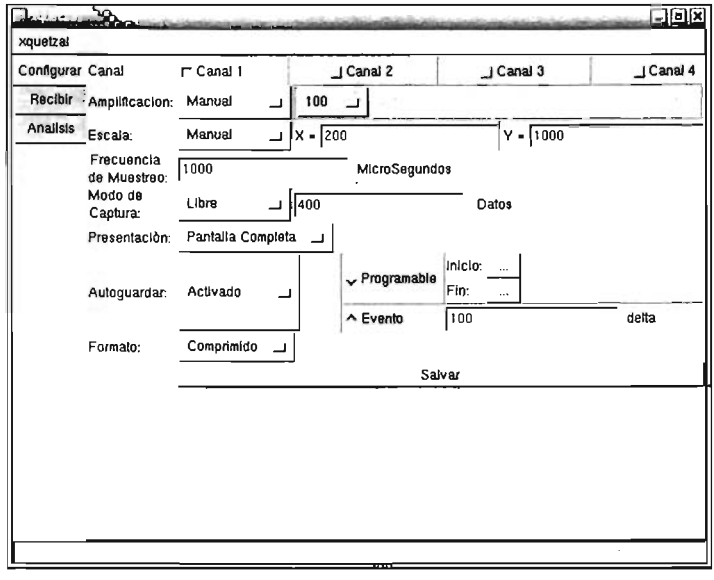

Figura C.3: Modelo Funcional para xquetzal-caption/Configurar.

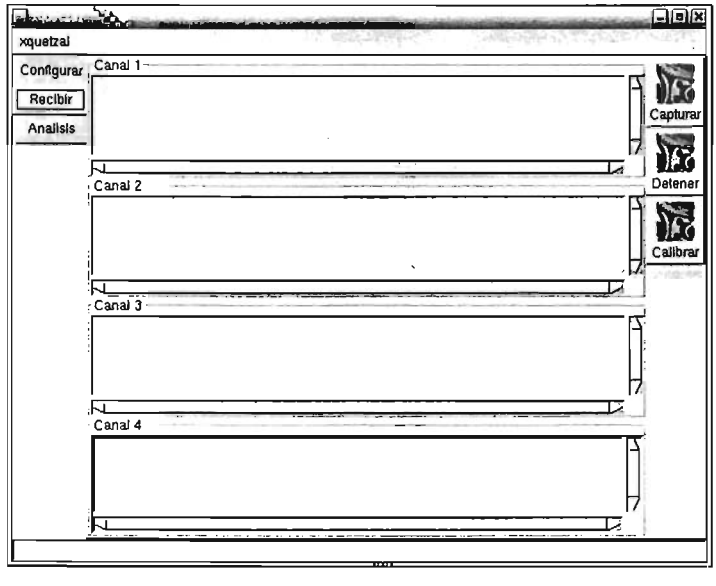

Figura C.4: Modelo Funcional para xquetzal-caption/Recibir.

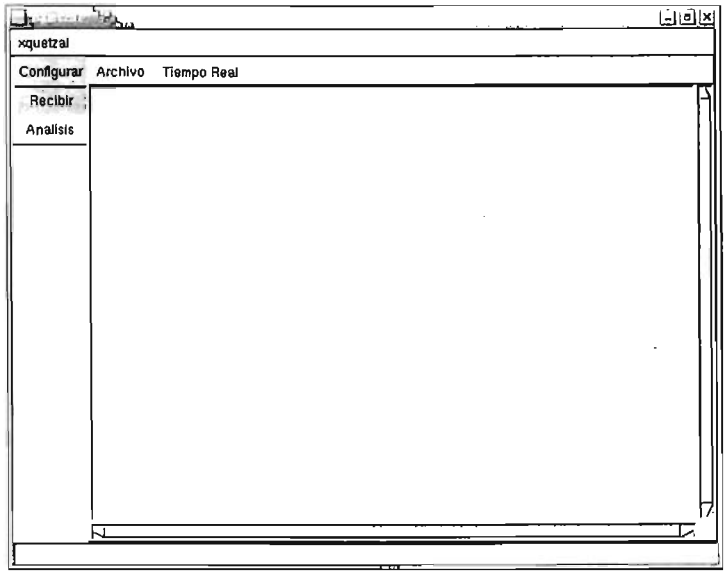

Figura C.5: Modelo Funcional para xquetzal-caption/Análisis.

C.3.4. Modelo de Datos

La forma en que establece sus estados la aplicación es mediante un arreglo de enteros. Este arreglo de enteros tiene como nombre Memoria y tiene la disposición de acuerdo a la tabla C.1 y funciona de la siguiente manera:

El registro 0 es el registro de cambio de estado, cuando la aplicación cambie de estado (Cambio de amplificación, cambio de frecuencia, cambio de asignación de canales, etc) este registro se pondrá en 1, en otro caso en 0. Cuando el registro 0 este en 1, revisara el registro 27, el cual indica el cambio que se realizo. Sabiendo que cambio se realizo, verificamos ese registro y obtenemos el valor de cambio.

Otra estructura importante es la estructura del Buffer. En esta estructura guardamos un historial de los estados del sistema, así como los datos que se vayan generando.

El buffer tiene 3 modo principales: El modo 0 o de Datos, 1 de Medias y 2 para cambios de estado.

La estructura para el modo 0 se encuentra representado en la figura C.6. La tarjeta convertora tiene como máximo de almacenamiento 12 bits por dato, la estructura del buffer tiene una longitud de 16 bits, 2 para indicar que es un dato, 2 no importan y los restantes 12 sirven para almacenar el dato.

Porque tener una estructura para dato y otra para media? porque se tiene pensado que en un futuro se pueda depurar el modo de guardar, para que mientras no hay eventos solares solo se vaya guardando la media, mientras que si hay un evento, los datos deben de estar íntegros. La estructura para el modo 1 o de media se encuentra representado en la figura C.7 y es igual que en el modo 0, excepto por el head y que el valor máximo de la media es de 14 bits.

En el caso del modo 2 (Figura C.8), la longitud del registro es de 40 bits. Los primeros dos

Índice	Valores	Descripción
0	0	No hay cambio
	1	Hay un cambio
1	0	En pausa o detenido.
	1	Continua la captura.
2	0	Continuar Proyecto.
	1	Nuevo Proyecto.
3	0	Amplificación Manual
	1	Amplificación Automática
4	0..7	Cantidad de Amplificación.
5	0..7	Frecuencia de Muestreo.
21		Memoria Temporal
22		Memoria Temporal (usada en Amplificación).
23		Memoria Temporal (usada en frecuencia).
24		Memoria usada para actualizar barra de progreso.
27	0	Nada
	1	Cambio de Amplificación
	2	Cambio de Frecuencia
	-1	Terminamos

Tabla C.1: Registros para la estructura de Intercambio de Estados.

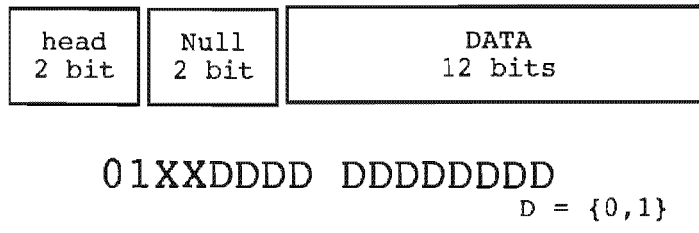

Figura C.6: Estructura para guardar un dato en el buffer.

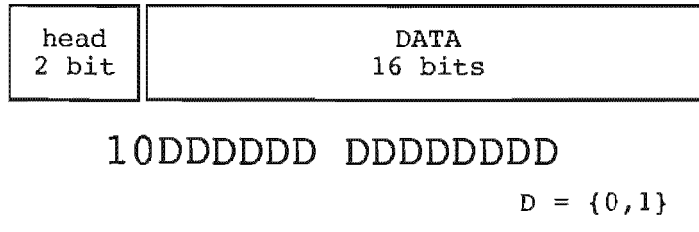

Figura C.7: Estructura para guardar una media en el buffer.

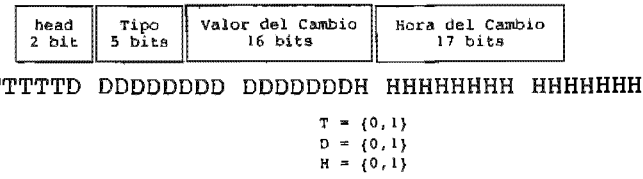

Figura C.8: Estructura para guardar un cambio de Estado en el Buffer.

Valor	Descripción	Valor	Descripción
0	Inicio	5	Termina Evento
1	Fin	6	Datos para la media
2	Cambio de Amplificación	7	Valor de Épsilon
3	Cambio de Frecuencia	8	Numero de Épsilon
4	Inicia Evento	9	Estado de Calibración

Tabla C.2: Codificación de Cambios de Estado usados por el modo 2 del Buffer.

bits son el head, los siguientes 5 bits son para guardar el tipo de cambio con el esquema que se muestra en la tabla C.2. Los siguientes 16 bits son para guardar el valor del cambio. Por ejemplo, si el tipo de cambio es 2 (Cambio de Amplificación) entonces en los siguientes 16 bits se encontrara el valor de ese cambio de amplificación. Los siguientes 17 bits guardan la hora del cambio con el siguiente formato: El primer bit es 0 si es AM y 1 si es PM, los siguiente 4 bits son para la hora (1..12), los siguiente 6 bits para los minutos (0..59) y los últimos 6 bits para los segundos (0..59).

El buffer ira guardando dinámicamente estas estructuras en la memoria para que posteriormente puedan ser guardadas en la unidad de almacenamiento secundario.

Otro elemento importante es la manera en que los datos estarán organizados en el disco. Los datos serán guardados en un directorio nombrado */database*, en este directorio se encontrara un directorio llamado *actual*, en este directorio se encontraran los datos que se están generando en tiempo real. Dentro del directorio */database* se encontrara una estructura con el siguiente prototipo: */AAAA/MM/DD* Donde AAAA es el año, MM el mes y DD el día, donde se guardaran 6 archivos importantes:

1. canal0.xdat : Datos en formato binario para el canal 0.
2. canal1.xdat : Datos en formato binario para el canal 1.
3. canal2.xdat : Datos en formato binario para el canal 2.
4. canal3.xdat : Datos en formato binario para el canal 3.
5. README : Historial de la captura en este día.
6. imagen.png : Imagen en formato png donde se grafican los 4 canales.

El directorio */database* sera montado mediante NFS en la computadora de publicación en donde se encuentre instalado xquetzal-publish.

Para crear las modificaciones en tiempo real, la pagina de tiempo real en xquetzal-publish refrescara los datos en pantalla del directorio */database/actual/*

C.3.5. Modelo de Flujo de Control

El modelo de flujo de control se estableció en el apéndice B en las figuras B.1 y B.2

C.3.6. Definición de Recursos

Los requerimientos mínimos para xquetzal-publish son:

1. Procesador: Pentium o superior.
2. Memoria RAM: 256 Mbyte o superior.
3. Disco Duro: 10 Gbytes o superior.
4. Tarjeta de Red: 100 Mbps o superior.

Los requerimientos mínimos para xquetzal-caption son:

1. Procesador: Pentium o superior.
2. Memoria RAM: 256 Mbytes o superior.
3. Disco Duro: 500 Mbytes o superior.
4. Tarjeta de Red: 100 Mbps o superior.
5. Monitor SVGA 800x600 o superior.

C.4. Descripción de los componentes

Para xquetzal-caption (Figura C.1):

main.c Se encarga de llamar al constructor de la Interfaz (createWindow() en interface.h); desplegar la interfaz en pantalla (gtk_widget_show()); leer las configuraciones del disco y a partir de estas configuraciones crear una estructura lógica en la memoria; colocar las configuraciones para que cualquier función a través de la aplicación pueda consultar el estado actual del sistema. La última función del main es llamar a gtk_main() para atender los eventos.

interface.c Se encarga de construir la interfaz gráfica y conectar los widgets con las funciones a partir de las señales (gtk_signal_connect).

1. support.c: Se encarga de leer los paths de configuración para las imágenes que se vayan a desplegar en la aplicación.

callbacks.c Aquí están las instrucciones que debe seguir el programa una vez que hay algún evento. En general la manera en que funcionan es alterando la estructura lógica donde se encuentra la configuración actual del programa.

Objeto.h Define un elemento de un buffer.

Buffer.c En esta biblioteca se encuentra la implementación de un buffer en donde se pueden guardar los datos de una manera estructurada. El buffer esta estructurado en bloques, cada bloque contiene 2 elementos importantes: la cabecera y el área de datos. En la cabecera se encuentran una secuencia de códigos que definirán el área de datos. En esencia hay 3 tipos de códigos: 0 para datos, 1 para media de datos y 2 para cambio de estado.

Guardar.c Guarda en el Disco Duro las estructuras contenidas en un Buffer. Puede guardar los elementos de 2 maneras: De forma binaria (Exactamente como vienen en la estructura Buffer) o en formato ASCII (Convierte la forma binaria en elementos legibles para el humano).

LabPC.c Funciones de bajo nivel para comunicarse con la tarjeta LabPC+.

procesar.c Procesa la señal obtenida de la tarjeta convertora análogo digital y se la entrega a graficos.c.

graficos.c Gráfica los datos obtenidos después de precesa.c. Analogo/Digital.

Para xquetzal-publish (Figura C.2):

index.html Documento html estático para la publicación web. En esta página damos una introducción de lo que es el interferómetro, sus características físicas y su localización geográfica. En esta pagina se encuentra un menú con 2 opciones: Tiempo Real y Eventos Solares.

emporeal.html Pagina html estática en donde se muestra la imagen que se esta obteniendo en estos momentos. Esta imagen es generada por xquetzal-caption. La pagina se actualiza automáticamente cada 5 minutos.

atosSolares.jsp Pagina dinámica que despliega un calendario en donde se muestran los días con mediciones y los días con eventos solares.

Build.jsp Pagina dinámica que muestra los datos de un día en particular.

C.5. Factibilidad y estimación de recursos

El proyecto continuará pero con la advertencia de que la actual infraestructura es insuficiente, pudiendo llegar el caso de que el proyecto sea inviable.

Para tener una mayor certeza de éxito, el proyecto necesita por lo menos dos computadoras personales con procesadores mayores a 1 GHz y con por lo menos 256 Mbytes en RAM, Un disco duro de 80 Gbytes para almacenar los datos en por lo menos 3 años.

Si el proyecto llega a fracasar con la actual infraestructura y se decide comprar estas máquinas y todas las interfaces de red, podríamos sugerir un cambio de modelo de desarrollo y un cambio de paradigma. El proyecto tomaría el esquema cliente-servidor, teniendo una computadora para capturar datos, otra para procesarlos y finalmente una ultima para publicarlos.

Apéndice D

Análisis de la tarjeta LabPC+ De National Instruments

La tarjeta convertidora Análogo/Digital LabPC+ de National Instruments cuenta con una serie de registros de 1 byte de longitud de lectura y de escritura. La lista de registros se encuentra en el manual de usuario de LabPC[6].

Los registros de lectura sirven para saber el estatus de la tarjeta, solo pueden ser leídos. Los registros de escritura sirven para cambiar el estado de la tarjeta, es importante saber que estos registros vienen agrupados en 8 bits y cada vez que se modifica un bit los otros 7 deben de reescribirse para no ver modificado su estado.

La tarjeta cuenta con 2 estados de captura: Libre y Controlada. Para la captura Controlada hay 2 contadores, uno que sirve como controlador de intervalos de tiempo (A0 y B0) y el otro como contador de capturas (A).

Uno puede decirle a la tarjeta: quiero 1000 datos a .2 segundos de intervalo entre cada uno.

La captura libre es mas fácil, solo se configura un reloj que es el de intervalo (A0 o B0) y se le indica a la tarjeta que inicie la captura. Esta comenzara la captura hasta que se le ordene detenerse.

Hay una serie de pasos que hay que realizar antes de iniciar la captura. Los primeros es iniciar a la tarjeta con los siguientes comandos:

1. Escribe 0x00 en el Command Register 1.
2. Escribe 0x00 en el Command Register 2.
3. Escribe 0x00 en el Command Register 3.
4. Escribe 0x00 en el Command Register 4.
5. Escribe 0x34 en el Counter A Mode Register.
6. Escribe 0x0A en el Counter A0 Data Register.
7. Escribe 0x00 en el Counter A0 Data Register.
8. Escribe 0x00 en el DMATC Interrupt Clear Register.
9. Escribe 0x00 en el Timer Interrupt Clear Register.

10. Escribe 0x00 en el A/D Clear Register.

11. Lee el dato del A/D FIFO Register(2 veces). Ignorar el dato.

12. Escribe 0x00 en el DAC0L y despues escribe 0x00 en el DAC0H si el DAC0 está configurado en salida unipolar. Escribe 0x00 al DAC0L y después escribe 0x08 al DAC0H si el DAC0 está configurado en salida bipolar.

13. Escribe 0x00 en el DAC1L y después escribe 0x00 en el DAC1H si el DAC1 está configurado en salida unipolar. Escribe 0x00 al DAC1L y después escribe 0x08 al DAC1H si el DAC1 está configurado en salida bipolar.

Una vez iniciada la tarjeta, se configura para realizar las conversiones.
Hay una serie de definiciones que la tarjeta LabPC usa para su configuración:

1. MultiEscaneo o Escaneo Simple: El multi escaneo funciona cuando se requiere capturar en mas de 1 canal y el Escaneo Simple es cuando solo se requiere capturar de un solo canal.

2. Selección de Canales: Una vez que se elige escaneo simple o múltiple, es necesario que elegir cuales canales se van a escanear.

3. Forma Bipolar y Unipolar: Hay dos formas de escanear canales, en forma unipolar o bipolar, si es bipolar podemos escanear hasta 4 canales y si es unipolar hasta 8 canales. La diferencia es que en bipolar podemos tener datos positivos y negativos (-2048 hasta 2047) mientras que en el unipolar solo positivos (0 hasta 4095).

4. Amplificación: La amplificación va de 1 a 100x.

5. Escaneo Libre/Controlado: Cuando tenemos restringido el número de elementos a convertir, se usa el escaneo Controlado, cuando queremos escanear datos de manera ininterrumpida y sin tomar en cuenta el número de ellos, se usa el escaneo libre.

6. Contadores: Cuando se utiliza escaneo libre solo se necesita un reloj, en el caso de escaneo controlado es necesario avisar a la tarjeta que usara también el contador.

7. Complemento A2: Si estamos capturando en forma bipolar tenemos que avisar a la tarjeta que queremos obtener los resultados en complemento A2, de lo contrario los entregará con complemento A1, es preferible el complemento A2 porque es el más usado y es como C maneja los números negativos.

Una vez configurados estos puntos, estamos listos para iniciar la captura, esto se hace avisando a la tarjeta a través del registro de inicio de captura.

Cuando se da el aviso de inicio de captura hay un esquema muy simple para comenzar a adquirir los datos, pero primero explicaremos la forma en que la tarjeta LabPC adquiere los datos:

1. La tarjeta LabPC tiene una pequeña cola (registro FIFO) donde se van acumulando los datos una vez convertidos, después de que este lista la conversión se enciende el bit 0 del Status Register.

2. Los datos que se obtienen de la tarjeta tienen una longitud de 12 bits, por lo que necesitamos leer 2 veces el registro FIFO y por software armar el dato.

3. Primero obtenemos los primeros 8 bits menos significativos y una lectura posterior nos da los siguiente 8 bits de los cuales solo los últimos 4 tiene información real.

4. Una vez leído el registro FIFO, la cola elimina al primer dato y da espacio para el siguiente. Puede pasar que la tarjeta convierta los datos demasiado rápido y no haya lectura del FIFO, la manera en que actúa la tarjeta es tirando los datos que se estén convirtiendo (OverFlow). Pero cuando esto pasa, el bit 2 del Status Register se activa y es cuando nosotros nos podemos enterar de que hubo perdida de datos por falta de atención al registro FIFO, cuando queremos acceder a datos que no están disponibles se enciende el bit 1 del Status Register (OverRun).

5. La forma en que escanea los canales es de acuerdo a la configuración del reloj, si esta es cada 0.5 seg, entonces leerá del canal 4, se esperara 0,5 seg después el canal 3 y esperara 0.5 seg, etc.

6. Para obtener lo datos de la tarjeta tenemos que estar revisando constantemente el bit 0 del Status Register, una vez que se encienda este registro (bit = 1) procedemos a ver si hay algún error por falta de atención (OVERFLOW o UNDERFLOW) si no hay ningún error, leemos dos veces el registro FIFO y posteriormente por software armamos el dato.

7. La manera en que reconstruimos el dato es sabiendo de antemano bajo que formato estamos trabajando, en este proyecto usamos complemento A2, así que lo que hacemos es revisar el primer byte, revisamos el 4 bit de derecha a izquierda, si es 1 entonces el numero es negativo, como el resultado final es un entero en C, los primeros 20 bits estarán en 1.

8. Cuando terminamos la captura solo reiniciamos la tarjeta.

9. Si queremos cambiar la configuración de la tarjeta tenemos que modificar los registros y reiniciar la captura.

Los registros son accesados a partir de una dirección base (en nuestro caso es 0x0260) y una dirección interna de la tarjeta (Ver Tabla D.1).

Entre los registros mas importantes tenemos:

1. Command Register 1: Indica el canal de entrada a ser leído, la amplificación, la configuración física de la tarjeta y el modo bipolar o unipolar.

2. Status Register:Da información de la situación de la conversión actual, determina si la conversión esta lista y si hay algún tipo de error.

3. A/D FIFO Register: Registro en donde se encuentra el valor de la conversión actual.

4. A/D Clear Register: Limpia el A/D FIFO Register.

5. Start Convert Register: Inicia la conversión de datos.

6. Contadores/Tiempo: En su conjunto configuran el intervalo entre conversiones.

En el Apéndice I se encuentra un programa que configura e imprime datos de la tarjeta LabPC+.

Tal vez el punto mas importante de la tarjeta LabPC+ es la programación de la frecuencia de muestreo.

Nombre	Dirección	Tipo	Tamaño
Grupo de Registros de Configuración y Estatus			
Command Register 1	00	Solo Escritura	8-bit
Status Register	00	Solo Lectura	8-bit
Command Register 2	01	Solo Escritura	8-bit
Command Register 3	02	Solo Escritura	8-bit
Command Register 4	0F	Solo Escritura	8-bit
Grupo de Registros de Entrada Análoga			
A/D FIFO Register	0A	Solo Lectura	8-bit
A/D Clear Register	08	Solo Escritura	8-bit
Start Convert Register	03	Solo Escritura	8-bit
DMATC Interrupt Clear Register	0A	Solo Escritura	8-bit
Grupo de Registros de Salida Análoga			
DAC0 Low-Byte Register	04	Solo Escritura	8-bit
DAC0 High-Byte Register	05	Solo Escritura	8-bit
DAC1 Low-Byte Register	06	Solo Escritura	8-bit
DAC1 High-Byte Register	07	Solo Escritura	8-bit
Grupo A, Contadores/Tiempo			
Counter A0 Data Register	14	Solo Lectura	8-bit
Counter A1 Data Register	15	Solo Lectura	8-bit
Counter A2 Data Register	16	Solo Lectura	8-bit
Counter A Mode Register	17	Solo Escritura	8-bit
Timer Interrupt Clear Register	0C	Solo Escritura	8-bit
Grupo B, Contadores/Tiempo			
Counter B0 Data Register	18	Solo Lectura	8-bit
Counter B1 Data Register	19	Solo Lectura	8-bit
Counter B2 Data Register	1A	Solo Lectura	8-bit
Counter B Mode Register	1B	Solo Escritura	8-bit
Grupo de E/S Digital			
Port A Register	10	Solo Lectura	8-bit
Port B Register	11	Solo Lectura	8-bit
Port C Register	12	Solo Lectura	8-bit
Digital Control Register	13	Solo Escritura	8-bit
Grupo de Contadores de Intervalo			
Interval Counter Data Register	1E	Solo Escritura	8-bit
Interval Counter Strobe Register	1F	Solo Escritura	8-bit

Tabla D.1: Mapa de Registros de Lab-PC+

Esta tarjeta cuenta con múltiples relojes que se pueden configurar, los mas importantes son los relojes B0 y A0, estos relojes se pueden configurar para trabajar juntos y son independientes de la PC.

Si necesitamos frecuencia de muestreo entre 0.000002 seg y .065535 seg es suficiente el contador A0. Para programarlo se necesitan ejecutar los siguientes pasos:

1. Escribir 0x34 en Counter A Mode Register.
2. Escribir el byte menos significativo en el Counter A0 Data Register.
3. Escribir el byte mas significativo en el Counter A0 Data Register.

Si queremos frecuencias de muestreo que van desde 0.0000005 seg hasta .032767 seg entonces necesitamos configurar un segundo reloj, el B0. Para programar el B0 ejecutamos los siguientes pasos:

1. Activar el TBSEL bit en el Comand Register 2 (xxx1xxx).
2. Escribir 0x36 en Counter B Mode Register.
3. Escribir el byte menos significativo en el Counter B Data Register.
4. Escribir el byte mas significativo en el Counter B Data Register.

Para frecuencias de muestre mas grandes se necesita hacer el control por software.

Apéndice E

Script de configuración de la puerta de Enlace Cintli a través de NAT

E.1. Introducción

Usando NAT podemos usar una simple PC como puerta de enlace para una red privada hacia Internet. Esto centraliza la labor de administración y de seguridad en una sola máquina y hace posible que toda la red interna tenga comunicación con el exterior, no así el exterior con nosotros. Explicar el protocolo NAT no es una meta de este trabajo así que solo enunciaremos el script que se encuentra en la máquina Cintli y tiene que ser ejecutado si la máquina se cae. El script se encuentra en /root/IPTABLES.sh

E.2. IPTABLE.sh

```
#!/bin/bash
iptables -F
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -F
iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -d 0.0.0.0/0 -j MASQUERADE
iptables -A INPUT -i eth0 -p ICMP -j ACCEPT
iptables -A INPUT -p TCP -dport 80 -j ACCEPT
iptables -A INPUT -p TCP -dport 22 -j ACCEPT
iptables -A INPUT -p TCP -dport 1:1024 -j REJECT
iptables -A INPUT -p TCP -m state -state RELATED -j ACCEPT
```

Apéndice F

Manual de Procedimiento para el RIS

F.1. xitris-server

Si el servicio esta detenido: iniciar sesión como root en quetzal y ejecutar en una terminal:

1. nohup xitris-server&
2. Es posible salir de la sesión y xitris-server seguirá ejecutándose.

F.2. xitris-client

Para ejecutar xitris-client: iniciar sesión como xitris en tlaloc y ejecutar en una terminal:

1. xitris-client

F.3. xitris-www

Para ejecutar xitris-www: iniciar sesión como xitris en cintli y ejecutar en una terminal:

1. xitris-www

F.4. xitris-www

Para ejecutar xitris-analyze: inciar sesión como xitris en cintli y ejecutar en una terminal:

1. xitris-analyze

F.5. Instalación

La forma de instalar cada paquete se encuentra en la ultima distribución del código, favor de consultar cada README y cada INSTALL. el código lo puedes encontrar en:

http://sourceforge.net/projects/xquetzal/

Apéndice G

Prueba 1 (HolaMundo.glade)

G.1. Glade

Glade es un constructor de Interfaces Libre para GTK+ y GNOME, está bajo los términos de GNU/GPL [15]. Glade puede producir código fuente C, C++, Python, Perl entre otros. Actualmente cuenta con dos versiones: glade-1.2 y glade2. Su principal diferencia es que una implementa a GTK+1.2 y el otro a GTK+2. Glade genera una jerarquía de directorios; en la raíz se encuentran una serie de archivos que permiten autogenerar los MakeFiles para compilar la aplicación. El script que revisa las dependencias y genera los MakeFiles es ./autogen.sh y el lugar en donde se encuentra el código fuente es *src*.

Glade genera 7 archivos:

1. main.c Es el main principal de la aplicación, aquí se manda llamar la generación de interfaces y el loop principal de GTK.

2. support.h/support.c Se configuran entre otras cosas los paths relativos para encontrar imágenes o mapas de pixeles.

3. interface.h/interface.c Se encuentra el código que genera las interfaces gráficas, en general esta lleno de código superfluo, debido a la automatización del proceso. Como por ejemplo, para crear 10 cajas de texto con un formato especifico, podriamos hacer una función que construyera una sola caja de texto bien formateada y mandarla llamar 10 veces, Glade lo que hace es repetir 10 veces el código de esa caja de texto generando una gran cantidad de codigo repetido, sin depurar.

4. callbacks.h/callbacks.c Es aquí donde tenemos que generar nuestro código.

5. Make El script para compilar nuestra aplicación.

De todos estos archivos generados el que mas nos interesa es callbacks.c ya que Glade te genera el código de manera que solo implementes aquellas funciones que se activan al dispararse eventos específicos, mas adelante con el ejemplo quedara mas claro.

Es importante decir que Glade genera muchos mas archivos, estos los usa para control interno y para generación del binario final.

Para presentar el ejemplo incluyo una imagen de la interfaz generada y los 3 archivos principales de Glade (main.c, interface.c y callbacks.c).

Figura G.1: El HolaMundo corriendo en mi escritorio.

G.2. Código Fuente de HolaMundo

```
/*
 * Initial main.c file generated by Glade. Edit as required.
 * Glade will not overwrite this file.
 */

#ifdef HAVE_CONFIG_H
# include <config.h>
#endif

#include <gtk/gtk.h>

#include "interface.h"
#include "support.h"

int
main (int argc, char *argv[])
{
  GtkWidget *window1;

  gtk_set_locale ();
  gtk_init (&argc, &argv);

  add_pixmap_directory (PACKAGE_DATA_DIR "/pixmaps");
  add_pixmap_directory (PACKAGE_SOURCE_DIR "/pixmaps");

  /*
   * The following code was added by Glade to create one of each component
   * (except popup menus), just so that you see something after building
   * the project. Delete any components that you dont want shown initially.
   */
  window1 = create_window1 ();
  gtk_widget_show (window1);

  gtk_main ();
  return 0;
}
```

Figura G.2: Codigo fuente del main.c de HolaMundo

```
/*
 * DO NOT EDIT THIS FILE - it is generated by Glade.
 */

#ifdef HAVE_CONFIG_H
# include <config.h>
#endif

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>

#include <gdk/gdkkeysyms.h>
#include <gtk/gtk.h>

#include "callbacks.h"
#include "interface.h"
#include "support.h"

GtkWidget*
create_window1 (void)
{
  GtkWidget *window1;
  GtkWidget *button1;

  window1 = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_object_set_data (GTK_OBJECT (window1), "window1", window1);
  gtk_window_set_title (GTK_WINDOW (window1), "window1");

  button1 = gtk_button_new_with_label ("button1");
  gtk_widget_ref (button1);
  gtk_object_set_data_full (GTK_OBJECT (window1), "button1", button1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (button1);
  gtk_container_add (GTK_CONTAINER (window1), button1);

  gtk_signal_connect (GTK_OBJECT (window1), "delete_event",
                      GTK_SIGNAL_FUNC (on_window1_delete_event),
                      NULL);
  gtk_signal_connect (GTK_OBJECT (button1), "clicked",
                      GTK_SIGNAL_FUNC (on_button1_clicked),
                      NULL);

  return window1;
}
```

Figura G.3: Codigo fuente de interface.c de HolaMundo

```
#ifdef HAVE_CONFIG_H
# include <config.h>
#endif

#include <gtk/gtk.h>

#include "callbacks.h"
#include "interface.h"
#include "support.h"

gboolean
on_window1_delete_event         (GtkWidget       *widget,
                                 GdkEvent        *event,
                                 gpointer         user_data)
{
  gtk_main_quit();
  return FALSE;
}

void
on_button1_clicked              (GtkButton       *button,
                                 gpointer         user_data)
{
  g_print("\nHola");
}
```

Figura G.4: Codigo fuente de callbacks.c de HolaMundo

Apéndice H

Prueba 2 (Iniciar/Detener)

H.1. Introducción

La idea es hacer la interfaz en Glade con una ventana y dos botones (Iniciar y Detener) e implementar el archivo callbacks.c para realizar lo siguiente: Tenemos una variable *semaforo* que puede estar en verde (1) o en rojo (0). Al iniciar esta en rojo. Cuando clickeo el botón Iniciar pongo el semáforo en verde e inicio un bucle que a cada paso observe el semáforo si esta en verde continuo, si esta en rojo me detengo. Cuando clickeo el botón Detener, simplemente pongo el semáforo en rojo y listo.

H.2. Código Fuente de Iniciar/Detener

A continuación se encuentra el código fuente de IniciarDetener. Esta compuesto por tres archivos principalmente: main.c, interface.c y callbacks.c (Figuras H.2, H.3 y H.4). Si miramos detenidamente el callbacks.c (figura H.4) encontraremos las dos funciones que implementan nuestro semáforo.

Al ejecutar el programa (Ver figura H.1) sucede algo inesperado, la aplicación deja de responder, toda la ventana se queda congelada y los botones no responden quedándose en un bucle infinito.

Revise varias veces el código, cambiando la secuencias de los connects pero no respondía, también señale a la variable semáforo como estática y mande dormir con la función sleep el bucle y tampoco respondió. Decidí preguntar a compañeros, buscar en libros sobre memoria compartida y semáforos y la manera en que GTK maneja los eventos y continuar con las siguientes pruebas.

Para terminar esta aplicación fue necesario mandarle una señal kill.

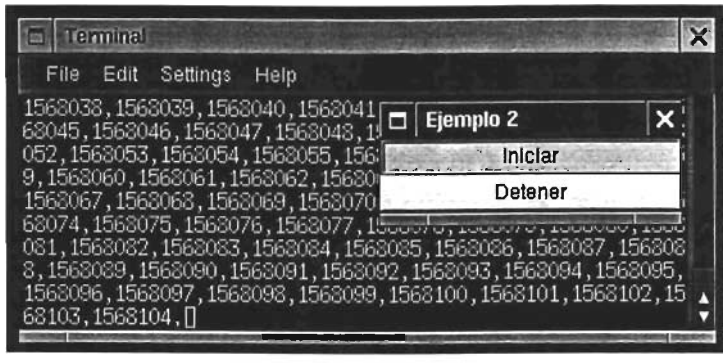

Figura H.1: IniciarDentener corriendo en mi escritorio.

```
/*
 * Initial main.c file generated by Glade. Edit as required.
 * Glade will not overwrite this file.
 */

#ifdef HAVE_CONFIG_H
# include <config.h>
#endif

#include <gtk/gtk.h>

#include "interface.h"
#include "support.h"

int
main (int argc, char *argv[])
{
  GtkWidget *window;

  gtk_set_locale ();
  gtk_init (&argc, &argv);

  add_pixmap_directory (PACKAGE_DATA_DIR "/pixmaps");
  add_pixmap_directory (PACKAGE_SOURCE_DIR "/pixmaps");

  /*
   * The following code was added by Glade to create one of each component
   * (except popup menus), just so that you see something after building
   * the project. Delete any components that you dont want shown initially.
   */
  window = create_window ();
  gtk_widget_show (window);

  gtk_main ();
  return 0;
}
```

Figura H.2: Código fuente del main.c de Iniciar/Detener

```
/*
 * DO NOT EDIT THIS FILE - it is generated by Glade.
 */

#ifdef HAVE_CONFIG_H
# include <config.h>
#endif

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>

#include <gdk/gdkkeysyms.h>
#include <gtk/gtk.h>

#include "callbacks.h"
#include "interface.h"
#include "support.h"

GtkWidget*
create_window (void)
{
  GtkWidget *window;
  GtkWidget *vbox1;
  GtkWidget *button1;
  GtkWidget *button2;

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_object_set_data (GTK_OBJECT (window), "window", window);
  gtk_window_set_title (GTK_WINDOW (window), "Ejemplo 2");

  vbox1 = gtk_vbox_new (FALSE, 0);
  gtk_widget_ref (vbox1);
  gtk_object_set_data_full (GTK_OBJECT (window), "vbox1", vbox1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (vbox1);
  gtk_container_add (GTK_CONTAINER (window), vbox1);

  button1 = gtk_button_new_with_label ("Iniciar");
  gtk_widget_ref (button1);
  gtk_object_set_data_full (GTK_OBJECT (window), "button1", button1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (button1);
  gtk_box_pack_start (GTK_BOX (vbox1), button1, FALSE, FALSE, 0);

  button2 = gtk_button_new_with_label ("Detener");
  gtk_widget_ref (button2);
  gtk_object_set_data_full (GTK_OBJECT (window), "button2", button2,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (button2);
  gtk_box_pack_start (GTK_BOX (vbox1), button2, FALSE, FALSE, 0);

  gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                      GTK_SIGNAL_FUNC (on_window_delete_event),
                      NULL);
  gtk_signal_connect (GTK_OBJECT (button1), "clicked",
                      GTK_SIGNAL_FUNC (on_button1_clicked),
                      NULL);
  gtk_signal_connect (GTK_OBJECT (button2), "clicked",
                      GTK_SIGNAL_FUNC (on_button2_clicked),
```

```
#ifdef HAVE_CONFIG_H
# include <config.h>
#endif

#include <gtk/gtk.h>

#include "callbacks.h"
#include "interface.h"
#include "support.h"

gint semaforo = 0; //El semaforo esta en rojo.

gboolean
on_window_delete_event          (GtkWidget     *widget,
                                 GdkEvent      *event,
                                 gpointer       user_data)
{
  gtk_main_quit();
  return FALSE;
}


void
on_button1_clicked              (GtkButton     *button,
                                 gpointer       user_data)
{
  gint i=0;
  semaforo = 1;
  while(semaforo){
    i++;
    g_print("%i,",i);
  }
}


void
on_button2_clicked              (GtkButton     *button,
                                 gpointer       user_data)
{
  semaforo = 0;
}
```

Figura H.4: Código fuente de callbacks.c de Iniciar/Detener

Apéndice I

Accediendo a Lab-PC+

De acuerdo con el estudio realizado en el apéndice D se construyo tarjeta.cI.1 que imprime un dato en la consola a partir de una configuración base.

Este programa utiliza una serie de funciones en C:

1. ioperm: Se encarga de tener acceso a un puerto lógico de la computadora, ioperm establece los permisos para lectura y escritura en dicho puerto.
2. inb: Lee 1 byte de un puerto especificado.
3. outb: Escribe un byte al puerto especificado.

El programa funciona de la siguiente manera:

1. Establece permisos mediante ioperm.
2. Configura la tarjeta con los valores iniciales contenidos en el apéndice D, dejando al sistema en las siguiente condiciones[6]:
   a) Contador A0 alto (bit=1).
   b) Contador A1 alto (bit=1).
   c) Todas las interrupciones deshabilitadas.
   d) Contador Base de A0 a 1 MHz.
   e) Salida Análoga en el canal 0 con amplificación 1.
   f) El A/D FIFO limpio.
   g) El Command Register iniciado.
   h) Circuitos de Salida iguales a 0.
3. Iniciamos la Captura.
4. Esperamos que haya un dato listo.
5. Cuando el dato esta listo, leemos los registros de error.
6. Si hay algún tipo de error, lo desplegamos.
7. En otro caso, leemos dos veces el FIFO y armamos el dato en complemento A2.

El programa se compila con: gcc -O2 -o tarjeta tarjeta.c

```
#include <stdio.h>   /* Libraria estandard de C        */
#include <unistd.h> /* ioperm()                        */
#include <asm/io.h> /* outb(), intb()                  */

#define BASEPORT 0x0260

  int lec1,lec2;
  int MASK = 128;
  int MASKCLEAR = 0x000000FF;

int main(){
  /*Pidiendo Permisos*/
  if (ioperm(BASEPORT, 25, 1) == 0)
    println("Tarjeta Disponible.");
  else
    exit(0);
  /*INICIANDO TARJETA*/
  outb(0x00,BASEPORT);
  outb(0x00,BASEPORT+0x0001) ;
  outb(0x00,BASEPORT+0x0002) ;
  outb(0x00,BASEPORT+0x000F) ;
  outb(0x34,BASEPORT+0x0017) ;
  outb(0x0A,BASEPORT+0x0014) ;
  outb(0x00,BASEPORT+0x0014) ;
  outb(0x00,BASEPORT+0x000A) ;
  outb(0x00,BASEPORT+0x000C) ;
  outb(0x00,BASEPORT+0x0008) ;
  /* limpia el registro de salida FIFO*/
  lec1 = inb(BASEPORT+0x000A);
  lec2 = inb(BASEPORT+0x000A);
  /* Empezar Conversion*/
   outb(0x04,BASEPORT+0x01);
  /* Espera a que haya un dato Listo */
  while(!(inb(BASEPORT) & 1));
  if ((0x004 & inb(BASEPORT)) >> 2){ //Error de Overflow!!
    println("Error de OverFlow");
  }else if ((0x002 & inb(BASEPORT)) >> 1){ //Error de Overrrun!!
    println("Error de OverRun");
  }else{
    lec1= (MASKCLEAR & inb(BASEPORT+0x000A)); //byte bajo
    lec2= (MASKCLEAR & inb(BASEPORT+0x000A)); //byte alto
    if ( (lec2 & MASK) == 128){ //es negativo
      lec2 = (0xFFFFF000 | (lec2 << 8));
    }else{
      lec2 = (lec2 << 8); //es positivo
    }
    res = (lec1 | lec2);
    println("%i",res);
}
```

Figura I.1: Código fuente de tarjeta.c

Apéndice J

Codigo Generado

Información:	Codigo Generado		
Proyecto:	Xquetzal	Versión:	03052004
Preparador por:	Victor De la Luz	Fecha:	3 / 05 / 2004
Recibido por :	A. Lara	Status:	Aprobado

J.1. LabPC.c

```
#include "LabPC.h" /* Cabezeras de la propia biblioteca */
#include <stdio.h>  /* Libraria estandard de C          */
#include <unistd.h> /* ioperm()                         */
#include <asm/io.h> /* outb(), inb()                    */


int tarjetaDisponible( void ){
  if (ioperm(BASEPORT, 25, 1))
    return 0;
  else
    return 1;
}

/*
 * Inicia la tarjeta para comenzar a recibir datos
 */
void iniciaTarjeta( void ){
  unsigned lec1;
  unsigned lec2;
  outb(0x00,BASEPORT);
  outb(0x00,BASEPORT+0x0001) ;
  outb(0x00,BASEPORT+0x0002) ;
  outb(0x00,BASEPORT+0x000F) ;
  outb(0x34,BASEPORT+0x0017) ;
  outb(0x0A,BASEPORT+0x0014) ;
  outb(0x00,BASEPORT+0x0014) ;
  outb(0x00,BASEPORT+0x000A) ;
  outb(0x00,BASEPORT+0x000C) ;
  outb(0x00,BASEPORT+0x0008) ;
  /* limpia el registro de salida FIFO*/
  lec1 = inb(BASEPORT+0x000A);
  lec2 = inb(BASEPORT+0x000A);
}
```

```
int ADFIFOregister(void){
  int lec1,lec2;
  int res; /*aqui falta para c2*/
  int MASK = 128;
  int MASKCLEAR = 0x000000FF;
  if ((0x004 & inb(BASEPORT)) >> 2){ //Error de Overflow!!
    return 5000; //estos valores jamas son alcansados por la tarjeta
  }else if ((0x002 & inb(BASEPORT)) >> 1){ //Error de Overrrun!!
    return -5000; //estos valores jamas son alcansadospor la tarjeta
  }else{
    lec1= (MASKCLEAR & inb(BASEPORT+0x000A)); //byte bajo
    lec2= (MASKCLEAR & inb(BASEPORT+0x000A)); //byte alto
    if ( (lec2 & MASK) == 128){ //es negativo
      lec2 = (0xFFFFF000 | (lec2 << 8));
    }else{
      lec2 = (lec2 << 8); //es positivo
    }
    res = (lec1 | lec2);
    return res;
  }
}

void empezarConversion(void){
   outb(0x04,BASEPORT+0x01);
}

void datoListo(){
while(!(inb(BASEPORT) & 1)) /*inb(0x260)*/ ;
}

void commandRegister1(char var){
  outb(var,BASEPORT); /*commandregister1*/
}

void commandRegister2(char var){
  outb(var,BASEPORT + 0x01); /*commandregister1*/
}

void commandRegister4(char var){
  outb(var,BASEPORT+0x0F); /*commandregister4*/
}

void counterAModeRegister(char var){
  outb(var,BASEPORT+0x17);
}

void counterA0DataRegister(char var){
  outb(var,BASEPORT+0x14);
}

void ADClearRegister(char var){
  outb(var,BASEPORT+0x08);
}

int leerReg(){
  return inb(BASEPORT+0x000A);
}
```

J.2. Objeto.h

```
/*Estructura para definir nuestra entidad minima de informacion
 (16 bits)*/

typedef struct _buffer{
  char *datos;
  int tamano; //cantidad de datos a meter.
  int indice; //indice en el arreglo.
  int actual; //contador de datos.
}Buffer;

typedef struct _cola{
  char *datos;
  int tamano;
  int indice;
}Cola;
```

J.3. Buffer.c

```
#include<stdio.h>
#include<stdlib.h>
#include<Objeto.h>

Buffer nuevoBuffer(int tamano){
  Buffer buffer;
  buffer.datos = (char *)malloc(5*sizeof(char)*tamano+1); //la estructura
                                                          //maxima es de 5 bytes
  buffer.tamano = tamano;
  buffer.indice = 0; //el indice del buffer de 1 en 1 (byte X byte)
  buffer.actual = 0; //El indice de la estructura!
  return buffer;
}

/* regresa 1 para verdadero*/
int bufferLleno(Buffer buffer){
  if (buffer.actual == buffer.tamano) return 1; //verdadero
  else
    return 0;
}

/*Regresa 0 si esta lleno */
/* modo 0 dato
        1 media
        2 cambio de estado
Para dato y media : tipo = hora = 0;
para cambio de estado
        tipo: 0 INICIO
              1 FIN
              2 CAMBIO DE AMPLIFICACION
              3 CAMBIO DE FRECUENCIA
              4 INICIA EVENTO
              5 TERMINA EVENTO
```

```
            6 DATOS PARA LA MEDIA
            7 VALOR DE EPSILON
            8 NUMERO DE EPSILONS
            9 EDO DE CALIBRACION

Representacion simbolica binaria
modo                                                                    40
0 -> 00XXDDDD DDDDDDDD
1 -> 01DDDDDD DDDDDDDD
2 -> 10TTTTTD DDDDDDDD DDDDDDDF FFFFFFFF FFFFFFFF
     87654321 87654321 87654321 87654321 87654321
Donde X no importa
      D {0,1} Que representan el dato
      T {0,1} Que representan el tipo de cambio
      F {0,1} Que representan la hora en el sig formato
              T HHHHMMMM MMSSSSSS
              T 0 = AM                                                  50
                1 = PM
                H = {0,1} horas (1..12)
                M = {0,1} minutos (0..59)
                S = {0,1} segundos (0..59)
*/

int agregarABuffer(Buffer *buffer, int modo, int tipo,int dato, int hora){
  // int empaquetador = 0;
  int mask;
                                                                        60


  if (bufferLleno(*buffer)) return 0;
  else{
    buffer->actual++; //anadimos otro
    switch(modo){
    case 0:
      mask = 0x0F00; //0000 1111 0000 0000
      //00XXDDDD
      buffer->datos[buffer->indice] = (char)((dato & mask) >> 8); // 0000 1111   70
      mask = 0x0FF; // 0000 1111 1111
      buffer->indice++;
      buffer->datos[buffer->indice] = (char)(dato & mask); // 1111 1111
      buffer->indice++;   //ok
      return 1;

    case 1:
      mask = 0x3F00; //0011 1111 0000 0000
      //01DDDDDD
      buffer->datos[buffer->indice] = (char)(((dato & mask) >> 8) | 0x40); //ponemos 01   80
      buffer->indice++;                                          // 0111 1111
      mask = 0x0FF; //0000 1111 1111
      buffer->datos[buffer->indice] = (char)(dato & mask); // 1111 1111
      buffer->indice++;
      return 1;

    case 2:
      buffer->datos[buffer->indice] = (char)((0x02 << 6) | ((tipo & 0x01F) << 1) | ((dato & 0x8000) >> 15)) ;
      buffer->indice++;
      buffer->datos[buffer->indice] = (char)((dato & 0x7F80) >> 7);          90
      buffer->indice++;
      buffer->datos[buffer->indice] = (char)(((dato & 0x7F) << 1) | ((hora & 0x10000) >> 16));
```

```
    buffer->indice++;
    buffer->datos[buffer->indice] = (char)((hora & 0x0FF00) >> 8);
    buffer->indice++;
    buffer->datos[buffer->indice] = (char)(hora & 0x0FF);
    buffer->indice++;
    /*
    mask = 0x01F; // 0000 0001 1111
    mask2 = 0x8000; // 1000 0000 0000 0000                                    100
    //10TTTTTD                    ??TTTTT?      10??????     ???????D
    op1 = (tipo & mask) << 1;
    //      printf("op1 %u\n", op1);
    op2 = (dato & mask2) >> 15;
    //printf("op2 %u\n", op2);
    op3 = (op1 | op2);
    //printf("op3 %u\n", op3);
    op4 = (op3 | 0x80);
    paquete = op4;
    //printf("op4 %u\n", op4);                                               110
    //printf("op4 %i\n", paquete);
    buffer->datos[buffer->indice] = paquete;
    //      printf(" %u", buffer->datos[buffer->indice]);
    buffer->indice++;
    mask = 0x7F80;          //   (0000111111110000000 >> 7) => 00000000 11111111
    buffer->datos[buffer->indice] = (dato & mask) >> 7;
    buffer->indice++;
    mask =0x7F;             //     000000001111111 << 1 => 0001111111?
    mask2 = 0x10000;
    buffer->datos[buffer->indice] = ((dato & mask) << 1) | ((hora & mask2) >> 16);  120
    buffer->indice++;
    mask = 0xFF00;    //          0001111111100000000 >> 8 => 0000000111111111
      buffer->datos[buffer->indice] = (hora & mask) >> 8;
      buffer->indice++;
    mask = 0xFF;    //          00011111111
      buffer->datos[buffer->indice] = (hora & mask);
      buffer->indice++;*/
    return 1;
   }
  }                                                                           130
}

void vaciarBuffer(Buffer *buffer){
  buffer->indice = 0;
  buffer->actual = 0;
}

void borrarBuffer(Buffer *buffer){
  free(buffer->datos);
  buffer->indice = 0;                                                         140
  buffer->tamano = 0;
  buffer->actual = 0;
}

/*Buffer cambiarTamano(Buffer buffer);*/
int agregarChars(Buffer *buffer, int modo, char *datos){
  if (bufferLleno(*buffer)) return 0;
  else{
    buffer->actual++; //anadimos otro
    if (modo == 0){                                                           150
      buffer->datos[buffer->indice] = datos[0];
```

```
      buffer->indice++;
      buffer->datos[buffer->indice] = datos[1];
      buffer->indice++;
      return 1;
    } else {
      buffer->datos[buffer->indice] = datos[0];
      buffer->indice++;
      buffer->datos[buffer->indice] = datos[1];
      buffer->indice++;
      buffer->datos[buffer->indice] = datos[2];
      buffer->indice++;
      buffer->datos[buffer->indice] = datos[3];
      buffer->indice++;
      buffer->datos[buffer->indice] = datos[4];
      buffer->indice++;
      return 1;
    }
  }
  return 1;
}
```

J.4. Guardar.c

```
#include<stdio.h>
#include<stdlib.h>
#include<Guardar.h>
#include <unistd.h>

void guardarCabezera(FILE *archivo)
{
  fprintf(archivo,"<!------------------------------------------------->\n");
  fprintf(archivo,"<!-- Por xquetzal                    -->\n");
  fprintf(archivo,"<!--      Tabla de Codigos           -->\n");
  fprintf(archivo,"<!-- modos:                          -->\n");
  fprintf(archivo,"<!--    00: Dato                     -->\n");
  fprintf(archivo,"<!--    01: Media                    -->\n");
  fprintf(archivo,"<!--    02: Cambio de Estado:        -->\n");
  fprintf(archivo,"<!--                00: Inicio       -->\n");
  fprintf(archivo,"<!--                01: Fin          -->\n");
  fprintf(archivo,"<!--                02: Amplificacion -->\n");
  fprintf(archivo,"<!--                03: Frecuencia   -->\n");
  fprintf(archivo,"<!--                04: Evento (Inicia) -->\n");
  fprintf(archivo,"<!--                05: Evento (Termina) -->\n");
  fprintf(archivo,"<!--                06: Datos para la media -->\n");
  fprintf(archivo,"<!--                07: Datos para la Epsilon -->\n");
  fprintf(archivo,"<!--                08: Numero de Epsilon -->\n");
  fprintf(archivo,"<!--                09: Estado de Calibracion -->\n");
  fprintf(archivo,"<!------------------------------------------------->\n");
  fprintf(archivo,"<!-- Ejemplos                        -->\n");
  fprintf(archivo,"<!-- 00 1234                         -->\n");
  fprintf(archivo,"<!-- Dato = 1234                     -->\n");
  fprintf(archivo,"<!-- 01 34                           -->\n");
  fprintf(archivo,"<!-- Media = 34                      -->\n");
```

```
  fprintf(archivo,"<!-- 02 02 3 12:10:33                    -->\n");
  fprintf(archivo,"<!-- Cambio de Estado, Amplificacion, con un valor-->\n");
  fprintf(archivo,"<!-- de 3 hecho a las 12:10:33            -->\n");
  fprintf(archivo,"<!------------------------------------------------->\n");
}


int convierteHora(int ampm, int hora, int min, int seg){
  int tiempo = 0;
  tiempo = (0x01 & ampm) << 16;                                          40
  tiempo = tiempo | (0x0F & hora) << 12;
  tiempo = tiempo | (0x3F & min) << 6;
  tiempo = tiempo | (0x3F & seg);
  return tiempo;
}

int guardarBuffer(Buffer buffer, FILE *archivo, int modoDGuardar){ //modoDGuardar::= 0->ASCII | 1->Binario.
  int tam;
  //int modo;
  int i,lec0,lec1,lec2,lec3,lec4;                                        50
  int res; /*aqui falta para c2*/

  char z;
  if (modoDGuardar){ //==1
    tam = buffer.indice;
    for(i = 0; i < tam; i++){ //aqui es -1
      fwrite(&(buffer.datos[i]),sizeof(char),1,archivo);
    }
    fflush(archivo);
    return 1;                                                            60
  }else if (modoDGuardar == 0){
    /*Guardando en modo texto*/
    int index=0;
    while(index < buffer.indice-1){ //mientra hay datos
      z=buffer.datos[index]; //leo un caracter!
      //0xC0 1100 0000
      switch ( (z&0xC0) >>6){ //Leo el modo: 0 = dato; 1 = media; 2 = cambio
      case 0 : //es un dato! 00XXDDDD DDDDDDDD
        lec2 = z&0x0F; //4 primeros digitos 0000 1111
        lec1 = 0x0FF & buffer.datos[++index]; // 8 segundos digitos      70
        if ((lec2 & 0x08) == 8) { //es negativo 0000 1000
          lec2 = (0xFFFFF000 | (lec2 << 8));
        }else{
          lec2 = (0x00000FFF & (lec2 << 8)); //es positivo
        }
        res = (lec1 | lec2);
        fprintf(archivo,"00 %i\n",res);
        index++;
        break;
      case 1 : //es Media! 01DDDDDD DDDDDDDD                           80
        lec2 = z&0x3F; //6 primeros digitos 0011 1111
        lec1 = 0x0FF &buffer.datos[++index]; // 8 segundos digitos
        if ((lec2 & 0x020) == 32){ //es negativo
          lec2 = (0xFFFFC000 | (lec2 << 8)); // 1100 0000 0000 0000
        }else{
          lec2 = (0x00003FFF & (lec2 << 8)); //es positivo
        }
        res = (lec1 | lec2);
        fprintf(archivo,"01 %i\n",res);
```

```
        index++;
        break;
      case 2 : //es Cambio de Estado! 10DDDDDD DDDDDDDD
        lec0 = z;
        lec1 = 0x0FF &buffer.datos[++index];
        lec2 = 0x0FF &buffer.datos[++index];
        lec3 = 0x0FF &buffer.datos[++index];
        lec4 = 0x0FF &buffer.datos[++index];
        index++;
        fprintf(archivo,"02,%i, ", ((lec0 & 0x3E) >> 1)); //pinto el modo y el tipo
        fprintf(archivo,"%i, ", ((lec0 & 0x1) << 15) | ((lec1& 0xFF) << 7) | ((lec2 & 0xFE) >> 1));//dato 100
        if(lec2 & 0x01) fprintf(archivo,"%i:",((lec3 & 0xF0) >> 4) + 12); //pm
        else
          fprintf(archivo,"%i:",(lec3 & 0xF0) >> 4); //am
        fprintf(archivo,"%i:",((lec3 & 0x0F) << 2) | ((lec4 & 0xC0) >> 6)); //minutos
        fprintf(archivo,"%i\n",lec4 & 0x3F);
        break;
      }
    }
    fflush(archivo);
    return 1;
  }
  return 0;
} //Termina guardaBuffer.

int guardarCola(Cola cola, FILE *archivo, int modoDGuardar){ //modoDGuardar::= 0->ASCII | 1->Binario.
  return 0;
}
```

J.5. interface.c

El código de interface es demasiado grande para incluirlo como un apéndice, puede ser encontrado en:

http://cintli.igeofcu.unam.mx/xquetzal

J.6. main.c

```
#ifdef HAVE_CONFIG_H
# include <config.h>
#endif

#include <gtk/gtk.h>
#include "callbacks.h"
#include "interface.h"
#include "support.h"
#include "stdio.h"
#include "LabPC.h"

gint *configuracion;
GtkWidget *drawingarea1 = NULL;
GtkWidget *drawingarea2 = NULL;
GtkWidget *drawingarea3 = NULL;
```

```
GtkWidget *drawingarea4 = NULL;


int
main (int argc, char *argv[])
{
  GtkWidget *windowPrincipal;

#ifdef ENABLE_NLS
  bindtextdomain (PACKAGE, PACKAGE_LOCALE_DIR);
  textdomain (PACKAGE);
#endif

  if(tarjetaDisponible()){
  gtk_set_locale ();
  gtk_init (&argc, &argv);
  add_pixmap_directory (PACKAGE_DATA_DIR "/pixmaps");
  add_pixmap_directory (PACKAGE_SOURCE_DIR "/pixmaps");


  configuracion = leerConfiguracion();
  windowPrincipal = create_windowPrincipal ();
  gtk_widget_show (windowPrincipal);
  popUp("xquetzal Ver 0.5");
  gtk_main ();
  }else{
    g_print("\nERROR 01: TARJETA NO DISPONIBLE\n");
  }
  return 0;
}
```

J.7. callbacks.c

```
#ifdef HAVE_CONFIG_H
# include <config.h>
#endif

#include <gtk/gtk.h>
#include <stdlib.h>
#include <stdio.h>
#include "callbacks.h"
#include "interface.h"
#include "support.h"
#include "graficos.h"

extern gint *configuracion;
extern GtkWidget *drawingarea1;
extern GtkWidget *drawingarea2;
extern GtkWidget *drawingarea3;
extern GtkWidget *drawingarea4;
GdkPixmap *pixmap = NULL;


void
CloseDialog(GtkWidget *widget, gpointer data){
  gtk_widget_destroy(GTK_WIDGET(data));
```

```
}

void
ClosingDialog(GtkWidget *widget, gpointer data){
  gtk_grab_remove(GTK_WIDGET(widget));
}

/*funcion que abre una ventana y despliega un mensaje*/
void popUp(char *szMessage){
  static GtkWidget *label;
  GtkWidget *button;
  GtkWidget *dialog_window;

  dialog_window = gtk_dialog_new();
  gtk_signal_connect(GTK_OBJECT(dialog_window), "destroy",
                     GTK_SIGNAL_FUNC(ClosingDialog),
                     &dialog_window);

  gtk_window_set_title(GTK_WINDOW(dialog_window), "MENSAJE");
  gtk_container_border_width(GTK_CONTAINER(dialog_window),5);

  label = gtk_label_new(szMessage);
  gtk_misc_set_padding(GTK_MISC(label),10,10);
  gtk_box_pack_start (GTK_BOX(GTK_DIALOG(dialog_window)->vbox),
                      label,TRUE,TRUE,0);
  gtk_widget_show(label);

  button = gtk_button_new_with_label("Aceptar");

  gtk_signal_connect(GTK_OBJECT(button),"clicked",
                     GTK_SIGNAL_FUNC(CloseDialog),dialog_window);
  GTK_WIDGET_SET_FLAGS(button, GTK_CAN_DEFAULT);
  gtk_box_pack_start(GTK_BOX(GTK_DIALOG(dialog_window)->action_area),
                     button,TRUE,TRUE,0);

  gtk_widget_grab_default(button);
  gtk_widget_show(button);
  gtk_widget_show(dialog_window);
  gtk_grab_add(dialog_window);
}


void
SiNo(char *sMessage, void (*YesFunc)(), void (*NoFunc)()){
  GtkWidget *label;
  GtkWidget *button;
  GtkWidget *dialog_window;

  dialog_window = gtk_dialog_new();

  gtk_signal_connect(GTK_OBJECT(dialog_window),"destroy",
                     GTK_SIGNAL_FUNC(ClosingDialog),
                     &dialog_window);

  gtk_window_set_title(GTK_WINDOW(dialog_window),"Informacin");
  gtk_container_border_width(GTK_CONTAINER(dialog_window),5);

  label = gtk_label_new(sMessage);
```

```
  gtk_misc_set_padding(GTK_MISC(label),10,10);
  gtk_box_pack_start(GTK_BOX(GTK_DIALOG(dialog_window)->vbox),
                     label,TRUE,TRUE,0);

  gtk_widget_show(label);

  button = gtk_button_new_with_label("Aceptar");
  gtk_signal_connect(GTK_OBJECT(button),"clicked",                          90
                     GTK_SIGNAL_FUNC(YesFunc),
                     dialog_window);

  GTK_WIDGET_SET_FLAGS(button,GTK_CAN_DEFAULT);
  gtk_box_pack_start(GTK_BOX(GTK_DIALOG(dialog_window)->action_area),
                     button,TRUE,TRUE,0);
  gtk_widget_show(button);


  button = gtk_button_new_with_label("Cancelar");                           100
  gtk_signal_connect(GTK_OBJECT(button),"clicked",
                     GTK_SIGNAL_FUNC(NoFunc),
                     dialog_window);

  GTK_WIDGET_SET_FLAGS(button,GTK_CAN_DEFAULT);
  gtk_box_pack_start(GTK_BOX(GTK_DIALOG(dialog_window)->action_area),
                     button,TRUE,TRUE,0);
  gtk_widget_grab_default(button);
  gtk_widget_show(button);
                                                                            110
  gtk_widget_show(dialog_window);

  gtk_grab_add(dialog_window);
}

void
salirSi(GtkWidget *widget, gpointer data){
  gtk_main_quit();
}
                                                                            120
void
salirNo(GtkWidget *widget, gpointer data){
  gtk_widget_destroy(GTK_WIDGET (data));
}

void
on_salir1_activate                      (GtkMenuItem *menuitem,
                                         gpointer     user_data)
{
  SiNo("Salir de xquetzal",salirSi,salirNo);                                130
}



/*metodo que lee de .xquetzal*/
gint *leerConfiguracion(void){
  gint *config;
  gint iesimo;
  FILE *origen;
  config = (gint*)malloc(sizeof(gint)*35);                                  140
  origen = fopen(".xquetzal","rb");
```

```
  /*leemos la configuracion*/
  for(iesimo=0;iesimo < 35;iesimo++){
    fread(&config[iesimo],sizeof(gint),1,origen);
    fflush(origen);
  }
  fclose(origen);
  return config;
}

  // if(gtk_toggle_button_get_active(checkbuttonCanal1) printf("REY!!!");


void
on_checkbuttonCanal1_toggled       (GtkToggleButton *togglebutton,
                                    gpointer         user_data)
{
   if(((gint*)configuracion)[0] == 0)
    ((gint*)configuracion)[0] = 1;
  else
    ((gint*)configuracion)[0] = 0;

}

void
on_checkbuttonCanal2_toggled       (GtkToggleButton *togglebutton,
                                    gpointer         user_data)
{
   if(((gint*)configuracion)[1] == 0)
    ((gint*)configuracion)[1] = 1;
  else
    ((gint*)configuracion)[1] = 0;
}


void
on_checkbuttonCanal3_toggled       (GtkToggleButton *togglebutton,
                                    gpointer         user_data)
{
   if(((gint*)configuracion)[2] == 0)
    ((gint*)configuracion)[2] = 1;
  else
    ((gint*)configuracion)[2] = 0;
}

void
on_checkbuttonCanal4_toggled       (GtkToggleButton *togglebutton,
                                    gpointer         user_data)
{
  if(((gint*)configuracion)[3] == 0)
    ((gint*)configuracion)[3] = 1;
  else
    ((gint*)configuracion)[3] = 0;
}


void
on_buttonSalvarConfiguracion_clicked(GtkButton *button,
                                    gpointer         user_data)
```

```
{
  FILE *destino;
  // int iesimo;
  int j;
  destino = fopen(".xquetzal","w+b");
  for(j=0;j<35;j++){
    fwrite(&((gint*)configuracion)[j],sizeof(int),1,destino);
  }
  fclose(destino);
}

void
amplificacionAutomaticaActivate(GtkButton *button, gpointer user_data){
((gint*)configuracion)[4] = 0;
// printf("HOOLA");
}

void
amplificacionManualActivate(GtkButton *button, gpointer user_data){
((gint*)configuracion)[4] = 1;
//printf("PIANOLA");
}

void visualiza(gpointer user_data){
  gint iesimo;
  printf("\n");
  for(iesimo=0;iesimo < 35;iesimo++){
    printf("%i,",((gint*)configuracion)[iesimo]);
  }
}

/* SALVAR CONFIG AMPLIFICACION MANUAL TIPO*/
void
amplificacionManual1(GtkButton *button, gpointer user_data){
configuracion[5] = 0;
}

void
amplificacionManual1_25(GtkButton *button, gpointer user_data){
configuracion[5] = 1;
}

void
amplificacionManual2(GtkButton *button, gpointer user_data){
configuracion[5] = 2;
}

void
amplificacionManual5(GtkButton *button, gpointer user_data){
configuracion[5] = 3;
}

void
amplificacionManual10(GtkButton *button, gpointer user_data){
configuracion[5] = 4;
}
```

```
void
amplificacionManual20(GtkButton *button, gpointer user_data){
configuracion[5] = 5;
}

void
amplificacionManual50(GtkButton *button, gpointer user_data){
configuracion[5] = 6;
}

void
amplificacionManual100(GtkButton *button, gpointer user_data){
configuracion[5] = 7;
}

void
escalaManual(GtkButton *button, gpointer user_data){
configuracion[6] = 1;
}

void
escalaAutomatica(GtkButton *button, gpointer user_data){
configuracion[6] = 0;
}


void
capturaLibre(GtkButton *button, gpointer user_data){
  configuracion[10] = 0;
}

void
capturaControlada(GtkButton *button, gpointer user_data){
configuracion[10] = 1;
}


void
presentacionVentana(GtkButton *button, gpointer user_data){
configuracion[12] = 0;
}

void
presentacionPantallaCompleta(GtkButton *button, gpointer user_data){
configuracion[12] = 1;
}

void
autoGuardarActivado(GtkButton *button, gpointer user_data){
configuracion[13] = 0;
}


void
autoGuardarDesactivado(GtkButton *button, gpointer user_data){
configuracion[13] = 1;
}
```

```
void
formatoASCCI(GtkButton *button, gpointer user_data){
configuracion[27] = 0;
}

void
formatoComprimido(GtkButton *button, gpointer user_data){
configuracion[27] = 1;
}

void
escalaX(GtkButton *button, gpointer user_data){
  configuracion[7] = atoi(gtk_entry_get_text(GTK_ENTRY(button)));
}

void
escalaY(GtkButton *button, gpointer user_data){
  configuracion[8] = atoi(gtk_entry_get_text(GTK_ENTRY(button)));
}

void
frecuencia(GtkButton *button, gpointer user_data){
  configuracion[9] = atoi(gtk_entry_get_text(GTK_ENTRY(button)));
}

void
datos(GtkButton *button, gpointer user_data){
  configuracion[11] = atoi(gtk_entry_get_text(GTK_ENTRY(button)));
}

void
delta(GtkButton *button, gpointer user_data){
  configuracion[25] = atoi(gtk_entry_get_text(GTK_ENTRY(button)));
}

void
autoguardarEncendido(GtkButton *button, gpointer user_data){
  configuracion[14] = 0;
}


void
evento(GtkButton *button, gpointer user_data){
  configuracion[14] = 1;
}

void
revisarNoteBook(GtkNotebook *note, gpointer windowPrincipal){
  int r;
  GtkWidget *pagina;
  // GList *lista;
  // GtkWidget *windowPrincipal;
  //GtkWidget *note;
  static GtkWidget *hbox1;
  GtkWidget *vboxDeCanales;
  GtkWidget *frameCanal1;
  GtkWidget *scrolledwindowPrincipal;
  GtkWidget *viewport4;
  GtkWidget *frameCanal2;
```

```
GtkWidget *scrolledwindow2;
GtkWidget *viewport5;
GtkWidget *frameCanal3;
GtkWidget *scrolledwindow3;
GtkWidget *viewport6;
GtkWidget *frameCanal4;
GtkWidget *scrolledwindow4;
GtkWidget *viewport7;
GtkWidget *scrolledwindow8;
GtkWidget *viewport8;
GtkWidget *toolbar1;
GtkWidget *tmp_toolbar_icon;
GtkWidget *buttonInicia;
GtkWidget *buttonPara;
GtkWidget *buttonCalibrar;
// GtkWidget *label3;

r = gtk_notebook_get_current_page(GTK_NOTEBOOK(note));
if(r==1){
  /*LA PAGINA HA SIDO CAMBIADA PARA EMPEZAR CAPTURA!*/
  pagina = gtk_notebook_get_nth_page (GTK_NOTEBOOK(note),r);
  if (configuracion[33] == 1){
    gtk_container_remove (GTK_CONTAINER (pagina), hbox1);
  }
  hbox1 = gtk_hbox_new (FALSE, 0);
  gtk_widget_set_name (hbox1, "hbox1");
  gtk_widget_ref (hbox1);
  gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "hbox1", hbox1,
                (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (hbox1);
  vboxDeCanales = gtk_vbox_new (FALSE, 0);
  gtk_widget_set_name (vboxDeCanales, "vboxDeCanales");
  gtk_widget_ref (vboxDeCanales);
  gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "vboxDeCanales", vboxDeCanales,
                        (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (vboxDeCanales);
  gtk_box_pack_start (GTK_BOX (hbox1), vboxDeCanales, TRUE, TRUE, 0);

  if (configuracion[0] == 1){
    frameCanal1 = gtk_frame_new ("Canal 1");
    gtk_widget_set_name (frameCanal1, "frameCanal1");
    gtk_widget_ref (frameCanal1);
    gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "frameCanal1", frameCanal1,
                          (GtkDestroyNotify) gtk_widget_unref);
    gtk_widget_show (frameCanal1);
    gtk_box_pack_start (GTK_BOX (vboxDeCanales), frameCanal1, TRUE, TRUE, 0);
    scrolledwindowPrincipal = gtk_scrolled_window_new (NULL, NULL);
    gtk_widget_set_name (scrolledwindowPrincipal, "scrolledwindowPrincipal");
    gtk_widget_ref (scrolledwindowPrincipal);
    gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "scrolledwindowPrincipal",
                          scrolledwindowPrincipal,(GtkDestroyNotify) gtk_widget_unref);
    gtk_widget_show (scrolledwindowPrincipal);
    gtk_container_add (GTK_CONTAINER (frameCanal1), scrolledwindowPrincipal);

    viewport4 = gtk_viewport_new (NULL, NULL);
    gtk_widget_set_name (viewport4, "viewport4");
    gtk_widget_ref (viewport4);
    gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "viewport4", viewport4,
                          (GtkDestroyNotify) gtk_widget_unref);
```

```
  gtk_widget_show (viewport4);
  gtk_container_add (GTK_CONTAINER (scrolledwindowPrincipal), viewport4);

  drawingarea1 = gtk_drawing_area_new ();
  gtk_widget_set_name (drawingarea1, "drawingarea1");
  gtk_widget_ref (drawingarea1);
  gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "drawingarea1", drawingarea1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (drawingarea1);
  gtk_container_add (GTK_CONTAINER (viewport4), drawingarea1);
}
if (configuracion[1] == 1){
  frameCanal2 = gtk_frame_new ("Canal 2");
  gtk_widget_set_name (frameCanal2, "frameCanal2");
  gtk_widget_ref (frameCanal2);
  gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "frameCanal2", frameCanal2,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (frameCanal2);
  gtk_box_pack_start (GTK_BOX (vboxDeCanales), frameCanal2, TRUE, TRUE, 0);

  scrolledwindow2 = gtk_scrolled_window_new (NULL, NULL);
  gtk_widget_set_name (scrolledwindow2, "scrolledwindow2");
  gtk_widget_ref (scrolledwindow2);
  gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "scrolledwindow2", scrolledwindow2,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (scrolledwindow2);
  gtk_container_add (GTK_CONTAINER (frameCanal2), scrolledwindow2);

  viewport5 = gtk_viewport_new (NULL, NULL);
  gtk_widget_set_name (viewport5, "viewport5");
  gtk_widget_ref (viewport5);
  gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "viewport5", viewport5,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (viewport5);
  gtk_container_add (GTK_CONTAINER (scrolledwindow2), viewport5);

  drawingarea2 = gtk_drawing_area_new ();
  gtk_widget_set_name (drawingarea2, "drawingarea2");
  gtk_widget_ref (drawingarea2);
  gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "drawingarea2", drawingarea2,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (drawingarea2);
  gtk_container_add (GTK_CONTAINER (viewport5), drawingarea2);
}
if (configuracion[2] == 1){
  frameCanal3 = gtk_frame_new ("Canal 3");
  gtk_widget_set_name (frameCanal3, "frameCanal3");
  gtk_widget_ref (frameCanal3);
  gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "frameCanal3", frameCanal3,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (frameCanal3);
  gtk_box_pack_start (GTK_BOX (vboxDeCanales), frameCanal3, TRUE, TRUE, 0);

  scrolledwindow3 = gtk_scrolled_window_new (NULL, NULL);
  gtk_widget_set_name (scrolledwindow3, "scrolledwindow3");
  gtk_widget_ref (scrolledwindow3);
  gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "scrolledwindow3", scrolledwindow3,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (scrolledwindow3);
```

```
    gtk_container_add (GTK_CONTAINER (frameCanal3), scrolledwindow3);

    viewport6 = gtk_viewport_new (NULL, NULL);
    gtk_widget_set_name (viewport6, "viewport6");
    gtk_widget_ref (viewport6);                                                    500
    gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "viewport6", viewport6,
                              (GtkDestroyNotify) gtk_widget_unref);
    gtk_widget_show (viewport6);
    gtk_container_add (GTK_CONTAINER (scrolledwindow3), viewport6);

    drawingarea3 = gtk_drawing_area_new ();
    gtk_widget_set_name (drawingarea3, "drawingarea3");
    gtk_widget_ref (drawingarea3);
    gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "drawingarea3", drawingarea3,
                              (GtkDestroyNotify) gtk_widget_unref);                510
    gtk_widget_show (drawingarea3);
    gtk_container_add (GTK_CONTAINER (viewport6), drawingarea3);
  }
  if (configuracion[3] == 1){
    frameCanal4 = gtk_frame_new ("Canal 4");
    gtk_widget_set_name (frameCanal4, "frameCanal4");
    gtk_widget_ref (frameCanal4);
    gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "frameCanal4", frameCanal4,
                              (GtkDestroyNotify) gtk_widget_unref);
    gtk_widget_show (frameCanal4);                                                 520
    gtk_box_pack_start (GTK_BOX (vboxDeCanales), frameCanal4, TRUE, TRUE, 0);

    scrolledwindow4 = gtk_scrolled_window_new (NULL, NULL);
    gtk_widget_set_name (scrolledwindow4, "scrolledwindow4");
    gtk_widget_ref (scrolledwindow4);
    gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "scrolledwindow4", scrolledwindow4,
                              (GtkDestroyNotify) gtk_widget_unref);
    gtk_widget_show (scrolledwindow4);
    gtk_container_add (GTK_CONTAINER (frameCanal4), scrolledwindow4);
    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolledwindow4), GTK_POLICY_AUTOMATIC, GTK

    viewport7 = gtk_viewport_new (NULL, NULL);
    gtk_widget_set_name (viewport7, "viewport7");
    gtk_widget_ref (viewport7);
    gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "viewport7", viewport7,
                              (GtkDestroyNotify) gtk_widget_unref);
    gtk_widget_show (viewport7);
    gtk_container_add (GTK_CONTAINER (scrolledwindow4), viewport7);

    scrolledwindow8 = gtk_scrolled_window_new (NULL, NULL);                        540
    gtk_widget_set_name (scrolledwindow8, "scrolledwindow8");
    gtk_widget_ref (scrolledwindow8);
    gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "scrolledwindow8", scrolledwindow8,
                              (GtkDestroyNotify) gtk_widget_unref);
    gtk_widget_show (scrolledwindow8);
    gtk_container_add (GTK_CONTAINER (viewport7), scrolledwindow8);

    viewport8 = gtk_viewport_new (NULL, NULL);
    gtk_widget_set_name (viewport8, "viewport8");
    gtk_widget_ref (viewport8);                                                    550
    gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "viewport8", viewport8,
                              (GtkDestroyNotify) gtk_widget_unref);
    gtk_widget_show (viewport8);
    gtk_container_add (GTK_CONTAINER (scrolledwindow8), viewport8);
```

```
  drawingarea4 = gtk_drawing_area_new ();
  gtk_widget_set_name (drawingarea4, "drawingarea4");
  gtk_widget_ref (drawingarea4);
  gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "drawingarea4", drawingarea4,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (drawingarea4);
  gtk_container_add (GTK_CONTAINER (viewport8), drawingarea4);
}
toolbar1 = gtk_toolbar_new (GTK_ORIENTATION_VERTICAL, GTK_TOOLBAR_BOTH);
gtk_widget_set_name (toolbar1, "toolbar1");
gtk_widget_ref (toolbar1);
gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "toolbar1", toolbar1,
                          (GtkDestroyNotify) gtk_widget_unref);
gtk_widget_show (toolbar1);
gtk_box_pack_start (GTK_BOX (hbox1), toolbar1, FALSE, FALSE, 0);

tmp_toolbar_icon = create_pixmap (windowPrincipal, "config.xpm");
buttonInicia = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar1),
                                           GTK_TOOLBAR_CHILD_BUTTON,
                                           NULL,
                                           "Capturar",
                                           NULL, NULL,
                                           tmp_toolbar_icon, NULL, NULL);
gtk_widget_set_name (buttonInicia, "buttonInicia");
gtk_widget_ref (buttonInicia);
gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "buttonInicia", buttonInicia,
                          (GtkDestroyNotify) gtk_widget_unref);
gtk_widget_show (buttonInicia);

tmp_toolbar_icon = create_pixmap (windowPrincipal, "config.xpm");
buttonPara = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar1),
                                         GTK_TOOLBAR_CHILD_BUTTON,
                                         NULL,
                                         "Detener",
                                         NULL, NULL,
                                         tmp_toolbar_icon, NULL, NULL);
gtk_widget_set_name (buttonPara, "buttonPara");
gtk_widget_ref (buttonPara);
gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "buttonPara", buttonPara,
                          (GtkDestroyNotify) gtk_widget_unref);
gtk_widget_show (buttonPara);

tmp_toolbar_icon = create_pixmap (windowPrincipal, "config.xpm");
buttonCalibrar = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar1),
                                             GTK_TOOLBAR_CHILD_BUTTON,
                                             NULL,
                                             "Calibrar",
                                             NULL, NULL,
                                             tmp_toolbar_icon, NULL, NULL);
gtk_widget_set_name (buttonCalibrar, "buttonCalibrar");
gtk_widget_ref (buttonCalibrar);
gtk_object_set_data_full (GTK_OBJECT (windowPrincipal), "buttonCalibrar", buttonCalibrar,
                          (GtkDestroyNotify) gtk_widget_unref);
gtk_widget_show (buttonCalibrar);
gtk_container_add (GTK_CONTAINER (pagina), hbox1);
gtk_widget_queue_draw (GTK_WIDGET (note)); /* refrescamos el widget*/
configuracion[33] = 1;
```

```
    /*COnectamos los botones*/
    gtk_signal_connect(GTK_OBJECT (buttonInicia), "clicked",
                       GTK_SIGNAL_FUNC (graficador),
                       drawingarea1);
    gtk_signal_connect(GTK_OBJECT (buttonPara), "clicked",
                       GTK_SIGNAL_FUNC (para),
                       NULL);

  }

}
```

J.8. graficos.c

```
#include <math.h> /* roundf()                */
#include <gtk/gtk.h>
#include "LabPC.h"
#include "graficos.h"
#include "Guardar.h"
#include "Buffer.h"
#include <asm/io.h> /* outb(), inlb()          */

extern gint *configuracion;
extern GdkPixmap *pixmap;
//static int cont = 0;
static int miniBuffer[600]; /*los 100 son para graficar bien en pantalla*/

//Crea un mapa de pixeles de tamao adecuado
static gint configure_event(GtkWidget *widget, GdkEventConfigure *event){
  if (pixmap){
    gdk_pixmap_unref(pixmap);
  }
  pixmap =gdk_pixmap_new(widget->window,
                         widget->allocation.width,
                         widget->allocation.height,
                         -1);
  return TRUE;
}

/*Esta funcion es invocada cuando hay que redibujar la pantalla*/
gint expose_event(GtkWidget *widget, GdkEventExpose *event){
  gdk_draw_pixmap(widget->window,
                  widget->style->fg_gc[GTK_WIDGET_STATE(widget)],
                  pixmap,
                  event->area.x,event->area.y,
                  event->area.x,event->area.y,
                  event->area.width,event->area.height);
  return FALSE;
}

void fabricaGrafica(int x, gpointer lienzo){
 int i;
  GtkWidget* areaDeDibujo = (GtkWidget *)lienzo;
  GdkRectangle actualiza;
  gint B = areaDeDibujo->allocation.width;
  gint A = areaDeDibujo->allocation.height;
```

```
  gint j=0;
  gint t = 0;
  gint factor;

  /*limpiamos segundo plano*/
  gdk_draw_rectangle(pixmap,areaDeDibujo->style->white_gc,TRUE,0,0,B, A);
  /*verificamos si la escala es manual o automatica */
  if(configuracion[6] == 0){ /*automatica*/                                    50

  }else{/*acaba escala automatica*/
    /*escala manual*/
    /*recorremos todo a la derecha*/
    if( (B %configuracion[7]) < ((int)((configuracion[7])/2))) t = 0;else t = 1;
    factor = (B/configuracion[7])+t;
    for(i=0; i<599; i++ ){
      miniBuffer[i] = miniBuffer[i+1];
    }
    miniBuffer[599] = x; /*siguiente punto*/                                  60
    /*graficamos el buffer en segundo plano*/
    j = B;
    i = 600;
    while(j >= 0){
      j -= factor;
      i--;
      gdk_draw_line(pixmap,areaDeDibujo->style->black_gc,
                    j , miniBuffer[i],
                    j-factor , miniBuffer[i-1]);
    }                                                                         70
    /*repintamos (invocamos a expose_event)*/
    actualiza.x = actualiza.y = 0;
    actualiza.width = B;
    actualiza.height = A;
    gtk_widget_draw(areaDeDibujo, &actualiza);
  }/*termina escala manual*/
}

int configurarCanales(int variable){
  int var = 10;                                                               80
  if((configuracion[0]+configuracion[1]+configuracion[2]+configuracion[3])> 1){
    /*e.d. hay mas de un canal seleccionado*/
    /*CONFIGURAR MULTICANAL*/
    //    var = 10;/*!*/
    /*TERMINA CONFIGURAR MULTICANAL*/
  }else{
    if (configuracion[0]) var = 0;
    if (configuracion[1]) var = 2;
    if (configuracion[2]) var = 8;
    if (configuracion[3]) var = 12;                                           90
    /*Configurando Canal*/
    commandRegister1|(var | variable));
    //  outb(var1,BASEPORT); /*commandregister1*/
  }
  return (var | variable);
}

int configurarAmplificacion(int variable){
  int var = 0;
  if(configuracion[4]== 0){                                                   100
    /*e.d. Automatica*/
```

```
    var = 10;/*!*/
    /*TERMINA CONFIGURAR Automatica*/
  }else{
    /*Configurando Canal*/
    commandRegister1(var);
  }
  return (var | variable);
}

void graficador(GtkButton *menuitem, gpointer lienzo){
  Buffer buffer;
  FILE *destino;
  char varRegister1 = 8;//complemento a2
  int v2 = 0;
  char c = 0;
  int dato;
  int cCont =0;
  int i;
  GdkRectangle actualiza;
  GtkWidget* areaDeDibujo = (GtkWidget *)lienzo;
  unsigned ancho = areaDeDibujo->allocation.height;
  buffer = nuevoBuffer(100);
  destino = fopen("/DATOS/temporal.bin","wb"); //destino
  // guardarCabezera(destino);
  // agregarABuffer(&buffer, 2, 0, 0, )
  /*limpiamos minibuffer*/
  for(i=0; i<500+100; i++ ){
    miniBuffer[i] = ancho;
  }
  iniciaTarjeta(); /*Iniciamos la tarjeta*/
  /*--> CONFIGURACION DE LA TARJETA */
  varRegister1 = configurarCanales(varRegister1); /*configuracion de canales*/

  varRegister1 = configurarAmplificacion(varRegister1);/* configurando ganancia*/

  commandRegister1((char)varRegister1);

  /*Configuraciones multiples*/
  // outb(0x34,BASEPORT+0x17);
  /*Seleccionando Modo 2*/
  counterAModeRegister(0x34);

  /*escribiendo el byte menos significativo*/
  //outb(0x50,BASEPORT+0x14);
  v2 = (int)configuracion[9];
  v2 = v2 & 0xFF;
  c = v2;
  counterA0DataRegister(c);
  /*escribiendo el byte mas significativo*/
  //outb(0xC3,BASEPORT+0x14);
  v2 = (int)configuracion[9];
  v2 >>= 8;
  c = v2;
  counterA0DataRegister(c);
  outb(0X70,BASEPORT+0x17);
  outb(0x00,BASEPORT+0x08);
  outb(0x08,BASEPORT+0x000F) ;
```

```
ADFIFOregister(); /*limpiamos*/

/*TERMINA CONFIGURACION DE LA TARJETA*/

/*COMIENZA CAPTURA */
empezarConversion();
//Seal utilizada para manejar el mapa de pixeles de segundo plano
gtk_signal_connect(GTK_OBJECT(areaDeDibujo),"expose_event",(GtkSignalFunc)expose_event,NULL);
gtk_signal_connect(GTK_OBJECT(areaDeDibujo),"configure_event",(GtkSignalFunc)configure_event,NULL);
/*Libre o controlada*/                                                                    170
if(configuracion[10]){ //es controlada?
  /*CAPTURA CONTROLADA*/

}/*TERMINA CAPTURA CONTROLADA*/
else{ /* captura libre*/
  cCont = 0;
  if (pixmap == NULL){
    pixmap =gdk_pixmap_new(areaDeDibujo->window,
                           areaDeDibujo->allocation.width,
                           areaDeDibujo->allocation.height,                               180
                           -1);
  }
  gdk_draw_rectangle(pixmap,areaDeDibujo->style->white_gc,TRUE,0,0,
                     areaDeDibujo->allocation.width, areaDeDibujo->allocation.height);
  actualiza.x = actualiza.y = 0;
  actualiza.width = areaDeDibujo->allocation.width;
  actualiza.height = areaDeDibujo->allocation.height;
  gtk_widget_draw(areaDeDibujo, &actualiza);
  do{
    datoListo();                                                                          190
    /*      dato = ancho - (ADFIFOregister()-4294900000); */
    dato = ADFIFOregister();
    if(agregarABuffer(&buffer, 0, 0, (0x0FFF & dato), 0) == 0) { //esta lleno
      guardarBuffer(buffer,destino,1); //binario
      vaciarBuffer(&buffer);
      agregarABuffer(&buffer, 0, 0, (0x0FFF & dato), 0);
    }

    //      g_print("%u\n",dato/configuracion[8]);
    fabricaGrafica(100+dato/configuracion[8],lienzo);                                     200
    cCont++;
  }while(cCont < configuracion[11]);
} /* termina captura libre*/
/*TERMINA CAPTURA */
guardarBuffer(buffer,destino,1); //binario
borrarBuffer(&buffer);
fclose(destino);
}

                                                                                          210
void para(GtkButton *button, gpointer dato){
  g_print ("DETENIENDO");
}
```

Bibliografía

[1] ANSI. Ansi incits 135-1992 (r1998). http://www.w3.org/TR/xhtml1/, 1998.

[2] Kent Beck. *Extreme Programming Explained: Embrace Change.* Addison Wesley Longman, 2000.

[3] Silvia Bravo. *Encuentro con una Estrella.* Fondo de Cultura Economica, 1997.

[4] Asociación BULMA. Usuarios de gnu/linux de mallorca y alrededores. http://www.bulma.net, 2004.

[5] Damon Chaplin. Glade, gtk+ user interface builder. http://glade.gnome.org, 2004.

[6] National Instruments Corporation. *Lab-PC+, User Manual.* National Instruments Corporation, 1994.

[7] National Instruments Corporation. Labview. http://www.ni.com/labview/, 2004.

[8] BLFS Equipo de Desarrollo. Mas alla de linux from scratch. http://es.tldp.org/Manuales-LuCAS/blfs-es/blfs-es-5.0/general/glib2.htm%l, 2005.

[9] Desconocido. Modernización y reubicación del radiointerferometro solar ik. *IGF*, 35:39–78, 1997.

[10] Cofradia Digital. La cofradia digital: Donde el software libre se une. http://www.cofradia.org, 2004.

[11] Eckstein and Robert. *Java Swing.* O'Reilly, 1998.

[12] Institute for Geophysics Astrophysics and Meteorology. Kanzelhohe solar observatory. http://www.solobskh.ac.at/, 2005.

[13] Martin Fowler. *Refactoring : Improving the Design of Existing Code.* Addison Wesley Longman, 1999.

[14] Nathan Froyd, Tony Gale, and Shawn T. Amundson. Gtk+ faq. http://www.ii.uam.es/~so2/gtk1.2/gtkfaq-es-5.html, 2005.

[15] Free Software Fundation. The gnu general public license. http://www.gnu.org/licenses/licenses.html#GPL, 2004.

[16] Geary and David M. *Graphic Java : Mastering the AWT.* SunSoft, 1997.

[17] Eric Harlow. *Desarrollo de aplicaciones Linux con GTK+ y GDK.* Prentice Hall, 1999.

[18] National Instruments. Labview professional development system for windows. `http://sine.ni.com/nips/cds/view/p/lang/en/nid/2441`, 2005.

[19] Mark J. Kilgard. *OpenGL programming for the X Window System.* Addison-Wesley Developers, 1996.

[20] Mario and Nikis. *ESA software engineering standards Issue 2.* ESA Board for Software Standardisation and Control, 1991.

[21] Mario and Nikis. *Guide to the user requirements definition phase.* ESA Board for Software Standardisation and Control, 1991.

[22] Mario and Nikis. *Guide to the software architectural design phase.* ESA Board for Software Standardisation and Control, 1995.

[23] Mario and Nikis. *Guide to the software detailed design and production phase.* ESA Board for Software Standardisation and Control, 1995.

[24] Mario and Nikis. *Guide to the software engineering standards.* ESA Board for Software Standardisation and Control, 1995.

[25] Mario and Nikis. *Guide to the software operations and maintenance phase.* ESA Board for Software Standardisation and Control, 1995.

[26] Mario and Nikis. *Guide to the software requirements definition phase.* ESA Board for Software Standardisation and Control, 1995.

[27] Mario and Nikis. *Guide to the software transfer phase.* ESA Board for Software Standardisation and Control, 1995.

[28] Mario and Nikis. *Guide to applying the ESA software engineering standards to small software projects.* ESA Board for Software Standardisation and Control, 1996.

[29] Francisco Manuel Marquez Garcia. *Unix programacion avanzada.* Ra-Ma, 1994.

[30] Microsoft. Windows 2000 pricing and licensing. `http://www.microsoft.com/windows2000/server/howtobuy/pricing/default.as%p`, 2001.

[31] Microsoft. Sql server 2000. `http://www.microsoft.com/sql/howtobuy/default.asp`, 2005.

[32] Microsoft. Visual studio .net 2003 professional special edition. `http://msdn.microsoft.com/howtobuy/vstudio/vstudiose/default.aspx`, 2005.

[33] Nagaratnam and Nataraj. *Java networking and AWT API superbible : the comprehensive reference to the Java programming language.* Waite Group, 1996.

[34] NAOJ. Nobeyama radio observatory: Solar. `http://solar.nro.nao.ac.jp`, 2005.

[35] NASA and ESA. The solar and heliospheric observatory. `http://sohowww.nascom.nasa.gov/`, 2005.

[36] University of Hawaii. Mees solar observatory. `http://www.solar.ifa.hawaii.edu/mees.html`, 2004.

[37] OpenDX.org. Opendx. `http://www.opendx.org`, 2005.

[38] Monica Pawlan. Javasever pages technology. `http://java.sun.com/developer/onlineTraining/J2EE/Intro2/jsp/jsp.html`, 2000.

[39] Havoc Pennington. *GTK+/Gnome Application Development.* Sams, 1999.

[40] Postgresql. Tutorial de postgresql. `http://es.tldp.org/Postgresql-es/web/navegable/tutorial/x56.html`, 2005.

[41] Apache Jakarta Project. Apache jakarta tomcat. `http://jakarta.apache.org/tomcat/`, 2005.

[42] Apache Jakarta Project. Licenses. `http://www.apache.org/licenses/`, 2005.

[43] RSL. Idl the data visualization and analysis plataform. `http://www.rsinc.com/idl/`, 2005.

[44] Adrian De Leon Saldivar. Gtk: Desarrollo de aplicaciones para gnome en minutos. `http://linuxuanl.org/presentaciones_p/html/python/text8.html`, 2005.

[45] Torres Mendoza Samuel. Calibración y evaluzación de la sensibilidad del radio interferometro solar de base pequena del instituto de geofisica. *UNAM*, 1:6–15, 2002.

[46] Scheifler and Robert W. *X window system : The complete reference to xlib, x protocol, icccm, xlfd.* Digital press x and motif series, 1992.

[47] NOAA SEC. Goes x-ray flux. `http://www.sec.noaa.gov/rt_plots/xray_5m.html`, 2005.

[48] Olivier Sessink. bluefish. `http://bluefish.openoffice.nl/`, 2005.

[49] GTK Team. Gtk+ reference manual. `http://developer.gnome.org/doc/API/gtk/index.html`, 2000.

[50] GTK Team. Gtk v1. tutorial. `http://www.gtk.org/tutorial1.2/`, 2000.

[51] w3c. Html 4.01 specification. `http://www.w3.org/TR/xhtml1/`, 1999.

[52] w3c. Xhtml 1.0 the extensible hypertext markup language (second edition). `http://www.w3.org/TR/xhtml1/`, 2002.

[53] Wikipedia. Common gateway interface. `http://es.wikipedia.org/wiki/CGI`, 2005.

[54] Seiji Yashiro. Soho lasco cme catalog. `http://cdaw.gsfc.nasa.gov/CME_list/`, 2005.

[55] Zhao and Mark Overmars. Forms library. `http://world.std.com/~xforms/`, 2005.