Universidad de la República
Facultad de Ingeniería
Documentación Final grupo HAOPL(Home Automation Over PowerLines)

Juan Martin Gonzalez Dibarboure
Pablo Biagioni Walpert
Matias Tassano Ferres
Tutor: Michel Hakas Sochaczewski

16 de febrero de 2009

Agradecimientos

Debemos sin duda agradecer a las siguientes personas:

Familias González, Dibarboure, Biagioni, Cortizo, Tassano Ferrés por aguantarnos durante el desarrollo.

Alejandro Cirino por su asesoría y por el préstamo del hub utilizado para el desarrollo.

Ricardo Arvisa por la fabricación del circuito impreso y por su apoyo a los estudiantes.

Leonardo Steinfeld por su darnos las primeras armas en el ámbito de los microprocesadores y por cedérnoslo en préstamo.

Fernando Manacorda por el préstamo del dominio web y protoboard, y por sus sugerencias y enseñanzas.

Michel Hakas por su colaboración en el momento justo.

Santiago Morrison por su colaboración en el suministro de microcontroladores varios.

Santiago Gondelman por la búsqueda del chip Ethernet en Brasil.

Sin su apoyo, las cosas hubieran sido bastante más complicadas.

Índice general

Índice de figuras

Resumen

El siguiente documento corresponde a la documentación final del proyecto de fin de carrera del grupo **HAOPL**.
El producto desarrollado en este proyecto busca satisfacer la necesidad de controlar remótamente un hogar, sin tener una PC dedicada a esto instalada en dicho hogar. Por controlar debe entenderse comandar artefactos eléctricos sencillos. Este proyecto se vincula con el área de la domótica y emplea la tecnología PLCbus. El producto final se compone de dos partes: un módulo, que conllevó un desarrollo tanto de hardware como del firmware de su microcontrolador, y una página web, que conllevó el desarrollo de varios scripts escritos en lenguajes de páginas web dinámicas. El módulo se instala en el hogar y se comunica con el usuario del sistema a través de la página web, y además es quien se encarga de comandar los dispositivos eléctricos.
El documento se divide en tres partes: gestión, módulo, y servidor y en general no se ahondará en extremo en cuanto al desarrollo técnico del firmware, ya que se adjunta documentación específica del mismo.

Parte I

GESTIÓN

Capítulo 1

Introducción general

Existen en plaza diferentes dispositivos para diversos protocolos, que permiten el manejo de redes domóticas[1][1] a través de Internet. Dado que existen productos que sirven de puente entre una PC y una red domótica, se decidió usar uno de estos para lograr un producto que diera la posibilidad de precindir de una PC y tuviese a grandes rasgos las mismas funcionalidades que ofrecen los productos PC-dependientes, pero a un costo signficativamente menor.
Inicialmente se pensó en emplear el protocolo de domótica X-10[2] dado su uso estandarizado en el sector. Quien luego sería nuestro tutor nos hizo observar que existen un sinnúmero de productos desarrollados para este dada su trayectoria en el mercado y que la idea no sería muy viable. Por ese motivo y dado que existen hoy en día protocolos que mejoran sustancialmente la performance y aumentan las funcionalidades, se pensó en cambiar de protocolo. Se consideró entonces utilizar PLCbus[3], ya que no existe ningún módulo que permita la conexión remota a la red domótica. Además, se eligió este pues en comparación con X-10, presenta algunas mejoras importantes como las siguientes: mayor robustéz frente a los ruidos eléctricos, más comandos y direcciones (se permite agregar más artefactos a controlar) y quizá lo más importante: bidirecionalidad. Es por tanto que PLCbus se presenta como el sucesor de X-10, el protocolo de comunicación por líneas eléctricas más usado en la actualidad.
Si bien la alternativa era tentadora, significaba también investigar y diseñar en un área para la cual no hay prácticamente información disponible. En diferentes oportunidades se pagó el precio de incursionar en aguas desconocidas.

[1] Por domótica se entiende el conjunto de sistemas capaces de automatizar una vivienda u oficina.

Capítulo 2

Objetivo general del proyecto

El proyecto tiene como objetivo diseñar y construir un sistema de control remoto que permita a un usuario final sin conocimientos técnicos, el comando a través de Internet de una red domótica.
Permite al usuario conectarse de forma rápida y segura a través de un PC o similar (cualquiera que maneje TCP/IP) a un dispositivo central situado en el hogar, oficina, etc. que se encargará de comandar, utilizando como canal de comunicación la línea eléctrica, actuadores conectados a los electrodomésticos.
Para esto, el módulo realiza el comando de dispositivos del tipo PC-powerline[1] (existentes en mercado), módulos que permiten el control de la red domótica por intermedio de una PC. De esta forma, se aprovecharán los beneficios de los productos existentes en el mercado, eliminando la necesidad de tener un PC dedicado a brindar dicho servicio.
El sistema debe tener la capacidad de ejecutar comandos de forma instantánea así como la posibilidad de agendar los mismos y permitirle al usuario la creación y modificación de eventos agendados así como la posibilidad de crear, agendar y ejecutar escenas[2]. Este debe funcionar de forma autónoma en el sentido de que solo se necesita acceder a él para su configuración y programación de eventos.
Con esto se busca brindarles a los consumidores una manera sencilla, eficiente, rápida y cómoda de comandar remotamente aparatos eléctricos de uso doméstico. También es de fundamental importancia diseñar un producto que sea robusto y acorde a los precios del mercado de modo de satisfacer las expectativas del cliente, por lo que se debió optimizar el diseño para reducir el costo lo máximo posible.
Se busca cubrir la necesidad de una forma efectiva y eficiente de controlar de forma remota las redes domóticas sobre powerlines existentes y brindar una solución de bajo costo a un segmento del mercado actualmente desatendido, principalmente a usuarios del protocolo PLCBus quienes no cuentan actualmente con un dispositivo similar.
En la Figura 6.1 se observa un diagrama de la configuración del sistema completo. Se aprecian tres divisiones, el servidor web, el domicilio del usuario y la casa u oficina del usuario. En amarillo se muestra precisamente en que consiste el producto que se desarrolló en este proyecto.

[1] Línea eléctrica, instalación eléctrica en el hogar. Estos dispositivos se comunican mediante el cableado previamente existente.
[2] Entendido como un conjunto de comandos que se ejecutan en forma simultánea.

Capítulo 3

Autoevaluación del grupo

Los motivos del retraso al cual nos enfrentamos ya han sido expuestos en partes anteriores, pero fundamentalmente se deben a una planificación en exceso optimista. En la mayoría de las tareas los tiempos previstos fueron correctos pero no así en otras.
Resultó que la demora en estas pocas (algunas pertenecientes al camino crítico) llevó a un atraso importante. Además, se debieron realizar algunas tareas que no habían sido consideradas, y existieron algunos inconvenientes que no habían sido correctamente evaluados, o que no habían sido siquiera previstos.
Si bien podíamos haber estimado más tiempo para las importaciones previendo paros, etc., de forma de contemplar posibles problemas; bajo ningún concepto hubiéramos imaginado que íbamos a tener que invertir tiempo y recursos para conseguir drivers apropiados para el módulo 1141.
Por otra parte, inicialmente se contaba únicamente con una plataforma de desarrollo por lo cual no podían trabajar todos los integrantes del proyecto de forma simultánea lo que implicaba que mientras algunos se encontraban en una intensa actividad, otros solo podían dedicarse a tareas de preparación de las etapas siguientes. Afortunadamente esto fue solucionado al agregarse una segunda plataforma la cual, si bien era más limitada que la primera, permitió que se pudiera trabajar en paralelo lo cual contribuyó significativamente a reducir los tiempos.
En cuanto a las horas de dedicación al proyecto, la distribución no fue uniforme como se había planeado. Inicialmente se evaluó esto como algo normal debido a las actividades curriculares realizadas por el grupo durante el primer semestre, sin embargo no se revirtió luego de terminado el semestre. Además, el hecho de no contar con fechas límites estrictas dentro del periodo de ejecución del proyecto contribuyó a que algunas actividades se dilataran más de lo debido por no dedicar un esfuerzo extra para llegar en fecha. De hecho, hasta el momento quedan actividades inconclusas, que deberían haberse terminado a principio de Enero, previo al comienzo de la etapa de documentación.
En cuanto al funcionamiento como equipo, nunca se logró el nivel de comunicación requerido para el correcto entendimiento entre los integrantes. Si bien se había decidido elaborar informes semanales sobre la evolución de las tareas, esta práctica se abandonó sobre el final del proyecto. A pesar de esto, hubieron momentos de mucha colaboración entre las partes, pero fueron los menos y en general sin involucrar a la totalidad del equipo. Afortunadamente, dado el cam-

bio de enfoque del proyecto hacia tareas autónomas por medio de la división del mismo en tres módulos auto contenidos permitió reducir el impacto de la falta de comunicación, y mejorar considerablemente la eficacia y eficiencia del equipo.

Capítulo 4

Conclusión

A pesar de la culminación formal de las actividades en el marco del proyecto de fin de carrera, queda abierta la posibilidad de continuarlo con expectativas de una futura comercialización del mismo. En particular, quedan tareas pendientes relacionadas a la seguridad del intercambio de la información, así como la estandarización del protocolo de intercambio de datos y del procesamiento de los mismos. Por otra parte, se tienen en mente diversas funcionalidades no contempladas en el proyecto que lo harían un producto comercialmente más atractivo.

Desde el comienzo, el proyecto tuvo como objetivo principal ganar experiencia en el diseño de sistemas embebidos en tiempo real, y su aplicación en el tráfico de información sobre redes TCP/IP, siendo su aplicación específica en el área de la domótica una forma de establecer una restricción al diseño, y permitir de esta forma tener objetivos claros. Por otra parte, se optó desde un principio por la utilización de un microcontrolador de gama baja, de forma de lograr desarrollar habilidades en la programación con recursos escasos, tanto por un desafío personal, como por el convencimiento de que no es necesaria una gran capacidad de procesamiento para implementar un producto atractivo.

El mayor desafío que debimos enfrentar fue esta escasez de recursos de procesamiento, lo que nos obligó a utilizar nuestra inventiva con el fin de poder superarlo. Por otra parte, una complejidad no menor es el hecho de que el sistema diseñado debe ser absolutamente autónomo, y capaz de sobrellevar por si mismo problemas como el corte de energía, desconexión de la red, etc.

Todo esto nos obligó a hacer un diseño por demás acabado del sistema, exigiendo rever constantemente las decisiones tomadas, buscando darle mayor estabilidad al sistema, y optimizando el consumo de recursos como una tarea constante y sistemática.

Por otra parte estamos satisfechos con los logros obtenidos en el área de programación web, ya que la solución que requería el sistema era bastante singular, ya que el servidor debe actuar de intermediario entre el módulo y el usuario, y la experiencia en programación web era nula.

Llegada esta instancia y en empresas rumbeadas de la forma que fue rumbeada la nuestra, es decir, anteponer bajo costo de fabricación contra horas de diseõ y desarrollo que se plantee la incógnita de si esta filosofía valío la pena. Hay que decir que sin duda lo fue desde el punto de vista académico y como medio de adquisición de conocimientos para los integrantes. Por otro lado, aunque el

sistema desarrollado fue desde un inicio de recursos acotados de procesamiento y memoria del lado del módulo, no lo es del lado del servidor web de aplicaciones lo que hace posible que se puedan integrar al sistema nuevas y variadas funcionalidades sin tener que migrar a otro microcontrolador y sin grandes cambios significativos. Esta filosofía hizo posible lograr un producto con un costo de fabricación de alrededor de USD130 (sin olvidar que este es el costo de la fabricación de una sola unidad y que el mismo disminuye sensiblemente conforme la fabricación sea de más unidades). A lo largo del proyecto se logró adquirir una gran experiencia en estos temas, lo cual creemos que es más destacable que el producto obtenido ya que abre las puertas a una gran variedad de posibles aplicaciones, sobre todo teniendo en cuenta el relativo bajo costo del hardware desarrollado.

En cuanto al producto obtenido, nos sentimos satisfechos con el mismo, aunque reconocemos que aún puede desarrollarse más. Debemos destacar en este punto, que a pesar de las dudas iniciales que tuvimos acerca del objetivo del proyecto, el mismo nos satisface gratamente por ser un producto funcional y completo. Por otra parte, consideramos que la elección del protocolo a utilizar, y la apuesta a cubrir este segmento del mercado fue acertada ya que al día de hoy todavía no se dispone de un producto similar en el mercado, siendo este un producto necesario.

Finalmente, en cuanto al proyecto y la gestión del mismo, queda un sentimiento de insatisfacción, a pesar de que debemos reconocer que si se tiene en cuenta que la gestión del mismo fue realizada enteramente por los estudiantes, y la inexperiencia en proyectos de esta magnitud, la gestión debe ser considerada buena.

Se debe señalar que, la falta de responsabilidades claras y de algún tipo de liderazgo o mecanismo de toma rápida de decisiones, sumado al afán de evitar discusiones y de no "mandar" al resto de los integrantes, llevó a que en muchos casos se dilataran excesivamente tareas y decisiones que no lo ameritaban, restándole tiempo a otras que debían llevarse a cabo con mayor conciencia.

Por otra parte, el compromiso de los diferentes integrantes no fue parejo, derivando en una distribución de tareas desigual que si bien en el balance general no representa una diferencia excesiva, si lo represento en la etapa inicial del proyecto, desgastando la relación entre los integrantes y desgastando al grupo, dificultando la gestión del proyecto.

El balance general del proyecto es bueno por haber obtenido un producto funcional, de gran potencial y con recursos súmamente escazos, si bien reconocemos que hubieron falencias en cuanto a la comunicación intragrupo.

Capítulo 5

Proyección a futuro

A continuación se describen algunas funcionalidades que en algún momento del transcurso del proyecto se sugirieron, pero que luego por falta de tiempo no se implementaron.

1. **Cámaras IP**

   La idea es integrar cámaras IP al producto de manera que funcionen de forma completamente independiente del módulo. Es decir, que generen un stream de video que sea enviado al servidor de manera segura y mostrado en la página web. Una ventaja de esto es que no se requiere realizar ningún cambio al módulo y una desventaja es el costo de las cámaras IP.

2. **Calendario**

   Actualmente los comandos se agendan en forma semanal en la página web. Si por ejemplo el usuario desea agendar un comando en este momento para dentro de un mes exacto, no puede hacerlo. Esta funcionalidad prevee que los comandos se agenden en un calendario anual o mensual.
   Se implementaría directamente en el servidor sin agregar nada extra al módulo.

3. **Video Integrado**

   Tratando de satisfacer la misma necesidad que con las cámaras IP, otra opción es usar el propio módulo conectado a una cámara web para generar el stream de video. Los cambios del lado del servidor serían similares a los de utilizar cámaras IP, pero habría que hacer un trabajo adicional en el firmware del módulo para lograr el streaming de video. Esto muy probablemente llevaría a un cambio del microcontrolador, ya que sus recursos serían desbordados por esta nueva funcionalidad.

4. **Ampliación del log de eventos**

   Actualmente el log de eventos solo almacena datos referidos a la ejecución de comandos agendados. Esta funcionalidad podría ampliarse de manera de poder almacenar más comandos o incluso otro tipo de datos como:

reinicios del microcontrolador, usuarios que se han logeado a la página web, de quienes ha recibido paquetes, etc..

5. **Idiomas**

   Para obtener un producto comercialmente más atractivo sin duda habría que considerar la opción de tener un acceso web al menos en español e inglés. También sería bueno en otra etapa incluír portugués.

6. **Skins**

   Skins o apariencias. Tomando como ejemplo nada más y nada menos que a gmail, quien recientemente ha incluído skins, es una opción que sin lugar a dudas torna mucho más atractivo, customizable y menos monótono el acceso web. Más aún si es uno pensado para que el acceso al mismo sea prácticamente diario.
   No se tiene claro en el momento el esfuerzo de programación que demandaría.

7. **Captura de escenas**

   Existen dispositivos PLCbus que son capaces de generar escenas automáticamente. Es decir, si el usuario lo solicita, ingresan a la red domótica todos los comandos que se corresponden con alguna de las escenas que tienen almacenadas. Sería bueno dotar al módulo de inteligencia de manera que pueda capturar las escenas que son enviadas por dispositivos externos, y almacenarlas de manera que puedan ser ejecutadas desde la página web. Esto demandaría un esfuerzo de programación relativamente grande tanto en el servidor como en el módulo.

8. **Configuración de dispositivos externos**

   Resulta que la configuración de algunos dispositivos PLCbus o X-10 es bastante tediosa. Algunas veces requiere mantener botones apretados por varios segundos y otros mecanismos un tanto rudimentarios. Sería deseable dotar al sistema de un modo "configuración", al que se ingrese desde la página web y permita configurar otros dispositivos de la red domótica. Esto sin duda requiere un esfuerzo de programación importante.

9. **Página de configuración**

   Este punto fue quizá el más hablado durante el transcurso del proyecto. Resulta que el módulo tiene almacenados una dirección IP del grupo de las privadas, una dirección MAC y un identificador. La única forma de modificar estos parametros del módulo hoy en día es a través de la interfaz JTAG. Lo que se quería era poder configurar el módulo o incluso hacer un upgrade del firmware por ejemplo a través de una página web almacenada en el mismo, y la interfaz RJ-45. Esto en principio no sería tan complejo de llevar a cabo.

10. **Seguridad**

Quizá la tarea de mayor prioridad que quedó sin completar es hacer todos los intercambios de información de forma segura. Actualmente la comunicación entre el módulo y el servidor se realiza de dos maneras distintas[1], y solo una de ellas puede ser considerada relativamente segura.

[1] Ver capítulo 9 *Comunicación entre webserver y módulo*

Parte II

MÓDULO

Capítulo 6

Hardware

6.1. Introducción

A la hora del planteo de qué requerimientos debería cumplir el módulo a desarrollar, se concluyó que un punto de interés era lograr un sistema que fuese económico, sobretodo poniendo en la balanza la potencialidad de insertar comercialmente el producto final.
Por otro lado, había que tener en cuenta las demás necesidades: el módulo se pensó para el control remoto por medio de Internet de los actuadores PLCBus que existen en un hogar, por lo que son necesarias tanto la conectividad TCP/IP como la posibilidad de que el módulo envíe comandos PLCBus hacia el hogar.
Antes de continuar, y para evitar posteriores confusiones, creemos conveniente hacer algunas aclaraciones sobre licencias tomadas en el uso de algunos términos a lo largo de este documento.
Protocolo de comunicación con PLCBus-1141:
Haremos referencia al mismo como protocolo PLCBus o el protocolo pero debemos tener en cuenta que una cosa es el protocolo PLCBus de trasmisiones sobre líneas eléctricas, y otra muy diferente es el protocolo utilizado para la comunicación serie entre una PC y el módulo PLCBus-1141.
RS232:
Es conveniente aclarar que haremos referencia a RS232 en el sentido de la comunicación serie asíncrona y del conector DB9 (con su pinout correspondiente) pero dejando de lado los pines de señalización y utilizando únicamente los de RX, TX y GND.
Módulo:
Se mencionará módulo haciendo referencia tanto a unidades físicas bien delimitadas (ej:PLCBus-1141) como a unidades de software bien delimitadas, determinadas por la función que cumplen dentro del sistema y que pueden ser pensadas como un objeto independiente de los demás.
Ej: Módulo TCP/IP hace referencia tanto al hardware como al software cuya única función es establecer comunicaciones vía TCP/IP y que puede funcionar de forma independiente al resto del sistema.

En la Figura 6.1 se observa la arquitectura de todo el sistema

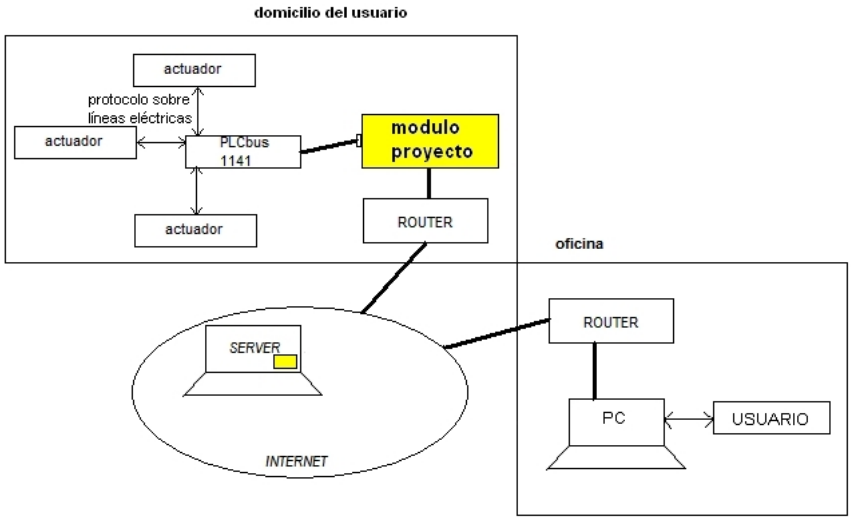


Figura 6.1: Arquitectura del sistema

6.2. Descripción por bloques

En primera instancia, una decisión importante a tomar era la del microcontrolador. Por varias razones el seleccionado fue el ATmega32, un AVR RISC de 8 bits del fabricante ATMEL[5]. Este microcontrolador posee integrados en el mismo chip numerosos módulos que proveen soluciones a las necesidades que debían solventarse. Al proveerse tantas soluciones en un mismo chip, y con lo que además hablamos de un microcontrolador de bajo costo, el costo de desarrollo de el producto se reduce sensiblemente, tanto en tiempo como en dinero. Un punto a tener en cuenta y que viene de la mano con la elección del microcontrolador a utilizar es el hardware adicional necesario para la programación y debugging del firmware. Para este propósito existe una plataforma muy económica del mismo fabricante del ATmega32 llamada AVR Dragon, y que se encontraba disponible dentro del mismo Instituto de Ingeniería Eléctrica, lo cual fue otro punto a favor de la elección del ATmega32 para el proyecto.

Para superar el hecho de que se necesitaba conectar el módulo en una red Ethernet se buscó un controlador stand-alone capaz de lograr esto y que además tuviese una interfaz soportada por el ATmega32. El elegido fue el ENC28J60 de Microchip[6], el cual posee interfaz serial SPI. Se trata de un controlador compatible con el estándar IEEE 802.3 y que integra funcionalidades de MAC y 10BASE-T PHY.

Con respecto al envío de comandos PLCBus dentro del hogar, se llegó a una implementación que no siempre fue la apuntada. Inicialmente se estudió fuertemente la posibilidad de que el propio módulo generase y transmitiese los comandos hacia la línea eléctrica. La falta de información provista del protocolo llevó a que se descartara esta opción. La solución que le siguió consiste en utilizar un módulo comercial que implementa una pasarela entre una interfaz PLCBus y RS232, el PLCBus-1141[7].

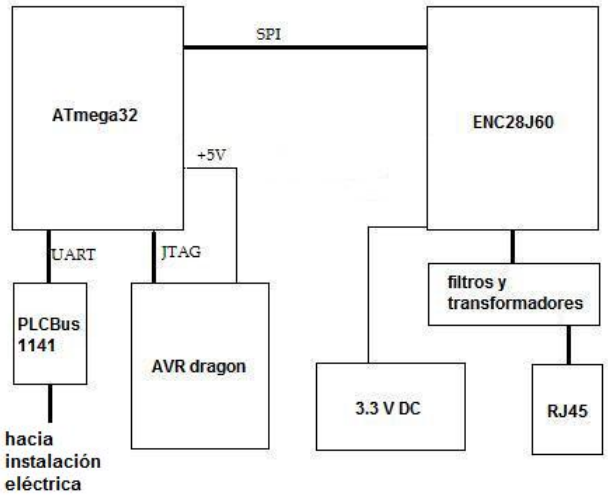


Figura 6.2: Diagrama de bloques del hardware.

En la Figura 6.2 puede observarse un diagrama de bloques del hardware utilizado para el desarrollo del sistema. Se aprecia la plataforma AVR Dragon en conexión con el ATmega32 a través de la interfaz JTAG. Esta plataforma también se utilizó para alimentar al microcontrolador en las etapas de trabajo con el hardware de prototipado.

6.3. El microcontrolador y sus interfaces

Entre otros, el ATmega32 posee (y son utilizados en el sistema) 32KB de memoria flash programable, 1KB de memoria EEPROM, 2KB de memoria SRAM, 32 líneas de I/O de propósito general, tres contadores/timers, USART, puerto serial SPI, interfaz JTAG y soporte para on-chip debugging y programación.
Para la comunicación con el controlador Ethernet se utiliza la interfaz serial SPI (Serial Peripheral Interface). Al ser un medio de transmisión sincrónico, la frecuencia de reloj debe ser la misma en ambos lados, y en este caso el que restringe la frecuencia a el valor fijo de 8MHz es el ENC28J60. El ATmega32 posee la característica de que su interfaz SPI funciona a una frecuencia igual a la mitad de su frecuencia de reloj y por esto debió suplirse a este con un cristal de 16MHz, ya que aunque el ATmega32 ofrece varias posibilidades de reloj autogenerado, no se logra esta frecuencia por estos medios.
El medio para la comunicación entre la plataforma de programación y el microcontrolador es la interfaz JTAG, la cual permite utilizarse también para el debugging y emulación. Esta fue la elegida ya que otros medios de comunicación ofrecidos por el AVR Dragon no proveen simultáneamente estas dos posibilidades (High Voltage Serial Programming, Parallel Programming, debugWire). Por el lado de la comunicación entre la PC y la plataforma se provee interfaz USB. Ya en el lado de la PC, el software utilizado para controlar el AVR Dragon se trata de el AVR Studio 4[8], provisto gratuitamente por ATMEL y que permite integrar tanto el compilador gratuito de ATMEL como el AVR-GCC, el compilador

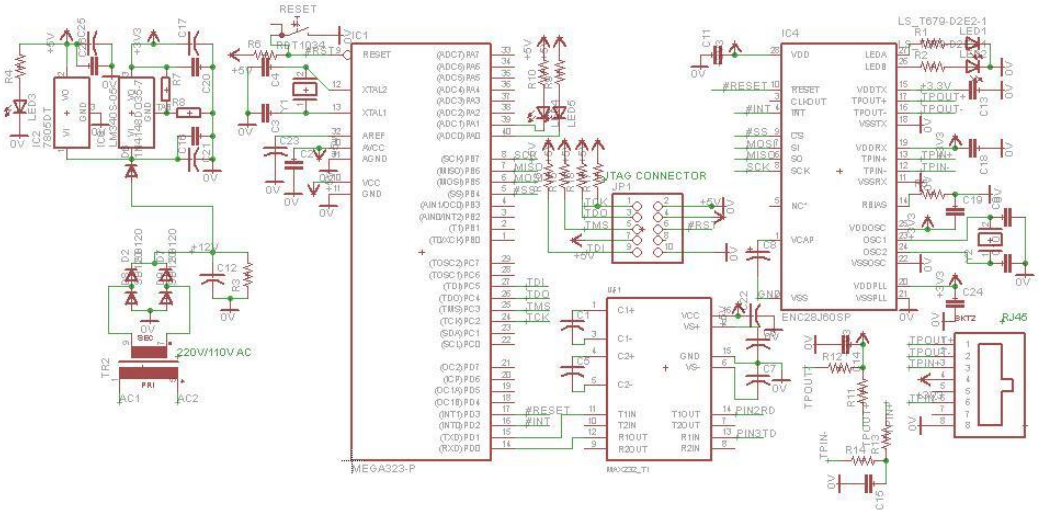


Figura 6.3: Esquemático del circuito completo.

de código libre más utilizado por la comunidad AVR.
Con respecto a la UART, esta es utilizada para la comunicación con la pasarela PLCBus-1141 la cual convierte comandos generados por el procesador a comandos del protocolo PLCBus sobre la instalación eléctrica y que se comunica con los distintos actuadores distribuidos a lo largo del hogar para que estos ejecuten las acciones deseadas por el usuario (por ej: apagar/prender lámparas, subir persianas, abrir portón, etc.). La UART del ATmega32 genera niveles de tensión que no son los especificados por el (pseudo)estándar RS232, el cual establece señales válidas en el rango [-15V, -3V] para los unos lógicos y en [+3V, +15V] para los ceros lógicos. Por este hecho es que se utiliza el conversor de niveles MAX232 de Maxim[9].

6.4. Controlador Ethernet

El controlador ofrece una implementación efectiva de capa física y capa MAC, sin embargo para lograr la integración en redes LAN fue necesario desarrollar y programar en el microcontrolador un stack TCP/IP que implementase la capas superiores. Ofrece un buffer de 8KB para transmisión y recepción de paquetes, implementa automáticamente funciones como padding programable y cálculo del CRC e incluso pueden programársele filtros de paquetes entrantes según diferentes criterios, como por ejemplo, que los paquetes entrantes posean como dirección de capa MAC de destinó la dirección que utiliza el ENC28J60.

Para lograr el correcto funcionamiento de este es necesario agregar un cristal de 25MHz entre sus pines OSC2:1 para que este pueda generar su señal de reloj. Otro requerimiento de este controlador es suministrarle una tensión de +3.3V. Para esto fue necesario armar una fuente de tensión a partir del integrado LM317[10], de donde se convierte la tensión de salida de un regulador de tensión de 220VAC/7.5VDC a 3.3VDC. Es necesario que la salida del integrado sea lo más estable posible puesto que el ENC28J60 es sensible a variaciones de

la tensión de alimentación. Se agregaron varios capacitores de desacople para superar este hecho.

Para la conexión física entre el chip ENC28J60 y el cable de Ethernet se utilizó un conector RJ45 modelo j00-0014NL de Pulse con filtros magnéticos anti EMI (Electromagnetic Interference) incorporados. El ENC28J60 recibe como entrada y salida hacia el conector 2 pares de pines, uno para transmisión y el otro para recepción.

6.5. PLCBus 1141

Esta pasarela posee dos canales de comunicación. Por un lado se conecta a la UART del procesador mediante un conector DB9, instalado por el grupo y que cortocircuita el conector USB, y por el otro se conecta a la instalación eléctrica del hogar. Sin embargo, en un principio el módulo no se alimentaba desde ninguna de estas dos posibilidades, sino que la alimentación la obtenía de la interfaz USB. Este módulo está concebido para que se le envíen instrucciones mediante su interfaz USB, que no consiste más que en una UART interna conectada a un chip conversor RS232-USB. Inicialmente se evaluó la implementación de USB dentro del módulo en desarrollo, pero por diversas razones se terminó optando por puentear la entrada al chip conversor (conexión serie de la UART del PLCBus-1141) con la conexión de la UART del microcontrolador y utilizar la interfaz USB existente para lograr la alimentación +5V. Actualmente, luego de haber superado la etapa de hardware armado sobre protoboard y habiendo logrado un circuito impreso del sistema, la pasarela se alimenta del regulador de tensión +5V existente y se comunica con el ATmega32 por medio de RS-232.

6.6. Desarrollo cronológico

Este proyecto está basado en muchos de sus aspectos iniciales en el proyecto ya existente y accesible en la web AVRnet[11] y lo relacionado al hardware no es la excepción. Pueden observarse varios aspectos en común con el proyecto antes mencionado, pero también puede observarse que aparecieron diferencias entre el hardware diseñado en cada proyecto conforme avanzaba el desarrollo del nuestro.

Inicialmente se pasó a armar una versión de prototipo sobre un protoboard. Para lograr esto, primero fue necesario realizar una importación de varios elementos al no encontrarse estos en plaza. Los artículos importados fueron el controlador ENC28J60 así como el respectivo cristal de 25MHz, el cristal de 16MHz para el microcontrolador y el conector RJ45. Luego de algunas semanas se pudo reunir todos los elementos y se procedió al armado del prototipo.

No todos los elementos utilizados actualmente lo fueron desde un principio. Para la fuente de alimentación del controlador Ethernet se empezó utilizando el integrado LP2950 el cual requiere menos componentes externos que el LM317, pero se encontró que el primero recalentaba demasiado y que su salida no era

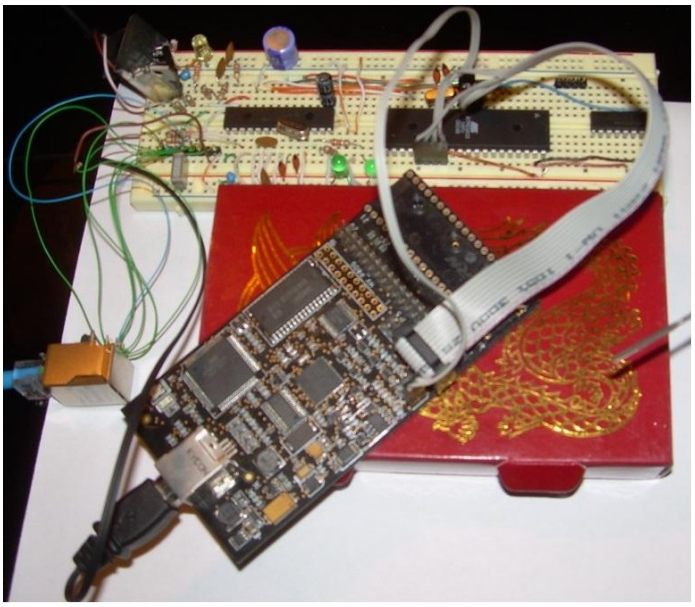


Figura 6.4: Hardware utilizado para el desarrollo.

lo estable que se requiere para que el ENC28J60 funcione correctamente. Otro punto modificado fue la utilización de un conversor de niveles entre los puertos del microcontrolador y el controlador Ethernet. El ENC28J60 funciona en el rango de [0V, +3.3V] mientras que el ATmega32 funciona entre [0V, +5V]. De esta manera se llegó a utilizar el buffer triestado 74ACT125 entre señales comunes a ambos. Una lectura en más detalle de las respectivas hojas de datos concluyó que el conversor de niveles no era necesario y se quitó[12].

Como parte del proyecto se debía controlar el envío/recepción de comandos y notificaciones por intermedio de un dispositivo PLCBus-1141. Dicho dispositivo existe en mercado en 2 versiones, una con interfaz USB y otra con RS232. Para el proyecto se pretendía utilizar la versión con RS232 ya que esto simplificaba notablemente la comunicación entre nuestro Hardware y el PLCBus-1141 sin embargo, se tenía en cuenta que esto podría ser un problema ya que el mercado tiende a dejar de lado los dispositivos con conexión RS232 y cambiar por versiones con USB. De hecho, el PLCBus-1141 que fue suministrado por nuestro socio (XTEND) trae como interfaz de fábrica un puerto USB por lo que se debió adaptar el proyecto para asegurarse que el mismo pudiera implementarse incluso en el caso hipotético de que solo existieran dispositivos con USB en el mercado.

Se consideraron diferentes opciones para solventar este tema, tales como migrar a un microcontrolador que implementase la interfaz USB o incluso utilizar una implementación de un host USB mediante una librería firmware escrita en C, pero estas opciones no resultaron atractivas ya que encarecía el producto y requería adaptar todo el proyecto a un nuevo microcontrolador para la primer opción, además de agregar el problema de que la comunicación vía USB es mucho más complejo que por medio de RS232.[13][14][15]

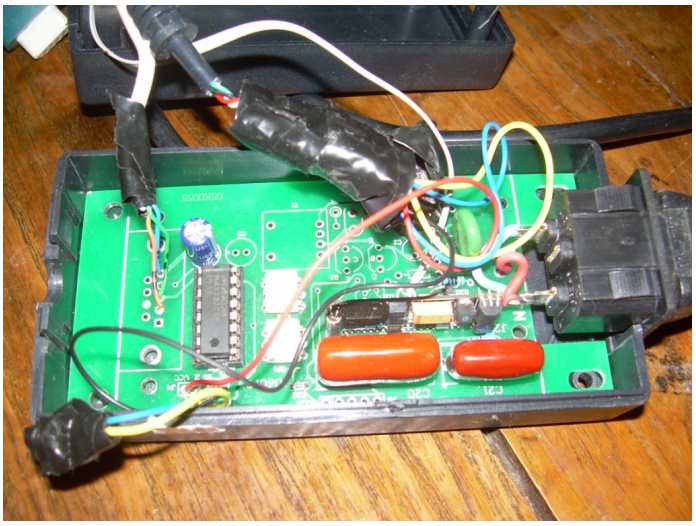


Figura 6.5: Pasarela PLCBus-1141 por dentro.

Por otra parte, se intuyó con certeza que la única diferencia entre el diseño del modelo con RS232 y el USB era un adaptador RS232-USB por lo cual podríamos tener (a menos de pequeñas modificaciones) una interfaz serie. Al desarmar el PLCBus-1141 vimos que efectivamente esto era así. De hecho se puede ver en la figura Figura 6.5 la huella en la placa para un conector DB9 utilizado en el caso del modelo con interfaz RS232. Esto reforzó la idea de trabajar con un puerto serie a pesar de todo ya que a nuestro entender carece de sentido hacer múltiples conversiones USB-RS232.

Ya llegadas las instancias finales del proyecto se empezó con la construcción de un circuito impreso de todo el sistema, exceptuando el circuito del PLCBus-1144 el cual permanece en la placa original. El diseño se llevó a cabo utilizando la versión *freeware* del famoso software de diseño Eagle[16]. El resultado puede verse en la figura Figura 6.6 y en la figura Figura 6.7 se aprecia todo el sistema armado dentro de una caja.

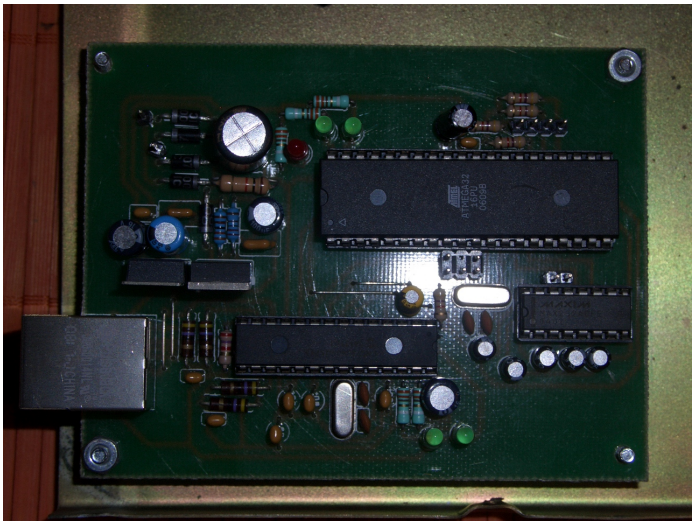


Figura 6.6: Circuito del sistema.

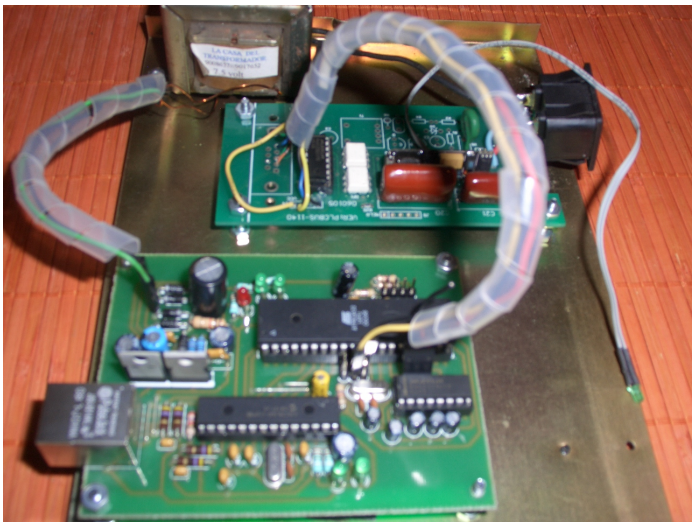


Figura 6.7: Hardware completo del sistema.

Capítulo 7

Archivos de Tablas

7.1. Introducción

El módulo en desarrollo ofrece la posibilidad al usuario de programar y agendar eventos que deben llevarse a cabo a la hora y día que este estipule. Un ejemplo de estos eventos podría ser que el usuario, quien previamente posee actuadores PLCBus para lámparas instalados, desea apagar ciertas lámparas todas las noches a las 11:30 PM. Para poder lograr agendar los eventos con diferentes comandos PLCBus que el usuario decida y que luego estos se ejecuten de la manera y en el tiempo correcto fue necesario establecer una estructura de tablas de agendas de eventos. Las mencionadas tablas se archivarán dentro de los 32kB de memoria flash que posee el microcontrolador ATmega32.

7.2. Memoria Flash

El microcontrolador utilizado, el AVR de 8bits ATmega32 de ATMEL, posee tres tipos de memoria útil para utilizar:

- Memoria RAM. 2kB. Utilizada para datos y variables en tiempo de ejecución.
- Memoria EEPROM. 1kB. Utilizada a criterio del programador.
- Memoria flash. 32kB. Utilizada principalmente para guardar código programa.

Los procesadores AVR fueron creados en una arquitectura Harvard, en donde la RAM se usa para guardar variables y la flash para guardar programa, y las dos están en espacios separados de memoria. La complicación a la hora de querer utilizar la memoria flash para guardar información aparte del código del programa se agrava más aún por el hecho de que el lenguaje C fue escrito para arquitecturas Von Neumann, en donde datos y programa coexisten en el mismo espacio de memoria.[18]
Para poder indicarle al compilador AVR-GCC de que queremos trabajar con datos en el espacio de programa, se utiliza la librería <avr/pgmspace.h>de avr-libc, en donde se definen macros para la utilización de la flash[18].

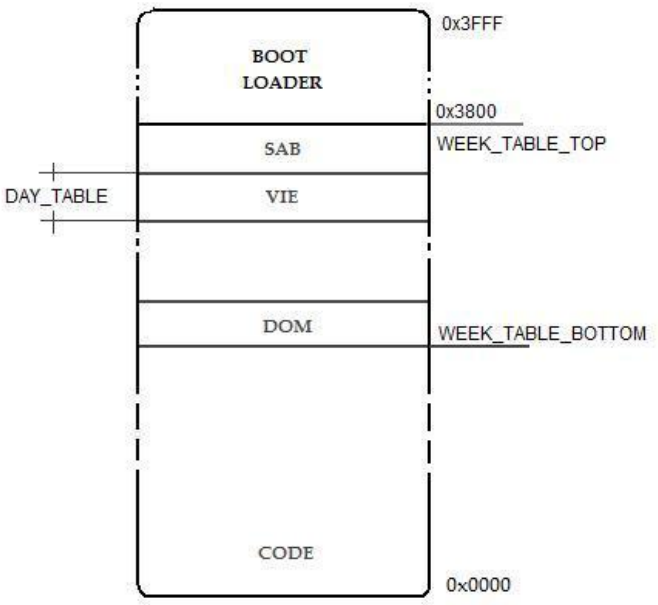


Figura 7.1: Organización de las tablas en memoria. Las direcciones están expresadas en words.

7.3. Estructura de tablas

Para que se fuese posible agendar eventos para ejecutar por parte del usuario, debe ser posible guardar la información generada a partir de esos eventos. Para ello se diseñó una estructura de tablas por día las cuales estarán ubicadas dentro de la memoria de programa.
La estructura consta de una tabla semanal, repartida en 7 tablas equivalentes, una para cada día de la semana. A cada tabla diaria se le asignó un tamaño de 10 páginas, o 1280 bytes (1,25kB).

Como se aprecia en la figura Figura 7.1, en los 2kB más altos de la memoria de programa se encuentra alojado el Boot Loader e inmediatamente por debajo se encuentra el tope superior del conjunto de las tablas. El direccionamiento a esta región de programa se logra mediante la definición y posterior utilización de las constantes WEEK_TABLE_TOP, WEEK_TABLE_BOTTOM, DAY_TABLE_LEN, entre otras.
WEEK_TABLE_TOP y WEEK_TABLE_BOTTOM marcan el final y el inicio de la tabla semanal y determinan un intervalo de 8960 bytes (8.75kB) el cual se reparte uniformemente en las tablas de cada día, empezando desde la dirección más baja como el día cero el domingo y terminando en la tabla del día seis, el sábado.
A partir de esto, las funciones que acceden a la tabla del correspondiente día reciben como argumento un número del 0 al 6 que indica a la tabla de cuál día se pretende acceder (0 indica domingo, 6 sábado); este multiplicado al largo de tabla de cada día (indicado por DAY_TABLE_LEN) es sumado a la dirección de comienzo de todas las tablas. Esto resulta la dirección de comienzo de la tabla del respectivo día.

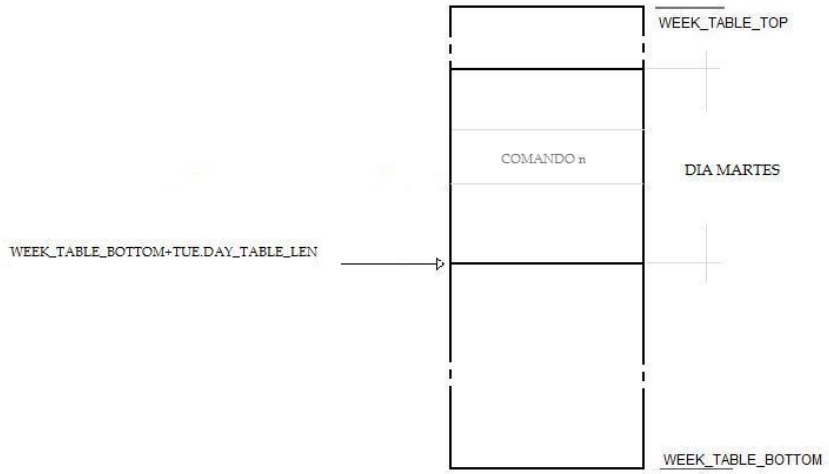


Figura 7.2: Direccionamiento en las tablas.

A lo largo de las tablas diarias están guardados los eventos que deben ejecutarse en su respectivo horario a lo largo del día. Cada evento contiene información acerca del horario a ejecutarse, la cantidad de comandos PLCBus a ejecutarse y la codificación de estos comandos. Estos eventos se encuentran ordenados por orden cronológico: el primero en ejecutarse en el día se encuentra en las direcciones más bajas de la tabla del día correspondiente.
La información de horario a ejecutarse se compone de dos campos dentro del evento: el campo OFFSET DE TIEMPO, de 16 bits, y el campo AM_PM (P en la Figura 7.3), de 1 bit. Este último indica con un 0 si la hora de ejecución del evento es en horario AM, y con un 1 si la hora es en horario PM. El campo de OFFSET DE TIEMPO indica la hora de ejecución en segundos relativa a las 12 AM/PM (dependiendo del estado del campo AM_PM). Para que efectivamente se lleven a cabo las acciones requeridas en cada evento a la hora estipulada, existen ciertas variables en el programa principal que se actualizan en tiempo real y que hacen las veces de reloj, pero no con el formato habitual, hh:mm:ss, sino con el formato utilizado en estas tablas, AM_PM:OFFSET (ver Figura 7.2). Básicamente, estas variables se comparan con los campos correspondientes del próximo evento a ejecutarse a cada segundo para verificar si la hora de ejecución es la corriente.

Para poder agendar los comandos PLBUS se requieren 4 bytes por comando. Esto lleva a que cada evento ocupe como mínimo 7 bytes,es decir, se puede lograr agendar un máximo de 182 eventos diarios.

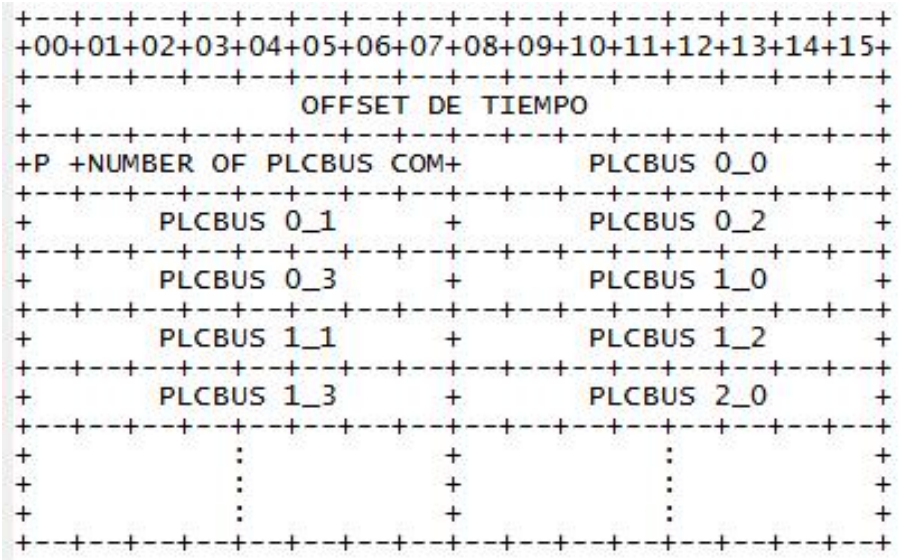


Figura 7.3: Diagrama con el formato con el que se almacenan los eventos diarios.

Capítulo 8

Stack TCP/IP

8.0.1. Diseño

Dado que el sistema es en su mayoría reactivo (en el sentido de que los procesos se bloquean a la espera de eventos) y que el flujo de los mismos es predefinido e invariante (visto desde la perspectiva de comunicaciones exitosas), se decidió utilizar para el flujo del programa el mecanismo de round-robin con interrupciones[1], y diseñar los procesos como máquinas de estado.
Para implementar este modelo, se crearon funciones wrappers de los distintos protocolos, con la finalidad de dividir el flujo en la ejecución de diferentes tareas o procesos.
Se implementaron para estas, conjuntos de funciones estandarizadas de control de flujo (XXX_start(), XXX_init(), XXX_error(), etc.)[2]y una estructura de banderas y campos de flujo para la intercomunicación entre procesos. Dicha estructura es global al proyecto, y es utilizada por el loop principal para determinar si se debe ejecutar o no algún paso de la máquina de estados del proceso XXX, y por las interrupciones y otros procesos para conocer datos como ser el estado, quien lo ejecuta, y si fue o no exitoso. Además, cada proceso tiene un campo de dirección MAC e IP correspondiente.

8.1. Estructura del main

```
   void main (void)
{

//inicializaciones

for(;;){
```

[1] Arquitectura de sistema en la cual se chequea en el loop de programa principal banderas de diferentes procesos, las cuales se activan cuando se da paso a las rutinas de atención de las correspondientes interrupciones. Al activarse una bandera, en el loop principal se llama a una función handler la cual atiende el respectivo proceso. Al finalizar se desactiva la bandera correspondiente para permitir que se vuelva a repetir el proceso citado anteriormente.[19]

[2] Aquí las XXX denotan los nombres de los protocolos tales como ARP, ICMP, TCP, SNTP, etc.

```
if(FC.ARP.on_process) ARP_step();
if(FC.ICMP.on_process) ICMP_step();
if(FC.SNTP.on_process) SNTP_step();

}
}
```

En el loop principal, se chequea si los procesos están o no activos y en caso de estarlo se da un paso en la máquina de estados. Los procesos se ponen inactivos en caso de estar a la espera de algún evento externo, y son despertados cuando este evento ocurre. El paso en que se encuentran es manejado por el handler del proceso, dentro de la función XXX_step() o por otros procesos o interrupciones en caso de error a través de la función XXX_error().
Dada la gran cantidad de archivos, interfaces publicas y funciones, se debió ser muy cuidadoso en la organización física del código. Para esto, se utiliza un archivo de inclusión global del proyecto (includes.h) el cual incluye las interfaces públicas de todos los protocolos y handlers. Por otra parte, dada la gran cantidad de estructuras utilizadas, las mismas están definidas en un único archivo (struct.h) para evitar diseminarlas por todo el código y hacer más fácil su modificación.
Un comentario aparte, es que los handlers de los protocolos también utilizan otros protocolos mediante las funciones XXX_start() lo que justifica utilizar campos XXX_requester para poder saber quien debe ser notificado en caso de éxito o error de un proceso.
Como ejemplo, SNTP_step llama a ICMP_start, que inicia un proceso ICMP. A su vez ICMP_step() llama a ARP_start() que inicia un proceso ARP.

8.2. Ejemplo de flujo normal del programa

Para mostrar el flujo del programa para el caso de uso aplicado en las pruebas Ethernet (ver página subsección 12.1.1), usamos las siguientes convenciones:

El diagrama de estados del programa es el siguiente:

Si miramos la figura Figura 8.2 horizontalmente se aprecia la evolución de los estados de cada tarea asociada a un protocolo. Si seguimos los números, se aprecia la secuencia de eventos que se suceden desde que se inicia el pedido de tiempo desde la UART hasta que se procesa el timestamp que devuelve el servidor SNTP. Como se ve, SNTP despierta a ICMP e ICMP despierta a ARP.

8.3. Recepción de paquetes

Al recibir el módulo un paquete por medio del chip controlador Ethernet, este se lo indica al microcontrolador por medio de un pedido de interrupción. Acto seguido, la rutina que atiende este pedido dentro del firmware realiza dos acciones: la primera es prender una bandera la cual indica que existe un paquete por procesar y que se chequea en el loop principal del programa y la segunda es

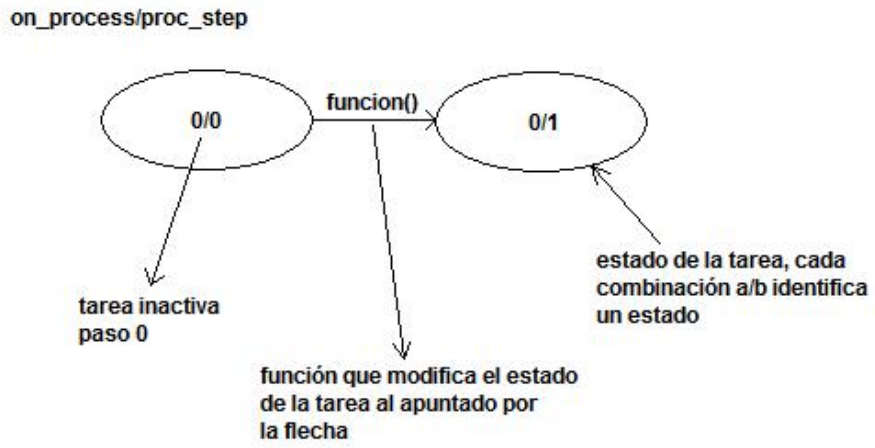


Figura 8.1: Convenciones del diagrama de estados del programa.

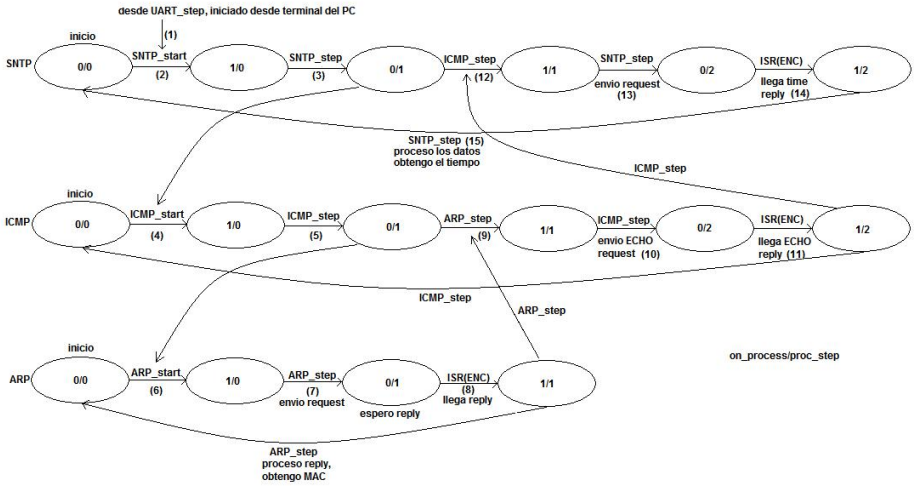


Figura 8.2: Diagrama de estados del programa para el caso de las pruebas de comunicación Ethernet realizadas.

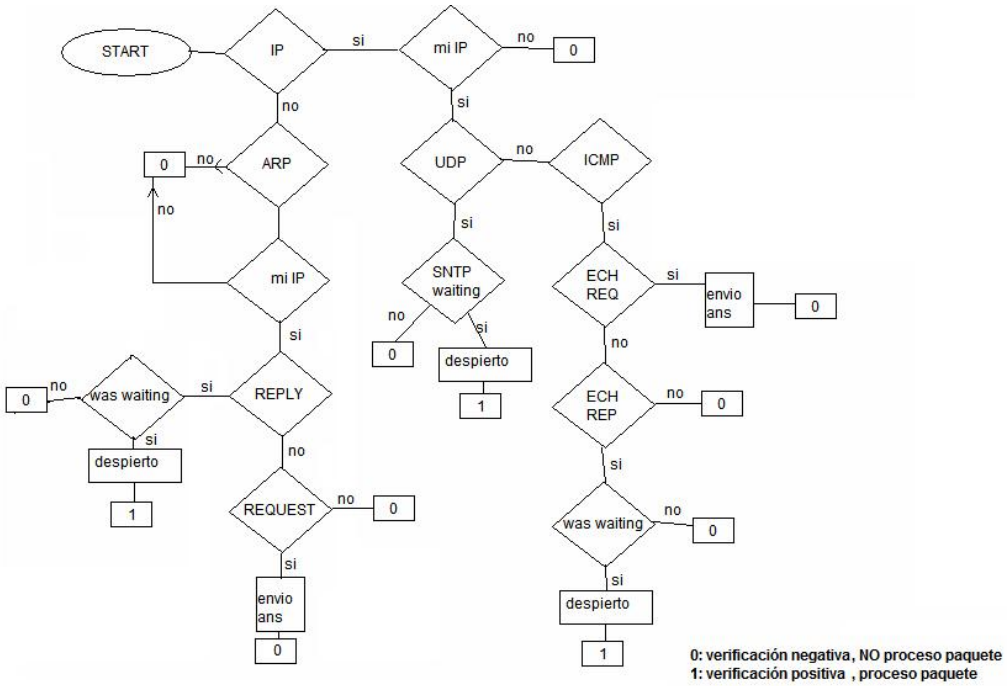


Figura 8.3: Diagrama del chequeo de protocolo

la desactivación de las interrupción mencionada.
Ya dentro del loop principal se da paso a una función (check_prot) que chequea el protocolo del paquete recibido, y en base a esto y a los estados (process_step) de los diferentes procesos se los procesa o descarta. Si un paquete es descartado (ya sea por ser de un protocolo, IP o puerto inválido o porque no se estaba esperando), se retorna de la función sin modificar ninguna de las banderas de flujo y dejando la interrupción respectiva habilitada. Por otra parte, si el paquete es válido, puede requerir procesamiento inmediato y ser este el único procesamiento necesario (p.e un ICMP_request, un PING contra el módulo, necesita una respuesta inmediata) o dejarse almacenado para ser procesado posteriormente. En el primer caso, se procesa dentro de la función y se sale de la misma dejando las interrupciones habilitadas. En el segundo caso, se despierta al proceso dueño del paquete y se sale de la función deshabilitando la interrupción correspondiente, que será habilitada luego que el paquete sea procesado. De esta forma el sistema queda pronto para recibir nuevos paquetes.
La forma en que se chequea el protocolo en la función check_prot puede verse en el diagrama de la figura Figura 8.3

Capítulo 9

Comunicación entre webserver y módulo

El módulo y el webserver realizan sus procesos de comunicación utilizando básicamente el protocolo TCP de capa de transporte, y en otros casos puntuales se utiliza HTTP sobre capa de aplicación. El proceso siempre consiste en la creación de un socket TCP llevando a cabo un *three-way handshake*[20] y luego existen dos posibilidades dependiendo del mensaje a transmitir. Por un lado, para la información que el módulo envía al server se utiliza el método HTTP GET a ciertas páginas. Por el otro lado, para enviar información contra el módulo se ahorran todos los bytes que significan los encabezados utilizados en HTTP y directamente sobre el socket TCP, que además se abre a determinados puertos dependiendo del mensaje a transmitir, se envían ciertas strings prefijadas.
Los procesos de comunicación que se llevan a cabo se resumen en:

- Transmisiones que inicia el webserver
  - Transmisión de las tablas de agenda de eventos
  - Petición de ejecución de un comando PLCBus instantáneo
  - Petición de envío del *log* de eventos
  - Petición de envío del estado de los artefactos (*status*)
- Transmisiones que inicia el módulo
  - Envío de dirección IP del módulo
  - Envío del *log* de eventos, luego de la petición
  - Envío del *status*, luego de la petición

Los sockets TCP que son abiertos por medio del módulo, se generan al puerto 80(HTTP) ya que se utiliza el método GET[20]. Por otro lado, los puertos del módulo utilizados se pueden observar en la tabla 9.1

Las siguientes descripciones detallan los procesos mencionados en la tabla 9.1. Cabe mencionar que las mismas muestran los procesos en el estado de

PUERTO	FUNCIÓN
60030	Envío de una tabla de agenda actualizada
60031	Petición de envio de todas las tablas
60032	Envío de paquetes con IP del módulo
60033	Ejecución instantánea de comando PLCBus
60034	Envío de status
60035	Envío del log

Cuadro 9.1: Puertos utilizados en el módulo

desarollo anterior a la implementación de la encriptación y autenticación en los mismos, por lo que las descripciones no muestran exactamente lo que en realidad ocurre. Sin embargo, este hecho se aprovecha desde el punto de vista de que puede mostrarse los mencionados procesos de una forma clara y entendible.

9.1. Envío de tablas

Las tablas diarias de agendado se transmiten en dos circunstancias diferentes:

- Por iniciativa del webserver, transmisión de alguna tabla actualizada por parte del usuario
- Por iniciativa del módulo, se pide al webserver el envío de todas las tablas, no necesariamente actualizadas. Este último proceso se lleva a cabo cuando el módulo se reinicia.

Para el primer caso, el desarrollo de este proceso se resume en los siguientes pasos:

1. Webserver abre socket TCP/IP persistente (*three-way handshake*)
2. Webserver manda una tabla. En el como carga útil de TCP (TCP payload) se agrega: [”TABLEn”]+[word con largo de la tabla]+[tabla del día n]
3. Módulo contesta: [”OKn”][1]
4. Server cierra socket

Se repiten los pasos 2 y 3 hasta que se hayan enviado las tablas necesarias. En estos pasos puede ser que el paquete de repuesta [”OKn”] hacia el servidor de aplicaciones se pierda y no sea recibido. En este caso, el webserver reenviará hasta 3 veces el paquete con la tabla. Si luego de esto la respuesta no es recibida, se muestra un mensaje en la página web indicando que existe un error de conexión con el módulo.

Para el segundo caso, el desarrollo del proceso es como sigue:

[1] n indica un número del 0 al 6 en formato ASCII para indicar la tabla de qué día se trata

1. Módulo abre socket TCP/IP al puerto 80 del server y envía paquete GET con el siguiente encabezado:
   GET /send_all.php?mod_id=X HTTP/1.1\r\n
   Host: www.radiocostadeoro.com:80\r\n\r\n[2]

2. Se realiza el proceso para algunas tablas descrito anteriormente con la salvedad de que al abrirse un nuevo socket, este será contra el puerto 60031 y no el 60030

3. Luego que se envían las 7 tablas, se responde con un mensaje de éxito HTTP 200 OK, se cierra el socket

El detalle de estos procesos puede observarse en la figura Figura 9.1. Se observa en los primeros paquetes, en color verde, el establecimiento de la conexión llevando a cabo el *three-way handshake* y luego se envía el mensaje de GET. Acto seguido se abre un nuevo socket, pero esta vez contra el puerto 60031 del módulo. Se aprecian los dos paquetes consecutivos con las banderas [PSH, ACK] y sus detalles que muestran que dejan ver claramente la conformación de estos.

9.2. Actualización de IP

Ya que el módulo está concebido para estar instalado en un hogar, el tipo de conexión a Internet más común para este caso es con dirección IP dinámica, es decir que el proveedor de Internet otorga una nueva IP cada cierto período de tiempo a la conexión del hogar. Esto requiere que se haya implementado una solución para que el módulo se mantenga "reportándose" al servidor e indicándole su IP, para que cuando este quiera contactar al módulo sepa a dónde hacerlo.
La forma de reportarse es simplemente cada 250 segundos enviar un mensaje GET a cierta página que capta la IP de destinó de ese mensaje. A saber:

1. Módulo abre socket TCP/IP al puerto 80 del server

2. Se envía paquete GET con el siguiente encabezado:
   GET /get_ip.php?mod_id=X HTTP/1.1\r\n
   Host: www.radiocostadeoro.com:80\r\n\r\n

3. Se responde con un mensaje de éxito HTTP 200 OK y se cierra el socket

9.3. Ejecución de un comando

Cuando el usuario final así lo desee, puede accionar instantáneamente cualquier actuador que poseea instalado. Al ocurrir esto se desarrolla entre webserver y módulo de esta forma:

1. Webserver abre socket TCP/IP persistente (*three-way handshake*)

[2] la variable *mod_id* está vinculada al usuario final del sistema. Cada usuario y módulo correspondiente tiene asociado un único *mod_id*

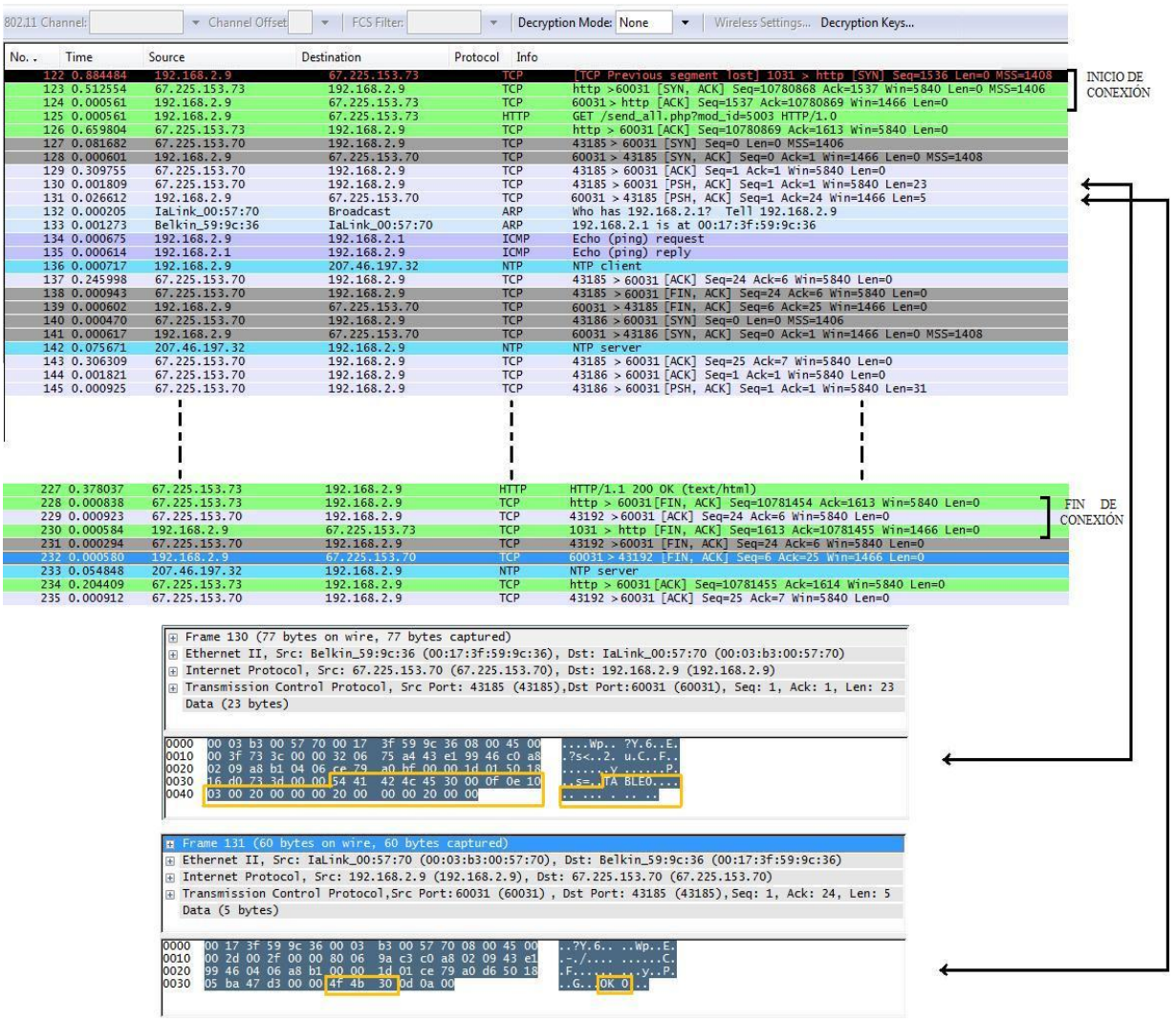


Figura 9.1: Extracto de una captura en el Wireshark mostrando envíos de tablas

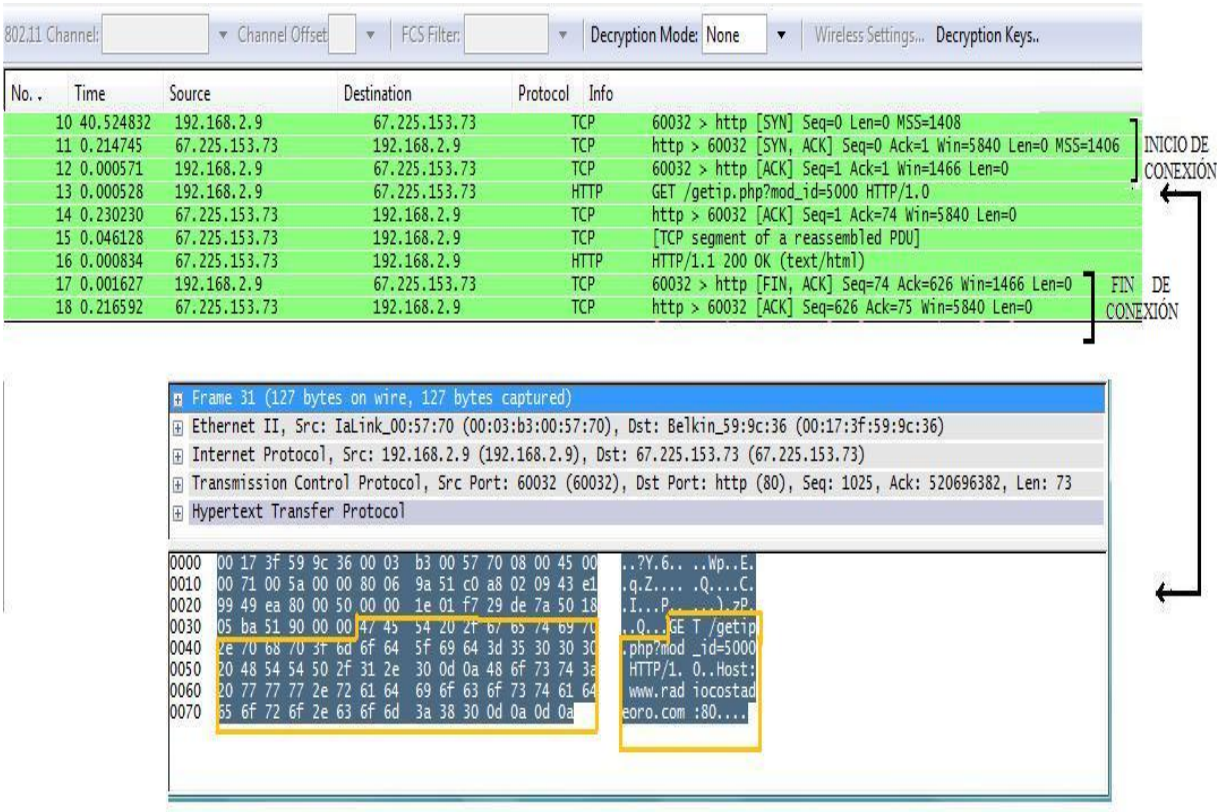


Figura 9.2: Extracto de una captura en el Wireshark mostrando proceso de actualización de IP

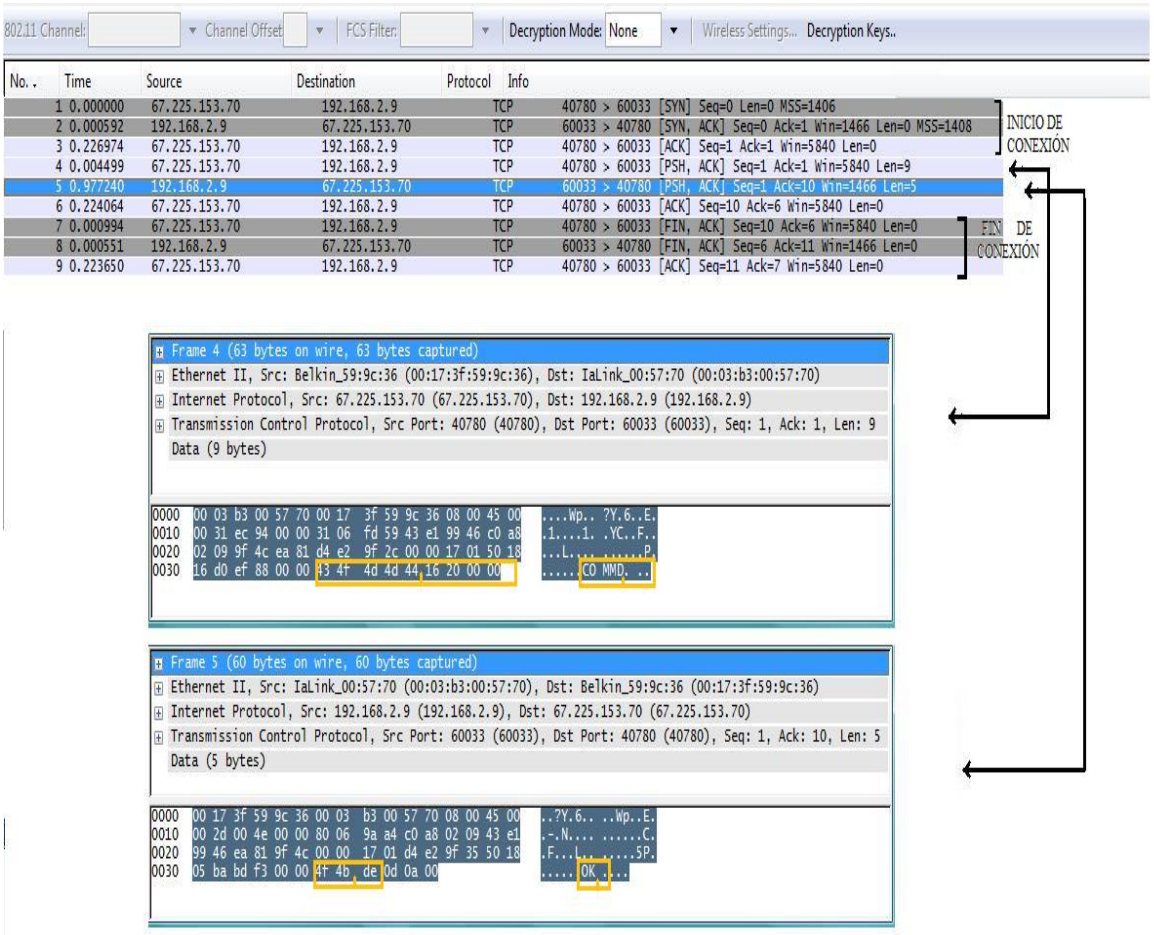


Figura 9.3: Extracto de una captura en el Wireshark mostrando proceso de envío de comando

2. Webserver manda como TCP payload: [”COMMD”]+[4 bytes de comando PLCBus]
3. Módulo contesta: [”OK”]+[byte de respuesta PLCBus]
4. Server cierra socket

Puede ser que el paquete de repuesta hacia el servidor de aplicaciones se pierda y no sea recibido. En este caso, el webserver reenviará hasta 3 veces el paquete con la tabla. Si luego de esto la respuesta no es recibida, se muestra un mensaje en la página web indicando que existe un error de conexión con el módulo.

9.4. Actualización del status

Este intercambio sucede cuando desea saberse el estado de los artefactos instalados en el hogar. Es el único intercambio que se realiza en dos etapas: la

primera consta de la apertura de un socket TCP/IP, el envío de la petición de status y el cerrado del socket. La segunda consta del envío de un paquete GET hacia el servidor incluyendo la información en las variables incluidas.

1. A pedido del usuario, el webserver abre socket TCP/IP persistente.
2. Webserver manda como carga útil de TCP: ["STATS"]+[byte con cantidad de artefactos a encuestar]+[direcciones PLCBus a encuestar]
3. Módulo contesta: ["OK"]
4. Server cierra socket y el módulo encuesta a los artefactos
5. Al recibir la respuesta de los actuadores, el módulo abre socket al puerto 80 y manda un paquete GET el siguiente encabezado:
   GET /status_receive.php?mod_id=X&dir1=B1B2B21B30B31&dir2=B1B2B21B30B31&...&dirz=B1B
   HTTP/1.1\r\n
   Host: www.radiocostadeoro.com:80\r\n\r\n
6. Servidor manda mensaje de respuesta HTTP 200 OK y cierra socket

Los bytes B20 B21 y B30 B31 se envian dependiendo del artefacto e indican nivel de brillo y dimmer rate respectivamente. El objeto de cerrar el socket inicial y luego incluir la información relevante dentro de las variables del paquete GET se debe a que el proceso de encuestar a los artefactos puede llegar a tomar cierto tiempo dependiendo de la cantidad a encuestar.

9.5. Envío de log de eventos agendados

El módulo guarda un registro histórico de los eventos agendados que han sido ejecutados (*log* de eventos). Esta información también se encuentra en el servidor de aplicaciones, aunque no en su versión más actual. Si el usuario así lo desea esta información puede actualizarse en el servidor y el proceso de intercambio de información transcurre así:

1. Webserver abre socket TCP/IP persistente (*three-way handshake*)
2. Webserver manda como TCP payload: ["EVLOG"]
3. Módulo contesta: [128 bytes de respuesta de log]
4. Server cierra socket

Puede ser que el paquete de repuesta hacia el servidor de aplicaciones se pierda y no sea recibido. En este caso, el webserver reenviará hasta 3 veces el paquete con la tabla. Si luego de esto la respuesta no es recibida, se muestra un mensaje en la página web indicando que existe un error de conexión con el módulo.

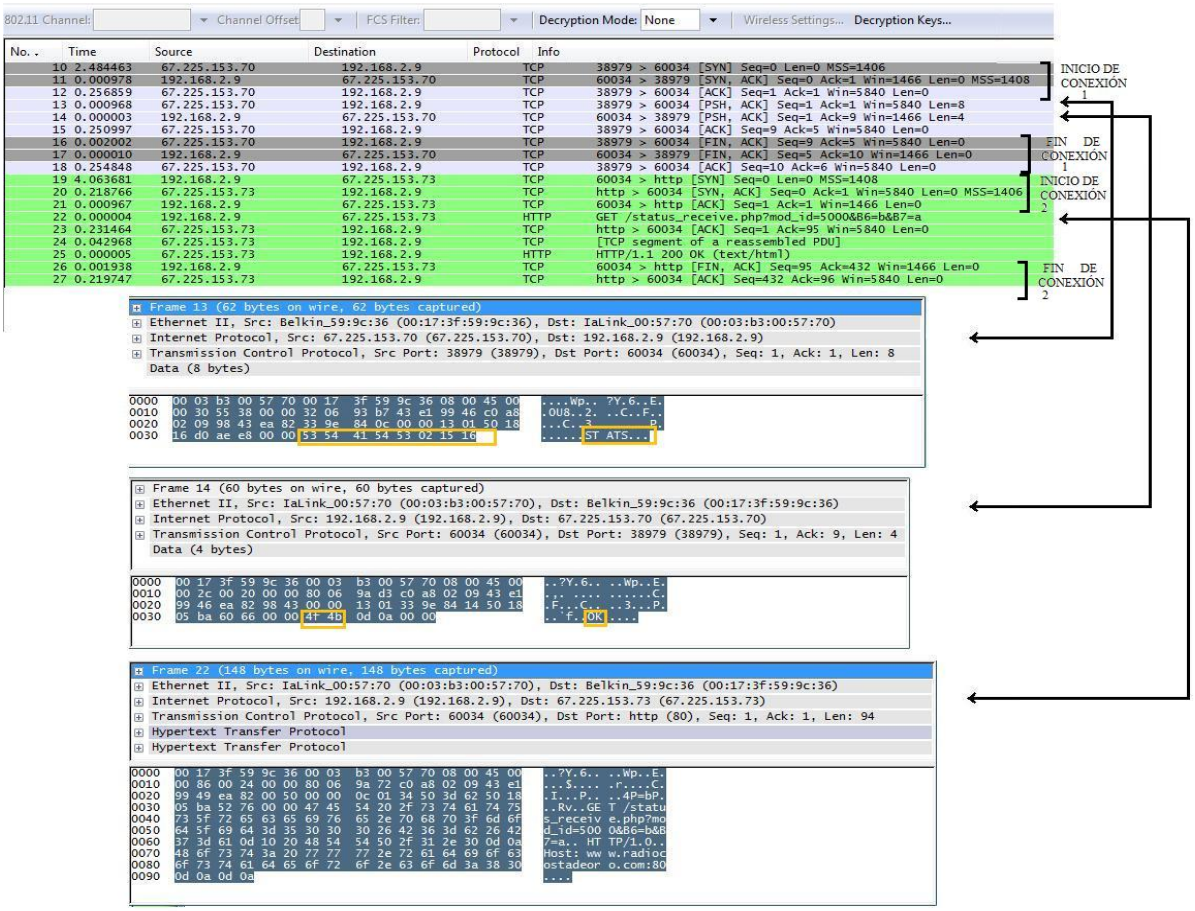


Figura 9.4: Extracto de una captura en el Wireshark mostrando proceso de envío de status

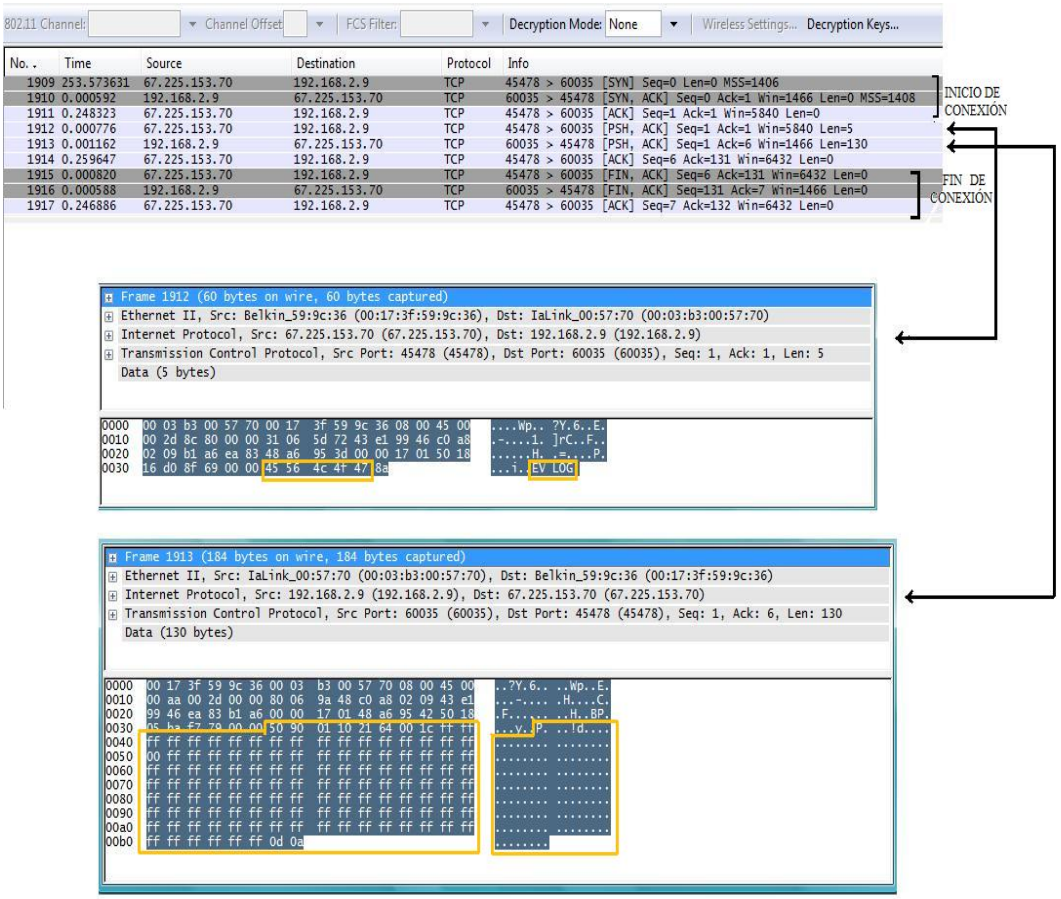


Figura 9.5: Extracto de una captura en el Wireshark mostrando proceso de envío de comando

Capítulo 10

Ejecución y manejo de eventos y comandos

10.1. Introducción

En la siguiente sección del documento se hace un breve recuento del trabajo realizado en lo relativo a la comunicación ejecución y manejo de los eventos y comandos, así como para las tareas necesarias para la implementación de la comunicación entre el módulo PLCBus y nuestro módulo. Se recomienda leer previamente la documentación sobre el protocolo de comunicación con el PLCBus-1141[21], en especial para la comprensión de la subsección que trata sobre las pruebas con el protocolo.

10.1.1. Definiciones previas

Evento

Refiere a un conjunto de comandos, ejecutados de forma "simultánea"[1] en el sentido de que su ejecución es consecutiva, y sin tiempos de espera entre ellos.

Comando

Refiere a una trama conformada de acuerdo a la especificación del protocolo de comunicación de PLCBus, cuya finalidad es provocar una acción de los actuadores.

Respuesta

Refiere a la o las tramas recibidas como consecuencia del envío de un comando, y que contienen información referida a la ejecución del mismo.

[1] Relativo, ya que el mecanismo implementado no permite enviar un comando hasta no recibir la respuesta del anterior.

10.2. Corroboración del protocolo PLCbus

10.2.1. Configuración del dispositivo PLCbus 1141

Por una cuestión de orden, antes de realizar las modificaciones al dispositivo PLCBus-1141 necesarias para adaptarlo a nuestro proyecto, se decidió realizar una verificación "rápida" de los comandos del protocolo PLCBus. En este punto se comenzaron a encontrar problemas vinculados al protocolo y a la empresa PLCBus que fabrica y distribuye los dispositivos. En primer lugar, encontramos que la documentación era escasa y errónea (p.e. confundían byte con bit sistemáticamente). Además, fue imposible conseguir un driver para conectar la PC y el PLCBus-1141 que funcionara. Luego de haber probado todos los drivers que se nos recomendaba usar por parte de los distribuidores y fabricantes del producto sin éxito, se decidió probar un driver VCOM[22][2] genérico. Para la configuración del mismo se requieren los parámetros VID y PID[3] del dispositivo. Estos pueden conocerse utilizando un sniffer de puerto USB o editando los archivos *.inf del driver correspondientes. Encontramos que los VID y PID del PLCBus-1141 obtenidos con el sniffer, no coincidían con los registrados en el driver distribuido por PLCBus lo cual hacía que el mismo no funcionara. Si bien nuestra teoría era que esto se debía a un cambio en el proveedor de los cables adaptadores USB-RS232 (por eso la diferencia de VID y PID), no se podía descartar que se debiera a un error en el mismo por lo que se decidió eliminar el cable USB y hacer la modificación para RS232 sin haber podido verificar el funcionamiento del módulo previamente.
Posteriormente se pudo verificar el funcionamiento del cable haciendo un loopback con el mismo, lo cual hace suponer que el problema se debió efectivamente a un cambio de proveedor.
Una vez solucionado este problema, nos dispusimos a corroborar el protocolo utilizando el software "HomePlanner 2.0"[23] y un sniffer de RS232. Para corroborarlo, se enviaron comandos de encendido/apagado a diferentes direcciones sin importar si las mismas correspondían o no a módulos existentes en la red domótica. Con la información de los comandos recabada por medio del sniffer, complementamos la documentación existente del protocolo pudiendo corregir varios errores y ambigüedades planteadas en el mismo.
En este punto se consideró prudente comenzar una corroboración manual del protocolo utilizando una terminal para puertos COM. Esto era un paso necesario ya que hasta el momento no se había podido verificar si la información acerca del formato de entramado y baudrate con la que se contaba era correcta. De hecho, la especificación del protocolo solo menciona el baudrate y no da ningún detalle acerca del entramado. Asumimos que la misma se realizaba con 8 bits, 1 de parada y sin paridad (8-N-1) dado que esta es la configuración más común en comunicaciones serie asíncronas. Para nuestra sorpresa, el PLCBus-1141 no respondía a ningún comando enviado, a los que sí respondía cuando se enviaban por medio del HomePlanner 2.0. Se probaron entonces diferentes combinaciones de repeticiones de comandos, tiempos entre comandos, chequeo de paridad, etc. pero no se obtuvo resultado alguno. Se probó entonces cambiar de software para descartar que este fuera la raíz del problema, lo que fue exitoso. Luego de probar

[2] Puerto COM virtual, se utilizó USB CDC/ACM Device Driver Demo Version, 2008, Thesycon
[3] Vendor ID y Product ID que identifican al dispositivo USB

varios programas de Terminal diferentes y ver que no todos funcionaba correctamente con el 1141, se concluyó de que existían detalles en la implementación que escapaban a nuestro control los cuales hacían que la comunicación fuera o no exitosa. Afortunadamente, esto no tuvo mayor incidencia en el resto del desarrollo aunque si implicó un retraso considerable respecto a lo planificado.

10.2.2. Pruebas de funcionamiento

Para verificar que la configuración utilizada funcionara correctamente, se planearon las siguientes pruebas:

1. Prueba de envío de comandos corruptos
2. Prueba de saturación del transmisor

Para la primera prueba, se enviaron comandos supuestamente erróneos de acuerdo a lo especificado por el protocolo, de modo de simular errores en la comunicación con el PLCBus-1141 y ver si se producía algún tipo de respuesta que pudiese interferir con el normal funcionamiento del driver (p.e. respuesta de envío satisfactorio de un comando inexistente). Se encontró que, a diferencia de lo que se esperaba, el módulo respondía a comandos erróneos. Sin embargo, para que el módulo responda a los mismos, estos deben ser "deformaciones" de los comandos aceptados por el protocolo y producen el mismo efecto que los comandos correctos.
Para aclarar un poco este punto, tomemos el siguiente ejemplo:
El mensaje constituido por la siguiente trama 02 05 0E 12 22 00 00 03 (encender módulo de dirección B3, con verificación de encendido)[4]. Si en su caso se envía la trama 02 FF 0E 12 22 (largo de trama erróneo, los campos DATA1 y DATA2 perdidos así como ETX) se ejecuta correctamente el encendido del módulo y se devuelve la confirmación.
Esto es preocupante porque demuestra que el protocolo no es respetado por parte de los fabricantes lo cual les quita seriedad. Además, en caso de que una trama se cortara, podría ser interpretada como un comando diferente al que se pretendió enviar:
Si a la trama 02 05 02 12 22 00 00 03 (encender B3 con verificación) se le "pierden" los primeros 2 bytes tenemos resultante el mensaje:
02 12 22 00 00 03 (apagar todas las unidades en la dirección de hogar HOME A).
El potencial error se encuentra si se utiliza 02 en el campo USER_CODE, o A3 en el campo HOME_UNIT. Es necesario entonces tener mucho cuidado con el tiempo que se interrumpe un envío de comandos, lo cual podría incluso ameritar que el envío de los mismos se haga de forma atómica (deshabilitando los pedidos de interrupción en el procesador). Esto no requiere mayores modificaciones al software y no implica afectar significativamente el funcionamiento del microcontrolador ya que el envío propiamente dicho dura apenas 6.7ms cada 400ms[5].
Dado que en cualquier caso, la confirmación contiene la información del comando enviado (lo que el PLCBus-1141 interpretó y envió por la línea) basta con verificar que la confirmación corresponda al comando para asumir que se envió

[4] Los números representan valores hexadecimales
[5] Todavía esta siendo discutido en el grupo si es o no necesario.

correctamente.
La otra prueba planeada fue una prueba de saturación consistente en enviar una gran cantidad de comandos consecutivos y verificar que los mismos fueran ejecutados. Esta prueba se realizó enviando paquetes de hasta 20 comandos espaciados 1 segundo, obteniendo como tasa mínima de éxito el 80 %, lo cual nos permitió comprobar definitivamente que la configuración elegida era la correcta.

10.2.3. Ensayos sobre tiempos de respuesta

Como parte de la verificación del protocolo PLCBus, se realizaron ensayos con el fin de determinar los tiempos involucrados en el envío de los comandos. Para esto, se procedió iniciando un contador al completar la transmisión de un comando y terminarlo al recibir la primer confirmación valida.

variable	tiempo (ms)	delta(ms)
231	288,288	0,2496
233	290,784	-2,2464
232	289,536	-0,9984
228	284,544	3,9936
228	284,544	3,9936
231	288,288	0,2496
234	292,032	-3,4944
232	289,536	-0,9984
231	288,288	0,2496
232	289,536	-0,9984
promedio: 288,5376		

El mismo experimento se repitió para los comandos con verificación los cuales dan 2 mensajes de respuesta al comando: el primero es el interno del 1141 y el segundo es la verificación del módulo (p.e. encendió correctamente o no). En este caso se hicieron medidas para averiguar el tiempo máximo que tarda en llegar el segundo comando, utilizando la misma metodología.

variable	tiempo (ms)	delta(ms)
251	313,248	-4,368
245	305,76	3,12
250	312	-3,12
250	312	-3,12
249	310,752	-1,872
246	307,008	1,872
245	305,76	3,12
247	308,256	0,624
248	309,504	-0,624
244	304,512	4,368
promedio: 308,88		

De los resultados anteriores se desprende que en realidad, la cota de 400ms que dice la documentación del protocolo esta sobredimensionada y que se puede mejorar la performance del sistema si se utiliza la cota hallada experimentalmente.

10.3. Descomposición modular

El análisis permitió determinar 4 módulos bien diferenciados de acuerdo a su responsabilidad y alcance:

1. Scheduler
2. Reloj
3. UART
4. Motor de PLCBus

El **módulo Scheduler** es un motor de tareas de varios pasos, que se encarga del control de los eventos agendados a ejecutar, así como de tareas accesorias necesarias.[6]
El **módulo Reloj**, se encarga de llevar la hora del sistema a partir de la interrupción de un Timer y de avisar al Scheduler si es necesario que actúe[7].
El **módulo Motor de PLCBus**, se encarga del control general de la ejecución de comandos de los tipos de eventos descriptos anteriormente, recibiendo para ello peticiones (de TCP en caso de comandos instantáneos y status, y Scheduler en caso de agendados).
Garantiza que se ejecuten correctamente de acuerdo a las prioridades e implementa retrasmisiones en caso de ser necesario. Además, se encarga del procesamiento de las respuestas, generando logs, o enviando las respuestas al Server.
El **módulo de UART** es el encargado de manejar la comunicación serie con el PLCBus 1141, implementando el envío y recepción, verificación de la conformación de las tramas y filtrado de tramas que no correspondan a la comunicación en curso.
La siguiente figura muestra un diagrama básico de la interacción que se da entre los diferentes módulos para lograr la correcta ejecución de eventos y comandos[8].

10.4. Módulo UART

10.4.1. Envío y recepción

Este módulo implementa el enlace con el módulo PLCBus 1141, intercambiando tramas con el mismo de acuerdo al formato especificado por el protocolo de comunicación con PLCBus.
Para implementar la comunicación con el módulo PLCBus-1141, se utiliza la UART del microprocesador configurada a una velocidad de 9600bps de acuerdo a lo especificado en el protocolo.
El formato de la trama no se encuentra especificado en el protocolo pero se determino que era la configuración típica de 8 bits, 1 bit de parada y sin chequeo de paridad.
El módulo esta compuesto por un trasmisor, implementado utilizando la ISR de UDRE[9] desencadenada al vaciarse el registro de salida, luego de finalizar el

[6] Ej: re sincronizar la hora, buscar el próximo evento a ejecutar y comunicar a PLCBus la necesidad de ejecutar un evento.
[7] Ej: es hora de ejecutar un evento, es hora de re sincronizarse
[8] Ver Figura 10.1
[9] Uart Data Register Empty, indica que el registro de salida está vacío.

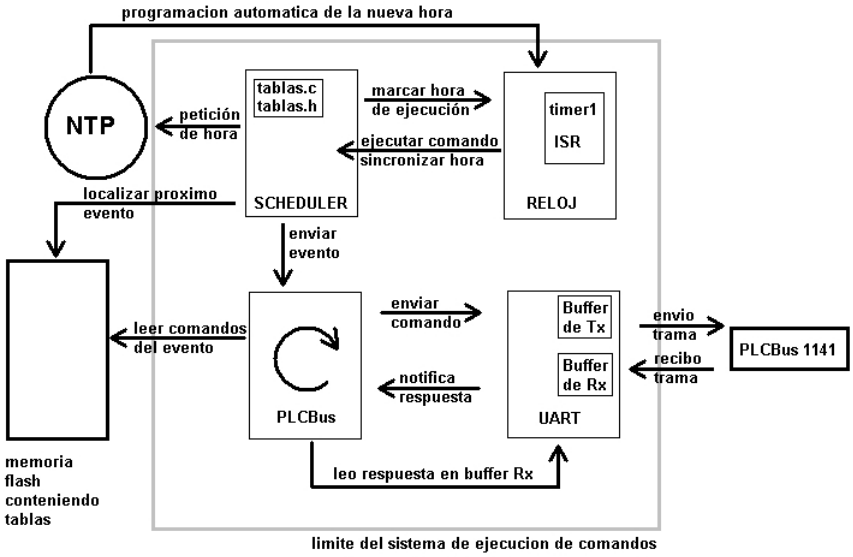


Figura 10.1: Interacción entre los módulos que componen el software

envío de un byte, y por un receptor implementado utilizando la ISR de RX_C la cual se ejecuta al terminar de recibir correctamente un byte.
El envío de una trama se realiza escribiendo la misma en un buffer de salida y colocando en el registro de salida el primer byte. A continuación, la ISR de UDRE incrementa el índice del array correspondiente al comando el cual es un índice de 3 bits que cuenta circularmente de 0 a 7 y coloca en el registro de salida el byte siguiente.
Para la recepción de las respuestas, los bytes recibidos se almacenan en un buffer de entrada a partir de que se detecta el comienzo de la trama (STX).
Se verifica byte a byte la correcta conformación de la trama recibida (STX, length, dirección, ETX) y en caso de no superar los filtros, se descarta y se continúa a la espera de la respuesta.
El sistema funciona en modalidad half-duplex, lo que significa que cuando se trasmite no se escucha, y viceversa, de forma de organizar la comunicación en un régimen de pregunta-respuesta, un criterio de diseño adoptado desde el principio con la finalidad de simplificar el desarrollo.

10.4.2. Detección de errores

Una de las razones por las cuales fue elegido el protocolo PLCBus, fue por su menor tasa de error en comparación con otros protocolos sobre líneas eléctricas. El sistema pretende ser autónomo, y funcionar sin un usuario verificando que se hayan ejecutado los comandos, por lo que es crítico poder detectar y manejar errores por menos probables que sean.
Durante el estudio y verificación del protocolo, se observo que todas las respuestas se identifican con el mismo código de usuario y dirección del módulo a quien

se envió el comando, salvo comandos enviados a varios módulos[10] que pueden no tener la misma dirección.
A su vez, durante las pruebas de verificación del protocolo, se detectaron algunos errores frecuentes:

1. Largo de la respuesta diferente a lo establecido por el protocolo
2. Respuestas con código de usuario invalido
3. Respuestas con dirección inválida
4. Trama mal conformada

Por esto, se implementa a nivel de la recepción un filtrado de las tramas teniendo en cuenta el código de usuario y la dirección, además del cumplimiento del formato de trama.
Dado que existen excepciones, se implementa un mecanismo que permite desactivar el filtrado por direcciones, por medio del control de una bandera dedicada a este fin.
Como los parámetros necesarios para validar las respuestas dependen del comando enviado, se optó por un esquema de trasmisiones del tipo half-duplex, lo que facilita la identificación de la relación entre comandos y sus respuestas.
Para realizar la verificación de la conformación de las respuestas, se utiliza un índice para la recorrida del array de entrada. Este índice se pone en 0 luego de que se trasmite, lo que indica que se está a la espera del comienzo de una trama, y continúa valiendo 0 mientras no se reciba el STX (02h) de forma de que, si por alguna razón se recibiera una trama empezada, la misma no sería reconocida.
Al recibirse el STX, se auto incrementa el índice y los datos sucesivos son almacenados de forma consecutiva en el array de entrada, hasta recibir el noveno byte, correspondiente a ETX (03h), lo que marca la finalización de la recepción.
Al recibirse el último byte de la trama, se llama a una función que verifica si la trama representa o no la finalización de la comunicación[11].
Si por algún motivo se debe descartar una trama, se marca el penúltimo byte de la trama en el buffer (correspondiente a RX_TX_SWITCH) con valor 0, lo que es interpretado por nuestro motor como la ausencia de respuesta. Además, se reinicia el índice a 0, habilitando la recepción de una nueva trama de respuesta.

10.5. Reloj (*time.c y time.h*)

Este módulo implementa un reloj de tiempo real que segundo a segundo determina si es necesario realizar alguna acción, en función de la hora interna y la hora de los eventos programados. Se implementa por medio de una rutina de atención a las interrupciones generadas por el Timer1[12], el único de los timers capaz de generar interrupciones cada un segundo a partir del reloj interno[13].
La rutina incrementa la variable interna que se encarga de llevar la hora[14], y

[10] Un ejemplo es el comando ALL LIGTHS OFF, al cual se responde desde una dirección fija.
[11] Ver sección 10.7 *Motor de PLCbus*
[12] timer/counter de 16 bits
[13] Contando hasta 15625 a frecuencia f/1024 =15625*1024/16Mhz = 1s
[14] Llevada en forma de segundos a partir de las 0 horas AM o PM

al llegar a las 12hs (43200s) modifica la bandera que indica si la hora es AM o PM. Además, en caso de llegar a las 12hs PM cambia el día actual. Cuando esto sucede, se llama a una función auxiliar la cual busca el primer evento agendado para el día corriente y lo configura como el próximo evento a ejecutar.
En cada instancia de la interrupción, se compara el valor de la hora del sistema con la del próximo evento y con horas prefijadas para la re sincronización. En caso de coincidir con alguna de estas notifica al scheduler de la necesidad de ejecutar una acción, y es este quien se encarga de las tareas necesarias.
Como es imposible ajustar la frecuencia exactamente a 16Mhz, puede existir una pequeña deriva en la hora interna, que acumulada podría significar un desfasaje considerable. Por esto se implementa un mecanismo que ejecuta la re sincronización periódica del reloj interno.
Esta re sincronización se realiza al llegar una hora determinada (MOD_ID segundos). MOD_ID es un número identificador del sistema, en el rango 200-43000 que permite diferenciarlo de los demás sistemas. El objetivo de realizar la sincronización a esta hora es evitar que todos los módulos pidan la hora al mismo tiempo, lo cual produciría una sobrecarga del servidor de fecha y hora[15], e incluso podría significar la negación del servicio por confundirse con un ataque al servidor.
Como se desconoce a priori que tan grave puede llegar a ser este desfasaje al momento de la re sincronización, es necesario tomar algunas precauciones para evitar un funcionamiento anómalo. Se contempla entonces la posibilidad de que el reloj interno adelante en relación a la hora real, lo cual implicaría un salto hacia atrás en el tiempo al llegar la hora de re sincronizar. Esto puede producir que se desencadene una nueva sincronización por volver a transitarse la hora de la misma. Esto, si bien no es crítico para el sistema, puede ser fácilmente evitable y ya que fue detectado en una etapa temprana de desarrollo, se decidió implementar una bandera que desactive la sincronización durante 40 segundos luego de la primera (tiempo más que prudencial) como forma de evitar errores.
En algunos casos es necesario mantener en funcionamiento el reloj pero evitar la ejecución de tareas programadas por lo que se implementa una bandera (FC.SCHEDULER.ON) que habilita o deshabilita la ejecución de eventos agendados, sin afectar al resto del sistema.

10.6. Scheduler (*scheduler.c y scheduler.h*)

Este módulo se encarga de llevar a cabo las secuencias necesarias para la ejecución de comandos agendados. Actúa como intermediario entre el reloj, quien notifica la necesidad de ejecutar eventos, y el motor de PLCBus, quien los ejecuta. Además, se encarga de realizar tareas accesorias, como ser la actualización de la hora.
La estructura de este módulo es similar al de los demás módulos de tareas, es decir, una función _step() llamada desde el loop principal dentro del round-robin, y funciones para la inicialización, comienzo y detención del mismo.
Se implementan tres tareas:

INICIAL

[15] En caso de que se transforme en un producto comercial de distribución masiva

Esta tarea es desencadenada al recibir tablas actualizadas provenientes del servidor. Se encarga de pedir inicialmente la fecha y hora utilizando para esto el módulo de SNTP. Esto es necesario ya que la actualización de tablas puede requerir un tiempo considerable de uso exclusivo del procesador, produciendo un desfasaje entre el reloj interno y la hora real. Al obtener respuesta, el módulo SNTP se encarga de configurar la hora e iniciar el módulo de reloj. La tarea se encarga también de buscar en las tablas el próximo evento a ejecutar y configurarlo para su ejecución, utilizando para esto funciones auxiliares[16].

EJECUTAR_CMD

Esta tarea es desencadenada por el módulo reloj[17] al llegar la hora de ejecución del evento agendado.
Transfiere el evento a ejecutar (por medio de un puntero a la ubicación en memoria flash) al motor PLCBus para que se encargue de su ejecución, y luego busca y agenda el próximo evento a ser ejecutado.

RESYNC

Esta tarea es desencadenada por el módulo reloj al llegar la hora de la re sincronización. Se encarga de actualizar fecha y hora[18], corrigiendo así posibles desfasajes.
Este desfasaje puede ser un retraso del reloj interno respecto a la hora real, por lo que al re sincronizarse podría producirse un salto hacia delante en el tiempo, y esto a su vez producir que un evento agendado no sea ejecutado. Basados en la teoría de que es mejor tarde que nunca, al terminar de obtener la nueva fecha y hora, la rutina verifica que el evento apuntado no se haya vencido sin ser ejecutado, y en caso de ser así, procede con su ejecución.

10.7. Motor de PLCbus

El motor de PLCBus es el encargado de controlar y ordenar la ejecución de comandos, así como de verificar que estos sean ejecutados correctamente, retrasmitir en caso de ser necesario, y efectuar las tareas de post proceso de los mismos.
El funcionamiento del motor de PLCBus se basa en un núcleo que implementa el motor de tareas, el cual se encarga del flujo de este módulo motor y de la toma de decisiones. El proceso se descompone en varias etapas, que incluyen el envío de los comandos, la verificación de las respuestas, el reenvío de los comandos y el post proceso[19].
Al igual que los demás motores de tareas implementados, el núcleo se implementa por medio de una función_step() llamada desde el loop principal en caso de encontrarse en ejecución la tarea. Al ejecutarse esta rutina se chequea si existe alguna tarea de post proceso pendiente y se ejecuta. Si no es así, se envía

[16] Estas funciones se encuentran implementadas en tablas.c.
[17] ISR de timer1 en time.c
[18] Se realiza una petición a SNTP, quien consulta al servidor NTP sobre la hora
[19] Entendido como las tares posteriores, consistentes en reconfigurar el sistema y prepararse para la próxima ejecución, la generación de registros y envío de información al servidor.

uno de los comandos de los eventos pendientes respetando las prioridades. Al realizarse el envío, el motor se coloca en un estado busy, del cual sale al recibir una respuesta al comando.

10.7.1. Tipos de eventos

En esta etapa, se extendió el análisis de requerimientos para el envío y recepción de comandos identificándose 3 tipos de eventos diferenciados entre si por características como el numero de comandos que los conforman, la criticidad de su ejecución y el procesamiento necesario para las respuestas obtenidas.
Estos son:

1. Evento Agendado
2. Comando Instantáneo
3. Encuesta de estado de los actuadores (status)

1. **Evento Agendado**

   Corresponde al caso de un evento almacenado en memoria flash, el cual es ejecutado al llegar una hora determinada y que consiste de uno o más comandos. No es necesario comunicar la ejecución de los mismos al servidor, ni el resultado de dicha ejecución, ya que se pretende que el sistema sea autónomo. Sí es deseable generar un registro de dicha ejecución y de los resultados, que permita realizar algún tipo de diagnostico del sistema para lo que se implementa un registro del resultado de la ejecución de los comandos.

2. **Comando Instantáneo**

   Corresponde al caso de un comando enviado por el usuario a través del servidor, el cual requiere ser ejecutado de inmediato. Es necesario comunicar la respuesta al servidor, el cual la informara al usuario, lo que implica que los tiempos involucrados son críticos. Por esto, este tipo de ejecución es el de mayor prioridad para el sistema.

3. **Encuesta de estado de los actuadores**

   Corresponde al caso de un grupo de comandos de status, ejecutados con el objetivo de conocer el estado actual del sistema. Este pedido es ejecutado online por el usuario, y es deseable que la respuesta sea instantánea. Esto último depende en gran medida de la cantidad de módulos a encuestar, por lo que es necesario prever mecanismos para evitar que el sistema[20] se bloquee. Por esto, la implementación se realiza liberando la conexión con el servidor luego de aceptadas las direcciones de los módulos a encuestar, y enviando al servidor el resumen de status luego de culminada la tarea de encuestar a todas las direcciones de la lista.

[20] Principalmente, para no retener al usuario en espera de una respuesta, y no dejar un socket abierto innecesariamente.

10.7.2. Manejos de prioridades para eventos

Existe la posibilidad de que se solapen la ejecución de diferentes tipos de comandos por lo que es necesario fijar algún tipo de orden de prioridades para los mismos. Este orden de prioridades, esta dado por la criticidad de los tiempos de respuesta al servidor, permitiendo ordenarlos de más a menos:

- Comando Instantáneo
- Evento Agendado
- Encuesta de estado de los actuadores[21]

En el caso de los dos últimos, el orden se justifica en que, de nada sirve conocer el estado de un módulo, si sabemos que inmediatamente se ejecutará un comando que probablemente lo altere.
En este punto, quedan algunos detalles a ser discutidos con mayor profundidad en una etapa posterior, sobre todo la posibilidad de cambiar estas prioridades, y de cambiar el tratamiento que se le debe dar a la superposición de comandos. Con esto ultimo, nos referimos a que, actualmente los eventos se pausan para dar lugar a otro de mayor prioridad, y luego de concluido, la ejecución continua desde el mismo punto. Con el mismo argumento planteado para la justificación de la prioridad entre estatus y evento agendado, podría ser interesante que, al retornar de la ejecución de un comando, se reiniciara la petición de estatus de modo de contar siempre con la información más actualizada.

10.7.3. Envío de comandos

El sistema contempla varios tipos de comandos, los cuales se deben procesar en forma diferente, por lo que se implementó una interfaz de envío especial para cada uno de ellos. Estas diferencias radican básicamente en la forma en que se almacenan internamente los eventos, y por no ser demasiado profundas se planea estandarizar el almacenamiento y unificar las funciones en una etapa futura.
Para el envío de los comandos, se rellena el buffer de salida de la UART a partir de los datos almacenados en memoria y se coloca en el registro de salida (UDR) el primero. Luego, la UART se encarga de enviarlos y filtrar las respuestas, y de llamar a la función encargada de procesarlas.
Como el sistema está diseñado para trabajar bajo un único USER CODE, la rutina de envío de comandos se encarga de rellenarlo, así como de rellenar los campos STX, ETX y length cuyos valores son fijos[22]. De esta forma, sólo se deben almacenar 4 bytes para cada comando en vez de 7 lo que representa un enorme ahorro de memoria.

10.7.4. Transmisiones exitosas y reintentos

Existe la posibilidad de que los comandos no lleguen a destinó, o de que se pierdan las respuestas a los mismos. Por esto es necesario implementar el reenvío de los mismos, de manera ágil y evitando tener tiempos de espera excesivos, pero sin realizar retrasmisiones innecesarias.
Para esto se implementó un contador de timeout[23], el que se encarga de solic-

[21] Referido también como pedido de Status
[22] 02h,03h,05h respectivamente
[23] de acuerdo a la especificación del protocolo son 0,4 segundos para ejecutar un comando

itar retrasmisiones al núcleo del motor PLCBus en caso de vencerse sin haber obtenido una respuesta externa.
Se observó que este tiempo especificado por el protocolo es bastante holgado[24], por lo que se decidió implementar además, un mecanismo que permita, en caso de recibir la información esperada antes de este tiempo, seguir con la ejecución del comando siguiente.
Se relevaron las respuestas para los diferentes tipos de comandos involucrados, con el fin de separarlas en diferentes grupos de acuerdo a las características de las mismas. Se identificaron 3 tipos de "terminaciones" dadas por el byte de FLAGS definido en el protocolo PLCBus (RX_TX_SWITCH). Estas terminaciones, pueden ser interpretadas:

- La trama es una respuesta externa al PLCBus 1141 (0X0C)
- La trama es un ACK proveniente de un actuador (0x20)
- La trama verifica que PLCBus 1141 coloco correctamente una señal en la línea (0x1C)

Es importante mencionar que pueden encontrarse más de una de las siguientes terminaciones presentes en la misma trama de respuesta, por ejemplo, podría ocurrir que llegara una respuesta con terminación 0x3C, lo que debe ser interpretado como 0x1C+0x20, o sea, que se colocó correctamente la señal en la línea, y que además el actuador verifica la ejecución del comando. Además, cabe destacar que no todos los comandos están contemplados en estos grupos, por lo cual no pueden tomarse como una condición obligatoria para el éxito del comando, sino una condición suficiente, y por esto no se verifican en la rutina de terminación por timeout.
Existe una suerte de jerarquía en cuanto a la información aportada por las tramas, ya que, si por ejemplo se recibe un ACK externo, queda implícito que se envió correctamente la trama, y que la misma fue entendida por el actuador involucrado.
Lo anterior se puede traducir rápidamente en condiciones de terminación para las secuencias de nuestros tipos de comandos de la siguiente forma:
STATUS: Sólo requiere la recepción de información externa, por lo cual la condición de finalización debe ser 0x0C.
COMANDO (instantáneo o agendado): Este caso es muy genérico, ya que en principio se podría solicitar ejecutar cualquiera de los comandos especificados en el protocolo. Sin embargo, la idea del mismo está concebida como la ejecución de comandos que produzcan un resultado físico en los actuadores (ej: encender, apagar, ajustar intensidad, etc), lo que reduce las posibles condiciones de finalización a 0x1C en caso de no requerir verificación externa, y 0x20 en caso de requerirla.
Se implementa entonces una rutina de control de finalización la cual es llamada por la UART al terminar de recibir una trama válida, que compara la información recibida con la condición de finalización del tipo de comando, y en caso de concordar, se encarga de cancelar el contador de timeout y de notificar la finalización para que se realice el post proceso.

[24] generalmente no tarda más de 0,3 segundos, Ver subsección 10.2.3 *Ensayos sobre tiempos de respuesta*

10.7.5. Post proceso

Al concluir una transmisión exitosa, se procesa la información útil generada a partir de la misma. El tratamiento que se le da a la información depende del tipo de evento al que pertenece el comando ejecutado por lo cual, antes de realizar el envío de cada comando, se registra a que tipo de evento pertenece.

1. **Evento Agendado**

   Dado que este tipo de eventos se caracteriza por su ejecución desatendida, no es necesario informar al servidor sobre los resultados de la ejecución. Sin embargo, se implementa un registro de los resultados de la ejecución de los eventos, cuya principal utilidad es brindar información acumulada para evaluar el correcto funcionamiento, y poder detectar la ocurrencia de errores.
   Este registro, permite almacenar la respuesta a los últimos 16 comandos ejecutados, un número que es sin duda muy acotado, por lo cual se planea ampliar la capacidad en una etapa posterior.[25] Como mencionamos anteriormente, las respuestas recibidas pueden constar de varias tramas, sin embargo, la información más útil se encuentra en la última trama recibida, por lo cual se almacena únicamente esta trama. Para cada evento se almacena en memoria flash la hora de ejecución y los bytes de la última trama de respuesta recibida, sin incluir los correspondientes a STX, length y ETX.

2. **Comando Instantáneo**

   Se comunica la respuesta obtenida al servidor, enviándole los datos de la misma, el cual se encargara de interpretarla y desplegar el resultado. Teniendo en cuenta lo dicho anteriormente, se envía únicamente la última trama recibida.

3. **Encuesta de estado de los actuadores**

   Este tipo de evento es solicitado online, por lo cual el resultado debe enviarse al servidor. Sin embargo, como ya mencionamos el mismo puede tardar mucho tiempo por lo cual el procesamiento se realiza desconectado del servidor. Luego de recibida la respuesta a cada comando se almacena la respuesta de manera temporal hasta concluir la lista de dispositivos a encuestar. Una vez concluida la lista se envía el resultado al servidor.
   A diferencia del registro de comandos ejecutados, no se necesita almacenar la hora de ejecución, ya que se supone la misma es instantánea, por lo que se almacena únicamente los bytes de dirección, estado del actuador[26] y dos bytes de datos[27].

[25] Inicialmente se implementa únicamente el registro de 16 comandos ya que esto corresponde con una página de memoria flash, lo que simplifica el manejo de punteros.
[26] Generalmente ON, OFF o dispositivo desconectado
[27] En cado de dispositivos del tipo dimmer, corresponden a Light brightness level y dimmer fade rate

10.7.6. Virtualización de la memoria

A lo largo del proyecto, una condicionante para el diseño fue la necesidad de ahorrar memoria RAM dado que se cuenta únicamente con 2Kbytes, y que además se pretende que el sistema evolucione en una etapa posterior lo que implicaría un mayor consumo de memoria.
Debido a que se manejan varios tipos de evento simultáneamente, dos de los cuales (comandos y estatus) implican la necesidad de almacenar gran cantidad de información temporalmente, necesaria tanto para la ejecución como para el post proceso, se decidió implementar una suerte de virtualización para reducir el consumo de RAM.
Para esto, se utilizan páginas de memoria flash para almacenar variables relacionadas a estos tipos de eventos, las cuales no se encuentran en uso temporalmente. Es importante recordar que esto se debe a que el sistema fue diseñado de forma de ejecutar los eventos de forma ordenada, y respetando prioridades.
Se concibe entonces el concepto de contexto de memoria, como el conjunto de variables en memoria RAM, cuyo uso se encuentra relacionado exclusivamente a estos dos tipos de eventos. Cada contexto esta conformado por un arreglo de 128 bytes[28] cuyos bytes representan a las diferentes variables utilizadas. Se utiliza además una bandera que indica cual es el contexto de memoria activo, de forma de que, al realizar alguna operación que requiera un contexto de memoria específico, se compara esta bandera y de ser necesario se realiza un cambio de contexto. Este cambio de contexto se realiza guardando el contexto actual en una página de flash, y cargando desde una página de flash el nuevo contexto.
En la versión actual de firmware, este procedimiento permite únicamente el ahorro de 128 bytes de RAM, pero brinda una herramienta muy útil para versiones futuras, en las que se plantea la necesidad de incrementar el consumo de memoria, al aumentar la capacidad de los registros, y la cantidad de actuadores a encuestar.

[28] El equivalente en tamaño a una pagina de flash.

Capítulo 11

Seguridad

En el siguiente capítulo se analizaran los detalles relacionados a la implementación de medidas de seguridad. Dichas medidas giran en torno a dos necesidades fundamentales, asegurar la autenticidad de la información intercambiada, e impedir que terceros puedan acceder a dicha información.
Se exploraron las implementaciones de algunos de los algoritmos utilizados en el mercado, relevando tiempos de ejecución, consumos de memoria, etc.
Se pudo concluir que estos algoritmos, si bien podían ser implementados en el microprocesador actual, requerían tiempos de procesamiento que, en relación a los tiempos de las demás tareas, eran considerablemente mayores.
Vale aclarar que, dado que al momento de comenzar el análisis de los algoritmos de seguridad no estaba completamente definido el sistema, se decidió optar por algoritmos propios, que fueran más rápidos e implicaran un menor consumo de memoria.
En la actualidad el sistema no se encuentra estandarizado, y coexisten diferentes métodos de intercambio de información, se planea estandarizar esto y utilizar únicamente el intercambio de información por intermedio de sockets, lo que permitiría extender las estrategias de seguridad a todo el sistema, y no parcialmente como se realiza actualmente.
Por otra parte, a pesar de todos los esfuerzos que se hagan para proteger y autenticar los datos trasmitidos por el módulo, nuestro sistema siempre tendrá como punto débil la obtención de la fecha y hora del servidor NTP, el cual esta basado en un estándar absolutamente inseguro y lo que lo hace susceptible a ataques, los cuales pueden estar orientados fundamentalmente a dejar el módulo fuera de servicio[1].
Por estas razones se consideró que lo más conveniente era implementar algoritmos propios no tan seguros en una primera etapa, verificar el funcionamiento del sistema completo y finalmente, en una etapa posterior, ajustar los detalles relativos a los intercambios, sustituir los algoritmos de seguridad por algoritmos estándar, dejar de lado el protocolo NTP para obtener la hora e implementar esta función consultando directamente al servidor.

[1] Se puede engañar al módulo suplantando al servidor de NTP, y enviando una hora diferente a la real, producir que no se ejecuten comandos.

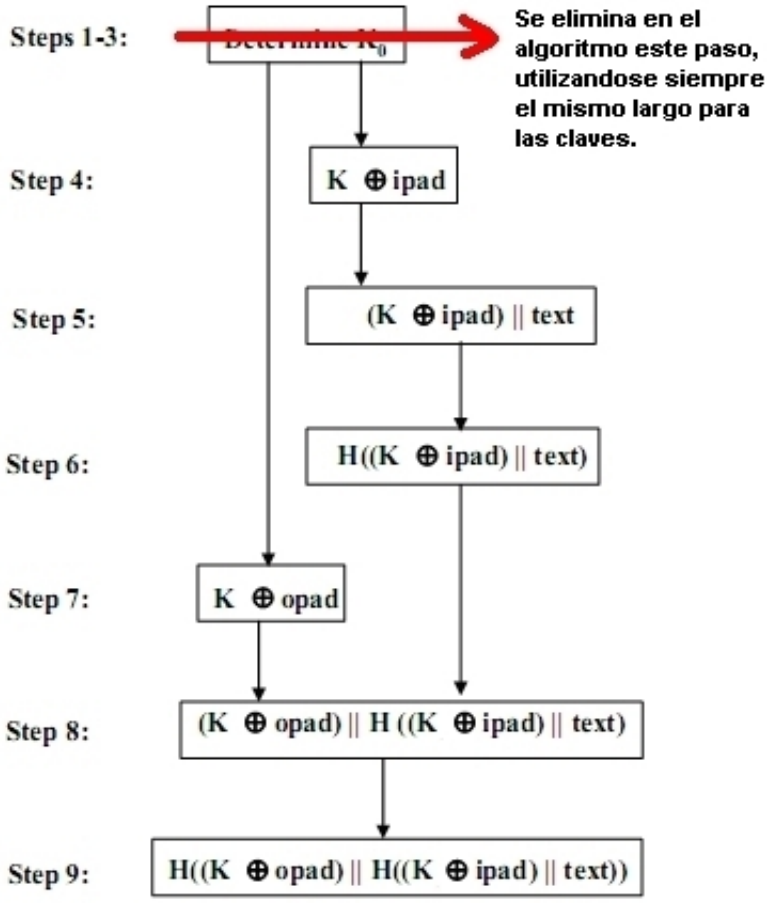
11.1. Autenticación

La autenticación de paquetes tiene como objetivo asegurar que la información recibida haya sido enviada por quien debiera, y no por un tercero. Para esto, una opción es firmar los paquetes, agregando al final de los mismos, información que solo el emisario podría generar. Estas firmas son generadas a partir de claves secretas, las cuales son conocidas únicamente por el emisor y el receptor, de modo que, si alguien intentara introducir información sin autorización, el receptor lo sabría y podría descartar el paquete. Se estudió el algoritmo de HMAC[24], el cual basa su funcionamiento en la utilización de una función de hash para generar un resumen a partir del mensaje y las claves. Las funciones de hash son algoritmos de encriptación de una vía, las que a partir de un bloque de información, generan un resumen (digest) de menor tamaño. Esto hace que sea imposible reconstruir la información a partir de un digest; además, es imposible generar el digest conociendo parcialmente la información. Por esto, utilizando como punto de partida la información que se desea enviar y una clave, la firma generada garantiza la autenticidad de la información. El nivel de seguridad estará dado entre otros por el largo de la firma, ya que, si bien la misma no puede ser calculada sin conocer las claves, puede ser acertada probando con firmas generadas al azar, por lo que una firma larga disminuye esta probabilidad. Como mencionamos, la firma generada a partir del mensaje y la clave, es enviada junto con la carga útil del paquete. Del otro lado, el receptor del mensaje utiliza la carga útil del paquete recibido y la clave del emisor (la cual debe conocer) para generar nuevamente la firma y compararla con la recibida. Esta forma de autenticación permite autenticar la carga útil paquete a paquete, lo cual brinda un nivel de seguridad aceptable. Como se puede ver en [24], documento correspondiente a la especificación del estándar HMAC, el nivel de seguridad del algoritmo esta directamente vinculado a la función de hash que el algoritmo utiliza para generar el digest. El estándar especifica que se debe utilizar como función de hash una de las aprobadas por el FIPS, o sea una de las de la familia de SHA[2]. Estas sin embargo son demasiado complejas, e implementadas en el ATmega32 conllevan un alto costo en recursos y tiempos de cómputo. A modo de ejemplo, el tiempo necesario para autenticar un bloque de 128 bytes (una pagina de flash) utilizando SHA es de unos 18 ms mientras que con una función no estándar es de menos de 1 ms. Es por esto que se decidió utilizar (al menos en un principio) una función de hash sencilla y rápida y que consumiera pocos recursos, de forma de poder implementar un nivel de seguridad mínimo y luego, en una etapa posterior mejorarlo. Además, dado que se pretendía elaborar un algoritmo más sencillo, se eliminaron algunos pasos especificados en el estándar, los cuales tienen como finalidad estandarizar el tamaño de la clave a ser utilizada, y que implican rellenar o acortar la misma en función de su largo inicial (ver Figura 11.1).

11.1.1. Elección del algoritmo de hash

Dada la carencia de formación del grupo en este tema, la elección del algoritmo de hash debió basarse en estudios y recomendaciones de terceros que tuvieran mayor conocimiento.

[2]SHA-1, SHA-224, SHA-256, SHA-384, y SHA-512 especificados en [25]



H:algoritmo de hash elegido
K:clave secreta compartida
ipad:inner pad, byte 0x36 repetido el largo de la clave
opad:outer pad, byte 0x5C repetido el largo de la clave
Text:los datos sobre los que se calcula HMAC
||:concatenacion de datos
^:operacion logica exor

Figura 11.1: Algoritmo de autenticación

En primer lugar, se recopilaron diferentes algoritmos implementados ya fuera directamente en lenguaje C, otros lenguajes, o escritos en forma de seudo código, de manera de poder llegar a una implementación en lenguaje C sin tener que realizar un estudio demasiado detallado de los mismos.
Se recabó información acerca de dichos algoritmos, comentarios de especialistas sobre los mismos y sus autores y en base a esta información se filtró la lista inicial.
Para elegir el algoritmo entre los candidatos, se tomaron como criterios principales el tiempo de ejecución del algoritmo y el consumo de memoria RAM de los mismos.
Se testeó la velocidad de los algoritmos candidatos y se los clasificó de acuerdo a la misma. Se observaron 2 grupos claramente diferenciados. Los algoritmos lentos, que demoraban entre 25-40ms para 128bytes eran los establecidos como estándar ya sea por el mercado o por alguna autoridad[3]. Por otra parte, los algoritmos rápidos eran aquellos no estándares, con largos de digest menores, los cuales demoraban menos de 1ms.
Se decidió entonces utilizar para la implementación de nuestro algoritmo una función de hash no estándar las cuales, si bien generan un digest más pequeño lo que implica menor seguridad, son notablemente más rápidas y consumen muchos menos recursos.
Inicialmente se había elegido un algoritmo teniendo en cuenta ensayos realizados por terceros como ser número de colisiones (digests iguales) obtenidos a partir de la ejecución del algoritmo sobre un pool de palabras, e información más subjetiva como ser el origen de los algoritmos, sus autores y otros proyectos en los que se hubieran utilizado.
Esta elección debió ser modificada ya que al momento de implementar la contraparte en PHP, surgieron problemas vinculados a los tipos de datos manejados y las operaciones permitidas sobre los mismos. Se intentó realizar la implementación en lenguaje C embebido en PHP, pero esto fracasó debido a problemas con la configuración en el servidor.
Debido a que la experiencia y habilidades en la programación en PHP era considerablemente menor que la adquirida en C, se decidió que lo más sensato era elegir el algoritmo de hash de la lista de algoritmos implementados en PHP, y realizar su implementación en C. Esta lista de algoritmos corresponde con la lista de algoritmos más utilizados, lo que redujo considerablemente las alternativas.
Finalmente, se eligió utilizar CRC32 como algoritmo de hash, lo que resultó una opción intermedia en cuanto a tiempos de ejecución y consumo de recursos.

11.1.2. Contraseñas temporales

Como se puede ver en las secciones anteriores, este sistema se basa en la autenticación de la carga útil paquete a paquete por medio de contraseñas. Es necesario plantearse entonces la siguiente pregunta: suponiendo que nuestro sistema fuera infalible en cuanto a asegurar la autenticidad de la información, ?'Qué impide que uno de los paquetes intercambiados sea capturado por un tercero y enviado posteriormente? ?'Cómo podemos evitar esto, y en caso de que suceda, descartar el paquete?

[3]Por ejemplo MD5 y la familia de SHA

La forma de evitar que esto suceda es por medio de la utilización de contraseñas de sesión. Para esto, las dos partes intercambian regularmente contraseñas, las cuales no son necesariamente del largo de la clave original, que se utilizan como parte de la autenticación[4]. De esta forma, se garantiza que la validez de los paquetes se encuentre limitada a una sesión, con lo que se evita el problema mencionado.
Esto es difícil de implementar ya que hay que considerar asuntos relacionados a la verificación de la recepción de los códigos nuevos, por lo que en esta etapa, sólo se analizaron y discutieron posibles mecanismos de intercambios de contraseñas temporales, pero no se llegó a una solución satisfactoria, y se optó por dejarlo para implementar en una etapa posterior.

11.2. Privacidad

Es importante tener en cuenta que, para poder cumplir con los dos requisitos mínimos planteados para la seguridad (autenticidad y confidencialidad), se debe además cifrar los mensajes a trasmitir de modo de no trasmitir texto claro que pueda ser capturado por terceros. Son varios los estándares que existen para esto, como por ejemplo DES, T-DES y AES, pero todos demostraron ser excesivamente costosos en recursos y en tiempo (menos de 512b/s). Por esto, se buscó una alternativa menos costosa en tiempos de ejecución que permitiera un nivel mínimo de confidencialidad.
Existen dos tipos de algoritmos de cifrado, los simétricos que utilizan la misma clave para cifrar y descifrar, y los asimétricos que utilizan claves diferentes. Se decidió implementar un mecanismo de clave simétrica, dado que estos son más sencillos de diseñar.
El cifrado clásico se basa en dos operaciones, la sustitución (cambio de los caracteres) y la trasposición (reordenación de los mismos), las cuales son combinadas para formar el algoritmo de cifrado. Para la sustitución se utilizan generalmente mecanismos como la sustitución por tablas, u operaciones reversibles. Dentro de este tipo de operaciones, la más utilizada es el OR exclusivo (XOR) que se caracteriza por ser su propia inversa.
El algoritmo diseñado se basa en el cifrado de Vernam[5], un tipo de cifrado de flujo de bytes que data del año 1917. Este consiste en realizar la operación XOR caracter a caracter entre el mensaje y una clave, y su seguridad radica en conservar la clave en secreto. Además, esta clave debe ser cambiada con cada comunicación ya que es muy susceptible a ser descifrada por medio de criptoanálisis si la información recabada es la suficiente; más aun si se conoce parcialmente la correspondencia entre el texto cifrado y el texto claro. Previo a aplicar el algoritmo de Vernam se aplica un algoritmo que traspone los bytes del mensaje a partir de un vector de trasposición aleatorio, el cual es adjuntado al mensaje. De esta forma se evita tener que cambiar periódicamente las claves y se aumenta la seguridad del algoritmo.
Para la implementación en C, se utiliza una función para generar números pseu-

[4]Una posibilidad puede ser intercambiar números aleatorios, los cuales se utilizarán para generar una nueva clave, utilizándolas como m]'ascara y aplicando una operación lógica a la última clave utilizada, y sustituyéndola.

[5]Gilbert S. Vernam, nativo de Brooklyn e ingeniero del MIT que trabajaba en los laboratorios de la empresa AT&T

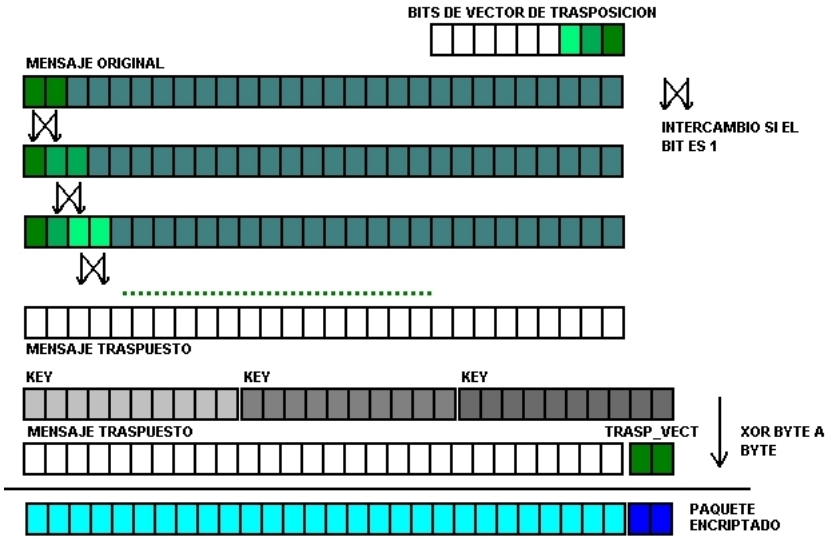


Figura 11.2: Algoritmo de encriptación

doaleatorios a partir de una semilla, esto implica que, si se usa siempre la misma semilla, la secuencia de números será siempre la misma. Para evitar esto, se utiliza como semilla la hora del sistema, de forma de evitar que se repita la secuencia.
Para cifrar el número aleatorio generado (vector de trasposición) se utiliza para decidir si se traspone o no un determinado byte del mensaje.
Se recorre el mensaje a la vez que se rota este vector, y byte a byte se decide en función del primer bit menos significativo del vector, si se traspone o no el byte con el siguiente (LSbit=1 =>se traspone). De esta forma, se "mezclan" los bytes, disminuyendo la probabilidad de obtener información sobre la clave en caso de conocer información parcial.
Al finalizar de recorrer el mensaje se le adjunta el valor final del número rotado, el cual es necesario para deshacer la trasposición. Posteriormente se realiza el XOR del mensaje traspuesto con la clave.
El proceso inverso consiste en repetir la operación de XOR a lo largo del mensaje, y deshacer la trasposición. Para esto, se comienza por el final, y con el valor final del vector de trasposición (últimos dos bytes del mensaje) se intercambian los bytes en función del LSbyte y se rota el vector hacia la izquierda. Ver Figura 11.2

11.3. Procedimiento completo

El procedimiento completo de envío y recepción de un paquete de datos puede resumirse en los siguientes pasos:

1. Se cifra el paquete
2. Se aplica HMAC sobre el paquete cifrado (incluyendo los datos agregados sobre el vector de trasposición). El resultado se agrega al final del paquete

3. Se envía el paquete
4. En el receptor, se verifica el valor del campo de autenticación, recalculando el HMAC de todo el paquete (por supuesto que sin incluir el campo de HMAC del mismo)
5. Si el valor de HMAC recibido corresponde con el calculado, se procesa el paquete
6. Se descifra el paquete recibido

Capítulo 12

Pruebas de funcionamiento

Para probar el sistema que estaba siendo desarrollado y que día a día aumentaba en tamaño y en cantidad de partes en las que podían encontrarse potenciales errores, la estrategia fue testear cada sección de código que podía considerarse completa de forma independiente al resto de las secciones, para luego aumentar progresivamente a distintos niveles de integración. Como paso final se testeó el sistema como un todo para poder observar si este cumplía o no sus funciones.

Las pruebas realizadas inicialmente se constituían generalmente de procesos automatizados, en los cuales se conocía su desarrollo de antemano y esto hacía posible que los resultados reales pudieran compararse contra estos. Pueden encontrarse a lo largo del firmware *snippets* de código que se desactivan o activan según el estado de definición de ciertos *macros* y que componen a estos procesos. Los módulos o secciones de firmware que recibieron este tratamiento son fundamentalmente

- UART
- Interfaz PLCBus
- Interfaz con memoria Flash y de tablas
- Interfaz Ethernet
- Scheduler
- Reloj
- Interfaz y motor web, escrito en php

Muchas de estas pruebas requirieron de hardware externo y elementos adicionales a nuestro módulo, a la plataforma de programación AVR Dragon o al computador que la controlaba, a saber

- modem y router para manejar la conexión a Internet y la formación de la LAN local
- Computador con puerto serie accesible para la utilización del *sniffer* del puerto

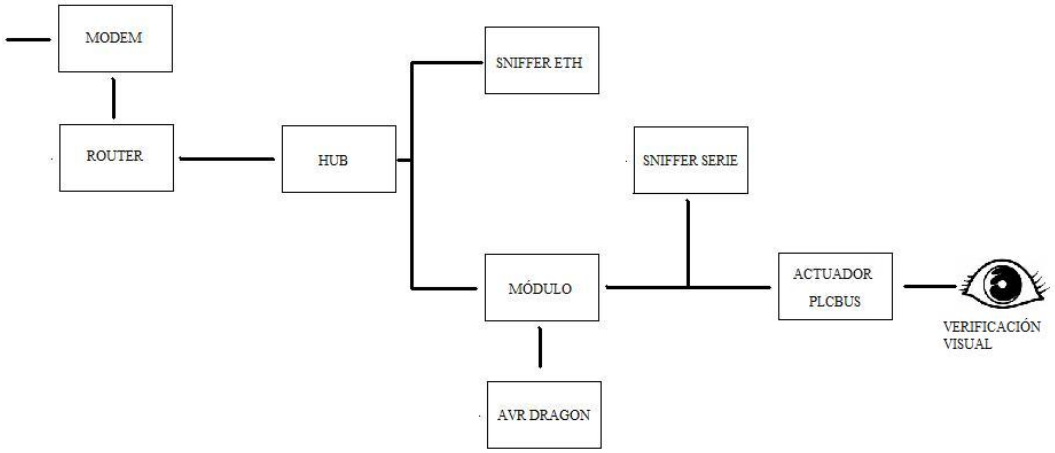


Figura 12.1: Diagrama completo de elementos utilizados para las pruebas de funcionamiento.

- hub para poder "husmear" el envío de paquetes del módulo
- actuadores PLCBus y lámparas

Conforme se integraban las piezas de código, las pruebas requerían más dedicación y tiempo, pues también empezaban a actuar factores externos a nuestro módulo en sí como pueden ser el funcionamiento intrínseco del servidor web que utilizamos, la pasarela PLCBus 1141 o el funcionamiento real de las funciones de php, de los cuales no se tenía ni la experiencia de trabajo previa ni la información completa para cada caso de utilización. Estos obstáculos no pudieron solventarse de otra forma que la experimentación y el ensayo y error.

Una vez que se integraron todas las partes, se realizaron numerosas y extensas pruebas de:

- *Actualización periódica de la IP de módulo en el servidor.* Este proceso requiere que cada 250 segundos se envíe un paquete GET con ciertas características al servidor de aplicaciones con el correspondiente inicio y corte de conexíon. En este punto se utilizó fundamentalmente el sniffer Ethernet (Wireshark[17])[1]
- *Ejecución de comandos instantáneos.* Se probaron la mayoría de las posibilidades de comandos que ofrece el protocolo PLCBus. En este punto se verificaba no sólo a través de sniffer Ethernet sino también visualmente que los actuadores PLCBus hubieran cumplido su tarea.
- *Petición y posterior envío del estado de los artefactos. Status* El uso del Wireshark aquí fue de vital importacia.
- *Ejecución de comandos agendados y actualización de las tablas de agenda diarias.* Mediante el Wireshark se observaban los intercambion servidor-módulo de las tablas diarias y posteriormente se requería que visualmente

[1] Al comunicarse el módulo directamente con el servidor web de aplicaciones, para husmear esta comunicación se requiere en todas los casos, aparte del software Wireshark, la utilización del *hub* o repetidor. De esta forma los paquetes transmitidos hacia o desde el módulo puedan ser captados por la computadora que corre el sniffer

se comprobara que los comandos se ejecutaran en los actuadores PLCBus a las horas y días fijados.

- *Petición y posterior envío del log de eventos.* Las capturas llevadas a cabo con el Wireshark fueron vitales para determinar si los paquetes se encontraban bien formados y si se agregaban entradas al log luego de que se ejecutaba alguún evento agendado.
- *Inclusión de escenas en la agenda.*
- *Correcta actualización del sistema de tablas luego del cambio de día* Dadas ciertas características del firmware, era muy probable que existiera un bug en este proceso. Sin embargo, testeando este hecho específicamente se llegó a la conclusión de que el firmware funcionaba correctamente (para este caso!).

- Una vez que los anteriores procesos funcionaban con normalidad, se apuntó a probar el funcionamiento del módulo en largos períodos de tiempo (lapsos de hasta 15 horas). En estos períodos se agendaban numerosos eventos a desarrollar y eventualmente se requería el status, el log de eventos o se actualizaban las tablas.

- Luego de superadas las anteriores y una vez que estaba implementada la parte de seguridad, esta se probó. En este punto se encontraron grandes dificultades ya que al tenerse que implementar los mismos algoritmos para dos diferentes lenguajes y arquitecturas de sistema tan disímiles, era natural que aparecieran diferencias.

Cabe recalcar que en todas las pruebas anteriores, el aspecto de la intercomunicación entre el módulo y el servidor web es troncal y de vital importacia. Este hecho llevó a la búsqueda constante de la optimización del funcionamiento del Stack TCP/IP, y las pruebas anteriores aportaron a esta causa lo suyo en todos los casos. Más aún, no hay que olvidar que los anteriores son intercambios de información entre dos partes, por lo que los bugs también se encontraban en cualquiera de las dos partes, hecho que también dificultaba la detección del error.

12.1. Pruebas de la interfaz Ethernet

Se describen las pruebas realizadas para testear ciertos aspectos de la comunicación Ethernet. Estas consisten en un simple pedido fecha y hora a un servidor SNTP, el cual debe ser generado y correctamente procesado por el módulo en desarrollo. Aunque de apariencia simple, la resolución exitosa de este proceso representa el funcionamiento también exitoso de una serie de procedimientos de manejo de software y hardware llevados a cabo en el módulo los cuales están vinculados de una forma no tan simple entre sí. Por ejemplo: la simple generación de una petición de hora al servidor SNTP significa que anteriormente se generó un pedido ARP, se procesó de forma correcta su respuesta, se generó un paquete ICMP al servidor, el cual fue respondido y esto fue captado

por el módulo para luego generarse la petición SNTP.

La red Ethernet en la cual se realizó la prueba consistió en una conexión del módulo a una PC mediante un cable cruzado.

La secuencia de la prueba fue:

1. Desde una terminal serie en una PC se pide el tiempo (enviando los caracteres 'W"T'con la terminal).
2. Esto desencadena un ARP request.
3. Llega el ARP reply y se procesa.
4. Se hace un Echo request (función de ICMP).
5. Llega el Echo reply y se procesa.
6. Se hace un SNTP request.
7. Llega una respuesta SNTP y se procesa.
8. Luego de hacer una lectura (enviando los caracteres 'R"D' con la terminal) desde una terminal se muestra el tiempo en pantalla.[2]

Esta prueba se llevó a cabo como presentación final de proyecto para la asignatura Sistemas Embebidos en Tiempo Real dictada por el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería, realizada el 25 de Junio del 2008 en el mencionado instituto.

12.1.1. Desarrollo

Se utilizó a lo largo de todo el proceso de desarrollo una conexión por medio de cable UTP cruzado CAT 5 entre la tarjeta de red de la PC utilizada y la placa de desarrollo. Como herramienta principal, se usó el analizador de protocolos de red Wireshark, para poder capturar y visualizar el tráfico, y observar la correcta formación de los paquetes enviados. Además, se utilizó la función ping de Windows para poder verificar el funcionamiento de ARP e ICMP reply, así como para desencadenar interrupciones de paquete entrante en las primeras etapas de diseño.

Dado que el diseño se centró en el stack TCP/IP, y la funcionalidad de comunicación por intermedio de la UART y el puerto serie de la PC se agregó a último momento, el sistema se probó a lo largo del desarrollo iniciando los procesos vía código, antes de entrar al loop principal.

Para realizar dichas pruebas, no se necesita realizar muchos cambios en la configuración estándar de la PC, simplemente se debe tener cuidado de deshabilitar el firewall de Windows, o habilitar en el mismo el puerto 123 de UDP (correspondiente al servidor NTP). Además, se deben configurar los datos de

[2]Se muestra el tiempo en la terminal en el formato dd/mm/aa hh:mm:ss

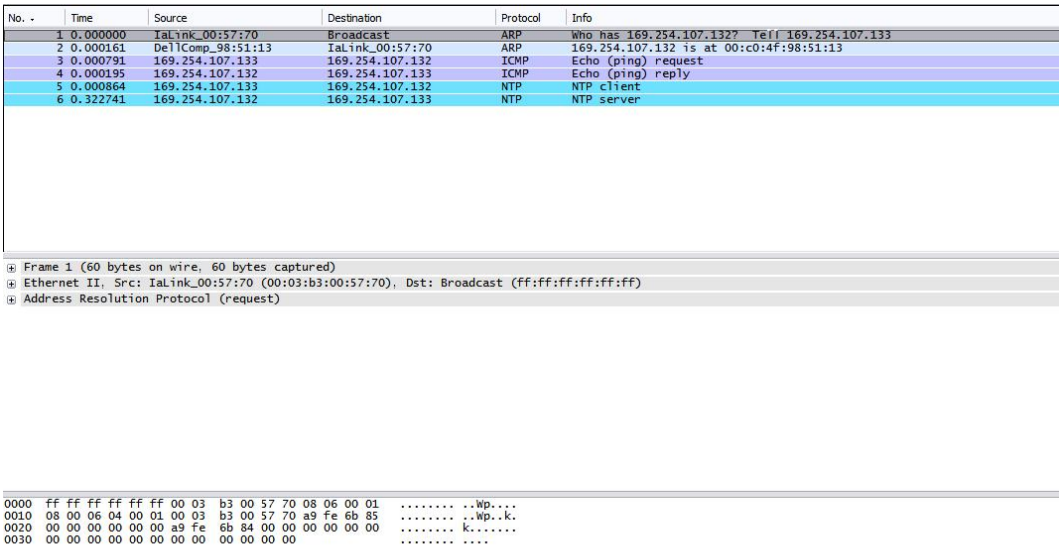


Figura 12.2: Secuencia desarrollada en la prueba de la interfaz Ethernet.

dirección IP del servidor SNTP y del gateway en el firmware previamente (grabados en EEPROM) con el IP de la PC. No es necesario instalar un servidor NTP (por lo menos en Windows XP) ya que este cuenta con uno propio aunque en otros SO podría llegar a ser necesario instalar uno.

Los protocolos se fueron probando a medida de que se escribían, primero ARP, luego ICMP y finalmente UDP. Las pruebas se realizaron utilizando la herramienta de debug de AVR Studio, por medio de la interfaz JTAG del AVR Dragon.

La función de SNTP timestamp_to_date() se probó pasándole por código un timestamp correspondiente a una fecha conocida y verificando los cálculos hechos paso a paso.

Luego de haber llegado a una versión definitiva del stack a entregar, se agregó la funcionalidad de comunicación vía puerto serie. Se utilizó el programa Hercules Setup 23, software muy similar al clásico Hyperterminal, para la comunicación vía puerto serie, configurándolo a 4800bps, caracteres de 8 bits, 1 bit de parada y sin paridad (8-N-1).

Las pruebas resultaron según lo esperado, la hora se procesó y desplegó de forma correcta. En la figura Figura 12.2 se aprecia el detalle de la captura de paquetes mediante el Wireshark en donde se aprecia la secuencia descrita más arriba.

Parte III

SERVIDOR

Capítulo 13

Introducción a las herramientas utilizadas

13.1. Asuntos previos

Recordemos una vez más de que se trata este proyecto: se parte del problema de controlar remotamente un hogar, sin tener una PC dedicada a esto instalada en el lugar. Por "controlar el hogar" nos referimos a manejar cualquier dispositivo eléctrico instalado allí, siendo el caso más simple el de encender y apagar una lamparita.
Existen varios protocolos para controlar dispositivos electrónicos del hogar, y podemos categoriozarlos según el medio a través del cual se comunican los nodos de la red domótica. Las categorías son básicamente tres:

1. de forma inalámbrica[27][26]
2. utilizando un cableado dedicado[28]
3. utilizando la red eléctrica[29]

Dado que una de nuestras premisas era desarrollar una solución lo más simple posible, tanto para el usuario como para el instalador; y a la vez del menor costo posible, optamos por utilizar el protocolo PLCbus[3]. La instalación de una red de este tipo requiere configurar algunas direcciones y conectar actuadores[1] a la red eléctrica (entre otras cosas), y los componentes que utilizan son relativamente económicos. Considerando todo lo anterior, la solución que se propone contiene tres componentes fundamentales. A saber, una página web (por decirlo de una manera sencilla), hardware propio, y el firmware de dicho hardware.
En esta parte de la documentación nos centraremos en el primer componente de la solución: la página web, o mejor dicho la programación de dicha página.

En los inicios del proyecto HAOPL (también **domoip**), se manejaron dos alternativas para resolver el problema de como el usuario se comunicaría con el

[1] su función principal es captar los comandos que les llegan por la red y en función de estos modificar la alimentación del "electrodoméstico" que tengan conectado.

módulo[2]: A. hacer una página web colgada en un servidor web estándar, que "hablaría" de alguna manera con el módulo; B. hacer un web server embebido en el módulo, y de alguna forma hacer que que el usuario vea una página también almacenada en este.
La solución B se descartó de inmediato dado que si bien ofrecía alguna ventaja, sobrecargaba aún más los escasos recursos de nuestro microcontrolador ATmega 32 de 8 bits[5].
Sin embargo, de haberse optado por esta solución, nunca hubiese existido el problema de comunicación entre el servidor y el módulo, ya que serían la misma cosa.
Se optó entonces por la opción A, cuya principal ventaja es la abundancia de información respecto a programación web, y la diversidad de lenguajes y scripts. Cabe resaltar que si bien los cursos y tutoriales disponibles en la web abundan, la calidad también es por lo general, pésima.

13.2. Selección de los lenguajes

El problema era ahora seleccionar qué lenguaje o lenguajes se usarían. Se comenzó haciendo una primera versión de la interfaz gráfica en HTML[30][31][32], eligiendo este dados su uso expandido, sencillez y abundante y no tan mala información disponible.
Utilizando HTML pueden hacerse páginas estáticas y muy poco atractivas para el usuario. Por eso surgió la necesidad de dotar la página de algo de dinamismo. Se incorporó entonces Javascript[33][34].
Si se quiere además de tener una página web con contenido dinámico, utilizar un estilo particular en el que están definidos los tipos de letra de cada sección de la página, los botones, los contenedores, los headers, etc., debe pensarse en CSS[35][36]. Con esto se estandariza todo lo que tiene que ver con el estilo de la página y se torna más "homogenea".
Una vez resueltos los problemas gráficos, deben resolverse los problemas relacionados a los procesos de comunicación entre el módulo y el servidor, al almacenamiento de datos y a la seguridad. Para esto se optó por PHP[37], script de programación web que corre en el server, a diferencia de HTML, Javascript y CSS que se interpretan en la máquina cliente. Se eligió PHP dado que es ampliamente utilizado hoy en día, y además puede encontrarse toda la infromación necesaria en su página web[37]; algo de altísimo valor considerando el área en la que se trabajó[3].
Lo único que resta ahora por saber es como se van a almacenar los datos que tienen que ver con el usuario y la red domótica.
Si bien el volumen de datos es pequeño, se optó por una base de datos MySQL[38] admnistrada desde PHP. Esta opción es natural dado que PHP tiene incorporado un paquete de funciones para acceder y administrar una base de datos MySQL, que hacen que su gestión sea extremadamente sencilla.

[2]Hardware desarrollado en este proyecto para hacer de puente entre una LAN Ethernet y un protocolo propio de comunicación de la interfaz PLCbus 1141. Por más detalles de la interfaz PLCbus 1141 ver sección 6.5 *PLCBus 1141*

[3]En general, en esta área de desarrollo no es fácil encontrar sitios que contengan información precisa y suficiente. En el caso de PHP, su página web oficial provee un manual muy completo y con variedad de ejemplos.

13.3. Introducción a los lenguajes utilizados

13.3.1. HTML

HTML o Hipertext Markup Language es un lenguaje destinado a la programación de páginas web. Está pensado para crear páginas sencillas y estáticas. HTML es utilizado para definir la estructura general de la página y algunas pocas cosas más. Al día de hoy todas las páginas incorporan scripts y archivos de estilo que las tornan dinámicas y estéticamente más refinadas.
Para crear un archivo HTML lo único que debe hacerse es crear un archivo con un editor de texto plano y guardarlo con extensión .html. La estructura básica de un archivo HTML es la siguiente:

```
<HTML>
<HEAD>
<TITLE>
Página más simple posible
</TITLE>
</HEAD>
<BODY>
Página HTML más simple posible
</BODY>
</HTML>
```

Si vemos el código anterior interpretado por un browser, veremos simplemente el texto contenido en el body y el título de la página[4] en la pestaña del explorador.
A grandes rasgos un archivo HTML está dividido en dos partes: el HEAD y el BODY. La primera contiene cosas tales como el título, información sobre el autor de la misma, paths a archivos de estilo (Ej.: CSS), scripts (Ej.: Javascript), etc.; la segunda contiene el contenido propiamente dicho.
La sección BODY puede contener texto de diferentes tamaños y estilos, tablas, formularios, botones, etc..
Veamos ahora un ejemplo más complejo:

```
<HTML>
<HEAD>
<TITLE>Ejemplo 15</TITLE >
</HEAD>
<BODY>
<H1>Formularios</H1 >
<FORM ACTION='mailto:unaprueba' METHOD='POST'>
Texto: <INPUT TYPE='text' NAME='nombre'><BR>
Password: <INPUT TYPE='password' NAME='contra'><BR>
Sexo:<INPUT TYPE='radio' NAME='boton1' VALUE='1'>Hombre
<INPUT TYPE='radio' NAME='boton1' VALUE='2'>Mujer<BR>
Vehiculo:<INPUT TYPE='checkbox' NAME='Moto' VALUE='Si'>Moto
<INPUT TYPE='checkbox' NAME='Coche' VALUE='' CHECKED>Coche
<BR ><BR>
<INPUT TYPE='submit'><INPUT TYPE='Reset'>
</FORM>
</BODY>
</HTML>
```

El browser interpreta el archivo anterior como lo muestra la 13.1.
La sección BODY del template HTML anterior contiene un formulario[5] para ingresar algunos datos, y los botones enviar consulta y limpiar.

[4] contenido entre ‹TITLE ›y ‹\TITLE›

[5] Un formulario (FORM) define una sección del código que usualmente espera recibir algún tipo de dato, y para la cual se prevee algún tipo de procesamiento, definido este último en la opción ACTION del tag de inicio del formulario.

Figura 13.1: Ejemplo HTML

Si un visitante oprime el botón "enviar consulta" los datos se envían utilizando el método de HTTP indicado en la opción METHOD[6], para su posterior procesamiento[7].

13.3.2. Javascript

Javascript permite mediante pequeños segmentos de código embebidos en el HTML, crear una página con contenido dinámico. Al igual que HTML Javascript es interpretado en la máquina cliente.
Si bien puede realizarse algún procesamiento de datos con Javascript, su principal función es de agregar detalles gráficos simples a la página.
Veamos un ejemplo:

```
<HTML >
<HEAD >
<TITLE >
Página con Javascript embebido
<\TITLE >
<SCRIPT=”Javascript” >
alert(”Saludo de Bienvenida JavaScript”);
<\SCRIPT >
<\HEAD >
<BODY>
Página HTML con Javascript embebido<BR >
<A HREF='JavaScript:alert('Enlace 1')'>Enlace 1<A><BR><A HREF='#' onClick='alert('Enlace
2')'>Enlace 2<A><BR><\BODY >
<\HTML >
```

Cuando la página anterior se carga en el browser se despliega el mensaje 'Saludo de Bienvenida JavaScript'. En la sección BODY se crean dos enlaces[8] tales que al hacer clic sobre el link despliegan 'Enlace 1' o 'Enlace 2'.

[6] Método POST en este caso
[7] En este caso se envía un mail
[8] El tag ANCHOR de HTML permite crear enlaces (links) a otras páginas.

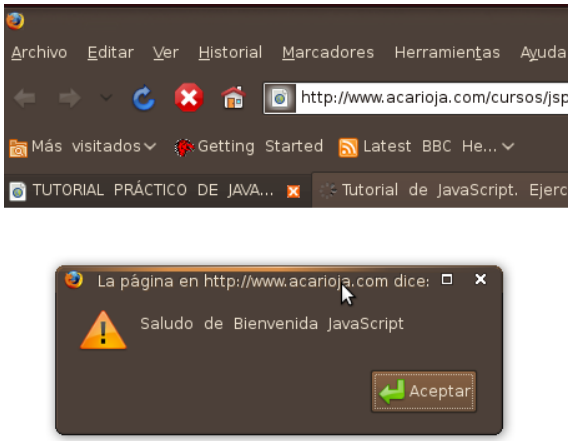


Figura 13.2: Al cargar la página

Figura 13.3: Luego de hacer clic en Aceptar en el mensaje de bienvenida

Puede verse la secuencia en la Figura 13.2 y la Figura 13.3
Es usual definir funciones en Javascript en la sección HEAD de HTML, y esas funciones son llamadas desde algún lugar de la sección BODY de HTML.
Javascript es extremadamente útil para conocer datos de la máquina cliente, y actuar en función de estos. Por ejemplo la página de nuestro proyecto esta optimizada para resoluciones 1280 x 800 y 1024 x 768. Dado que es posile conocer con Javascript la resolución de la máquina cliente, si se detecta al cargar la página de inicio que la resolución es menor a 1024 x 768 se despliega un mensaje. Es cuando se desea conocer este tipo de información que Javascript se vuelve la herramienta justa.

13.3.3. CSS

Lo único que hace falta para tener una página medianamente aceptable para el estándar actual es CSS. Cascading Styles Sheets permite homogeneizar el estilo de la página.
CSS define una sintáxis particular, y respetando esta se pueden definir los estilos de cada una de las partes que componen un archivo HTML. Podemos definir por ejemplo el tipo y color del texto de la sección BODY, el tamaño de las imagenes, hacer botones dinámicos, etc..
Veamos el siguiente ejemplo:

```
body {
padding-left: 11px;
font-family: Georgia;
color: red;
background-color: #d8da3d;
}
```

Con la secuencia de comandos anterior se define para la sección BODY de un archivo HTML: un padding ('relleno') de 11 pixeles, el tipo de letra del texto, el color del mismo, y el color de fondo. Lo mismo puede hacerse para cada elemento de HTML: headers, párrafos, enlaces, contenedores, imagenes, etc..
Hay dos formas de utilizar CSS:

1. Mediante un archivo externo con extensión .css.
2. Al igual que con Javascript escribir un SCRIPT CSS dentro de la seccón HEAD del archivo HTML en cuestón.

Para utilizar un archivo particular de estilo en un archivo HTML lo único que debe hacerse es indicarlo en el HEAD de la siguiente forma:

```
<link href='../styles.css' rel='stylesheet' type='text/css' />
```

en donde el path y nombre del archivo CSS se indican en la opción 'href'.
La ventaja de utilizar la primer forma, si se quiere utilizar el mismo estilo para más de una página (usualmente es así) es evidente, y radica en que no hace falta escribir cada uno de los comandos CSS cada vez que se crea una nueva página.
Para dar un último e ilustrativo ejemplo veamos como se ve la misma página web con y sin el archivo de estilo CSS:

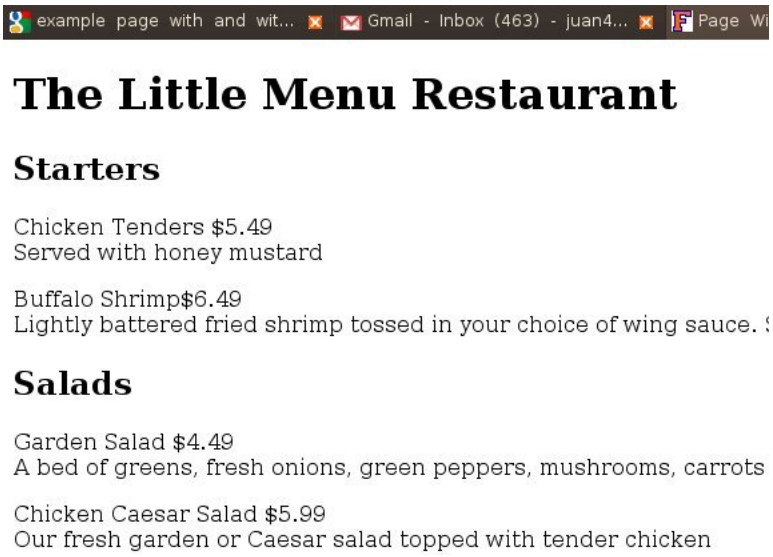


Figura 13.4: Página HTML sin el archivo CSS cargado

Figura 13.5: Página HTML con el archivo CSS cargado

13.3.4. PHP

PHP es un script de programación web muy poderoso. A diferencia de Javascript y HTML que se interpretan en la máquina cliente, un archivo PHP genera un proceso que corre en el servidor. PHP toma conceptos de Perl, C y otros lenguajes y scripts y agrega funcionalidades como la apertura y manejo de sockets TCP/IP que lo hacen muy completo y adecuado a nuestro desarrollo.
Es común sin embargo, utilizar HTML y Javascript desde PHP, utilizando el comando ECHO de PHP que imprime en pantalla la cadena de caracteres que se le pase como parámetro. Además en un mismo archivo con extensión PHP pueden escribirse fragmentos de HTML y PHP mezclando ambos.
Para escribir un archivo PHP debe utilizarse un editor de texto plano y almacenar el archivo con extensión .php. Veamos un ejemplo:

```
<?php
echo 'script php más simple posible';
?>
```

Lo que hace el archivo anterior es imprimir en el browser 'script php más simple posible'.
Con PHP se puede hacer prácticamente cualquier cosa en el servidor. Entre los comandos más útiles a efectos de nuestro desarrollo se encuentra la función *fsockopen()*, mediante la cual se abre un socket TCP/IP a la IP y puerto que se indique. Esto resultó fundamental pues resuelve una de las dos vías de comunicación entre el servidor web y el módulo.
Una limitación importante que trajo problemas de compatibilidad entre algoritmos implementados en el servidor y el módulo[9], fue la inexistencia de ciertos tipos de datos en PHP. Ocurre que en C y más aún programando un procesador es común definir variables con los tipos unsigned short, unsigned int o unsigned long entre otros. No es posible en PHP definir una variable como unsigned, y esto trajo problemas a la hora de implementar funciones de hash para autenticación[10].

13.4. Introducción a MySql

Una base de datos esta dividida en tablas. Cada tabla contiene filas y columnas. En el momento de la creación de una nueva tabla deben especificarse el nombre de cada columna y el tipo de dato[11] que contendrá.
El uso de la misma consiste en ir agregando o quitando filas, o modificando el contenido de una o varias filas (modificando registros).
Una de las principales ventajas es la facilidad para buscar dentro de una tabla en particular, filas cuyos registros coincidan con algunos parámetros definidos en la busqueda. Esto acorta mucho el código y es de extrema sencillez.

El uso local de una base de datos SQL requiere de la instalación de un servidor SQL y la creación de un cliente SQL en primer lugar.

[9] Usando C para el firmware y PHP para el servidor
[10] Ver sección 16.1 *Autenticación*
[11] Los tipos más comunes son varchar(characters) e int(enteros)

MySql tiene definida una sintaxis específica para cada uno de los comandos. Veamos ejemplos de algunos comandos:

1. Creación de una tabla

   'CREATE TABLE miTabla (registroChar varchar(3), registroInt int(5))'
   Con esto creamos la tabla 'miTabla' con dos tipos de datos: registroChar de hasta 3 caracteres, y registroInt de hasta 5 enteros.

2. Insertar un registro

   'INSERT INTO miTabla (registroChar) VALUES ('abc')'
   Con este comando agregamos una fila a la tabla 'miTabla' con el registro 'registroChar' inicializado con la cadena 'abc'.

3. Obtener el valor de un registro de una tabla en particular

   'SELECT registroInt FROM miTabla'
   Con este comando obtenemos un array que contiene el registro 'registroInt' de cada fila.

Esos son los comandos que podrían catalogarse como básicos de MySql.

Para poder utilizar el servidor MySql de un servidor web debe crearse un cliente, crear una base de datos asociada a ese cliente y para esa base de datos en particular pueden crearse todas las tablas que se deseen, obviamente luego de logearse.

13.5. Apache Web Server

Para poder realizar pruebas de entorno local, es decir utilizando nuestra propia máquina como servidor web, son necesarios tres pasos fundamentales.

1. Instalar alguna versión de PHP.
2. Instalar un servidor web.
3. Instalar un servidor MySql

Si se trabaja en linux[12] estos tres pasos son tan simples como descargar el paquete LAMP[13] con el gestor de paquetes. No requiere prácticamente ninguna configuración adicional salvo si se desea, la ruta en donde el servidor debe buscar las páginas.

Se optó por Apache Web Server[39] dado su carácter de freeware bajo licencia GPL, y su muy extendida utilización en la mayoría de los web servers que corren sobre alguna distribución de Linux. Además la información que ofrece su página web es muy clara y su configuración muy sencilla.
La configuración de PHP en un servidor Apache instalado sobre Windows XP no fue sencilla, y esto incidió en la utilización de Ubuntu[40] para el desarrollo

[12] Se utilizaron las distribuciones ubuntu 8.04 y ubuntu 8.10
[13] Linux-Apache-MySql-PHP

de la página web[14].
La configuración de un servidor Apache consiste en modificar algunas líneas de su archivo de configuración[15] y lo único que debe hacerse para ver una página en el localhost[16] es colocarla en una carpeta cuya ruta puede indicarse en dicho archivo de configuración.

[14] No debe confundirse la página web desarrollada en este proyecto con una página web estándar en el sentido de que generalmente una página estándar, ofrece una interacción entre un usuario y un servidor, mientras que esta página resulve la comunicación entre tres puntos: un usuario, un servidor web y nuestro módulo

[15] apache2.conf dependiendo de la versión del servidor

[16] http://127.0.0.1 en general

Capítulo 14

Interfaz Gráfica y procesos

14.1. Estructura general de la página

Cuando se accede a `http://www.radiocostadeoro.com`[1] se redirecciona a la página de login (ver Figura 14.1).

Una vez completados los campos de Username y Password se accede a la siguiente página de inicio (ver Figura 14.2). En función de los datos que se ingrese en el Login se crea una sesión[2] que asocia ese acceso con cierto módulo en particular[3]. De allí en más todas las acciones que se realicen tendrán como destinó el módulo que pertenece a ese usuario.

Una vez en la página de inicio vemos la pantalla dividida en tres partes: superiores izquierda y derecha y la barra inferior.
Como la mayoría de las páginas web actuales esta compuesta por frames. La ventaja de utilizar estas radica en que algún sector de la página puede permanecer estático mientras que otro cambia. Esto es útil si por ejemplo se desea mantener ciertas "funciones" accesibles siempre en uno de los lados de la página, como es el caso de la nuestra[4].
Vemos entonces que la porción superior izquierda tiene 7 links. Estos son:

- Ejecutar comando
- Registro de sucesos
- Estado del sistema
- Configurar dispositivos
- Escenas
- Agenda
- Help

[1] El dominio y hosting fue un prestamo gentileza del Sr. Fernando Manacorda, URSEC. En el mismo iba a ubicarse una radio online.
[2] Ver Anexo C *Página multiusuario*
[3] El sistema está pensado para que desde una misma página web, con distintos usuarios se puedan controlar varias casas.
[4] Los botones en el márgen izquierdo de la pantalla permanecen estáticos mientras el usuario navega

Figura 14.1: Login

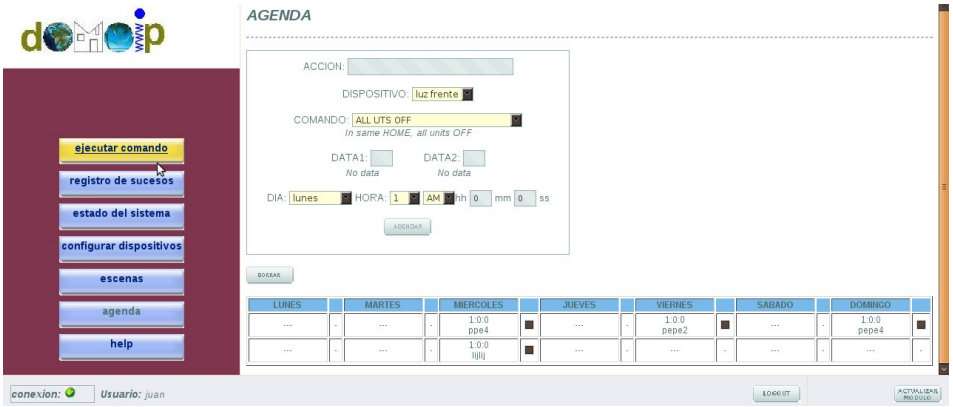


Figura 14.2: Página de inicio

Desde *Ejecutar comando* se puede simplemente enviar un comando a uno o varios actuadores. Si se pulsa allí se accede a un página desde donde luego de elegir el actuador y el comando se puede enviar este mediante TCP/IP a el hogar del usuario, provocando que uno o más actuadores actúen en consecuencia. Desde *Registro de sucesos* se pueden consultar los últimos 16 comandos que se ejecutaron en el hogar, y se puede solicitar la actualización de este registro. Desde *Estado del sistema* se puede consultar el estado de los actuadores y solicitar una actualización de dicho estado. Desde *Configurar dispositivos* se puede consultar que dispositivos están instalados en el hogar, agregar y borrar actuadores. Desde *Escenas* se puede ver que escenas fueron creadas, borrarlas, editarlas, crear nuevas o agendarlas. Desde *Agenda* se pueden agendar comandos, borrarlos y también borrar escenas. Cuando se tiene activo este link se observa un sector de la pantalla desde donde se puede agendar un comando, y más abajo se muestra una tabla semanal con los comandos y escenas agendados.
En la porción superior derecha de la pantalla observamos la página que se correponde con el link activo, es decir luego de pulsar uno de los botones a los que nos referimos arriba, se modifica el contenido de este frame.
Por último en la porción inferior de la pantalla tenemos una barra que indica en su sector izquierdo el estado de la conexión con el módulo, al lado el usuario con el que nos hemos logeado, y en el sector derecho de la misma tenemos la opción de deslogearnos o actualizar el módulo si es necesario. Sucede que una vez que uno agrega o quita de la agenda un comando o una escena, debe actualizarse el módulo. Por esto se lleva un registro de cuando fue la última vez que se actualizó el módulo, y si ocurre algún cambio se muestra el botón de actualizar el módulo.

14.2. Descripción de las funcionalidades

La página web brinda las siguientes funciones al usuario:

1. Login, usuario normal y administrador del sistema
2. Actualización del módulo y estado de la conexión
3. Selección de la hora local (respecto a la GMT), hora ”amanecer” y hora “anochecer”
4. Consulta de los actuadores existentes, agregar y quitar actuadores
5. Ejecución de un comando
6. Consulta y actualización del registro de sucesos
7. Consulta y actualización del estado del sistema
8. Agendar o quitar comandos
9. Crear editar, borrar y agendar escenas

A continuación veremos una descripción de las mismas.

14.2.1. Login, usuario normal y administrador del sistema

Desde los inicios del proyecto se tuvo en cuenta la necesidad de tener un login. Al comienzo se pensó que esto sería una tarea sencilla, de una tarde y algo más, pero no fue tan así.
La idea inicial fue desarrollarlo nosotros mismos, pero dado que estabamos trabajando en PHP y dada la abundancia de scripts disponibles en la web se decidió buscar uno y adaptarlo. Luego de probar varios, se optó por la versión 1.1 del login de *roscripts.com*[41], que si bien no estaba completo, estaba escrito de una manera clara, de forma que la adaptación a nuestras necesidades fue relativamente sencilla. No obstante, a medida que la página fue creciendo y agregando nuevas funcionalidades, hubo que modificar algunos archivos de login en varias oportunidades. Las principales modificaciones fueron:

1. Hacer funcionar el "olvido de contraseña"
2. Modificar los datos que se ingresan al crear una cuenta, ya que por ejemplo es necesario ingresar el identificador del módulo.
3. Modificar los chequeos que se hacen al ingresar un usuario nuevo, para verificar que los registros en la base de datos sean únicos.
4. Modificar los datos que se inicializan en la base de datos al crear una nueva cuenta.
5. Agregar opciones para borrar datos al eliminarse un usuario.
6. Proteger cada una de los archivos que componen la página web, incluyendo para esto algunas líneas de código al inicio.

El acceso al contenido de la página implica la existencia de un usuario registrado. Dado el producto que se desarrolló, la idea es que puedan acceder al sistema solo aquellos usuarios que tengan el servicio instalado, por lo que el registro de usuarios no es libre, sino que debe solicitarse uno al administrador del sistema. Es este administrador quien tiene la capacidad para agregar, borrar o modificar las cuentas existentes.
El administrador lo único que debe hacer es logearse con su usuario y contraseña, y en lugar de acceder a la página normal de inicio, accede a la página de administración de cuentas.
El paquete de archivos de login lo que hace en definitiva es: bloquear el acceso al contenido de la página web, de aquellos usuarios que no se han registrado; permitir agregar, borrar y editar usuarios a través del administrador y enviar un mail en caso de olvido de contraseña si el usuario lo solicita.

14.2.2. Actualización del módulo y estado de la conexión

El módulo contiene almacenados en memoria flash una agenda semanal con comandos que debe ejecutar a cierta hora. Cuando un usuario agenda o quita un comando o una escena[5] en la página web, la agenda almacenada en la página web y la agenda almacenada en el módulo difieren. Es por esto que es necesaria

[5] Una escena no es otra cosa que una serie de comandos cada uno con un offset. Cuando se agenda una escena a una cierta hora, los comandos se ejecutan a partir de dicha hora

la actualización del módulo. Dependiendo de un registro almacenado en la base de datos del servidor se sabe si debe o no actualizarse el módulo, y en función de este registro se muestra un cartel (Módulo actualizado) en la porción inferior derecha de la pantalla, o se muestra un botón para actualizar el mismo.
La actualización del módulo implica enviar las tablas que se corresponden a cada uno de los días vía un socket TCP. El módulo simplemente actualiza la información de la agenda almacenada en memoria flash[6].

Si una actualización sale mal, ya sea por que no se pudo establecer la conexión con el módulo[7] o porque alguno de los intercambios no respeta los formatos preestablecidos, o porque el mensaje no es autentico[8], se avisa al usuario que la actualización falló, y el color de la conexión[9] pasa a rojo hasta que un intercambio se produzca normalmente.

14.2.3. Selección de la hora local, hora "amanecer" y hora "anochecer"

Dado que el módulo debe ejecutar comandos a horas determinadas, este de alguna manera debe conocer la hora[10]. El mecanismo que utiliza para mantener la hora actualizada se basa en la interrupción de un timer a intervalos de tiempo fijos, y en la sincronización con un servidor SNTP. Consultado este servidor, envía un paquete UDP que contiene la hora actual. Ocurre que esta hora es GMT[11], y dado que la página esta pensada para manejar módulos en principio en cualquier parte del mundo, es necesaria la configuración de la hora local. Esto puede hacerse desde la barra inferior.
La configuración de la hora local, del lado del servidor solo implica almacenar dicho valor en la base de datos, asociandolo al usuario.

Algo opcional que surgió de comparaciones con otros productos similares, es el poder contar con la hora en que amanece y la hora a la que oscurece, dado que son horas a las cuales un usuario querría particularmente agendar algún comando o escena. Cuando algo se agenda, puede hacerse a una hora cualquiera, al amanecer o al anochecer.
Del lado del servidor puede obtenerse la hora "amanecer" y la hora "anochecer" simplemente utilizando dos funciones de PHP, las cuales reciben como parámetro coordenadas de latitud y longitud. Por defecto, la hora local es la GMT, y las coordenadas alguna que pertenezca al meridiano de Greenwich. Cuando un usuario configura la hora local, se elige en base a una tabla las coordenadas correspondientes a dicho uso horario, y con este dato se utilizan las funciones auxiliares de PHP para conocer la hora "amanecer" y "anochecer" del hogar del usuario del sistema.

[6] El intercambio de datos se analiza en profundidad en la sección capítulo 9 *Comunicación entre webserver y módulo*
[7] El establecimiento demora más que un cierto tiempo predeterminado
[8] Ver sección 16.1 *Autenticación*
[9] Normalmente en verde e indicado en el extremo inferior izquierdo de la pantalla
[10] Ver sección 10.5 *Emph*
[11] Greenwich Mean Time

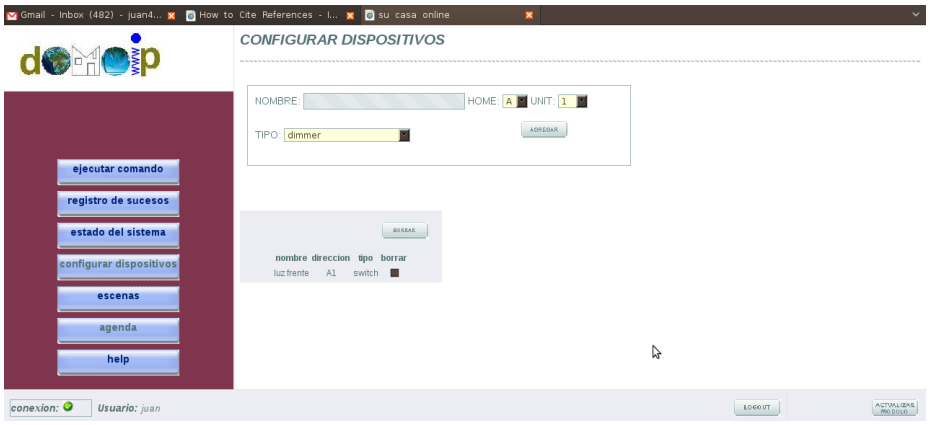


Figura 14.3: Página para consultar, agregar o quitar actuadores

14.2.4. Consulta de los actuadores existentes, agregar y quitar actuadores

Cada vez que se agrega un actuador a la red domótica debe configurarsele una dirección, de forma de poder destinar los comandos a un solo dispositivo de la red. Esta dirección, además de configurarla en el actuador mismo, debe configurarse en la página web. Es decir la informacón de la dirección de los actuadores está contenida en el propio actuador y en el servidor, pero no en el módulo. Una vez que el instalador configura la dirección del actuador, este debe ingresar a la página web y agregar el dispositivo.

A través de uno de los links laterales se accede a la página **configurar dispositivos**(ver Figura 14.3), mediante la cual se pueden consultar los dispositivos existentes en la base de datos, agregar un actuador o quitarlo.
Agregar un actuador es muy sencillo, simplemente se debe elegir un nombre, una dirección y un tipo, y presionar agregar. El página chequea que el nombre y la dirección sean únicos, y si es el caso no hay nada más que hacer. De la misma forma, si se quita un actuador de la red domótica, debe borrarse en la página web.

14.2.5. Ejecución de un comando

Además de agendar comandos y escenas, el usuario de la página web puede querer ejecutar una acción de inmediato[12]. Es por esto que incluímos esta función. La interfaz es simple: se accede a la página **ejecutar comando** a través de uno de los links en la barra lateral izquierda(ver Figura 14.4). Una ves allí, se debe elegir el actuador al que va dirigido el comando[13], seleccionar el comando y presionar enviar. Inmediatamente el server arma un paquete con los datos seleccionados y se envía a la dirección IP del módulo asociado a ese usuario.
Como siempre puede ocurrir que la conexión no pueda establecerse en cuyo caso el indicador de conexión cambia a rojo. Si puede establecerse la conexión y el

[12] Ej.: encender o apagar una lamparita, aumentar un dimmer, apagar todas las unidades
[13] Puede que esta elección sea irrelevante ya que hay comandos dirigidos a todos los actuadores

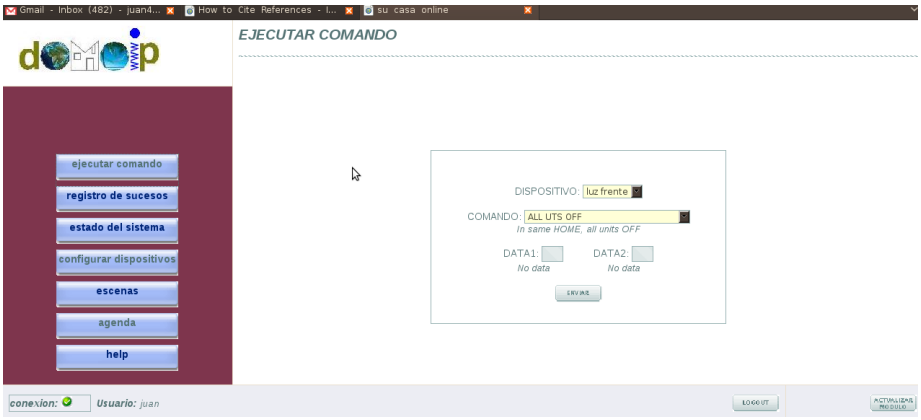


Figura 14.4: Página de ejecución de comando

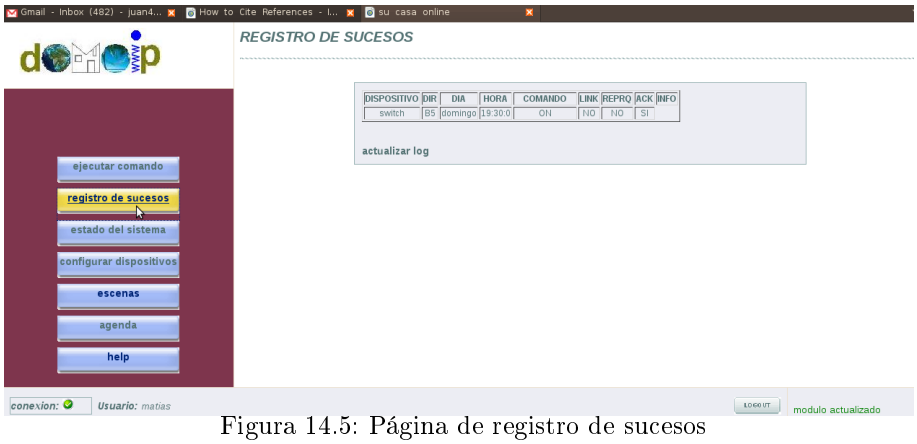


Figura 14.5: Página de registro de sucesos

paquete no se corrompe, el comando es procesado por el módulo y enviado al actuador. Inmediatamente el módulo envía una respuesta al servidor, indicando si el actuador pudo ejecutar el comando, o si no hubo respuesta del actuador. Esto se informa al usuario. Toda la secuencia de eventos demora menos de 2 segundos.

14.2.6. Consulta y actualización del registro de sucesos

En primer lugar por "registro de sucesos" debe entenderse una secuencia de comandos ordenados cronológicamente. Uno de los links laterales de la página le permite al usuario acceder a dicho registro(ver Figura 14.5). El mismo está pensado tanto para que el usuario pueda consultar que ha sucedido en su casa, como para que el administrador o supervisor tenga algún mecanismo que le permita conocer si las cosas están funcionando como se espera, y eventualmente poder corregir errores. El registro almacena datos como nombre del comando, hora a la que fue ejecutado, dia, actuador que lo ejecutó, si fue ejecutado de forma satisfactoria y otros datos. La cantidad de comandos que almacena esta registro son 16. Sabemos que es un número pequeño y está previsto que aumente. En principio fue elegido así dada la poca memoria con la que cuenta

Figura 14.6: Página de estado del sistema

nuestro microcontrolador[14].
En el registro se almacenan solamente aquellos comandos que se ejecutan por estar agendados[15] . Una vez que el módulo ejecuta un comando de este tipo, almacena información sobre el mismo en memoria flash de forma circular. Es decir, el tamaño del registro es fijo, y los comandos nuevos van reemplazando a los más viejos.
Cuando el usuario accede a la página **registro de sucesos**, se le presenta una tabla con los últimos 16 comandos[16] que contiene almacenados el servidor. Esta lista en general está desactualizada, por lo que el usuario tiene la opción de actualizar el registro. Al presionar actualizar "actualizar registro", se envía una solicitud de actualización de registro al módulo vía un socket TCP/IP. Al recibir dicha solicitud el módulo consulta el registro en memoria flash, arma un paquete con esta información y se la envía de regreso al servidor por el mismo socket[17]. Al recibir la información de actualización el servidor la almacena en la base de datos y se la muestra al usuario, avisando que el registro está actualizado. Todo el intercambio demora entre dos y tres segundos en condiciones de fundicionamiento normal.

14.2.7. Consulta y actualización del estado del sistema

Por "estado del sistema" nos referimos simplemente al estado en que se encuentran cada uno de los actuadores. Es decir si están encendidos, apagados y en caso que contengan algún valor regulable como el dimmer, cuanto vale el mismo. Al igual que con registro de sucesos, existe un link en la parte izquierda de la pantalla que le permite al usuario acceder al estado del sistema(ver Figura 14.6). Allí se listan los actuadores y al lado su estado.
Debemos observar que el estado de un actuador no necesariamente es la conse-

[14] El registro se almacena en memoria flash. Aún están disponibles algunas páginas de la misma (la flash está dividida en páginas) que pueden ser utilizadas para expandir el registro. Ver sección 7.2 *Memoria Flash*
[15] Recordar que también pueden enviarse comandos sin necesidad de agendarlos
[16] La imagen fue tomada en un caso en que el módulo se ha reiniciado hace poco, y el registro de sucesos aún no se ha completado.
[17] Ver algo de pablo

cuencia de un comando ejecutado por el módulo. Un actuador permite no solo activar un rele remotamente, sino que también cumple la función de una llave común de encendido (en el caso de un switch). Es decir una vez enviado en comando ON alguien dentro de la casa puede apagarlo de forma manual.
A diferencia del registro de sucesos, en el que la información está almacenada en el módulo, la información del estado del sistema está almacenada de forma distribuída en la red domótica, dentro de cada actuador. Más específicamente, cada actuador tiene un registro interno mediante el cual conoce su estado. Además todos los actuadores PLCbus tienen incorporada la funcionalidad de responder a solicitudes de status hechas por un dispositivo "encuestador"[18] conectado a la red domótica.
Al igual que en **registro de sucesos**, el usuario que consulta la página de **estado del sistema** tiene la posibilidad de actualizar la información, ya que muy probablemente la información contenida en el servidor web esté desactualizada.
Antes de analizar el intercambio debemos explicar algunos temas previos: a cada actuador de la red domótica debe configurarsele una dirección antes de conectarlo a la red eleéctrica. En simultáneo, debe agregarse dicho actuador en la página web, ya que este es el único lugar en donde queda almacenada dicha información, ademas de en el actuador mismo por supuesto. Es decir, el módulo no conoce las direcciones de los actuadores que están conectados, simplemente ejecuta comandos en función de datos que le llegan del servidor.
Una vez que el usuario presiona "actualizar estado", busca la lista de actuadores conectados en el hogar del usuario dentro de la base de datos, con estos arma un paquete, se lo envía al módulo y le avisa al usuario que el proceso puede tardar unos segundos. Luego de esto se cierra de inmediato la conexión, ya que se asume que el proceso de encuestar a cada uno de los actuadores lleva bastante tiempo[19]. Una vez que el módulo recibe una solicitud de actualización de estado, envía por la red eléctrica un comando específico de solicitud de estado a cada actuador según la lista que le llega del servidor, y estos envían su respuesta[20]. Con dichas respuestas arma nuevamente un paquete y le envía la respuesta al servidor. Una vez que el servidor la recibe, almacena la nueva lista en la base de datos y setea un registro de un bit en la base de datos, dando a conocer que la lista está actualizada. Cuando el usuario regresa a la página de "estado del sistema", se le notifica que la información está actualizada.

14.2.8. Agendar o quitar comandos

Además de poder ejecutar un comando de forma inmediata desde la páina **Ejecutar comando**, es posible agendar comandos en una agenda semanal, o quitarlos de la misma. A través del link **Agenda** se accede a la página para agendar comandos (ver Figura 14.7). Desde la misma luego de elegir un nombre, elegir el comando, el actuador el día y la hora, se presiona agendar y el comando pasa inmediatamente a la agenda.
También desde la agenda se pueden seleccionar los checkboxes de aquellos comandos que se desea borrar, y al presionar el botón borrar se eliminan de la agenda.
Una vez que se modifica la agenda, ya sea por agendar un comando, o borrar

[18] En nuestro caso el PLCbus 1141
[19] 10 segundos o más, dependiendo de cuantos actuadores estén conectados a la red
[20] Ver sección 10.4 *Módulo UART*

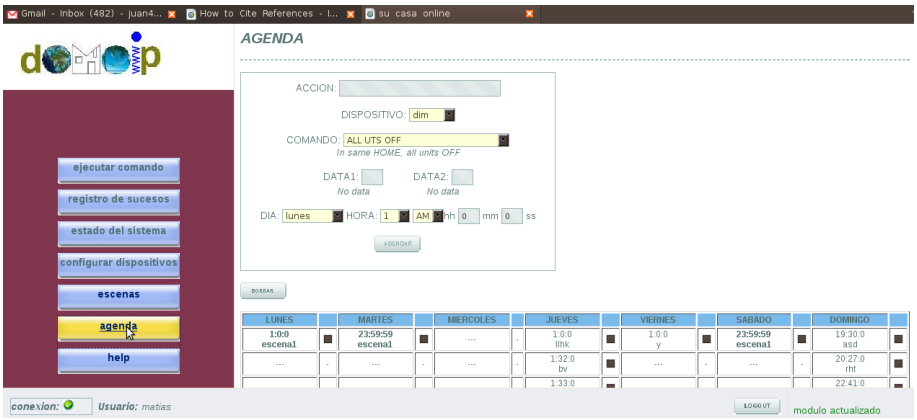


Figura 14.7: Página de agendado de comandos

uno, el módulo deja de estar actualizado. Es por esto, que luego de una modificación en la agenda, aparece en la esquina inferior derecha de la pantalla el botón **actualizar módulo**. Al presionarlo se le envían al módulo las actualizaciones de la agenda[21].

14.2.9. Crear editar, borrar y agendar escenas

En primer lugar aclaremos nuevamente que es una escena: una escena se identifica por un nombre, y contiene una serie de comandos cada uno con un offset. La utilidad de las mismas es tener uno o varios grupos (escenas) de comandos previamente definidos, y en lugar de agendar uno por uno, agendarlos todos al mismo tiempo, pero de una manera más sencilla. Un ejemplo de escena podría ser *Simular presencia*. En la misma podriamos definir que cada una hora prenda una luz distinta de la casa, y se apague el resto. El usuario del sistema, antes de salir del hogar, o desde su oficina, agenda *Simular presencia* a una hora determinada, y de allí en más las luces en su casa se encenderán o apagarán cada una hora.
Dicho lo anterior, se debe proveer una manera de consultar que escenas han sido creadas, de crear una nueva, de borrar escenas, editar las existentes y agendar. Desde uno de los links lateras se accede a la página de **Escenas** (ver Figura 14.8). En la parte inferior de la misma se observa la agenda, y en negrita las escenas que están agendadas en la misma. En la parte superior de esta página se puede elegir la escena, la hora y el día de la misma y agendar, tal como si fuera un comando. Inmediatamente la escena pasa a la agenda semanal. Desde dicha página se puede acceder a **Ver escenas** y a **Crear escenas**. En Ver escenas se muestran las escenas creadas y los comandos que las componen, con sus respectivos offsets (ver Figura 14.9), cada una en un bloque distinto. Además, se puede borrar o editar cada escena simplemente presionando el boton correspondiente, junto a la escena. La página de creación de escenas (ver Figura 14.10) permite crear una escena de manera sencilla. Primero se debe asignarle un nombre a la escena, y luego utilizar el cuadro 'agregar comando' para ir agregando comandos a la escena. Para crear la escena, el script correspondiente chequea que tenga al

[21] Ver sección 9.1 *Envio de tablas*

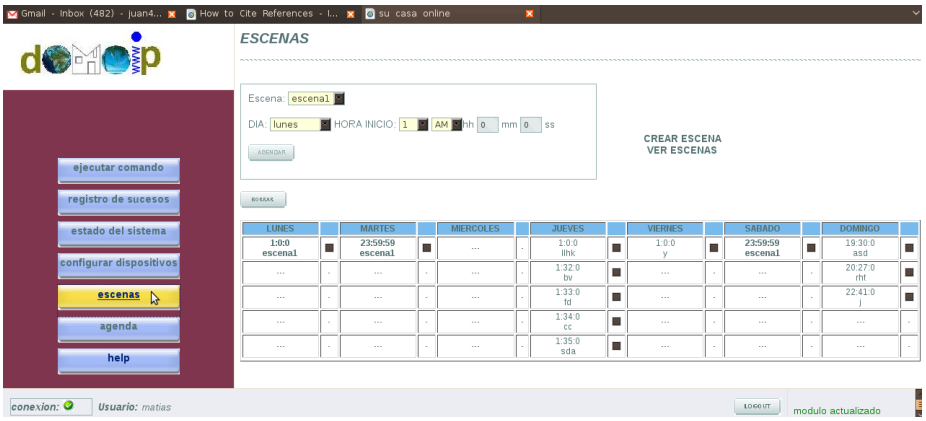


Figura 14.8: Página de agendado de escenas

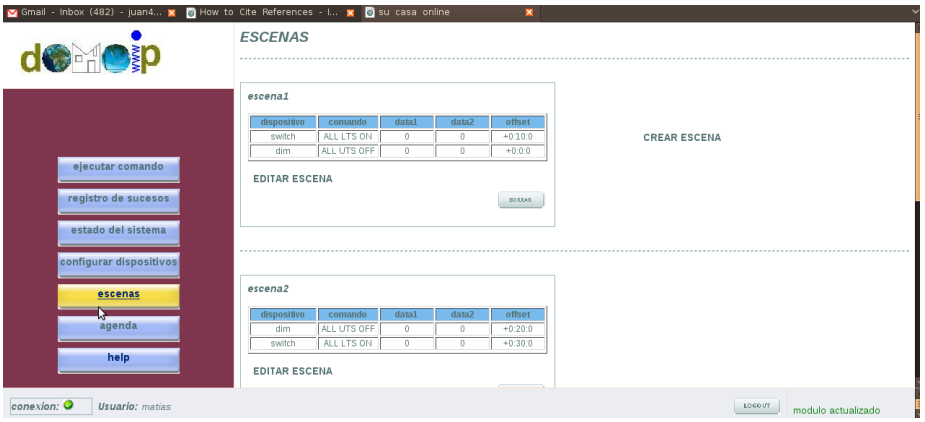


Figura 14.9: Página de consulta de escenas existentes

menos un comando, y que el nombre sea único. Una vez que la escena fue creada, se puede volver a la página principal de escenas, y agendarla. Desde la página **Ver escenas**, es posible acceder a editar una escena(ver Figura 14.11). Luego de presionar el link correspondiente a la edición de una determinada escena, se puede quitar o agregar comandos y cambiarle el nombre, de manera similar a la creación de una escena.

Figura 14.10: Página para crear una escena

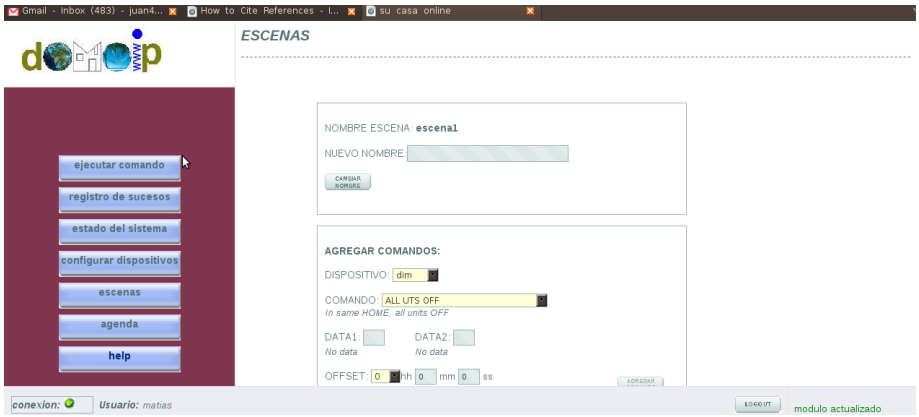


Figura 14.11: Página para editar escenas

14.3. Solución técnica de cada funcionalidad.

A continuación se expone cómo se utlilzaron las herramientas de programación web para resolver cada una de las funcionalidades del acceso web.

14.3.1. Gráficos

Para implementar la interfaz gráfica de cada una de las páginas que componen el acceso web, se utilizaron HTML, Javascript y CSS como se explicó en la sección 13.2 *Selección de los lenguajes.*

14.3.1.1. HTML

Para crear la estructura (layout) de cada una de las páginas que componen el acceso web, se utilizó HTML. El mismo es apropiado para lograr dicho objetivo, ya que si bien no permite crear páginas dinámicas o con un estilo moderno, es muy útil si es usado para definir el "esqueleto" de las páginas.

En primer lugar, la sección HEAD se utilizó para:

- Definir el título de la página[22]
- Definir funciones con Javascript, a ser utilizadas en la sección BODY
- Dar la ruta al archivo de estilo CSS

Se puede utilizar la sección HEAD para muchas cosas más: por ejemplo dar información sobre el autor de la página o dar la ruta a una imágen a ser mostrada en la pestaña del explorador[23].

La sección BODY como se dijo, define a grandes rasgos cómo va a ser la página. En primer lugar veamos cómo se usó HTML para dar la estructura global del acceso web, utilizando 4 frames. Las frames como ya se dijo son útiles ya que permiten que ciertas partes de la pantalla permanezcan estáticas, mientras que otras cambia a la vez que el usuario navega. Cada frame muestra un archivo distinto HTML y en nuestro caso usamos 4 frames como se observa en la Figura 14.12. La frame 1 es donde se muestran las páginas a las que se accede a través de los links laterales, ubicados en la frame 4. Las frames 2 y 3 forman la barra de información, en donde se muestra el estado de la conexión, el usuario, se ubica el boton para el logout y en la esquina derecha se muestra o bien *módulo actualizado* o bien se despliega un botón para actualizar el mismo. La idea de separar la barra inferior en dos frames, surge de que estas modifican su contenido de forma muy distinta: en función de si el módulo está o no actualizado la frame 3, y en función de si hubo o no problemas en la conexión con el módulo la frame 2. La barra inferior podría haberse implementado con una sola frame pero es más eficiente de la manera que se lo hizo.

Todas las páginas del acceso web contienen al menos un contenedor, definido entre los tags ‹DIV›y ‹/DIV›. Simplemente se completa la sección entre estos con algún contenido, y luego puede darse una posición y un estilo particular

[22] Este aparece en la pestaña del browser.

[23] Por ejemplo la carta en blanco y rojo que muestra `http://www.gmail.com`

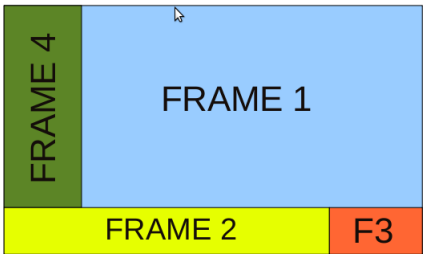


Figura 14.12: Frames utilizadas para la página web.

Figura 14.13: Ejemplo de contenedor

a ese contenedor. Veamos un ejemplo de un contenedor que define el área en donde se muestra el estado de los dispositivos en la página de **Estado del sistema** (ver Figura 14.13). Lo que se hace es definir las dimensiones que tendrá el contenedor, definir el borde y el color de fondo del mismo[24]. El contenedor que se muestra, contiene una tabla con la que se presenta la lista de los dispositivos instalados y su estado, y un link para actualizar la información[25].

También se utilizaron los tags TABLE, HR, H2, BR y P para definir tablas, líneas, headers, salto de línea y párrafo respectivamente.
En cuanto a las entradas de datos[26], todas están contenidas dentro de un FORM, mediante el cual se especifica que script toma esos datos para el posterior procesamiento de los mismos, de la siguiente forma:

```
<FORM name=”form0” action=”<?=$_SERVER['PHP_SELF']?>” method=”post”>
<INPUT name=”mto” type=”text” value=”0”>MINUTOS
<INPUT type=”submit” value=”agendar”/>
</FORM>
```

Un segmento de código HTML similar al anterior, se encuentra en todos los archivos que contienen algo de la interfaz gráfica del acceso web.
Con el atributo *action* del FORM, se especifica para este caso, que es el propio archivo PHP el que tratará a los datos. Dentro del FORM se incluye una entrada de texto y un botón de submit. Al presionar el botón los datos son enviados vía el método POST del protocolo HTTP[42][43]. El hecho de que sea el mismo archivo PHP (en el cual está embebido HTML), el que procesa los datos es algo bastante particular a la vez que eficiente. De esa manera se logra que la página que se muestra sufra pocas o ninguna modificación una vez que el usuario le solicita algo. La solución de redirigir el procesamiento a un script almacenado en otro archivo, representa generalmente que el usuario pierda tiempo esperando que se carguen nuevas páginas, mientras que si en un mismo archivo se tiene procesamiento y HTML (PHP y HTML), todo se torna un poco más rápido.

14.3.1.2. Javascript

Si bien ya se hizo una pequeña introducción a Javascript[33][34] en la subsección 13.3.2 *Javascript*, recordemos que Javascript es un lenguaje de programación interpretado, es decir que no requiere compilación, y es orientado a objetos al igual que java. Es utilizado fundamentalmente para agregar dinamismo a las páginas web, haciendo que la apariencia de estas cambie a medida que el usuario navega. Hay que tener presente que con HTML, Javascript y PHP se pueden hacer cosas completamente distintas, y sin duda complementarias. Digamos que con esos tres lenguajes más CSS se puede hacer programación web logrando interfaces gráficas muy atractivas, a la vez que potentes desde el punto de vista del procesamiento que se lleva a cabo en el servidor.
Javascript tiene dos grandes aciertos. En primer lugar permite conocer datos sobre la máquina cliente de una manera muy sencilla, ya que muchas veces es-

[24] Utilizando CSS.
[25] Ver subsección 14.2.7 *Consulta y actualización del estado del sistema*
[26] ya sea a través de texto, seleccionar un checkbox o presionar un botón

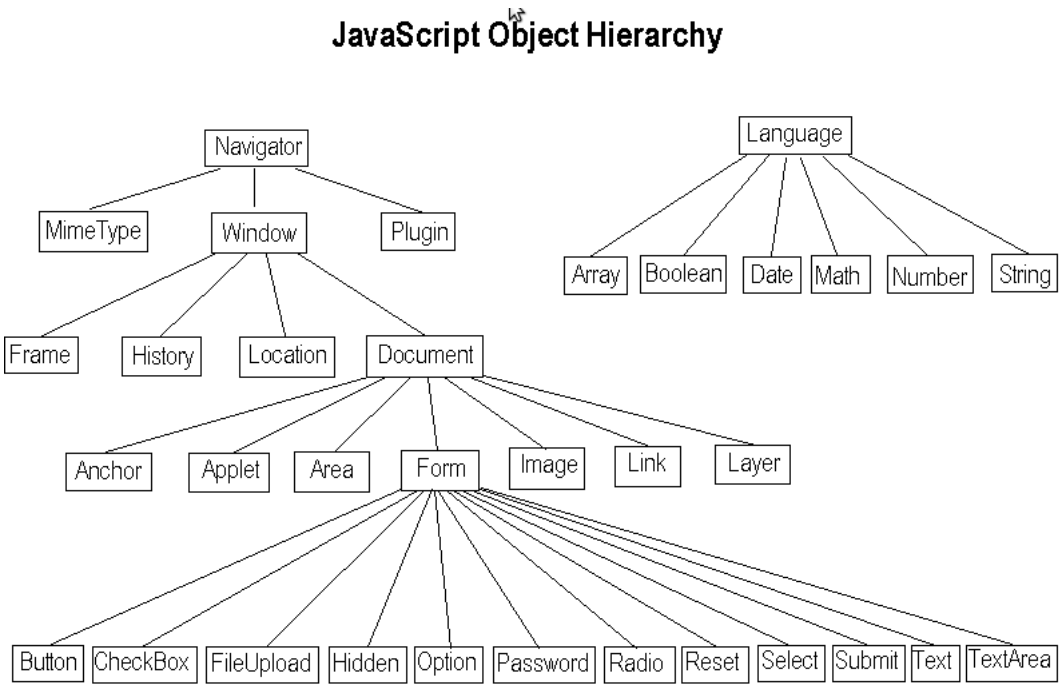


Figura 14.14: Jerarquía Javascript

tos son variables predefinidas; y en segundo lugar la jerarquía de objetos[27] le otorga mucho potencial.
Otro punto importante de Javascript son los eventos. Estos definen varias formas a través de las cuales se puede hacer llamadas a funciones Javascript. Por ejemplo uno querría que se llame a una función a partir de cierto evento como por ejemplo:

- Click sobre algo
- En el momento que se carga una imágen o una página
- Si el ratón esta parado sobre algo en particular
- Al seleccionar una entrada de datos
- Al enviar un formulario
- Al pulsar una tecla

Veamos el siguiente ejemplo que si bien es muy sencillo ilustra lo que queremos decir:

```
<BODY onload="alert('Se cargo la página')">
```

Alert es una función predefinida de Javascript, que simplemente despliega el

[27] Ver Figura 14.14

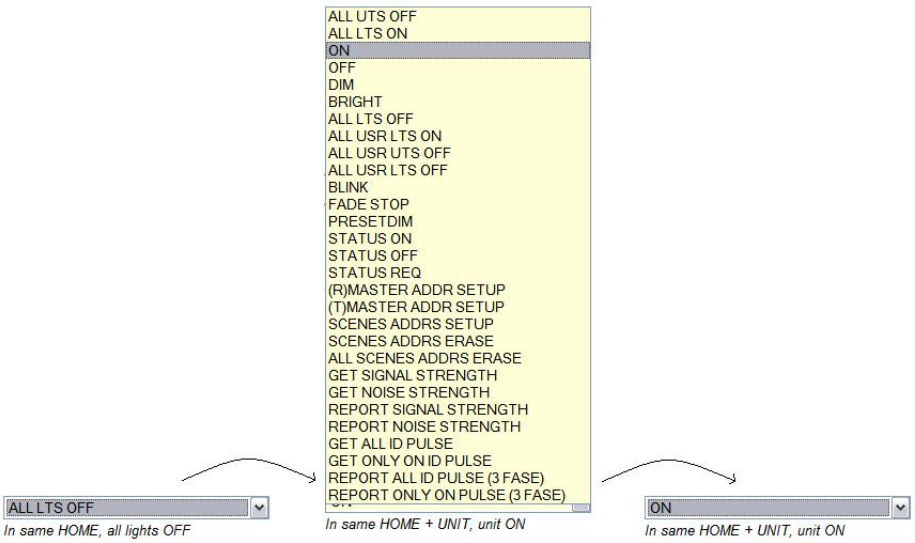


Figura 14.15: Efecto logrado con Javascript

mensaje que se le pasa como parámetro. Con el código anterior, lo que logramos es que cuando se carga la página se despliegue el mensaje "Se cargó la página". También están los eventos *onunload, onclick, onsubmit* y muchos más, que pueden ser llamados desde la declaración de un formulario, la definición de una imágen, etc..

En nuestro caso particular, utilizamos Javascript en casi todas las páginas para:

1. Desplegar mensaje debajo del selector de comando, y debajo de las entradas de datos(ver Figura 14.7)
2. Recargar una página
3. Romper frames de forma de mostrar una página con una sola frame en lugar de varias
4. Cargar una frame con una página nueva

Hay muchos editores de programación web que incorporan una librería con cientos o miles de scripts Javascript. El script que nos permitió lograr el item 1 fue tomado de la librería del editor *Eversoft First Page 2006*, y adaptado a nuestras necesidades. De esta forma logramos que a medida que el usuario selecciona un comando, ya sea para agendarlo o para enviarlo enviarlo de inmediato, se despliegue debajo del selector una breve descripción del comando. Puede observarse este efecto en la Figura 14.15.
El código de dicha función es muy sencillo:

```
function displaydesc(which, descriptionarray1, container1){

if (document.getElementById){
document.getElementById(container1).innerHTML=descriptionarray1[which.selectedIndex]
}

}
```

En primer lugar se define la función dentro del HEAD de HTML y luego se la llama utilizando el evento *onselect* desde una entrada[28] que muestra los diferentes comandos, con los parámetros correspondientes: nombre de la entrada[29], array con los mensajes a desplegar e identificador del contenedor respectivamente.

Otra función para la cual Javascript nos resultó útil fue para recargar una página. Esto también puede hacerse con HTML, pero Javascript lo prevee de una manera más simple. El problema era que queríamos en todos los casos recargar uno de los frames, ya sea el 1, 2 o el 3[30].Con el siguiente comando, se recarga[31] el contenido del frame bajo el nombre *update*[32]:

```
parent.frames["update"].location.reload()
```

Algo que complicó bastante más de lo que se pudiera pensar a priori, fue poder "romper los frames" en el sentido de que en lugar de seguir navegando en una página con 4 frames como la nuestra, a partir de cierta acción volver a tener una página con 1 solo frame. Esto era deseable ya que una vez que el usuario quería deslogearse, presionando el botón *logout* en la barra inferior, se quería volver a la página inicial de login, que tiene un solo frame. Ni bien nos encontramos con este pequeño problema se trató de buscar una solucón con HTML y al no encontrarse se pasó a Javascript. Finalmente la solución es un sencillo comando, que se desencadena con el evento *onsubmit*[33]:

```
parent.location.href = "/login/logout.php"
```

Con el comando anterior se carga la página *logout.php*, eliminando las frames existentes. En este tipo de cosas es que Javascript brinda su mayor potencial, por la simpleza en la utilización.
De manera similar se logra cargar el contenido de un frame con un archivo cualquiera:

```
parent.frames["update"].location.href="actualizar.php";
```

Con el comando anterior se logra cargar el contenido del frame *update* con el archivo "actualizar.php".

[28] Utilizamos para ello la entrada SELECT de HTML, que despliega una lista de opciones
[29] Ej.: document.form0.comandos. Observese la jerarquía de objetos.
[30] Ver Figura 14.12
[31] Si se presiona F5 se recarga el contenido de todos los frames.
[32] Cada frame está identificada por un nombre usando el atributo *name* de HTMl
[33] Este evento se desencadena al presionar un botón de submit para enviar un formulario.

14.3.1.3. CSS

Para completar con el diseño gráfico del acceso web se incorpora CSS.
Todas las páginas que componen este proyecto incorporan un archivo de estilo, en el que se indican entre otras cosas:

- Tamaño, tipo y color de letra de la sección BODY
- Tamaño, tipo y color de letra de las tablas
- Tipo y color de letra de los headers
- Color de fondo y ancho del borde de las entradas de texto (INPUT)
- Imágen y dimensiones de cada uno de los botones de entrada
- Dimensiones, color de fondo y borde de varios contenedores
- Color y tipo de letra de los párrafos[34]
- Color, tipo y espesor de las líneas[35]
- Color, tipo de letra e imágen de algunos links[36]

Veamos por ejemplo cómo se especifican los parámetros de la sección BODY en el archivo CSS utilizado:

```
body{font:0.85em Arial, helvetica, sens-serif;color: # 567475;}
```

Arriba se especifican el tamaño de la letra, la fuente y el color.
Veamos por último como se especifica una entrada tipo "button"[37] de forma que en lugar de mostrar un botón estándar, aparezca una imágen que oficia de botón, y que además al ubicar el ratón sobre ella, esta imágen cambie:

```
.agregar-disp{width: 65px;height: 24px;background:url(/login/images/agregar.gif)
no-repeat;outline: none;border:0;}

.agregar-disp:hover{background: url(/login/images/agregar.gif) no-repeat
0 -24px;display:inline}
```

El texto anterior se aplica a una entrada que tenga como identificador la palabra *agregar-disp*. En el comienzo de la primer línea están indicadas las dimensiones del botón y luego se especifica cual será la imágen de fondo entre otras cosas.
Con la segunda línea lo que se logra es que al poner el ratón sobre el botón[38], la imágen cambie. Se observa allí que se mostrará la misma imágen pero 24 pixeles más abajo. Lo único que resta es almacenar una imagen con las dimensiones adecuadas (65 x 48 pixeles) en la ruta indicada.
Veamos ahora un ejemplo concreto. A partir de la imágen de la Figura 14.16 se genera un botón que cambia cuando el ratón está sobre él (Ver Figura 14.17).

Observese la Figura 14.18 para ver el mismo botón pero sin el archivo de estilo.

[34] Definido por los tags ‹P ›
[35] Definido por los tags ‹HR ›
[36] Definido por los tags ‹A ›
[37] Las entradas(INPUT) de HTML aceptan varios tipos: "text", "button" entre otros.
[38] Atributo "hover"

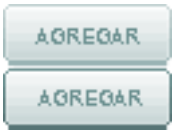


Figura 14.16: Imagen a partir de la cual se genera el botón para agregar actuador

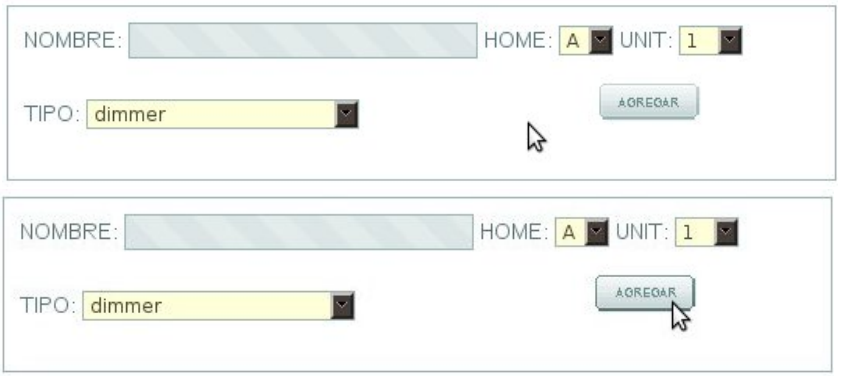


Figura 14.17: Botón de submit generado con un archivo de estilo

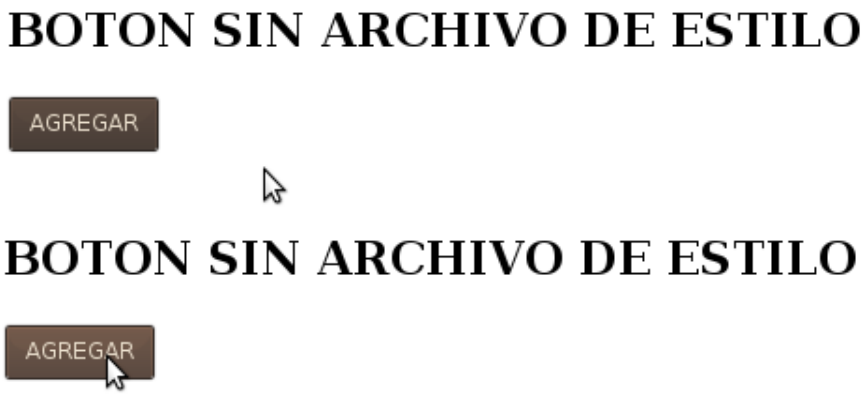


Figura 14.18: Botón de submit sin el archivo de estilo

14.3.2. Procesamiento de datos

Para todo lo que tiene que ver con procesamiento de datos y uso de la base de datos[39] se utilizó *PHP*.
PHP es un lenguaje muy potente, interpretado al igual que los que hemos visto hasta ahora, pero a diferencia de Javascript y HTML se interpreta en el servidor y no en la máquina cliente. Es por esto que es el candidato cuando tenemos que almacenar datos no volátiles[40] o establecer conexiones con otros servidores, o en nuestro caso con el propio módulo. PHP tiene varios años de desarrollo, y ha alcanzado un punto en el cual su potencial es enorme. Es muy difícil querer hacer algo del lado del servidor que no esté previsto en PHP, desde las operaciones aritméticas más sencillas hasta obtener la hora a la que amanece en una cierta región del planeta. No obstante tiene ciertas carencias que lo pueden hacer incompatible con scripts programados en C[41], dada la inexistencia de los tipos de variables unsigned y signed, comunes en el lenguaje C, y la limitación de los *bitwise operators*[42]. El código de un script php está contenido entre los caracteres ‹?php y ?›.
Todas los archivos que componen el acceso web del proyecto tienen extensión .php. Esto es así dado que es la única forma de protegerlos de accesos de usuarios no permitidos. Si bien hasta ahora hablamos de archivos HTML, en realidad son todos archivos PHP, que tienen HTML embebido.
A continuación describiremos las funcionalidades que implementamos con PHP, comenzando por las más sencillas.

14.3.2.1. Redirigir

Una funcionalidad importante es la de redirigir. Esta se utiliza en el login y en otras páginas, y se redirecciona a una u otra dependiendo de si el usuario ingresó un nombre y contraseña válidos o no, o si es administrador en el caso del login.
Esto puede resolverse con HTML y también con PHP. Ya que la mayoría del código de nuestros archivos es PHP, se optó por este para implementar la funcionalidad. Redirigir no es otra cosa que modificar el valor del header *Location*[43] del protocolo HTTP. Por esto, si se hace:

```
header('Location:/un-archivo-php.php');
```

a partir de esa línea el usuario verá el contenido del archivo *un-archivo-php.php*.

[39] Ver capítulo 15 *Almacenamiento de datos en el servidor*
[40] Básicamente todos los que no son cookies
[41] Antes comunes por la utilización de scripts CGI, que podían ser programados en C, Perl y otros lenguajes.
[42] Operadores a nivel de bits, por ejemplo rotate left.
[43] El protocolo HTTP o Hypertext Transfer Language es un protocolo definido para transferir datos entre una máquina cliente y una servidor. Actualmente la versión usada es la 1.1. Este define varios headers, que cambian dependiendo de si la máquina que arma el paquete es host o client. Los headers brindan o solicitan información a la máquina destinó de la transferencia. Otro header usado en la comunicación entre el servidor y el módulo iniciada por el módulo, es el *Host*. Con este se le indica a la máquina "Host", es decir a la que brinda información, a qué URL está dirigida la solicitud. Dicho header hace posible la existencia de servidores multidominio, como es usual. Sin el mismo, un servidor podría alojar solo a una página web, haciendo los servicios de hosting inviables. El header *Host* es uno de los pocos headers de uso obligatorio en la versión 1.1 de HTTP. Generalmente los headers son opcionales.

14.3.2.2. Funciones pack y unpack

Otra necesidad es la de intercambiar bytes entre el módulo y el servidor. Para esto es necesario contar con una forma de convertir los datos binarios crudos provenientes del módulo, a la representación que usa PHP.
Durante el transcurso del proyecto, siempre se tuvo la filosofía de transladar la mayor cantidad de procesos al servidor, de forma de alivianar el ya exigido procesador ATmega 32 utilizado en el módulo. Es por esto que si bien antes de enviar todo tipo de datos al servidor, el módulo podría modificar los datos para que sean entendibles por el servidor sin más procesamiento, preferimos que el módulo envíe los datos crudos directamente al servidor, y este los procese de forma de poder operar con ellos. A esos efectos se usaron las funciones *pack* y *unpack* de PHP. Estas convierten datos codificados PHP a binario y el proceso inverso. Es decir si tenemos un número almacenado en una variable dentro de un script PHP, luego de pasarlo por la función *pack* nos devolverá un byte con el valor de dicho número en representación binaria. También es posible pedirle a la función que devuelva dos o cuatro bytes. A la inversa, cuando se obtienen datos del módulo, se usa la funciín *unpack* que recibe el valor binario y devuelve la representación de ese número, según la codificación que usa PHP.

14.3.2.3. Conversión offset a horas, minutos y segundos

Otra sencillez necesaria de resolver es la conversión de la representación del tiempo que maneja el módulo, a la representación del tiempo que maneja el servidor. Esto es así ya que el módulo utiliza como su hora interna los segundos que transcurren desde la hora 0. El servidor utiliza el formato usual en horas, minutos y segundos, con hora AM o PM. La funciones que convierten en un sentido y el otro se implementan con PHP y están formadas por estructuras de control convencionales[44].

14.3.2.4. Manejo de strings

Lo que motivó el buscar formas de manejar cómodamente strings[45], fue que todos los datos que se reciben del módulo, así como muchos otros se obtienen en forma de string.
Las necesidades fundamentales son:

- conocer su largo
- obtener strings más pequeños a partir de uno
- buscar un string o un caracter dentro de otro string de mayor tamaño

Para conocer su largo simplemente se utiliza la función *strlen()* que recibe como parámetro el string.
Por otra parte un string puede verse como un array de caracteres, por lo que puede recorrerse con un índice. Esto nos permitió buscar un cierto caracter o string dentro de uno más grande. Esto fue motivado dado que en ciertos intercambios[46] el módulo envía al servidor el string 'OK', junto con otros datos

[44] FOR, IF, WHILE
[45] Un string no es más que un array de caracteres, con índice entero.
[46] Ver capítulo 9 *Comunicación entre webserver y módulo*

como forma de respuesta, para avisar al servidor que los datos fueron recibidos correctamente.
También, como el módulo envía en ciertos tipos de respuesta varios datos de largo fijo, contenidos todos en el mismo string, es necesario partir este para poder manejar cada dato de forma independiente o sea, partir el string en otros más pequeños. Esto se logra con la función *substr()*.

14.3.2.5. Desplegar mensajes

Otra problema que encontramos fue el de desplegar mensajes de información o error. Cuando se inicia una comunicación con el módulo que se sabe tardará más de cierto tiempo, se avisa al usuario que se inició la comunicación y que por favor espere. Esto es muy sencillo ya que es simplemente usar la función *echo()* que imprime en pantalla el parámetro que se le pasa. El caso que es poco más complicado el de los mensajes de error. Para iniciar una comunicación con el módulo se utiliza la función *pfsockopen()* que abre un socket persistente a la IP y puerto TCP que se le pasan como parámetros. El problema es que cuando esta función no logra abrir el socket, por defecto imprime un error en pantalla, y esto era inaceptable ya que no teníamos forma en principio, de escribir nosotros ese mensaje. Observando otros scripts vimos que la forma de anular dicho mensaje de error interno, de forma que no imprima nada luego de que se produce un error, era anteceder a la función con el símbolo '@'. Además para desplegar un mensaje de error definido por nosotros puede usarse la siguiente estructura:

```
@pfsockopen(...) or die('Ha ocurrido un error al intentar abrir el
socket');
```

De esta forma si se puede abrir el socket se continúa normalmente con la ejecución del script, y en caso contrario se fuerza la finalización del script desplegando el mensaje *Ha ocurrido un error al intentar abrir el socket.*
Además usamos dos estilos para mostrar mensajes de información o error, definidos en el archivo CSS. Tanto en la sentencia *echo* como en *die* usadas para desplegar los mensajes, en realidad no se pasa el mensaje directamente, sino que se lo hace a través de los tags ‹P id='msg'›o ‹P id='error'›de HTML, que permiten seleccionar el estilo del mensaje cambiando el atributo id.

14.3.2.6. Mostrar tablas

Dado que el acceso web cuenta con una agenda semanal, fue necesario desde el comienzo mostrar una tabla con las columnas lunes a domingo, que contenga los comandos agendados. Para esto se usa TABLE de HTML, que funciona primero definiendo entre los tags ‹TD›y ‹/TD›las columnas de la tabla, y luego cada nueva declaración con los tags ‹TR›y ‹/TR›define una nueva fila. Al principio dado que no se había incorporado MySql, se usaba una "base de datos" casera y muy rudimentaria usando archivos de texto plano. De esa manera, y dada la rigidez de TABLE de HTML, el algoritmo para formar la tabla de manera que se observen los comandos ordenados en horas crecientes, era muy complejo. Esto se simplificó muchísimo cuando se pasó a PHP, dado que el MySql[47] permite ordenar los datos de manera rápida y extremadamente fácil. Por esto

[47] Ver sección 15.2 *PHP y MySQL*

mostrar una tabla pasó a ser simplemente un while que considera la cantidad total de comandos agendados, que contiene a un for de 1 a 7 (por los días de la semana). Como el algoritmo se escribe en PHP, se usa la función *echo* para imprimir en pantalla el código HTML, de manera que al ser interpretado en la máquina cliente, se arme la tabla.
Algo necesario para lograr ésto es intercalar texto con strings almacenados en variables. Eso se logra haciendo lo siguiente:

```
echo(<td><center>{$row['hora']}</center></td>);
```

con lo cual se intercala la variable *$row['hora']* con código HTML. La sentencia anterior produce una nueva celda en la tabla.

14.3.2.7. Obtener la dirección IP del módulo

Algo que parecía un problema grande al comienzo era el hecho de que el servidor pudiera contar con la IP del módulo actualizada en todo momento. Esto es absolutamente necesario porque la mayoría de los contratos de ADSL suponen IP dinámica. Del lado del módulo se pensó instalar un servicio de DynDNS[44][48], ya sea en el propio módulo o en el router, de forma de que la actualización de la IP se haga "automaticamente". Se vió que esto no era posible, ya que este servicio estaba pensado para PCs y algunos routers particulares. Por eso se pasó a una solución casera, que consistió en que el módulo envíe periódicamente al servidor mensajes de actualización de IP a una URL[45] particular. El módulo no hace otra cosa que un GET HTTP a dicha URL.
Queda entonces ver cómo hace el servidor para obtener dicha IP. La solución pasa por las variables predefinidas de PHP: estas son variables disponibles en general para obtener información sobre el servidor, que brindan datos sobre conexiones del servidor, métodos GET y POST de HTTP, archivos que se han intercambiado, variables de sessión, cookies y muchos datos más. Cada variable en realidad es un array[49], con varios índices, cada uno apuntando a un dato particular de esa categoría. La variable predefinida $_SERVER, contiene información sobre el nombre del archivo que está corriendo actualmente, el nombre del servidor bajo el cual el archivo actual se está ejecutando, la versión del protocolo HTTP bajo la cual se ha solicitado el recurso actual, el puerto remoto, la IP remota entre otros muchos datos más. Si desde el módulo se hace un GET de la página *get_ip.php*, incluyendo en este archivo la línea

```
$ip = $_SERVER['REMOTE_ADDR'];
```

se obtiene la IP del módulo almacenada en la variable *$ip*. Obviamente luego debe almacenarse en la base de datos.
De esta forma se completa la solucón para tener actualizada la IP del módulo. Solo resta que el este envíe la actualización cada intervalos lo suficientemente

[48] DynDNS permite registrar un usuario DNS, de manera que si se quiere establecer una conexión con una máquina que cuenta con dicho registro, simplmente se hace la conexión a ese nombre y es el servicio DynDNS quien se encarga de mantener actualizada la IP en algún servidor de DNS.

[49] Un array en PHP puede tener un índice de cualquier tipo, es decir puede ser numérico o no. En general se usan como índices enteros o strings.

chicos como para que la probabilidad de disponer de la IP correcta cada vez que quiere iniciarse un intercambio, sea muy alta. Se vió que enviando dicha actualización cada 5 minutos era suficiente para tener un error aparentemente bajo, el cual obviamente depende de la frecuencia de uso.

14.3.2.8. Login

Otro problema que resuelve PHP es el del login. Para implementar el mismo se utilizan varios archivos que se tomaron de la versión 1.1 del login disponible en roscripts[41][50], y se adaptaron a nuestras necesidades. Ya se explicaron que funcionalidades provee dicho login en subsección 14.2.1 *Login, usuario normal y administrador del sistema*. Ahora vamos a explicar a muy grandes rasgos cómo se resuelven los problemas básicos que trae esta funcionalidad.
En primer lugar para poder contar con un login se necesita una tabla[51] específica en la base de datos, *user* en este caso. Los datos que deben almacenarse son al menos:

- Un identificador
- Un nombre
- Una contraseña
- Un email
- Un nivel de acceso
- Una bandera que indica si la cuenta está activa

La tabla users contiene otros datos y se verán en detalle más adelante.
Cuando se registra un usuario deben especificarse cada uno de estos datos, junto con otros no indicados. Luego se le envía al usuario una confirmación vía email, usando la función *mail()* de PHP, y después que el usuario presiona el link que le llega a su casilla, la cuenta está activa y lista para usarse. Esto se hace así para asegurarse que el mail es válido y que el usuario tiene acceso a esa cuenta de mail.
Cuando un usuario ingresa su Username y Password en la página de login de `http://www.radiocostadeoro.com` y presiona INGRESAR, se inicia una búsqueda de un registro asociado a ese nombre en la tabla users. Naturalmente si se encuentra que existe dicho usuario y que además la contraseña ingresada corresponde a ese usuario, entonces o bien se direcciona a la página de inicio (si es un usuario "común"), o bien se direcciona a la página de administración de cuentas, dependiendo del campo *nivel de acceso*.
Si no existe ese usuario en la base de datos, o si el nombre de usuario y la contrasenña no coinciden, se muestra un mensaje de error.
En el momento en que se verifica que coinciden el usuario y la contraseña, se guarda el identificador del usuario en la variable predefinida $_SESSION['user_id'],

[50] *roscripts* brinda libremente muchos scripts en más de 20 lenguajes de programación web diferentes, bien documentados y diseñados de forma de poder adaptarlos fácilmente a una página web en particular.
[51] Ver capítulo 15 *Almacenamiento de datos en el servidor*. Una base de datos está dividida en tablas. Cada tabla es una matriz. Para crear una tabla nueva debe indicarse el nombre y tipo de dato de cada columna, así como el nombre de la tabla

y se guarda TRUE en la variable $_SESSION['logged_in'] que por defecto es FALSE; previstas a estos efectos, de forma de poder manejar sessiones. Luego de esto la sesión esta creada y se puede acceder a las páginas que componen el acceso web.
Otro problema es cómo proteger dichas páginas, es decir todas las que no forman parte del login. La forma de hacerlo es justamente chequear al inicio de todos los scripts si $_SESSION['logged_in'] es TRUE[52]. Para esto es necesario una función auxiliar de PHP, *session_start()*, que "captura" las variables de session generadas anteriormente. De no llamarse esta función previo a chequear el valor de $_SESSION['logged_in'], el resultado sería que nunca se podría acceder a ningún contenido. Veamos entonces que es lo que se incluye en cada archivo, de forma que quede protegido frente a accesos de personas no habilitadas:

```
<?php
session_start();
if ($_SESSION ['logged_in'] ):
?>
Aquí va lo que se desea proteger
<?php
endif;
?>
```

De esa sencilla manera el contenido de la página queda protegido, en el sentido que si el usuario no está logeado no corre el script ni tampoco se muestra ningún contenido salvo un mensaje de error.

14.3.2.9. Conexión con el módulo

Uno de los problemas principales con los que nos enfrentamos era cómo se iba a instrumentar la comunicación entre el módulo y el servidor. Digamos que hasta este punto la comunicación entre el servidor, que hace de intermediario[53], y el usuario ya estaba resuelta mediante la interfaz gráfica de la página web, pero resta resolver con que mecanismos se va a comunicar el módulo y el servidor[54].

Veamos cada una de las vías de la comunicación:

1. **Vía del módulo al servidor**

   El módulo tiene un único mecanismo para iniciar una comunicación con el servidor: utilizar el método GET de HTTP y el puerto 80 tal como si fuera un navegador web cualquiera que solicita cierto recurso a un web server.
   El módulo puede o bien hacer un GET a una cierta página con el objeto por ejemplo de actualizar la IP, caso que ya explicamos en uno de los puntos anteriores, o bien hacer un GET a una cierta URL con el objeto

[52] Es por esto que es necesario que todas las páginas del acceso sean PHP, o al menos tengan extensión .php
[53] Ver Figura 14.19
[54] Esto se explica con más detalle en capítulo 9 *Comunicación entre webserver y módulo*. En esta sección se verán detalles de la implementación en PHP

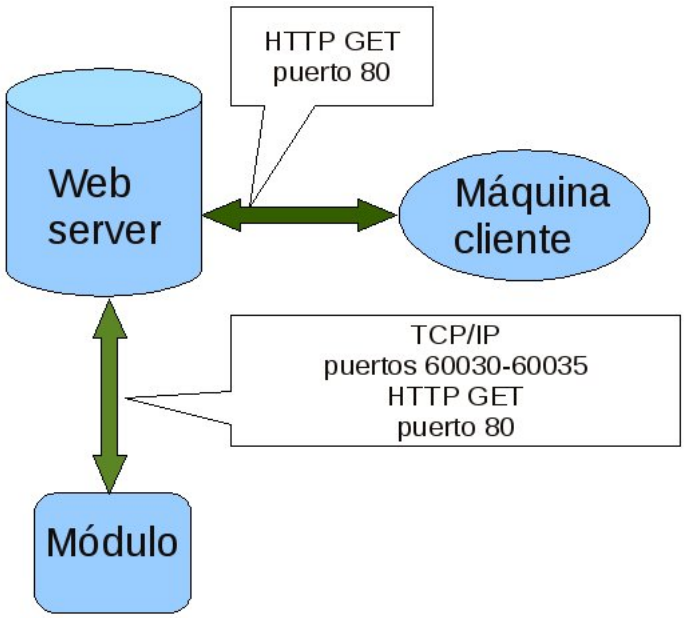


Figura 14.19: Comunicación entre del servidor y el módulo y el servidor y el cliente

de además de iniciar una comunicación, pasar información en la propia consulta.
Por ejemplo cuando un usuario realiza una solicitud de actualización de status desde la página web, el servidor abre un socket al módulo y le solicita esta información. A continuación el socket se cierra. Luego de que el módulo obtiene todos los datos del estado del sistema, consultando a cada uno de los actuadores conectados a la red domótica, tiene que enviar estos al servidor. Para ello arma un paquete HTTP GET, con los datos como información almacenada en variables del método GET. Veamos cómo se hace esto:

```
http://www.example.com?var1=46&var2=78
```

De esa forma, se envían en la propia consulta las variables var1 y var2 con los valores 46 y 78 respectivamente. De esta forma, el módulo arma el paquete HTTP GET asociando el nombre de cada variable con la dirección del actuador.
En el script al cual está dirigido el HTTP GET, lo que se hace es una vez más utilizar las variables predefinidas de PHP. En este caso el array $_GET. Para almacenar en una variable del script por ejemplo la variable *var1* del ejemplo anterior lo que se hace es:

```
$var = $_GET['var1'];
```

y de esa manera ya se capturó el dato enviado por el módulo. Claramente de antemano debe acordarse el nombre de variables que se usarán

en cada transferencia, o cómo se construirán estos nombres.

2. **Vía del servidor al módulo**

El servidor también dispone de una única manera de iniciar una comunicación con el módulo, y es abriendo un socket TCP/IP. Esto es requerido una vez que el usuario solicita actualizar la agenda del módulo, pedir el status, pedir el log o enviar un comando instantáneo.
Para los intercambios de datos vía socket se usan varias funciones de PHP. En primer lugar cabe observar que existen al menos dos tipos de sockets: persistentes y no persistentes. Estos difieren en cuando se cierra el socket. Los sockets no persistentes cierran la conexión luego de que una de los extremos envío el primer ACK correspondiente al primer envío de datos. Los sockets persistentes, soportan el intercambio de datos de ambas vías tantas veces como se quiera hasta que haya un timeout o hasta que una de las vías inicie el cierre de conexión. Inicialmente no conocíamos este detalle, y eso tornaba el intercambio muy ineficiente, ya que en la mayoría de las veces era deseable mantener el socket abierto.
Las funciones que abren un socket TCP/IP son *fsockopen()* y *pfsockopen()*, la primera de ellas abre sockets no persistentes. Lo único que debe pasarse como parámetro es la dirección IP y el puerto destinó, y si se desea el tiempo de timeout. Devuelven 0 en el caso de que la conexión no se pueda establecer, o un *handler* de la conexión, que debe ser usado posteriormente para enviar y leer datos o cerrar la conexión.
Una vez abierto el socket se usa la función *fwrite()* para enviar datos. A esta función se le pasa el handler de la conexión y el string que desea enviarse, y envía un stream conteniendo dicho string. Para leer los datos se utiliza de forma similar la función *fread()*. Luego para terminar se llama a *fclose()* que cierra la conexión que se correponde con el handler que se le pasa. Veamos un ejemplo de esto:

```
//abro el socket con 30 segundos de timeout
$handler = @pfsockopen(189.53.32.12, 60031, 30);
//envío los datos por el socket
fwrite($handler,'DATOS QUE ENVIO');
$rsp = "";
//leo la respuesta que llega del módulo. Hasta 8192 bytes
//o hasta que se alcance el caracter 0x0a
while(!($rsp[strlen($rsp)-1] == chr(0x0a)))
$rsp .= fgets($handler,8192);
//SE HACE ALGO CON LOS DATOS
//se sierra la conexión
fclose($handler);
```

Esa es la secuencia de comandos típica utilizada en los scripts que inician una comunicación con el módulo.
Luego de mandar los datos por el socket con la función *fwrite()*, espero una respuesta. Para el caso del ejemplo, la condición de parada es que el último caracter del string recibido como respuesta sea el 0x0a, más cono-

cido como LF o Line Feed. Ahora no se hcae exactamente así, ya que se termina de recibir datos del módulo cuando el string respuesta alcanza cierto largo. Esto se cambió porque al usar encriptación[55] uno de los caracteres encriptados podía coincidir con el 0x0a, no así tal como se hacía antes.

En cuanto al manejo de conexiones PHP prevee algunas funciones auxiliares que permiten monitorear el estado de la conexión, e incluso hacer que el script siga corriendo luego que el cliente ha presionado detener en el explorador[56], o en nuestro caso que el módulo haya cerrado la conexión.

14.3.2.10. Datos que se envían al módulo

Hay varias circunstancias en las que el servidor envía datos al módulo y ya las hemos nombrado en varias oportunidades. En esta sección veremos cómo se arman los paquetes, respetando el formato definido en capítulo 9 *Comunicación entre el webserver y el módulo.*

- **Status y log**

  En primer lugar, la solicitud de status al módulo requiere que se le envíen las direcciones de cada uno de los actuadores[57]. Para esto construye un string buscando en la base de datos la dirección de cada actuador que ese usuario tiene y va concatenando esta información de a un byte por actuador. Luego agrega el largo total de bytes que envía, abre un socket y envía el string con un encabezado que indica que se trata de una solicitud de status.
  En cuanto al log, no necesita enviar ningún dato ya que es el módulo quien cuenta con toda la información. Simplemente envía una cabecera por el socket que indica pedido de log.

- **Actualización de agenda**

  Hay dos tipos de envíos de comandos agendados: uno total y uno parcial. Cuando el módulo se conecta a la alimentación lo primero que hace en cuanto a comunicación con el exterior se refiere, es pedir la agenda completa al servidor. Para ello hace un GET a un script particular que envía cada uno de los días. En cuanto a la actualización de la agenda, es decir el envío de aquellos días para los cuales el servidor considera que el módulo tiene información desactualizada, es desencadenado por el usuario de la página web, luego de agregar o borrar comandos de la agenda. Cuando el usuario agenda o borra un comando, se setea un registro en la base de datos que se corresponde con el día en cuestión. Cuando el usuario solicita una actualización del módulo, el servidor chequea este registro y envía solo aquellos días para los cuales este registro ha sido seteado. Luego de enviar la actualización el servidor resetea el registro de días.

[55] Ver sección 16.2 *Encriptación*
[56] Función ignore_user_abort()
[57] Recordemos que el módulo no cuenta con esta información. A la hora de ejecutar un comando ya sea agendado o de tipo instantáneo necesita contar con toda la información almacenada en el propio comando.

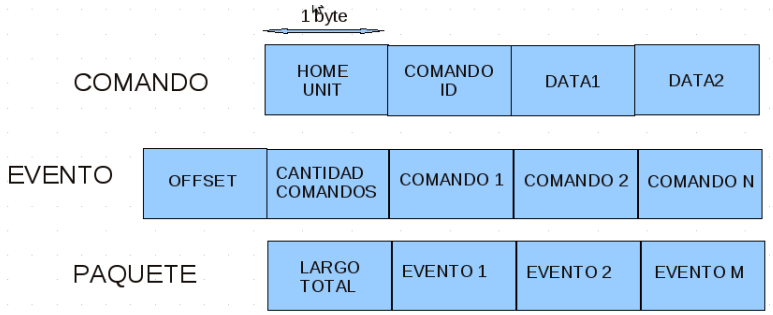


Figura 14.20: Almacenamiento de un comando

Ya sea el que el envío de comandos agendados es total o parcial, el paquete correspondiente a los comandos de un día se arma siempre igual.

Para entender cómo se implementa el algoritmo que arma el paquete de un día es necesario conocer cómo se almacenan los datos. Para ello se puede consultar la sección 15.1 *Estructura de las tablas de la base de datos.* Observe en la Figura 14.20 cómo se estructura el paquete que se va a enviar al módulo.
Por otra parte la tabla en la que se almacenan los comandos contiene una fila por comando. En cada registro (o fila) se almacenan los datos del comando propiamente dicho, el offset del comando[58] y el día. Hay registros que en lugar de representar a un comando, representan una escena. Para hacer esto simplemente se guarda un registro con un ID inválido[59]. Además los comandos que pertenecen a escenas se almacenan en una tabla independiente.
Al momento de formar el string correpondiente a un día, con el formato indicado en la Figura 14.20 se implementa el diagrama de flujo de la Figura 14.21. El diagrama de flujo está simplificado, por ejemplo no esta considerado el hecho de que al agendar una escena, algúno de sus comandos podría pasar al día siguiente, y esto debe ser considerado al momento de armar el paquete que se le envía al módulo. Esto torna el flujo un poco más complejo y no tiene sentido una explicación tan detallada.

14.3.2.11. Procesamiento de formularios

Algo particular de los scripts realizados es que el procesamiento de los formularios, y el código HTML que muestra el propio formulario está todo contenido en el mismo archivo. Generalmente, por un lado se tiene un archivo HTML que muestra el formulario, y el procesamiento se hace con un script almacenado en otro lado. Al unir estos dos archivos en uno, se gana en velocidad y se hace posible por ejemplo mostrar carteles producto del procesamiento, dentro de la misma página[60].
Los formularios que puede completar el usuario en el acceso web son:

- agendar de un comando

[58] El offset son los segundos desde la hora 0
[59] La cantidad de comandos es menor a 256, es decir 1 byte
[60] Por ejemplo: "la casilla ACCION no puede estar vacía", en la página de agenda

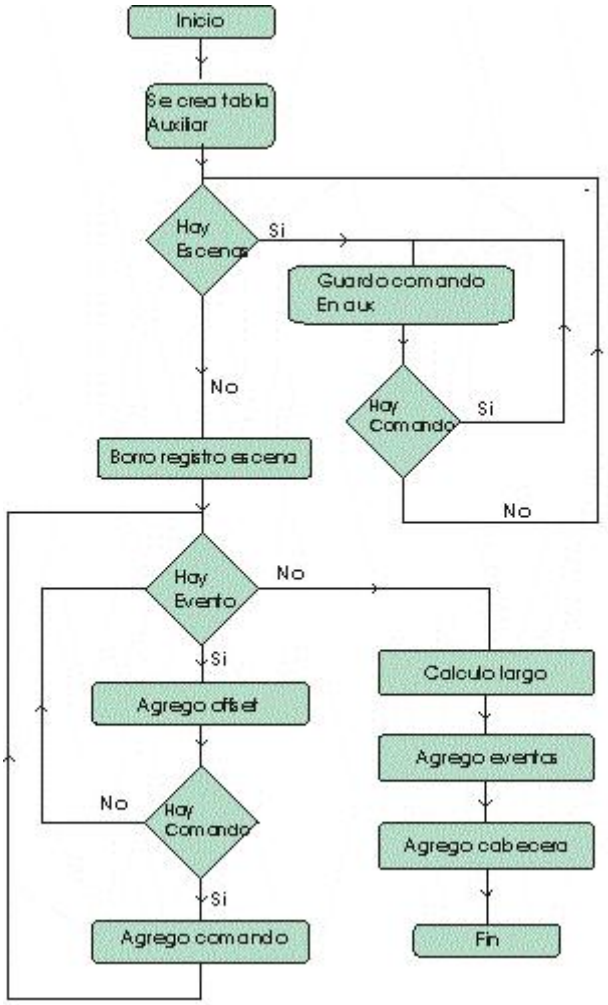


Figura 14.21: Diagrama de flujo del armado del paquete correspondiente a un día

- quitar un comando o escena de la agenda
- agendar de una escena
- envío de un comando instantaneo en página *Ejecutar comando*
- agregar un dispositivo
- quitar un dispositivo
- crear una escena
- borrar una escena
- editar una escena
- borrar el comando de una escena
- agregar un comando a una escena

entre otros formularios. Cada uno de esos ítems merece un tratamiento de datos distinto, obviamente.
Generalmente los datos son enviados al propio script vía método POST de HTTP, aunque para algunos pocos se usa el método GET. Esto es así ya que en ciertos casos enviar datos vía POST provoca que algunos browsers generen una advertencia, y esto es incómodo para el usuario.
Ya se vió cómo se arman en general los formularios en subsección 14.3.1.1 *HTML*.
Ahora mostraremos cómo se capturan los datos:

```
if (array_key_exists('_submit',$_POST)){
$var1 = $_POST['nombre'];
$var2 = $_POST['hora'];
//aquí se hace algo con los datos
}
```

Con el if anterior se capturan los datos del formulario *_submit*, enviados por el método POST. Como se observa, una vez más se usan las variables predefinidas para obtener el valor de las entradas. En el ejemplo se guardan temporalmente en las variables *$var1* y *$var2* las entradas bajo el nombre de *nombre* y *hora*.
Esto mismo puede hacerse con Javascript, pero dado que generalmente el procesamiento de los datos involucra el acceso a la base de datos MySql, carece de sentido.

Capítulo 15

Almacenamiento de datos en el servidor

15.1. Tablas de la base de datos

Como ya se dijo, al inicio del proyecto dado que se subestimaba brutalmente la cantidad de datos que iba a ser necesario almacenar, se inventó una "base de datos" usando archivos de texto plano. Con la incorporación del login que requiere el uso de una base de datos MySql administrada a través de PHP, no quedó otra opción que utilizar esta, y repensar lo que se había hecho hasta el momento en cuanto a almacenamiento se refiere.
MySql es una base de datos muy versátil y de uso muy expandido, y PHP cuenta con una familia de funciones destinadas a administrar una base de datos MySql. En cuanto al hosting `http://www.hostingenlaweb.com` que es el que utilizamos, su panel de control permite fácilmente crear una base de datos, registrar un usuario MySql y asociarlo a esa base de datos.
Recordemos que en el momento de la creación de una tabla MySql deben defnirse el nombre de cada columna y que tipo de dato contendrá la misma. Luego de creada la tabla, crear un registro nuevo significa agregar una fila a la tabla. Si bien en el momento de agregar un registro no es necesario especificar cada uno de los campos, cada registro se ve como una fila independiente.
A continuación veremos que tablas se crearon y que contenidos almacenan:

1. Tabla **users**

ID	User	Pwd	Temp	Temp_active	Email	Active	Level	mod_id	IP	actualizo

La tabla *users* como se explicó arriba se creó inicialmente para poder regsitrar usuarios almacenando datos como el nombre, la contraseña, un identificador, etc.. A medida que se requirió almacenar datos como la dirección IP del módulo[1] y el identificador del módulo se modificó la tabla. Las columnas que se muestran arriba son las que contiene la tabla actual. Los campos son:

[1] Ver Anexo C *Página multiusuario*

- *ID*
  Se define como un número entero que identifica al usuario de manera única. Cuando se crea una sesión se almacena este valor en la variable $_SESSION['user_id'] como se explicó en subsección 14.3.2.8 *Login*. Este campo se define en la creación de la tabla de forma que se autoincremente. De esta forma no es necesario especificar su valor nunca, simplemente se consulta este y además se asegura que no habrán dos repetidos. Es un número de hasta 11 enteros.

- *User*
  El nombre de usuario.
  Se define como un string.

- *Pwd*
  La contraseña actual del usuario.
  Es un string.

- *Temp*
  Es una contraseña temporal. En general no se usa, pero si el usuario olvida su contraseña y presiona el botón correspondiente, se genera una automáticamente y se le envía al mail.
  Es un string.

- *Temp_ active*
  Indica si la contraseña temporal está activa. Si el usuario no presiona sobre el link que se le envía al mail luego del envío de contraseña, la temporal no se activa y el usuario no puede utilizarla.
  Es un string.

- *Email*
  El email del usuario almacenado con el fin de enviarle la contraseña en caso de olvido, o de enviarle el link de activación de cuenta una vez que se ha registrado.
  Es un string.

- *Active*
  Indica si la cuenta del usuario está activa. Cuando se crea un nuevo usuario se le envía un mail de activación de cuenta y debe presionar el link enviado. Hasta que eso no sucede la cuenta permanece inactiva y no puede usarse. También permite al administrador del sistema deshabilitar usuarios momentáneamente.
  Es un número.

- *Level*
  Indica si el usuario es normal o administrador.
  Es un número.

- *mod_id*
  Es el identificador del módulo. El módulo envía este en cada comunicación dirigida al servidor, con el fin de que este pueda identificar de qué módulo proviene la comunicación ya que su IP cambia, y de forma de poder actualizar la IP[2].
  Es un número entero de entre 3 y 5 dígitos.

- *IP*
  Es la dirección IP del módulo.
  Es un string.

- *actualizo*
  Se utiliza para saber que días deben ser enviados al módulo, en caso de que el usuario solicite una actualización. Se utiliza como si fuera un byte, pero en su un lugar, para operar más cómodamente se usa un string de 7 caracteres. Por ejemplo si el caracter 0 está en 1 indica que debe enviarse actualización del día domingo.

2. Tabla **comandos**

cmd_ID	Offset	Hora	Ampm	PB0	PB1	PB2	PB3	name	User_ID	Dia

La tabla comandos almacena todos los comandos y escenas que el usuario agendó. Recordemos que un comando consiste en 4 bytes, PB0 a PB3. Por otro lado, se almacenan también las escenas agendadas guardando valores inválidos de comando en el registro PB1, y usando el registro PB2 para identificar a la escena.
Veamos cada uno de sus campos:

- *cmd_ID*
  Creado con el único fin de eliminar el comando agendado de la tabla. Se define en la creación de la tabla al igual que el ID de la tabla users como auto-increment. De esa manera nos aseguramos que no se repita[3] y además no debe especificarse nunca[4]. El cmd_ID es también muy útil cuando se quieren mostrar los comandos en una tabla HTML por ejemplo.
  Es un entero

- *Offset*
  Almacena la hora del comando como los segundos desde la hora 0. Como ya se vió es este el formato que se utiliza en el módulo para llevar un registro de la hora. Cuando se arma el paquete correspondiente a los comandos de día cuando se lleva a cabo la actualización del módulo, se envía este campo en 2 bytes.
  Es un entero.

[2]Ver Anexo C *Página multiusuario*
[3]Es la propia base de datos que asegura que un campo definido como auto-increment no se repita en una tabla.
[4]En cada nuevo registro se aumenta y se asigna internamente.

- *Hora*
  Es la hora a la que está agendado el comando en el formato estándar HH:MM:SS. Es información redundante con la anterior, y si bien hace un mal uso de la memoria del web-server, se gana en tiempo de respuesta. En el momento en que se agenda un comando se calculan el offset y la hora, pero ya sea cuando el comando se muestra en pantalla o cuando se envía al módulo, solo hay que consultarlo a la base de datos y ambos procesos se tornan algo más rápidos.
  Es un string

- *PB0 a PB3*
  Son 4 bytes que definen el comando. PB0 especifica la dirección del actuador, PB1 especifica el comando propiamente y PB2 y PB3 son datos.
  Cada uno es un caracter.

- *name*
  Es el nombre que el usuario debe asignarle al comando cuando se lo agenda. Esto se hizo así pensando en tener algun mnemónico de manera que el usuario mirando la agenda semanal pueda saber con este nombre para que fue que agendó ese comando. Por ejemplo "enciendo regadores".
  Es un string.

- *User_ID*
  Recordemos que el acceso web está pensado para soportar múltiples usuarios. Los comandos y escenas agendados de todos los usuarios están contenidos en esta tabla. Para diferenciar a qué usuario pertenece cada comando se creó este campo. Simplemente al momento de agendar un comando, junto con los otros campos se almacena el User_ID de manera de relacionar el comando con el usuario en cuestión. Este campo vale lo mismo que el campo ID de la tabla *users*.
  Es un entero.

- *Dia*
  Es el día en el que está agendado el comando o la escena.

3. Tabla **actuadores**

nombre	home	unit	tipo	b1	b2	b3	user_ID

La tabla actuadores se creo con el fin de almacenar los actuadores de todos los usuarios registrados en una tabla. Al igual que la tabla comandos utiliza el campo *user_ID* para asociar el actuador con un usuario en particular. No solo almacena la dirección del actuador[5], sino que también permite elegirle un nombre[6] y también almacena información sobre el estado del actuador. Veamos sus campos:

[5] Que desde el punto de vista del protocolo PLCbus es lo único que interesa
[6] Ej.: "lampara principal cocina"

- *nombre*
  Es el nombre del actuador. Fue creado con el fin de que el usuario simplemente recuerde el nombre y no la dirección, lo cual sería bastante más incómodo para el uso diario del sistema. Si se observa la Figura 14.7 de agendado de comandos se verá que se muestra una casilla que permite seleccionar el actuador, para esto se usa el nombre del mismo y no la dirección.
  Es un string.

- *home*
  Es el nibble alto de la dirección del actuador. La dirección del mismo es un byte, y se divide en dos partes: home y unit, partes alta y baja respectivamente. En la práctica, se suelen reservar algunos "home" para asignar direcciones especiales si se usan dispositivos que generan escenas.
  Al momento de agregar un actuador en la página debe especificarse la dirección, home y unit por separado.
  Es un character.

- *unit*
  Nibble bajo de la dirección PLCbus del actuador.
  Es un character.

- *tipo*
  Hay básicamente 5 tipos de actuadores: dimmer, switch, motor, controlador de escenas y control remoto. En el momento de agregar un actuador en la página debe especificarse uno de esos 5 tipos. Los que hemos utilizado para este proyecto dado que son los únicos con los que contamos son dimmer y switch. Un switch permite simplemente prender o apagar, mientras que el dimmer permite además graduar la intensidad.
  Es un character.

- *b1, b2 y b3*
  Los campos b1, b2 y b3 tienen que ver con la información del estado del sistema. Como se vió en la subsección 14.2.7 *Consulta y actualizacion del estado del sistema*, cuando se accede a la página correspondiente a *Estado del sistema*, se muestra una tabla que indica en que estado están cada uno de los actuadores. Esa información es la que se almacena en estos tres campos. B1 indica si el actuador está encendido o apagado, B2 indica el "brightness level" y B3 indica el "dimmer level". Obviamente B2 y B3 solo tienen sentido en el caso de que se trate de actuador dimerizable.
  Cada uno de estos campos se definen como character.

- *user_ID*
  Al igual que en la tabla *comandos*, user_ID se utiliza para asociar un actuador con un usuario, y de esta manera poder almacenar los actuadores de todos los usuarios en una misma tabla.

4. Tabla **log**

offset	ampm	dia	home	unit	cmd	info	data1	data2	user_ID

La tabla log se creó para almacenar los últimos 16 comandos de agenda que ejecutó el módulo. La idea de esta tabla es informar al usuario sobre que tan bien o mal se están ejecutando los comandos agendados. No se incluyen los comandos que se ejecutan de forma instantánea desde la página *Ejecutar comando*. Para ese tipo de comandos, se puede ver cómo resultó desde la propia página de ejecución instantánea. A diferencia de la tabla anterior que almacena información vital, en el sentido que es solo en el servidor que se mantienen almacenadas las direcciones de los actuadores, la información del log se almacena tanto en el servidor como en el módulo, y es el módulo quien tiene siempre la versión actualizada, ya que es él quien en definitiva ejecuta los comandos.
El caso de buen funcionamiento sería observar que en el log y en la agenda aparecen los mismos comandos, a horas que difieran en quizá algún segundo pero no más, y además ver en el log que cada uno de los actuadores involucrados confirmó la ejecución del comando.
Veamos sus campos:

- *offset*
  El campo offset indica a que hora se ejecutó efectivamente el comando, en el formato de hora que utiliza el módulo. La hora a la cual el comando fue agendado, y la hora a la cual el comando se ejecuta pueden diferir en algunos segundos. Este offset, se considera respecto de las 12hs últimas, ya sean las 0 o las 12 del medio día. Es decir a lo sumo vale 12*60*60 - 1 segundos.
  Es un entero.

- *ampm*
  Indica si el comando se ejecutó a una hora am o pm. En esta tabla no hay información redundante respecto a la hora, ya que el offset anterior como se vió se calcula con respecto a las 12 horas últimas.
  Es un character

- *dia*
  El día en que se ejecutó el comando.
  Es un character.

- *home*
  El nibble alto de la dirección del actuador que ejcutó el comando.
  Es un character.

- *unit*
  El nibble bajo de la dirección del actuador que ejcutó el comando.
  Es un character.

- *cmd*
  Es el comando que fue ejecutado por el actuador. Es un character.

- *info*
  Este campo indica si se el actuador confirmó o no la ejecución del comando. Una mejora que tiene el protocolo PLCbus con respecto a sus predecesores como X10, es que la comunicación es en dos vías. Es decir por un lado está el dispositivo que transmite la orden al actuador de ejecutar cierto comando, pero el actuador si se le solicita puede confirmar la ejecución, enviando por la línea eléctrica otra trama PLCbus, dirigida en nuestro caso al PLCbus 1141. La trama de respuesta incluye más de un byte de información, pero dado que no está muy claro que quieren decir varios de los bits de esa respuesta, decidimos almacenar simplemente si se obtuvo o no respuesta del actuador.
  Es un character.

- *data1 y data2*
  Los campos data 1 y data 2 son los 2 bytes de información que envían los actuadores como respuesta. Se incluyeron previendo un procesamiento. Actualmente no se utilizan para mostrar nada al usuario.
  Ambos son character.

5. Tabla **escenas**

ID	cmd_id	offset	pb0	pb1	pb2	pb3	nombre	usr_ID

Esta tabla permite almacenar todo lo referido a escenas. En el momento que se crea una escena se almacenan en esta tabla al menos un registro. El hecho de que se haga un registro en esta tabla no tiene ninguna dependencia con el agendado de escenas, que como se explico más arriba, se usa la tabla *comandos* para agendar escenas.
El campo nombre, es el nombre de la escena. Cada vez que se agrega un nuevo comando a la escena, debe copiarse este nombre también. Esto no es del todo eficiente, pero simplifica las cosas. Se podría por ejemplo tener otra tabla que solo almacene los nombres de las escenas, y vincular los comandos de las escenas y sus nombres a través de un identificador.
Veamos sus campos:

- *ID*
  El campo ID se creó para borrar más fácilmente a una escena y para poder agendarla. Para crear una escena se debe hacer un registro en esta tabla de al menos una fila. En el momento de creación de la escena se genera como identificador un número de entre 0 y 255 de único para cada escena. Esto es así debido a cómo se agendan las escenas. Como ya se vió, las escenas y los comandos se agendan todos en la misma tabla. Cuando se agenda una escena se guarda el campo PB1 con un número que no identifica a ningún comando, y el tercer byte, PB2 es el identificador de la escena, que a su vez es este ID.

Es un entero.

- *cmd_ID*
  Este campo es único para cada fila de esta tabla. Una escena consiste en al menos una fila. Cada fila, además de almacenar el nombre de la escena almacena un comando y un offset. Cuando se crea una escena, se asocia al menos un comando con un offset a dicha escena. La primer fila que se almacena en esta tabla, que registra la creación de esa escena, guarda también su primer comando. Cada vez que se agrega un comando a una escena, se almacena una nueva fila que tiene en común con el resto de las filas correspondientes a esta escena, el nombre y el ID. Este campo permite quitar un comando específico de una escena.
  Es un entero.

- *offset*
  Es el offset del comando. Como se vió en subsección 14.2.9 *Crear editar, borrar y agendar escenas*, cada comando que pertenece a una escena contiene un offset. Esto indica que si la escena se agenda a la hora X, ese comando se ejecuta a la hora X + offset.
  Es un entero.

- *pb0, pb1, pb2, pb3*
  Son los bytes específicos del comando. Su significado es el mismo que en la tabla *comandos*
  Son character

- *nombre*
  Es el nombre de la escena. Como se dijo, esta información se repite en cada comando que pertenece a la escena, es decir en cada fila que pertenece a una escena. La información del nombre de la escena en general es redundante, salvo que se tenga una escena de un solo comando, caso en el cual la escena no tiene utilidad alguna.
  Es un string.

- *usr_ID*
  Como siempre, permite asociar una escena a un usuario en particular. De esta forma se almacenan en esta tabla las escenas de todos los usuarios del sistema.

15.2. PHP y MySql

En esta sección veremos que comandos de MySql[7] se usaron y que funciones de PHP[8] se utilizaron para enviar dichos comandos al servidor MySql.

[7] MySQL versión 5.0.67-community
[8] PHP versión 5.2.6

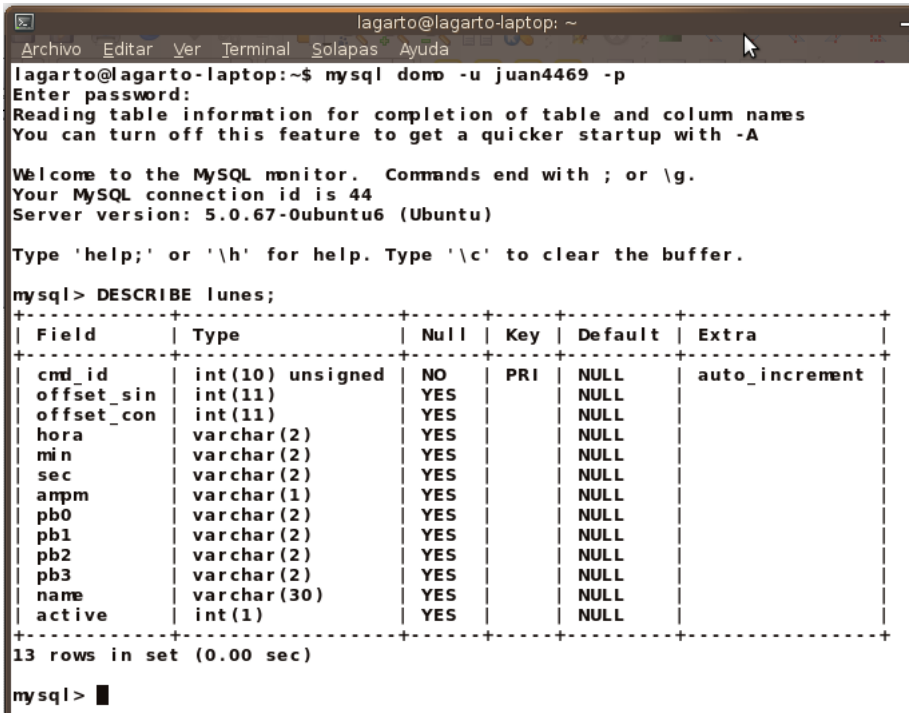


Figura 15.1: Acceso al servidor MySql desde la consola.

- **Comandos MySql**

  MySql define una serie de comandos a través de los cuales se manejan los datos que la base de datos almacena. Usando los mismos es posible crear una base de datos, crear una tabla, almacenar un dato en particular en una tabla y muchas cosas más. Estos comandos son propios de MySql y nada tienen que ver con PHP. Simplemente y ya que fue el lenguaje utilizado en el servidor, usamos PHP para enviar estos comandos al servidor MySql instalado en el servidor que nos provee nuestro hosting `http://www.hostingenlaweb.com`.

  Previo a ver los comandos volvamos sobre algunos puntos previos al uso de la base de datos.
  Para poder usar una base de datos MySql son necesarios un servidor MySql y un usuario que pueda utilizar dicho servidor. Nosotros desarrollamos el acceso web programando en PHP, pero para pruebas particulares de funcionamiento de la base de datos usamos el servidor MySql que instala el paquete LAMP de linux para ubuntu. Para realizar dichas pruebas se usa una consola cualquiera, debe crearse una base de datos y un usuario por líneas de comando, y luego de logearse con este usuario se pueden utilizar todos los comandos que describiremos más adelante. Puede observarse la Figura 15.1 para ver cómo luce el prompt MySql en ubuntu. En la figura se observa la salida del comando *DESCRIBE*, que descirbe cómo fueron definidos los campos de una tabla específica.
  Por otra parte en el servidor las cosas son un poco distintas ya que debe

usarse el panel de control para crear un usuario y una base de datos MySql, y si bien el entorno es más agradable visualmente, es también mucho más acotado.

Ahora sí veamos que comandos se utilizaron en el proyecto y para que se los usó[46]. Describiremos aquellos empleados en el web server.

- *Creación de una tabla*

  Para crear una tabla se usa el comando `CREATE`. La sintaxis del mismo, así como la de todos los comandos MySql es muy simple.

  ```
  CREATE TABLE una-tabla(variable-str varchar(3), variable-num
  int(4));
  ```

  Dentro del paréntesis se definen cada una de las columnas de la tabla. Por ejemplo la variable `variable-str` es un string de hasta 3 caracteres. MySql soporta varios tipos de datos enteros, varios para números decimales, tipos más específicos como `DATE`, `TIME`, `BLOB` para almacenar fechas, tiempo o bloques de bytes respectivamente, y muchos otros tipos más.

- *Selección de datos*

  El comando `SELECT` es por lejos el comando que más utilizamos. Su sintáxis es la siguiente:

  ```
  SELECT * from una-tabla;
  ```

  Con esa línea se seleccionan todas las columnas de la tabla una-tabla. En general MySql devuelve arrays de arrays[9], o un array. El caracter * implica todos, si en su lugar se hubiese usado el nombre de una columna la consulta hubiera arrojado un array conteniendo el valor de esa columna en cada una de las filas de la tabla. El caso del ejemplo es el más simple posible y no es muy utilizado. Generalmente en el propio comando `SELECT` se usa algún campo opcional como `WHERE`, que permite filtrar la salida con la condición que se quiera, por ejemplo:

  ```
  SELECT variable-str from una-tabla WHERE variable-num=8;
  ```

  Esa consulta devuelve un array que contiene el valor del campo `variable-str` de aquellas filas cuya `variable-num` vale 8.
  Otro campo opcional del comando SELECT, así como lo es WHERE es ORDER by. Con el mismo se puede solicitar que la respuesta devuelta esté ordenada según la variable que se elija, generalmente un número. Por ejemplo usamos este campo para mostrar los comandos

[9] O una matriz, como quiera verse.

de la agenda semanal ordenados por offset creciente.

- *Agregar una fila*

  Para agregar cualquier tipo de dato a una tabla MySql siempre debe agregarse una fila entera. Para esto se utiliza el comando `INSERT` de la siguiente forma:

  ```
  INSERT INTO una-tabla (variable-str,variable-num) VALUES ('uno',1)
  ```

  De esa manera se agrega a la tabla `una-tabla` una fila con los valores 'uno' y 1 para la primer y segunda columnas. Si no se hubiese especificado el valor de una de las columnas queda ese campo con valor NULL, salvo que se haya definido un valor por defecto o que sea un campo del tipo *auto-increment*, como se usó en la definición del campo *cmd_ID* en la tabla `comandos`.

- *Borrar una tabla o el contenido de la misma*

  Para borrar el contenido de una tabla se usa el comando `TRUNCATE` de la siguiente forma:

  ```
  TRUNCATE TABLE una-tabla
  ```

  Esa línea simplemente "limpia" la tabla, es decir borra todas las filas pero no destruye la tabla. Si se eliminar la tabla completa se debe usar el comando `DROP`.

- *Borrar filas o modificar el valor de alguno de sus campos*

  MySql permite usar un comando para borrar alguna fila en particular o varias. Su sintáxis es la siguiente:

  ```
  DELETE from una-tabla WHERE variable-str='uno'
  ```

  Ese comando elimina todas las filas de la tabla `una-tabla` que verifiquen la condición que sigue al campo `WHERE`, en este caso aquellas que contengan el string 'uno' en el campo `variable-str`. Obviamente si se usa una candición más exigente se borrarán menos filas.
  Existe otro comando que permite actualizar el valor de un campo de una o varias filas a la vez:

  ```
  UPDATE una-tabla SET variable-num=78 WHERE variable-str='uno'
  ```

  De esa manera se carga la columna `variable-num` con 78 de todas aquellas filas que cumplan la condición.

Ahora que hemos visto que comandos hemos usado, se ve más claramente la utilidad de incluír identificadores ya sea para el usuario, como para los

comandos como para las escenas. Casi siempre las sentencias de búsqueda o de borrado de algún registro incluyen al final la opción `...WHERE ID=una_variable_en_cuestion`.

- **Funciones mysql de PHP**

  Para gestionar la base de datos MySql alojada en el servidor web desde scripts PHP, se usaron funciones de una familia de PHP creadas específicamente a estos propósitos.
  Como siempre en cada script es necesario logearse con el usuario registrado en el control panel del hosting, y además conectarse a la base de datos asociada a ese usuario. Para eso se utilizan las siguientes funciones:

```
mysql_connect($host,$user,$password);//login contra el servidor
mysql_select_database($database);//conexión con la base de datos
asociada a ese usuario
```

  Como es necesario hacerlo en todos los scripts, se usa la función `include()` de PHP, que permite incluír código PHP alojado en otro archivo, y de esta forma se evita tener que repetir esas dos líneas y otras en cada script.
  Luego, son pocas funciones las que se usan: La más utilizada es `mysql_query()` que recibe como parámetro un string con un comando MySql de los que vimos en la sección anterior, y devuelve un handler. Una vez hecho eso se puede querer hacer básicamente 2 cosas: conocer el largo de la respuesta, es decir la cantidad de filas, o obtener una fila en particular. La función `mysql_num_rows()` recibe como parámetro el handler y devuelve el número de filas que arrojó la consulta, y la función `mysql_fetch_assoc()` que recibe también el handler y devuelve un array.
  El comportamiento de esta última función es extraño y a la vez práctico. `mysql_fetch_assoc()` tiene un puntero interno que al comienzo apunta a la primer fila arrojada por la consulta. Una vez que se llama a la función devuelve un array que contiene la primer fila. La siguiente vez que se llama a la función esta devuelve un array que contiene la segunda fila de la respuesta, y así sucesivamente va incrementando el puntero interno en cada llamada. En el siguiente ejemplo se muestra cómo imprimir en pantalla el contenido de la columna variable-str de una tabla:

```
$sql = 'SELECT variable-str from una-tabla';
$handler = mysql_query($sql);
while($row = mysql_fetch_assoc($handler)){
echo $row['variable-str'];
}
```

  El código anterior funciona porque además mysql_fetch_assoc() devuelve 0 cuando no hay más filas, por lo que se sale del loop.
  La última funcón de esta familia que se utilizó fue `mysql_error()`. Esta puede usarse para conocer una descripción de un error que se produzca al hacer una consulta. Por ejemplo:

```
mysql_query('DELETE from una-tabla WHERE variable-num=8') or die(mysql_error());
```

De esta forma, si se produce un error de cualquier tipo al realizar la consulta se termina el script y se imprime el error en pantalla.

Capítulo 16

Seguridad

Un problema que identificamos desde el comienzo del proyecto fue la seguridad, ya que un producto que permita controlar remotamente un hogar evidentemente debe ser robusto, o de otra manera no comercializable bajo ningún concepto. Este tipo de productos de automatización deberían funcionar para traer tranquilidad y comodidad al usuario, es por esto que deben ser lo más seguro posibles.
La tema seguridad se divide en 3:

1. *login* para bloquear el acceso a la página a quienes no están registrados
2. *autenticación* para identificar si quien me envía un mensaje es quien yo creo que es y no un impostor
3. *encriptación* para que un tercero no pueda ver el contenido de los mensajes

Claramente el tema login ya se vió en la subsección 14.2.1 *Login, usuario normal y administrador del sistema* y en la subsección 14.3.2.8 *Login* por lo que no vamos a entrar en detalle al respecto.
Para ver un análisis más profundo de los algoritmos empleados para encriptación y autenticación ver sección 11.2 *Encriptación* y sección 11.1 *Autentucación*.
Cabe aclarar que el tema de seguridad no esta completamente implementado. Actualmente como vimos la comunicación entre el módulo y el servidor se realiza mediante dos mecanismos: vía sockets TCP/IP y vía variables del método GET de HTTP. Lo relativo a autenticación y encriptación se aplica a sockets, y no a la información que el módulo le envía al servidor vía HTTP. Como se explica en el Anexo C *Página multiusuario*, no se puede enviar cualquier byte usando HTTP GET, por lo que implementar autenticación o encriptación sobre HTTP GET implicaría usar mecanismos como el que se describió mandando información de a nibble por byte, por lo que se decidió no incluír seguridad en esos intercambios, y en una etapa posterior del proyecto migrar todas las comunicaciones a sockets.

16.1. Autenticación

El mecanismo de autenticación que utilizamos consiste en generar a partir del mensaje que se quiere autenticar y una clave, 4 bytes que se agregan al final del mensaje. Además la función que se utiliza para generar los 4 bytes no tiene

inverso.
Este mecanismo se puede implementar con diversos algoritmos, pero teníamos varias limitantes. En primer lugar estaba el problema de la velocidad de procesamiento del módulo, que limita en gran manera los algoritmos disponibles. En general aquellos aceptados como buenos eran demasiado "pesados" para nuestro ATmega 32. En segundo lugar estaba el problema de la compatibilidad entre los lenguajes C usado para el firmware del módulo y PHP.
El algoritmo que utilizamos para autenticación es HMAC. Este a su vez utiliza una función de hash. Lo primero que se hizo entonces fue encontrar una función de hash tal que ambos algoritmos[1] coincidan. Se vió que esto era posible con el hash *CRC32b* de PHP[2]. Luego de identificar que la salida del hash en cuestión coincidía para más de un vector de prueba, pasamos al algoritmo HMAC que requiere una clave para generar el digest de un mensaje. PHP también incluye una función que implementa HMAC, pero esta no producía la misma salida que el algoritmo programado en el módulo. Por este motivo se tuvo que implementar una versión casera del algoritmo HMAC en el servidor. El único problema para implementar HMAC fue que la función lógica exor es solo aplicable a enteros, y no a caracteres en general como en C. Esto se soluciona utilizando una función auxiliar que convierte el caracter a entero.
Luego de alguna prueba y alguna modificación se llegó a una versión que generaba los mismos digest de 4 bytes.

16.2. Encriptación

Con la encriptación se completa lo referido a seguridad, aunque como se explicó arriba de forma parcial.
El algoritmo de encriptación a diferencia de los utilizados para autenticación es "casero". No se entrará en detalle en esta sección sobre porque se implementó como se lo hizo, dado que se explica en sección 11.2 *Encriptación*.
La familia de algoritmos de encriptación que utilizamos supone el uso de algoritmos con inverso que utilizan una sola clave.
Para implementar nuestro algoritmo de encriptación es necesario:

1. Generar un vector aleatorio
   Para lo que se usó la función `mt_rand` de PHP, que no necesita inicializar la semilla[3] ya que se hace internamente, a diferencia del lenguaje C.

2. Aplicar exor al mensaje byte a byte con la clave
   Para esto como se explicó en *autenticación* es necesario convertir cada caracter a entero, antes de aplicarle el exor. La operación se aplica byte a byte y cuando se alcanza el final de la clave se rota esta de manera circular.

3. Rotar el mensaje
   Rotar el mensaje consiste en intercambiar algunos bytes del mensaje en

[1] Módulo y server
[2] Como siempre PHP incluye casi todo lo que se busca. En particular más de 20 o 30 funciones de hash entre las que se encuentra el algoritmo CRC32B, de la familia de los códigos de redundancia cíclica.
[3] Muchas funciones que generan números aleatoriamente requieren inicializar una semilla, punto inicial de la iteración, cada vez que se la utiliza para obtener mejores resultados.

función del vector de rotación. No explicaremos el detalle de este procedimiento pero el mismo implica rotar circularmente un número, es decir rotar los bits del mismo. Resulta que en lenguaje C la rotación bitwise se hace de forma circular, y no así en PHP. Por este motivo, hubo que implementar esta rotación también, de forma que sea compatible con C.

Tanto aplicar el exor con la clave como rotar el mensaje necesitan del algoritmo inverso para poder implementar la decriptación. Lo bueno de utilizar la función EXOR es que su inverso es la propia función.

Antes de enviar un mensaje por el socket se lo encripta y al resultado de eso se le aplica el algoritmo de autenticación, agregando al final del mensaje encriptado el digest de la autenticación. A la inversa, cuando se recibe un mensaje por el socket primero se chequea que sea autentíco y si lo es se lo desencripta. Elegir este orden[4] implica que si el mensaje no es auténtico el proceso de descartar el mensaje es más rápido.

[4]Primero encriptar y luego autenticar el mensaje encriptado antes de enviarlo al módulo.

Apéndice A

Evolución del proyecto durante su ejecución

A.1. Objetivo inicial y desviaciones del mismo

A lo largo del proyecto no se hicieron mayores modificaciones al objetivo del mismo, de hecho, la única modificación significativa realizada fue que se decidió implementar el sistema para un único protocolo de domótica, y no para múltiples protocolos como se había planteado inicialmente.
Esto ya había sido considerado en la etapa de planificación inicial ya que por un lado, el objetivo principal era cubrir el vacío de mercado para módulos de control remoto vía Internet existente en PLCBus, y por otro, los tiempos manejados no eran suficientes. Esta funcionalidad se incluyó en el objetivo inicial ya que al momento de la entrega del mismo no se había cerrado la negociación con el socio (XTEND Argentina)[4][1].
Dado que su prioridad era obtener un producto robusto "lo antes posible", se optó por trabajar únicamente con el protocolo PLCBus, y dejar X10 para una posible adaptación posterior del sistema.
Por otra parte, dado que la única diferencia entre ambos productos sería a nivel de firmware, se concluyó que no tendría demasiado sentido hacer un producto de protocolo seleccionable, puesto que la idea de esto era que el distribuidor lo configurara por única vez previo a la instalación.

A.2. Supuestos y realidades

Algunos de nuestros supuestos no se cumplieron lo cual repercutió negativamente en la ejecución del proyecto.
En primer lugar, la información recabada además de ser escasa fue errónea (por ejemplo, la especificación del protocolo brindada por el fabricante confundía sistemáticamente byte con bit). Además, se descubrió que la comunicación serie entre la PC y el módulo PLCBus-1141 presentaba algunas particularidades que

[1] XTEND es una empresa dedicada desde hace 15 años a brindar soluciones en el área de los sistemas de control inteligentes de hogares. Actualmente es la única representante para toda Sudamérica del fabricante X10 USA Inc., pionera en el mundo en este rubro.

no eran mencionadas en la especificación de la comunicación.
El otro supuesto que no se cumplió fue que los tiempos de importación fueron mayores a los normales ya que hubo un paro de aduanas durante el proceso de importación lo que retuvo en el aeropuerto de carrasco los componentes comprados retrasando la etapa de comprobación del stack TCP/IP durante unos días.
Por otra parte, no fue bien estimada la repercusión que podría tener el contar con una única plataforma completa para desarrollo, dado que se contaba únicamente con un módulo PLCBus 1141. Afortunadamente, esto se pudo superar gracias al nuevo enfoque del proyecto adoptado y la división de las tareas de programación en 3 módulos auto contenidos.
Si bien estos supuestos no eran escenciales para la realización del proyecto, el que no se hayan cumplido (en conjunción con problemas en la implementación) retrasó significativamente las tareas sucesivas.

A.3. Evaluación del análisis inicial de riesgos

Parte de los problemas surgidos a lo largo del proyecto habían sido previstos en el análisis de riesgos. Sin embargo, debemos decir que en muchos casos se menospreció el impacto de los mismos por lo cual, o bien no se habían diseñado planes de contingencia, o los mismos demostraron no ser eficientes.
Un ejemplo de esto son los problemas surgidos durante la puesta en marcha de la red domótica. Este riesgo había sido estimado de probabilidad de ocurrencia moderada e impacto bajo y demostró ser un factor de impacto extremo el cual costó un retraso de al menos un mes en el proyecto además de la frustración y desgaste del equipo debido a la imposibilidad de solucionar un problema menor frente a los demás problemas encarados. Luego de dedicarle tiempo y esfuerzo mayores a los planteados incialmente este problema se logró sobrellevar.
Lo mismo sucedió con el análisis del riesgo de información insuficiente sobre el protocolo ya que en ningún momento consideramos que esta información pudiera ser errónea por lo cual no contábamos con ningún mecanismo para superar este inconveniente más que recurrir al ensayo y error.
En cuanto a los problemas vinculados a la compra e importación de los componentes electrónicos para el proyecto, creemos que, si bien se actuó correctamente buscando diferentes alternativas de proveedores, las decisiones tomadas en este punto podrían haber sido más rápidas ya que se perdió demasiado tiempo intentando comprarlo en países vecinos cuando se debería haber recurrido directamente a la compra on-line.
A modo de resumen, creemos que el análisis de riesgos fue correcto en cuanto a la identificación de los mismos pero, al menospreciar su impacto, se diseñaron planes de contingencia no del todo eficientes. Se deberían haber incluido por ejemplo tiempos extra-tareas para poder utilizar en caso de incidentes y no haber hecho una planificación tan ajustada del proyecto.

Apéndice B

Reestructura del proyecto

En general, la planificación del proyecto fue respetada a nivel de las tareas programadas y de la duración de las mismas en cuanto a horas destinadas. Sin embargo, la planificación del proyecto fue muy ambiciosa en cuanto al compromiso diario ya que se planifico para una carga media de 6 horas diarias. Esto, que fue pensado en un contexto en el cual ninguno de los integrantes del grupo tenía compromisos laborales, y cuyo objetivo era el de poder concluir el proyecto en tiempo para la presentación del mismo en una feria de domótica a realizarse en Buenos Aires a mediados de Noviembre de 2008, no pudo ser ejecutado dados los compromisos laborales adquiridos posteriormente por los integrantes.
Con el fin de reducir el impacto sobre las fechas limites preacordadas, los integrantes del equipo decidimos comprometernos en utilizar los fines de semana para compensar las horas necesarias para el proyecto. Este compromiso no fue respetado por todos los integrantes, e implicó que algunas tareas si se realizaran en tiempo y forma mientras que otras no, así como una distribución despareja del trabajo y las responsabilidades. Dado que la mayoría de las tareas planeadas tienen una fuerte dependencia con tareas realizadas por otros integrantes, el resultado fue negativo ya que implicaba sobre exigir a algunos recursos sin que esto permitiera cumplir con los plazos pactados.
En la planeación inicial, realizada sin mucha idea del alcance que se le pretendía dar al proyecto, o de las tareas implicadas, se incluyó como tarea "programación del sistema completo", algo demasiado genérico, lo cual fue tomando forma de tareas específicas con el transcurso del proyecto.
Se decidió entonces reestructurar el proyecto y subdividir esta tarea en 3 módulos auto contenidos que pudieran ser desarrollados de forma independiente por cada uno de los integrantes, de forma de reducir los tiempos de gestión necesarios y permitiendo a cada integrante disponer de sus tiempos de la forma que creyera más correcta. De esta forma, se disminuyó la necesidad de reuniones de coordinación, así como de sesiones de trabajo colectivo, reemplazándolas por informes semanales de avance en cada una de las tareas y la coordinación de actividades grupales destinadas únicamente a probar la correcta interacción de los diferentes módulos, así como el diseño de las interfaces necesarias para la comunicación entre los mismos. Estos 3 módulos y sus responsables son:

1. Programación del servidor e interfaz gráfica de usuario (Juan)
2. Ejecución de comandos y eventos vía comunicación con PLCBus-1141

(Pablo)

3. Interfaz de ATMega32 con servidor para intercambio de información y eventos (Matías).

Este nuevo enfoque, demostró ser mucho más eficiente que la planificación inicial, principalmente teniendo en cuenta que se torno muy difícil la coordinación de reuniones grupales dadas las diferencias en los horarios laborales de los 3 integrantes del grupo.
Por otra parte, esto permitió que cada uno de los integrantes se especializara en un área determinada del sistema, lo cual resultó en un mejor aprovechamiento de los recursos, tanto físicos (plataformas de desarrollo) como humanos, sobre todo ante la eventualidad de errores o inconsistencias en el diseño.
Como contrapartida de esto, existieron casos en los cuales esta especialización fue contraproducente ya que ante la eventualidad de errores en alguno de los módulos, la colaboración en la resolución por parte del resto de los integrantes exigía un mayor esfuerzo ya que los mismos debían disponer de tiempo para familiarizarse con la implementación del módulo antes de poder colaborar con la resolución.
Es importante aclarar que, si bien se distribuyo la responsabilidad sobre los diferentes módulos, no todos los módulos fueron implementados enteramente por su responsable puesto que esta nueva distribución se realizo luego de avanzadas las tareas de diseño y desarrollo. Además, para permitir un avance más rápido del proyecto, algunas subtareas fueron distribuidas a integrantes con mayor disponibilidad horaria o con mayor experiencia en determinadas áreas.

B.1. Asignación final de tareas

Dado que los tres integrantes cursaron en el primer semestre el curso de Sistemas Embebidos en Tiempo Real dictado en el Instituto de Ingeniería Eléctrica, para el cual es necesario un proyecto de fin de curso, se vió la posibilidad de hacer coincidir algunas de las tareas necesarias para el proyecto de fin de carrera con el proyecto de Sistemas Embebidos. Esto requirió una reprogramación de las tareas a realizar con el objetivo de lograr un proyecto auto contenido y atractivo para dicha materia.
Se eligió entonces dejar las etapas relativas al testeo del protocolo PLCBus y al diseño del driver para más adelante y comenzar con la implementación del Stack TCP/IP y la obtención de la fecha y hora de forma remota. Esto, que si bien puede parecer un objetivo menor en relación a las ambiciones del proyecto, implica desarrollar una plataforma de hardware y software bastante compleja que representa al menos la mitad del proyecto en cuanto a tiempo requerido.
Luego, el cambio de enfoque del proyecto y la subdivisión de la tarea "programación del sistema completo", implicó la creación de nuevas tareas, siendo la lista final de tareas realizadas la siguiente:

1. Negociación de colaboración con socio (Juan)
2. Estudio del chip Ethernet. (Pablo)
3. Estudio de redes de datos y protocolos necesarios para el proyecto.(Pablo)

4. Estudio del funcionamiento del stack TCP/IP usado como base.(Pablo)
5. Verificación del Stack.(Pablo)
6. Adaptación del stack a necesidades para ejecución desatendida.(Pablo)
7. Implementación de protocolo SNTP1.(Pablo)
8. Construcción de hardware de desarrollo.(Matías-Pablo)
9. Pruebas de la versión del stack y recepción de fecha y hora2. (Pablo-Matías-Juan)[1]
10. Diseño de funcionamiento interno.(Pablo-Matías-Juan)
11. Análisis de comunicación con exterior.
12. Estudio de lenguajes necesarios para interfaz de usuario.(Juan)
13. Diseño e implementación de la interfaz gráfica de usuario.(Juan)
14. Diseño de formatos de intercambio de datos.(Pablo-Matías-Juan)
15. Implementación de intercambio con servidor Web.(Matías-Juan)
16. Instalación de red PLCBus.(Matías-Pablo)
17. Corroboración del protocolo PLCBus (Pablo)
18. Implementación de driver básico para PLCBus.(Pablo)
19. Implementación de escritura y lectura de tablas en memoria Flash. (Matías)
20. Implementación de módulo de ejecución de eventos y comandos.(Pablo)
21. Implementación total de la comunicación cliente-servidor.(Juan-Matías)
22. Pruebas del sistema completo.(Juan-Matías)
23. Versión final del esquemático de la placa (Juan)
24. Diagramado de PCB. (Pablo)
25. Estudio de algoritmos de autenticación y cifrado.(Pablo)
26. Diseño o adaptación de algoritmos de autenticación y cifrado.(Pablo)
27. Implementación de los algoritmos de autenticación y cifrado.(Pablo-Juan)
28. Implementación de comunicación segura cliente-servidor.(Juan-Matías)
29. Pruebas finales del sistema completo. (Juan-Matías)[2]

[1] Finaliza el proyecto de Sistemas Embebidos (25 de junio).

[2] NOTA: Las tareas descritas anteriormente se encuentran ordenadas de acuerdo a sus fechas de inicio; muchas de ellas se ejecutaron en paralelo por lo que no pueden considerarse como consecutivas. No pueden ser consideradas como equivalentes ni en duración ni en complejidad.

B.2. Evaluación de las tareas realizadas

Procederemos a continuación a una breve evaluación de algunas de las tareas realizadas y de los inconvenientes presentados.

B.2.1. Negociación de colaboración con XTEND

Esta tarea no presentó inconvenientes y se finalizó antes de lo previsto aunque debemos decir que luego de las conversaciones iniciales con el socio la relación con el mismo no fue muy fluida y se limitó a pedir colaboración en momentos puntuales del proyecto, sobre todo en cuanto a insumos (actuadores para PLCBus principalmente). Además, no se llegó a ningún plan de negocios formal, simplemente a manifestaciones de interés por parte del mismo y a un compromiso "de palabra" por parte del equipo.

B.2.2. Adquisición de chip Ethernet, estudio de su funcionamiento y adaptación del stack TCP/IP

Este punto presentó inconvenientes los cuales se debieron sobre todo a la inexperiencia del grupo y a la mala toma de decisiones, ya que se subestimó el tiempo para realizar el proyecto, y no consideramos las verdaderas implicancias de atrasarse en el comienzo del mismo.
Inicialmente, como se debió viajar a Argentina por la tarea acerca de la negociación de colaboración con Xtend, se pensó en adquirir el ENC28J60 allí. Resultó que este no estaba en plaza. A continuación, dado que existen personas allegadas al grupo residiendo en Brasil y el ENC28J60 se fabrica allí, decidimos que dicho contacto nos lo enviara. Sin embargo, resultó que no se comercializa el producto al por menor por lo que tampoco era una opción viable. Finalmente, optamos por importarlo de USA y no solo llevó tiempo finalizar el pedido y arreglar el transporte, sino que nos topamos con un problema adicional: Aduanas del Aeropuerto de Carrasco no estaba funcionando por paro de funcionarios. Esto llevó a un atraso de la fecha prevista en algunos días.
El impacto de la mala toma de decisiones en este punto se redujo ya que se optó por no esperar a que llegara el chip para comenzar la verificación del stack TCP/IP ni la implementación de modificaciones al mismo de modo que, al llegar el chip, ya se había avanzado considerablemente en el desarrollo y solo restaban pruebas de funcionamiento. Esto, que afortunadamente dió resultado, implicó trabajar con un mayor grado de incertidumbre ya que no se podían ir testeando las modificaciones a medida que se realizaban, lo cual implicaba que cada decisión debía ser analizada con mucho más detalle y verificada contra la información disponible en las hojas de datos.
Si bien hubo muchos contratiempos en este punto, la reformulación de tareas y la buena voluntad de los integrantes del equipo para trabajar de forma más intensiva durante algunas semanas redujo la incidencia de dichos contratiempos en los tiempos manejados para el proyecto.

B.2.3. Diseño de funcionamiento interno

En la nueva distribución de tareas, este punto fue dejado de lado como una tarea en sí y fue hecho a lo largo del diseño de los demás módulos. Se

realizaron algunos bosquejos del diseño, pero no se ahondó demasiado en detalle, simplemente se tuvo en cuenta en cada paso las restricciones que se debían imponer.
Recién a medida que otras partes del proyecto fueron avanzando, se fue teniendo un panorama más claro de cómo debía hacerse el diseño interno del sistema y no fue hasta la finalización del proyecto de Sistemas Embebidos que se decidió hacer un diseño más detallado.

B.2.4. Análisis de comunicación con exterior

Al igual que la tarea del diseño interno, esta tarea fue casi inexistente en una primera etapa. Simplemente se exploraron las posibilidades cuando hubo que programar el sistema completo y aún en este momento, existen algunas posibilidades abiertas.

B.2.5. Construcción del hardware de desarrollo

La construcción del hardware de desarrollo fue evolucionando a medida que era requerido. Inicialmente se demoró dado que no se tenían todos los componentes (demora en la importación). Luego se implementó una primer plataforma orientada únicamente las pruebas de comunicación TCP/IP. En una segunda etapa, se realizó una plataforma adicional (carente de comunicación vía Ethernet) la cual se utilizó para el desarrollo del driver PLCBus. De esta forma, se pudo recuperar algo del tiempo perdido ya que se pudo trabajar en 2 tareas de forma simultanea, por un lado en la finalización del stack TCP/IP y por otro en el driver PLCBus.

B.2.6. Estudio de lenguajes necesarios, diseño e implementación de la interfáz gráfica de usuario

En este punto, se determino que era necesario encarar el proyecto con una estrategia más agresiva de división de tareas, por lo que se decidió que fuera únicamente un integrante el encargado de llevar a cabo las tareas de implementación de la interfaz de usuario.
Se destinó a esta tarea al integrante menos ocupado en ese momento, quien se encontraba ocioso debido a los retrasos en la adquisición del chip Ethernet y demás, con quien colaboraron puntualmente el resto de los integrantes en la resolución de algunos aspectos vinculados al diseño.
De esta forma se logró aprovechar de mejor forma las plataformas de desarrollo existentes, y disminuir el esfuerzo necesario para adquirir nuevos conocimientos, así como disminuir la incidencia del retraso en la adquisición de los componentes.
En balance, resultó ser un plan acertado, aunque se debe reconocer que fue una estrategia muy peligrosa ya que en caso de que hubieran surgido inconvenientes serios a nivel de programación, deberían haber sido resueltos únicamente por un único encargado responsable, sin asistencia de los demás.
Inicialmente se pensó en JavaScript como lenguaje principal para resolver todo el procesamiento, pero se decidió finalmente migrar a PHP ya que el primero es un lenguaje que ejecuta el procesamiento a nivel de la maquina del cliente, y el segundo a nivel del servidor, lo cual hace posible el almacenamiento de datos y la comunicación entre el módulo residente en el hogar y el usuario, utilizando

al servidor como intermediario.
Además, esta idea de utilizar al servidor como intermediario permite que las tareas de procesamiento más pesadas se ejecuten en el servidor, lo que ayuda a liberar recursos en el módulo residente en el hogar, los cuales son por demás escasos.

B.2.7. Instalación de red PLCBus y corroboración del protocolo

Los inconvenientes ocurridos en este punto fueron debidos a factores externos al grupo los cuales, si bien habían sido evaluados como posibles, jamás se pensó que llegaran a la gravedad que alcanzaron y por ende a tener la incidencia que tuvieron, pudiéndose considerar como el contratiempo más desgastante y de mayor incidencia.
Para comenzar, la documentación acerca del protocolo era errónea y confusa por lo que hubo que invertir mucho tiempo en su interpretación.
Además, surgieron problemas en la instalación de la red domótica para la prueba del protocolo dado que el PLCBus-1141 cedido por nuestro socio no funcionaba (debido a problemas en los drivers). Esto significó un gasto de tiempo mucho mayor al estimado ya que se debió solucionar dicho problema sin asistencia técnica alguna por parte de nuestro socio o de la firma encargada de la fabricación de dichos aparatos.
Si bien una alternativa hubiera sido devolver el dispositivo y solicitar uno nuevo, dados los tiempos de importación requeridos, esto se descartó y se optó por intentar solucionar el problema. Debemos reconocer que también pesó un poco en la decisión el orgullo propio, ya que nos reusabamos a tener que devolver el módulo por no haber sido capaces de instalar un driver.
Más tarde nos daríamos cuenta de que esto fue lo más acertado ya que el problema descubierto parece ser un problema sistemático y no hubiera sido resuelto con un módulo nuevo.
A pesar de que el problema encontrado es casi trivial y puede ser resuelto muy fácilmente, el no haber contado con soporte externo y haber tenido que diagnosticarlo y solucionarlo nosotros, insumió mucho tiempo haciendo que se atrasara la ejecución del proyecto cerca de un mes y medio. Además, dado que exigió destinar a varios integrantes para su resolución, hizo que se debieran suspender temporalmente algunas tareas, y finalmente redistribuirlas.

B.2.8. Diseño de formatos de datos e implementación del intercambio con el servidor

El formato del intercambio de datos debió ser revisado en varias oportunidades a lo largo de la implementación de la comunicación entre el módulo residente en el hogar con el servidor. En un principio el envío de datos desde el módulo al Server se realizaba a través del intercambio de variables del método GET de HTTP al acceder a una pagina existente en el servidor. Esto tiene limitantes en cuanto a los datos que se pueden intercambiar ya que no todos los caractéres son válidos. Por otra parte, el intercambio de datos iniciado por el

Server se realiza abriendo un socket de comunicación con el módulo. La coexistencia de estos dos formatos de intercambio de datos surgió de la necesidad de evaluar las ventajas de un método sobre el otro, de forma de lograr un método que fuera posible utilizar para todos los intercambios.
Otro aspecto que incidió negativamente en este punto, fue que al llevar a cabo inicialmente esta tarea, no se tuvo en cuenta la implementación de seguridad sobre las comunicaciones. Posteriormente se determino que el intercambio de variables por medio del método GET era sumamente inconveniente a la hora de implementar mecanismos de cifrado para evitar traficar texto claro, así como para la autenticación de los mensajes intercambiados.
Finalmente se acordó eliminar este método, pero dado que el tiempo destinado al proyecto ya se encontraba encima de lo planeado, y que aún quedaban tareas por finalizar, se decidió dejar esto para una etapa posterior.

B.2.9. Implementación de driver básico para PLCBus

El objetivo de esta tarea era diseñar un driver básico que configurara la UART y enviara por ella un comando al módulo PLCBus. Como se mencionó anteriormente, la documentación sobre el protocolo era escasa y errónea. Por esto, esta tarea insumió mucho más tiempo del necesario debido a problemas en la configuración del entramado. De hecho, la tarea culminó luego de un período de ensayo y error en el que básicamente se configuró la UART en todos los modos de entramado posibles. Finalmente se encontró una configuración de trama con la cual se enviaban y recibían correctamente los comandos, pero esta resultó ser una configuración marcada como "Reserved" en la hoja de datos del microcontrolador, lo cual fue desconcertante.
Posteriormente, en la etapa de integración de todos los módulos, esta configuración falló. Afortunadamente, el tiempo invertido en la etapa de implementación del driver había derivado en un conocimiento detallado de las posibles configuraciones, así como de rutinas de testeo robustas lo cual permitió detectar el error rápidamente.
La conclusión a la que se arribó fue que el microcontrolador utilizado en la plataforma de desarrollo era defectuoso, por lo cual funcionaba únicamente con esta configuración reservada. El microcontrolador utilizado finalmente, no tenía este defecto y por lo tanto, debía ser configurado con un tipo de entramado distinto.

B.2.10. Implementación de escritura y lectura de tablas en memoria Flash

En esta tarea surgieron inconvenientes que retrasaron considerablemente la finalización de la misma, la cual había sido evaluada inicialmente como una tarea accesoria. Estos inconvenientes surgieron debido al desconocimiento del funcionamiento del microcontrolador, y de los correctos mecanismos de escritura y lectura de la memoria flash. Por ser este tipo de memoria el destinado al almacenamiento del código del programa en ejecución, su acceso no es tan sencillo como en el caso de la memoria RAM, de hecho, todo el código que realice escritura en memoria Flash debe ser escrito en la sección de bootloader de la misma, la cual no puede ser leída y escrita al mismo tiempo. De esta forma se garantiza que el proceso de escritura no sea auto corrompible, o sea, que no se

sobrescriba a si mismo.
Dado que se contaba con un programa de ejemplo para la escritura en Flash, el que fue adaptado exitosamente en una primera instancia, no se realizo un estudio muy detallado del asunto y se dió por concluida la tarea. El problema surgió al integrar la rutina de escritura con el resto del programa, ya que el código resultante excedía el tamaño de la sección bootloader, y al estar configurado para colocar todo el programa dentro del sector de bootloader, se producía un error de compilación. Esto, que se debe simplemente a un error en la configuración del proyecto, insumió un gasto innecesario de tiempo en el estudio y análisis de los pormenores de la escritura en flash, antes de ser resuelto.

B.2.11. Implementación del módulo de ejecución de eventos y comandos

Esta tarea se efectuó sin demasiados inconvenientes, principalmente debido a la experiencia adquirida previamente en el diseño e implementación de motores de tareas. Los factores de mayor incidencia en esta tarea fueron los vinculados a los formatos de intercambio de mensajes y datos con el servidor, pero los mismos fueron resueltos de manera ágil por el grupo.

B.2.12. Pruebas del sistema completo

Esta tarea fue ejecutada a lo largo de más de un mes y en general bastante atrasados respecto a la replanificación realizada a la mitad del proyecto. Dado que las funcionalidades del producto diseñado son varias, y dada la complejidad de algunos intercambios, fueron necesarias varias reuniones de forma de lograr un producto que se comporte razonablemente bien[3].
Los principales inconvenientes encontrados fueron en primer lugar la inexperiencia con PHP y el manejo de sockets sumado a la relativamente escasa información al respecto; y en segundo lugar la inestabilidad de la plataforma de pruebas junto con la dificultad para debug. La plataforma de desarrollo consiste en nuestro módulo, el hardware específico para programación y debug del microcontrolador, y el software para manejar este último. La dificultad para "debugear" radica en que la mayoría de los procesos son en tiempo real y con tiempos fijados externamente por el servidor.
El balance de la tarea fue aceptable aunque se reconoce que se debió dedicarle tal vez un poco más de tiempo del que se le dedicó.

B.2.13. Versión final del esquemático y diagramado de PCB

La realización del esquemático, así como el diseño del PCB, implicó un esfuerzo extra, dado que no se tenía experiencia en la utilización de herramientas de diseño, y que se debió trabajar con una versión de evaluación de Eagle, la cual impone algunas restricciones que pueden enlentecer el trabajo. A pesar de esto, en balance la tarea fue ejecutada correctamente y en el tiempo estimado.

[3] Esto se alcanzó recien en la segunda quincena de enero. Ver sección capítulo 12

B.2.14. Estudio de algoritmos de autenticación y cifrado y diseño de uno propio

Esta tarea no fue considerada en la planificación inicial y requirió un tiempo considerable para su desarrollo ya que se carecía de formación alguna en el tema. La principal complejidad surgió a la hora de evaluar las alternativas ya que no se tenía ningún criterio claro para elegir o descartar algoritmos. Además, se desconocía a priori los requerimientos de recursos de los diferentes algoritmos y las diferentes implementaciones. Esto obligó a que se debieran relevar los requisitos de múltiples algoritmos para poder realizar la elección, para lo cual era necesario lograr primero las implementaciones en C de los mismos. Afortunadamente, estos algoritmos cuentan con varias implementaciones las cuales fueron adaptadas de forma de poder medir sus requerimientos, y en caso de perfilarse como candidatos, optimizadas. Lamentablemente, la mayor parte del trabajo realizado fue improductivo ya que luego de elegidos los algoritmos a utilizar, surgieron problemas con la implementación en PHP de los mismos, debiéndose rever la elección, limitándose en algunas casos a utilizar los algoritmos que ya estaban implementados en PHP. En otros casos como HMAC, la implementación del algoritmo se hizo en ambas partes, es decir el módulo y el servidor.

Apéndice C

Página multiusuario

En cuanto a la **conexión con el módulo**, para tener un acceso web que soporte múltiples usuarios, es necesario en primer lugar que cada módulo tenga un identificador único. Al registrar un usuario nuevo, uno de los datos que deben ingresarse es dicho identificador, el ya conocido mod_ID. Este dato se almacena junto con los demás datos del usuario.
El otro dato que se necesita para poder distinguir entre varios módulos es conocer la dirección IP de cada uno. Para esto cuando el módulo se enciende, envía una solicitud al servidor de actualización de agenda, en la cual incluye como dato inicial el mod_ID. El servidor identifica de qué módulo proviene la solicitud, toma esa IP y la asocia al usuario asociado a ese mod_ID. A partir de ese pedido el módulo envía actualizaciones de su IP cada intervalos regulares de tiempo, de forma que el servidor siempre tenga la IP actualizada[1]. De esta forma, con los datos Nombre de usuario, mod_ID e IP es posible manejar desde un mismo servidor varios hogares.
Obtener la IP de una conexión remota es muy simple si se usa PHP, ya que no es otra cosa que acceder al grupo de variables globales SERVER, y dentro de este al elemento REMOTE_ADDR.
Como se vió en la capítulo 9 *Comunicación entre webserver y módulo*, el módulo le envía datos al servidor a través de variables del método GET de HTTP. En cada una de esas comunicaciones como se explicó arriba debe enviar el mod_ID. Por este motivo, el formato genérico de mensajes del módulo hacia el servidor son consultas HTTP GET de la forma:

```
http://www.example.com?mod_ID=5500&var1=78&var2=53...
```

y luego el módulo en función de la variable predefinida $_GET['mod_ID'] almacena los datos en el usuario correspondiente[2].

En cuanto al **usuario de la página**, cada uno tiene un identificador único

[1] Eso para el caso que el domicilio del usuario tenga IP dinámica, que es lo más probable.
[2] Un problema que encontramos al enviar datos vía GET es la imposibilidad de asignar cualquier valor a las variables. Sólo pueden enviarse caracteres alfanuméricos y algunos más. Cuando intentamos enviar caracteres bajos de la tabla ASCII, vimos que estos no llegaban correctamente. Para ello hubo que implementar un mecanismo de forma que la información se envíe por nibble y no por byte, para asegurarnos que el caracter siempre caiga en la zona de la tabla ASCII aceptada para este tipo de intercambios vía variables GET.

en la base de datos, dentro de la tabla *users*. Una vez que el usuario se logea, se copia este valor a las variables de sessión del sistema, más específicamente al elemento session_id, y se modifica la flag logged_in de dichas variables a TRUE. De esta forma, desde cualquier script PHP lo que se hace para conocer cual es el usuario actual es llamar a la función `session_start()` de PHP que captura el valor del session_id, y con ese número se busca en la tabla *users* el usuario que le corresponde.

Apéndice D

Horas planificadas vs. horas reales

En la planificación inicial del proyecto se estimó que se trabajarían 30 horas semanales durante 23 semanas cuando culminaría el proyecto a principios de noviembre. Eso da un total de 690 horas por estudiante.
Las horas reales que demandó el proyecto fueron

- *Matías*: 720 horas
- *Pablo*: 730 horas
- *Juan*: 650 horas

Vemos que las horas reales están cercanas a lo planificado y más aún si se considera el promedio: 700 horas por estudiante.
Como era previsible la dedicación por estudiante fue algo despareja, pero no así la dedicación del grupo entero, que se mantuvo bastante constante. Es decir, mientras algunos permanecían con poca actividad, otros se dedicaban por completo al proyecto. Esto fue posible dado que se trabajó de manera muy independiente y con muy pocas reuniones grupales. La coordinación se llevó a cabo vía mails, llamadas y fundamentalmente por google chat.

Apéndice E

Contenido del CD

El CD contiene los siguientes archivos:

- **documentacion.pdf**

  Documentación completa del proyecto en versión *.pdf

- **ieee.pdf**

  Artículo con formato ieee en el que se presentan los aspectos técnicos del proyecto.

- **doxygen**

  Carpeta que contiene archivos HTML de documentación del firmware, generados automáticamente con DOXYGEN.

- **Código C**

  Carpeta que contiene todos los archivos que componen el firmware.

- **Código web**

  Carpeta que contiene todos los archivos del código web, documentados en el propio código.

En cuanto a los requerimientos del sistema solo se necesita una versión de Adobe Acrobat Reader 6 o superior e Internet Explorer 5.0 o superior.

Bibliografía

[1] Wikimedia Foundation, Inc., *Domótica*, 9 Set 2008, disponible: `http://es.wikipedia.org/wiki/domótica`, accedida 13 Set 2008

[2] Wikimedia Foundation, Inc., *X10*, 25 enero 2009, disponible:`http://es.wikipedia.org/wiki/X10`, accedida: febrero 2009

[3] SHANGHAI Super Smart Electronics Co., LTD., *News*, 2008, disponible: `http://www.plcbus.com.cn/new_1.htm`, accedida: abril 2008

[4] XTEND, *Quiénes Somos*, disponible en http://www.xtend.com.ar/quienes.asp, accedida el 13 Set 2008.

[5] Atmel Corporation, *ATmega32*, 2009, disponible: `http://www.atmel.com/products/avr/default.asp`, accedida: mayo 2008

[6] Microchip, *ENC28J60*, 2008, disponible:`http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en022889`, accedida: mayo 2008

[7] Automation@Home, *PLCBus USB Computer Interface*, Febrero 2006,disponible: `http://x10-hk.com/store/product_info.php?cPath=22_26&products_id=165`, accedida: Setiembre 2008

[8] ATMEL, *AVR Studio 4*, Mayo 2008, disponible: `http://www.atmel.com/dyn/products/tools_card.asp?family_id=607&family_name=AVR%AE+8%2DBit+RISC+&tool_id=2725` , accedida: Setiembre 2008

[9] MAXIM, *MAX232 +5V-Powered, Multichannel RS-232 Drivers/Receivers*, 17 Julio 2006, disponible: `http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1798`, accedida: Setiembre 2008.

[10] NATIONAL SEMICONDUCTORS, *LM317 3-Terminal Adjustable Regulator*, Setiembre 2008, disponible: `http://www.national.com/pf/LM/LM317.html`, accedida: Setiembre 2008

[11] AVR Portal, *AVRnet*, Agosto 2007, disponible: `http://ww.avrportal.com/?page=avrnet` , accedida: 13 Set 2008

[12] Microchip Technology Inc., *ENC28J60 Stand-alone Ethernet Controller with SPI*, Febrero 2008, disponible: `ww1.microchip.com/downloads/en/DeviceDoc/39662a.pdf`, accedida:Febrero 2009

[13] Objective Development Software GmbH, *AVR USB - A Firmware-Only USB Driver for Atmel AVR Microcontrollers*, 2009, disponible en: `http://www.obdev.at/products/avrusb/index.html`, accedida: Febrero 2009

[14] Cornell University, *ECE476 USB Host Controller*, disponible en: http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2007/blh36_cdl28_dct23/blh36_cdl28_dct23/index.html, accedida: Febrero 2009

[15] Craig Peacock, *USB in a NutShell*, 6 de Abril 2007, disponible en: http://www.beyondlogic.org/usbnutshell/usb1.htm, accedida: Febrero 2009

[16] Cadsoft Computer GmbH, *Cadsoft Online*, 2009, disponible: http://www.cadsoft.de/freeware.htm, accedida: Febrero 2009

[17] Wireshark, *About*, 2008, disponible: http://www.wireshark.org/about,accedida: Setiembre 2008

[18] Avr-libc, *Data in Program Space*, 21 Dec 2007,disponible:http://www.nongnu.org/avr-libc/user-manual/pgmspace.html, accedida: Setiembre 2008

[19] David Simon, *An Embedded Software Primer, 1 ed.*(Singapur, Pearson Education,2005), 119

[20] Andrew Tanenbaum, *Computer Networks, 4 ed.* (Pearson Education, 2003), p.501,534

[21] Automation@Home, *RS232 to PLCBUS Control Transport Protocol*, 2 Feb 2005, disponible: http://x10.hk.com/store/manual/plcbus/plcbus-1141.pdf, accedida:Setiembre 2008

[22] Thesycon System Software and Consulting GmbH, *USB Driver*, 2009, disponible: http://www.thesycon.de/eng/usc_cdcacm.shtml#cdcacmdemo , accedida: Setiembre 2008.

[23] HomeAutomation, *HomePlanner*, Mayo 2008, disponible: http://www.hometomation.com/homeplanner/, accedida: Setiembre 2008

[24] National Institute of Standards and Technology (NIST), *Keyed-Hash Message Authentication Code, FIPS PUB 198-1*, Julio 2008, disponible en:http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf, accedida en:Febrero 2009

[25] National Institute of Standards and Technology (NIST), *Secure Hash Standard (SHS), FIPS PUB 180-3*, Octubre 2008, disponible en:http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf, accedida en:Febrero 2009

[26] IEEE, "IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003),IEEE 802.15.4-2006", *ieeexplore.ieee.org*, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), 2006, disponible: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&arnumber=1700009&isnumber=35824, accedida: abril 2008

[27] ZigBee Alliance, *ZigBee Control your world*, 2009, disponible: http://www.zigbee.org/,accedida:abril2008

[28] Echelon Corporation, *The LonWorks Platform: Technology Overview*, 2009, disponible: `http://www.echelon.com/developers/lonworks/protocol/default.htm`, accedida: abril 2008

[29] ShangHai Super Smart Electronics Co.Ltd., *Ideas for gracious living*, 2008, disponible: `http://www.plcbus.com.cn/1_2.htm`, accedida: marzo 2008

[30] T. Berners-Lee, D. Connolly, *Hypertext Markup Language - 2.0*, November 1995, disponible: `http://www.rfc-editor.org/rfc/rfc1866.txt`, accedida: julio 2008

[31] W3Schools, *HTML Tutorial*, 2009, disponible:`http://www.w3schools.com/htmL/default.asp`, accedida: julio 2008

[32] Miko O'Sullivan, *HTML Code Tutorial*, 2007, disponible:`http://www.htmlcodetutorial.com/`, accedida: julio 2008

[33] W3C schools, *JavaScript Tutorial*, 2009, disponible: `http://www.w3schools.com/JS/default.asp`, accedida: agosto 2008

[34] Joaquín García, *javaScript*, 2006, disponible: `http://www.webestilo.com/javascript/`, accedida: agosto 2008

[35] W3C schools, *CSS Tutorial*, 2009, disponible:`http://www.w3schools.com/css/`, accedida: setiembre 2008

[36] Joaquín García, *Manual de CSS*, 2006, disponible:`http://www.webestilo.com/css/`, accedida: setiembre 2008

[37] The PHP group, *PHP*, 2009, disponible:`http://www.php.net`, accedida: agosto 2008

[38] MySQL AB Sun Microsystems Inc., *MySQL*, 2009, disponible:`http://www.mysql.com`, accedida: octubre 2008

[39] The Apache Software Foundation, *The Apache Software Foundation-Meritocracy in Action*, 2009, disponible:`http://www.apache.org`, accedida: agosto 2008

[40] Canonical Ltd. Ubuntu and Canonical are registered trademarks of Canonical Ltd.Linux-based operating system, *ubuntu*, 2009, disponible: `http://www.ubuntu.com/`, accedida: agosto 2007

[41] roscripts.com, *roScripts BETA*, 2009, disponible: `http://www.roscripts.com/`, accedida: octubre 2008

[42] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *RFC2616 - Hypertext Transfer Protocol – HTTP/1.1*, junio 1999, disponible:`http://www.faqs.org/rfcs/rfc2616.html`, accedida: agosto 2008

[43] James Marshall, *HTTP Made Really Easy* , agosto 1997, disponible:`http://www.jmarshall.com/easy/http/`, accedida: agosto 2008

[44] Dynamic Network Services Inc., *DynDNS.com*, 2009, disponible: `http://www.dyndns.com`, accedida: julio 2008

[45] T. Berners-Lee, L. Masinter, M. McCahill, *Uniform Resource Locators (URL)*, Diciembre 1994, disponible: `http://www.ietf.org/rfc/rfc1738.txt`, accedida: octubre 2008

[46] MySQL AB Sun Microsystems Inc., *MySql developer zone*, 2009, disponible: `http://dev.mysql.com/`, accedida: noviembre 2008

[47] Avr-lib, *<avr/pgmspace.h>:Program Space Utilities*, Diciembre 2007, disponible: `http://www.nongnu.org/avr-libc/user-manual/group__avr__pgmspace.html`, accedida: Setiembre 2008

[48] Arash Partow, *General Purpose Hash Function Algorithms*, sin fecha de modificada, disponible: `http://www.partow.net/programming/hashfunctions/`,accedida: noviembre 2008

[49] Literate Programs, *Hash function comparison (C, sh)*, sin fecha de modificada, disponible: `http://en.literateprograms.org/Hash_function_comparison_(C,_sh)`,accedida: noviembre 2008

[50] NIST, *Cryptographic Toolkit*, julio 2008, disponible: `http://csrc.nist.gov/groups/ST/toolkit/`, accedida: noviembre 2008

[51] Phrogz, *String Hashing Algorithms*, Mayo 2005, disponible: `http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/142054`,accedida: noviembre 2008

[52] smallcode, *Hash functions: An empirical comparison*, noviembre 2008, disponible: `http://smallcode.weblogs.us/hash_functions`,accedida: noviembre 2008

[53] Das-Labor org, *Crypto-avr-lib/en*, noviembre 2008, disponible: `http://www.das-labor.org/wiki/Crypto-avr-lib/en`,accedida: noviembre 2008

[54] John J. G. Savard, *A Cryptographic Compendium*, 2003, disponible: `http://www.quadibloc.com/crypto/intro.htm`,accedida: noviembre 2008

[55] Oded Goldreich, *The Foundations of Cryptography - a two-volume book*, Mayo 2004, disponible: `http://www.wisdom.weizmann.ac.il/~oded/foc-book.html`, accedida: noviembre 2008

[56] CriptoRed, *Libro ElectrÃşnico de Seguridad InformÃątica y CriptografÃıa VersiÃşn 4.1 de 1 de marzo de 2006*, marzo 2005, disponible: `http://www.criptored.upm.es/guiateoria/gt_m001a.htm`, accedida: noviembre 2008

[57] J. Postel, *RFC768 - User Datagram Protocol*, agosto 1980, disponible: `http://www.faqs.org/rfcs/rfc768.html`, accedida: mayo 2008

[58] J. Postel , *INTERNET PROTOCOL*, setiembre 1981, disponible: `http://www.rfc-es.org/rfc/rfc0791-es.txt`, accedida: mayo 2008

[59] DARPA INTERNET PROGRAM, *RFC793 - Transmission Control Protocol*, setiembre 1981, disponible: `http://www.faqs.org/rfcs/rfc793.html`, accedida: mayo 2008

[60] David C. Plummer , *RFC826 - Ethernet Address Resolution Protocol: Or Converting Ne*, noviembre 1982, disponible: `http://www.faqs.org/rfcs/rfc826.html`, accedida: mayo 2008

[61] Charles Hornig , *RFC894 - A Standard for the Transmission of IP Datagrams over E*, abril 1984, disponible: `http://www.faqs.org/rfcs/rfc894.html`, accedida: mayo 2008

[62] J. Postel, *RFC867 - Daytime Protocol*, mayo 1983, disponible: `http://www.faqs.org/rfcs/rfc867.html`, accedida: mayo 2008

[63] J. Postel, K. Harrenstien , *RFC868 - Time Protocol*, mayo 1983, disponible: `http://www.faqs.org/rfcs/rfc868.html`, accedida: mayo 2008

[64] T. Socolofsky, C. Kale, *RFC1180 - TCP/IP tutorial*, enero 1991, disponible: `http://www.faqs.org/rfcs/rfc1180.html`, accedida: mayo 2008

[65] C. Madson, R. Glenn, *RFC2403 - The Use of HMAC-MD5-96 within ESP and AH*, noviembre 1998, disponible: `http://www.faqs.org/rfcs/rfc2403.html`, accedida: noviembre 2008

[66] C. Madson, R. Glenn, *RFC2404 - The Use of HMAC-SHA-1-96 within ESP and AH*, noviembre 1998, disponible: `http://www.faqs.org/rfcs/rfc2404.html`, accedida: noviembre 2008

[67] G. Klyne, C. Newman, *RFC3339 - Date and Time on the Internet: Timestamps*, julio 2002, disponible: `http://www.faqs.org/rfcs/rfc3339.html`, accedida: junio 2008

[68] D. Mills, *RFC1361 - Simple Network Time Protocol (SNTP)*, agosto 1992, disponible: `http://www.faqs.org/rfcs/rfc1361.html`, accedida: abril 2008

[69] D. Mills, *RFC2030 - Simple Network Time Protocol (SNTP) Version 4 for IPv*, octubre 1996, disponible: `http://www.faqs.org/rfcs/rfc2030.html`, accedida: abril 2008

[70] R. Braden, D. Borman, C. Partridge, *RFC1071 - Computing the Internet checksum*, Setiembre 1988 , disponible: `http://www.faqs.org/rfcs/rfc1071.html`, accedida: abril 2008

[71] ATMEL Corporation, *AVR035: Efficient C Coding for AVR*, Enero 2004, disponible: `http://www.atmel.com/dyn/resources/prod_documents/doc1497.pdf`, accedida: Diciembre 2008

[72] ATMEL Corporation,*AVR109: Self Programming*, junio 2004, disponible: `http://www.atmel.com/dyn/resources/prod_documents/doc1644.pdf`, accedida: Febrero 2009

[73] ATMEL Corporation,*AVR106: C functions for reading and writing to Flash memory*, agosto 06, disponible: `http://www.atmel.com/dyn/resources/prod_documents/doc2575.pdf`, accedida: Febrero 2009