INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
Campus Ciudad de México

Escuela de Graduados en Ingeniería

Maestría en Ciencias Computacionales

Tesis

"Patrones y Estilos para Sistemas Embebidos"

Autor: Carlos Alberto Aceff Cordero
Asesor: Dr. José Martín Molina Espinosa

TECNOLÓGICO DE MONTERREY
BIBLIOTECA
Campus Ciudad de México

México D.F., diciembre de 2006

Contenido

1. Introducción 1
1.1. Definiciones Cortas de Algunos Términos Utilizados Frecuentemente en el Documento ... 2
1.2. Antecedentes Sobre los Sistemas Embebidos 3
1.3. Definición del Problema 4
1.4. Justificación y Valides de la Investigación 7
1.5. Alcances de la Investigación 8
2. Marco Teórico 9
2.1. Sistemas Auto-contenidos 10
2.2. Definición de patrones 13
2.3. Bases Teórica de la Ingeniería de Software Basada en Componentes 14
2.4. La Arquitectura de SW y Algunos Estilos Útiles 17
2.5. Listado de Patrones No Arquitectónicos Útiles Para los Sistemas Auto-Contenidos 42
2.6. Las Bases Existentes y el Control Remoto 65
3. Producto de la Investigación 66
3.1. Diseño Utilizado por la Empresa 66
3.2. Metodología Utilizada para Obtener la Nueva Propuesta 68
3.3. El Despachador 75
3.4. Interfaz con el Mundo Exterior y Lógica de Control 84
3.5. Arquitectura del Sistema 100
3.6. Validación 105
3.7. Pruebas Análisis e Interpretación de Resultados 106
4. Conclusiones 112
4.1. Recapitulación 112
4.2. Principales Contribuciones de la Investigación 113
4.3. Limitaciones 113
4.4. Trabajo a Futuro 114

Figuras

1. Representación del Estilo de Capas 23
2. Corazón Estándar de Software (*Standard Software Core-SSC-*) 24
3. Componente Requerido 26
4. Componente Genérico 26
5. Componente Adaptado 27
6. Filtros y Tuberías 29
7. Estructura para Compartir Información 32
8. Estructura Publicador Subscriptor 34
9. Modelo de un sistema operativo 36
10. Estructura de un Despachador de Tareas con Centinela de Fallas 37
11. Sistema centralizada utilizando el estilo cliente servidor 39

12. Despachador de Tareas Externo, con el estilo Cliente-Servidor 40
13. Bit de Formato 47
14. Despachador Cooperativo Tomado de un Patrón de Pont 51
15. Esquema de Ejecución Cooperativa 56
16. Porcentaje de Aprovechamiento del CPU del Despachador Cooperativo 58
17. Ejecución del Despachador Híbrido 61
18. Despachador Hibrido con Tarea Esporádica 61
19. Imprecisión en el Periodo de un Despachador Híbrido 62
20. Despachador Híbrido con Interrupción Planeada 63
21. Despachador de Tareas con Ejecución en *Background* 63
22. Tarea que Muestrea Durante la Transmisión de un Comando 67
23. División por Gradas del Sistema 68
24. Abstracción del Sistema 70
25. Repartición de los Procesos del Sistema en el CPU 73
26. Hilo Principal del Sistema 76
27. Código Equivalente 77
28. Hilos Despertadores 78
29. Código de los Hilos Despertadores 79
30. Hilo de Detección de *Transponder* en las Cercanías 80
31. Hilo de Transmisión de Comandos 81
32. Código del Hilo de Transmisión 82
33. Funciones de Soporte para el Llenado del *Buffer* de Transmisión 83
34. Diagrama de Actividad de Rutina "*Debounce*" 85
35. Código que Efectúa el "Debounce" de 1 Botón 86
36. Código que Efectúa el "*Debounce*" de Cada Botón 86
37. Diagrama de Estados de la Funcionalidad "*Remote Engine On y OFF*" 88
38. Decodificador de Secuencia de Programación 89
39. Maquina de Estados para Formar Mensaje a Transmitir 90
40. Maquina de Estados para Formar Mensaje a Transmitir 91
41. Código del Decodificador de Eventos Externos 96
42. Decremento de Temporizadores Virtuales 97
43. Código de la Máquina de Estados que Crea los Paquetes a Enviar 99
44. Vista Modular del Sistema 100
45. Módulo Descriptor de Conexiones 103
46. Descomposición en Subsistemas 104
47. Aplicación del patrón *Guarded Wait* 106
48. Poniendo la Bandera del *Guarded Wait* 107
49. Código que Pone a Dormir el Microcontrolador e Indica que está Dormido 108
50. Opción para Entrar en IDLE con el PCF7941 108
51. Macro para Borrar la Bandera de Estado IDLE 109
52. Hilo Despertador 109
53. Metodología de investigación 134

1. Introducción

La presente obra es un trabajo de investigación sobre formas de crear software para sistemas embebidos. Revisa técnicas de abstracción de sistemas, que son utilizadas para crear modelos que representen, con poca complejidad, la funcionalidad de un sistema compuesto por múltiples subsistemas. También explora algunas soluciones existentes, a problemas recurrentes en el área de sistemas auto contenidos. Con lo anterior, se tiene la finalidad de descubrir nuevos modelos de sistema, que sirvan para resolver problemas recurrentes, y difíciles de resolver para una industria mexicana de origen extranjero.

El trabajo está dividido en los siguientes 4 capítulos:

1. El capítulo uno es una introducción al problema a resolver.
2. El segundo capítulo es el marco teórico que se examinará para descubrir una solución innovadora.
3. El tercer capítulo trata sobre la solución innovadora.
4. El cuarto y último capítulo concluye sobre la solución obtenida.

Este capítulo considera la descripción del problema a resolver, sus orígenes, y el alcance esperado de la investigación.

1.1. Definiciones Cortas de Algunos Términos Utilizados Frecuentemente en el Documento

Antes de comenzar con la definición del problema a resolver, es necesario definir algunas palabras que se utilizarán con frecuencia en el resto del texto.

Se definirán los **sistemas auto-contenidos** ("*embedded*" o a veces conocidos como embebidos), en éste documento, como computadoras de propósito específico, que están completamente encapsuladas por el dispositivo que controlan, y que requieren ejecutar tareas periódicas con restricciones temporales en su ejecución; es decir, que tienen tareas que deben ser ejecutadas en un límite de tiempo o con una frecuencia específica.

Ejemplos de estas computadoras son las que controlan los sistemas automotrices, o las que controlan los sistemas de navegación de los aviones. Estas computadoras cuentan generalmente con recursos limitados de memoria temporal, (RAM) y de capacidad de procesamiento, y por lo general su producción en masa es barata comparada con la producción de una PC.

Además, en éste documento, se definirá en detalle, en las secciones siguientes, el concepto de **arquitectura de software**. Por el momento, se define como un conjunto de elementos de software que se relacionan entre si para forman estructuras que sirven para modelar, (o representar) subsistemas de software.

Finalmente, se definen los **patrones**, como soluciones documentadas, y probadas, a problemas recurrentes. Estas pueden ser soluciones de modelación de sistemas[1], de diseño de sistemas o de implementación de software.

[1] También conocidos como estilos

1.2. Antecedentes Sobre los Sistemas Embebidos

Debido a que el presente trabajo trata sobre sistemas embebidos, es necesario hacer una breve introducción a éstos.

Uno de los primeros sistemas auto-contenidos, utilizados en el continente americano, fue la computadora de guía *Apollo*, desarrollada en el laboratorio de instrumentación de la Universidad de tecnología de Massachussets. Este sistema fue utilizado para controlar algunos de los dispositivos de navegación de vehículos enviados a la luna; sin embargo, éstos no fueron producidos en masa hasta 1961, cuando se manufacturó un sistema embebido que se encargaba de guiar misiles.

En estas aplicaciones, el costo de dichos sistemas era muy elevado y el principal problema tecnológico radicaba en el hardware. A partir de los años 70's, con la introducción de los procesadores de 8bits, el poder de procesamiento de estas máquinas comenzó a aumentar y a abaratarse. Ahora, el desafío ya no se encuentra más en el desarrollo de hardware barato, sino en el desarrollo de software de bajo costo que sea capaz de controlar sistemas complejos y que pueda ser desarrollado rápidamente. Por lo que una posible solución, es la reutilización de elementos de software, (a nivel modelo o implementación), haciendo que el proceso de diseño cambie de construcción total del sistema, a reutilización y ensamblaje de piezas.

1.3. Definición del Problema

Esta investigación está orientada al desarrollo de software para la industria automotriz. Más específicamente, al área de los controles remotos, que son utilizados para activar funcionalidades tales como: el abrir y cerrar puertas, activar y desactivar la alarma, o el encender y apagar el motor del automóvil[2]. Dichos sistemas, transmiten por radio frecuencia los comandos a un receptor en el automóvil, siendo este último el encargado de ejecutar las órdenes del control.

El problema ha sido propuesto por ingenieros de la empresa Siemens V.D.O, que es una sucursal localizada en Guadalajara, de una matriz alemana, que hace diseño y producción de computadoras automotrices. Trabaja módulos de *body*, *restraint, cluster,* y otros, para clientes como G*eneral Motors*, *Ford*, y algunas otras japonesas.

Con las definiciones presentadas en 1.1, el tema de investigación se limitará a las siguientes áreas:

1. **Los patrones de diseño de sistemas auto-contenidos para el sector automotriz.**
2. **Los patrones arquitectónicos de sistemas auto-contenidos para el sector automotriz.**

Concentrándose en crear modelos del control remoto (patrones y estilos), que produzcan implementaciones que puedan ser fácilmente reutilizables en diferentes proyectos.

El enfoque principal estará en los planos de representación de los sistemas embebidos, y no en su implementación; no obstante, para demostrar viabilidad y obtener retroalimentación sobre el proceso de transformación modelo-código, se implementará, en C[3], una versión del sistema.

---

[2] El diseño de un control remoto para automóvil consiste en censar los botones presionados por el usuario y emitir un comando de radiofrecuencia de acuerdo a la combinación de botones usados.
[3] Utilizando un compilador de Keil para el 8051

Los planos, (patrones), deberán guiar la construcción de software que se pueda adaptar fácilmente a diferentes plataformas de hardware. Además, los elementos de software que poseen los comandos a transmitir, deberán ser fácilmente reemplazables.

1.3.1. Los Problemas Presentados por Siemens

Son dos los problemas a resolver[4]:

1. Presentar un diseño para un control remoto, que cumpla con los siguientes requerimientos:
    a. El sistema deberá detectar cada vez que un botón se presiona, y activar una rutina de envío de comando RF.
    b. El control remoto se alimenta con baterías, por lo que es necesario que el sistema ahorre energía, apagando ciertas funcionalidades, siempre que sea posible.
    c. El microcontrolador del sistema, además de ser usado como control remoto, también se utilizará como transmisor receptor de proximidad (*transponder*). De manera que cuando éste se encuentre cerca de una central transmisora-receptora, el dispositivo deberá terminar su funcionalidad de control remoto, e iniciar su funcionalidad como transmisor-receptor de proximidad.
    d. Existen combinaciones temporizadas de botones que causan la emisión de códigos de radiofrecuencia especiales, y éstas se deberán detectar. Para mayor información sobre las secuencias a detectar consultar el apéndice B:

2. El segundo problema a resolver es hacer lo anterior, procurando que los planos de

[4] Ver apéndice B para mayor información

diseño guíen implementaciones que se puedan reutilizar en diferentes proyectos. Para lograrlo, es necesario que el *hardware* que conforma los botones no forme parte del sistema, y que las secuencias temporizadas sean fácilmente reemplazables.

1.3.2. Precondiciones

1. Por motivos de confidencialidad, no fueron proporcionados los comandos de radiofrecuencia, por lo que esta funcionalidad se dejará en un nivel abstracto.
2. El software se pretende ejecutar en el microcontrolador-transmisor/receptor PCF7941 de Semiconductores *Philips*,[36] que es un pequeño chip de 20 terminales de conexión, que cuenta con los circuitos necesarios para generar las señales de radio frecuencia, transmisor/receptor. Es un microcontrolador sumamente pequeño de arquitectura RISC de 8 bits, y únicamente cuenta con 128Bytes de memoria RAM, 4KB de memoria ROM, y 512Bytes de memoria EEPROM para calibraciones.

1.3.3. Requisitos No Funcionales

1. Cuando el control remoto no se encuentre ejecutando tareas, se deberá apagar cierta funcionalidad del microcontrolador (poner al CPU en estado *idle*), con el fin de ahorrar energía.
2. Las entradas del control remoto deberán pasar por un filtrado de software antes de confirmar la activación de un botón. Lo anterior es con la finalidad de evitar falsas activaciones debidas al rebote eléctrico producido en los interruptores[5].
3. Por requerimientos de la empresa donde se realiza la investigación, la documentación debe estar hecha en UML, que se presumirá como un lenguaje formal. (Ver referencia

[5] Ver el patrón "*debounce*" en [25] para más información.

[37])

1.4. Justificación y Valides de la Investigación

1. Uno de los productos de esta investigación es la documentación de la resolución de un problema real y recurrente en el área de software automotriz; ésta, por su intrínseca naturaleza abstracta, podrá ser reutilizable en proyectos similares, pudiendo causar ahorros monetarios a la empresa receptora de la investigación, ya que el diseño no se deberá comenzar desde el principio cada vez que se requiera un sistema con características similares.
2. El segundo producto de esta investigación será la documentación de un patrón arquitectónico, que podrá ser utilizado por la empresa como un mecanismo de comunicación eficaz entre personas con diversos conocimientos técnicos, pero que requieren saber sobre el desarrollo de un producto.

1.5. Alcances de la Investigación

1. Encontrar un patrón de diseño que sirva como guía para implementar un control remoto con las características mencionadas en 1.3 más atrás.
2. Extraer y documentar, del patrón de diseño, un patrón arquitectónico.

2. Marco Teórico

El capítulo del marco teórico trata resume los conocimientos actuales[6] que existen sobre el desarrollo de software para sistemas embebidos, que puedan ser aplicados al sistema del control remoto.

En este capítulo, se incluirá lo siguiente:

1. Definición de los Sistemas Auto-Contenidos: Que será útil para caracterizar el contexto en el que el control remoto está inmerso.
2. Definición de los Patrones: Que extenderá la definición antes proporcionada.
3. Bases Teóricas de la Ingeniería Basada en Componentes: Que listará características necesarias para que un software pueda ser construido mediante componentes que se puedan reutilizar.
4. La Arquitectura de Software y Algunos Estilos Útiles para el Sistema a Desarrollar: Que tiene información actual sobre estilos utilizados en sistemas embebidos.
5. Un Listado de Patrones No Arquitecturales Utilizables en Sistemas Embebidos: Que contiene soluciones probadas para resolver problemas recurrentes en el área de codificación de sistemas embebidos.

[6] Al menos los que pudieron ser accedidos

2.1. Sistemas Auto-contenidos

Los sistemas auto-contenidos son dispositivos que se encargan de controlar uno, o más procesos, del "entorno real" donde están inmersos. Algunos lugares donde pueden ser encontrados son: lavadoras, equipo médico, automóviles, televisores, cajas de control de maquinaria industrial, etc. Los procesos, del "entorno real", donde son aplicados, varían desde lavar 5 kilos de ropa, hasta controlar la temperatura dentro de un automóvil. Éstos, a diferencia de las computadoras personales, están diseñados con un propósito de control específico, y todo el hardware y software necesario para cumplir su función está incluido dentro del dispositivo, siendo pocas las excepciones donde se puede agregar nuevo software o hardware para extender su funcionalidad.

El hardware, de dichos dispositivos, varía de acuerdo a la aplicación, y puede estar compuesto, por ejemplo, por un circuito analógico o digital de control, un microcontrolador con un programa, un micro procesador junto con su memoria volátil y no volátil, un chip de aplicación específica, cuyo hardware está grabado desde fábrica (*Application Specific* IC o ASIC), un arreglo de compuertas lógicas alambradas dentro de un chip para controlar algún proceso, (*Field-programmable gate array* o FPGA), un dispositivo de procesamiento de señales (DSP), u otros.

Con lo que respecta al software, éste es dependiente del hardware, y sus formas varían desde la construcción de los sistemas con redes de *Petri*, (para sistemas que utilicen tecnología del tipo FPGA[7]), hasta software para los microcontroladores y procesadores. En este trabajo me referiré únicamente a los sistemas basados en microcontroladores.

[7] Si desea más información sobre su diseño vea la referencia [16], y si desea aplicar un método formal para verificar la correctés de este tipo de diseño, vea la referencia [7], si desea el código en C, de verificación formal del método por modelos, (*model checking*, actualmente muy utilizado), vea el capítulo 4 de la referencia [5]

Estos sistemas generalmente se programan en ensamblador, C, C++, o mediante una herramienta de desarrollo por modelos como Matlab. Cuentan con recursos muy limitados de memoria y es difícil utilizar lenguajes que utilicen máquinas virtuales como Java. [27]

Generalmente éstos controlan diversos dispositivos conectados a su exterior, por ejemplo, en el sistema de aire acondicionado de un automóvil, (*Heating Ventilating and Air Conditionting* o HVAC), controlan motores que abren o cierran compuertas de aire caliente o frió, encienden y apagan el compresor de gas que enfría el aire, y posiblemente también encienden y apagan ventiladores auxiliares. Debido a lo anterior, estos controladores, deben manipular simultáneamente diferentes dispositivos de salida, además de su lógica de control, haciendo generalmente necesaria la presencia de un despachador de tareas, (en este texto también conocido como "sistema operativo") que le indique al CPU del microcontrolador, cuáles tareas ejecutar, y en que momento.

Existen diferentes formas de despachadores, como los mono-tónicos de prioridad fija para tareas in-interrumpibles, que ejecutan primero las tareas de frecuencia más pequeña, y luego las de frecuencia mayor; así, si el despachador tiene que decidir entre correr una tarea que debe ejecutarse cada 20 milisegundos (ms), y otra cada 100ms, elegirá ejecutar primero la de 20ms, y luego la de 100ms. Si las dos tienen la misma frecuencia de ejecución, elegirá aleatóriamente, o con una preferencia preasignada.

Otro, más complicado, busca, entre las tareas disponibles, la que tenga más urgencia por ser ejecutada, y de ser necesario interrumpe la ejecución de la actual, y ejecuta la de mayor prioridad.

El despachador se hace más complejo al ser requerida la ejecución de tareas esporádicas, ya que éste debe buscar un momento preciso para ejecutarlas, sin retrasar a las demás.

Generalmente los despachadores son un elemento fundamental en los sistemas auto-contenidos, y su tamaño, respecto del resto del software debe ser pequeño. Tomando algunos bytes en sistemas poco exigentes, y cientos, en sistemas más demandantes.

2.2. Definición de patrones

Los patrones son soluciones documentadas a problemas recurrentes encontrados en el área de ingeniería de software. Éstos, generalmente describen el problema a resolver, el contexto en el que el problema debe estar inmerso para que el patrón funcione, y el método, previamente probado, para solucionarlo.

Existen diferentes tipos de patrones, como son: los de estructura, y de codificación. Para este trabajo se estudiaran los estructurales, mejor conocidos como estilos arquitectónicos, que son descripciones de relaciones entre elementos de software. También mencionaré algunos patrones no estructurales, que son consejos para implementar soluciones.

2.3. Bases Teórica de la Ingeniería de Software Basada en Componentes

El objetivo primordial de la ingeniería de software basada en componentes, es la construcción de sistemas mediante el ensamblado de un conjunto finito de piezas, (o componentes) preconstruidos. De ésta forma, los sistemas pueden ser rápidamente terminados, y gozar de una homogeneidad en su calidad. Sin embargo, para alcanzar este objetivo, es necesario construir piezas de software con características especiales.

Una característica que deben tener éstos componentes es: que la dependencia hacia otros componentes sea configurable.[8] Por ejemplo, suponga que existe un componente que envía un *email*, y que depende de otro que le pasa el mensaje a enviar. Si los dos componentes están fuertemente unidos, es decir, que es muy difícil configurarlos para que cada uno funcione por separado con un substituto, entonces la reutilización de los componentes se dificulta.

Otra característica que deben tener éstos, es que se puedan ensamblar unos con otros. Dos o más componentes se podrán ensamblar si sus protocolos de comunicación son compatibles. Es decir, que un componente entienda, sin ambigüedad, la información que el otro le envía, (ya sea un comando de activación o simple transmisión de información). Para resolver esta restricción, generalmente se crea un estándar, que todos los componentes siguen.

De esta forma, se pueden utilizar simuladores para probar el correcto funcionamiento del componente. Primero verificando la interfaz de comunicación, y luego la lógica de funcionalidad de éste.

Por último, los componentes deberán estar bien documentados, de manera que su búsqueda se

[8] Un componente depende de otros, cuando éste requiere el funcionamiento correcto de otro para operar correctamente

facilite. Esta documentación, por lo menos, deberá indicar cuales servicios son proveídos, y qué servicios externos son requeridos. Así mismo, con el fin de dotar información relevante sobre el funcionamiento, se puede agregar una sección, en la documentación, que contenga un conjunto de premisas que el componente asuma como válidas, de esta forma, se puede establecer un contrato de servicios a condición de premisas; dichas premisas, describirán condiciones previas sobre la información que el componente debe de recibir; si dichas condiciones son válidas, las condiciones posteriores se cumplirán, además de que las condiciones invariantes, siempre se garantizarán. Esta forma de contrato puede garantizarse con lenguajes de programación que admitan aserciones, y que obliguen el cumplimiento de las premisas antes de compilar. Estando la documentación incluida en el código de implementación.

Cada componente debe tener un tamaño adecuado. Si se engloba mucha funcionalidad, es posible que se encapsule código que sea requerido en otras partes del sistema, haciendo que 2 elementos requieran ejecutar el código y variables al mismo tiempo, gastando recursos de memoria innecesariamente. Por otro lado, si el componente es muy pequeño, se generará una estructura de dependencias complicada, que puede dificultar la acción de poner y quitar componentes para probar cual queda mejor.

Para poner en marcha la construcción de sistemas basados en componentes, es necesario contar con un proceso de desarrollo; el que además de tener pasos como: definición de requerimientos, asignación de tareas, y otros, regularmente requeridos en cualquier desarrollo de sistemas, se incluyan los siguientes pasos: identificación de componentes, adaptación de requerimientos[9], selección de piezas de ensamble, y finalmente, ensamblado.

Para concluir con esta sección, se menciona que la construcción de sistemas basados en componentes implica la resolución de muchos problemas, como son: el aseguramiento de selección de componentes de alta calidad, la resolución de conflictos entre los requerimientos y las piezas disponibles, la resolución de incompatibilidades de interfaces entre dos o más componentes que se utilicen, etc.

Sin embargo, ésta habilita la construcción rápida de sistemas de software, y el aseguramiento de calidad, (si las piezas con las que se cuenta son de alta calidad).

[9]Requerido porque los componentes disponibles generalmente imponen restricciones sobre su compatibilidad con otros módulos.

2.4. La Arquitectura de SW y Algunos Estilos Útiles

Se sentarán las bases de la arquitectura de software, tal como se referirán en esta investigación, y a continuación se presentará un listado de patrones arquitectónicos, (estilos), que son útiles para el desarrollo de sistemas auto-contenidos.

La arquitectura de software es un tema nuevo de estudio, y aún no existe una única definición; por lo que para este trabajo utilizaré la siguiente: Actividad modeladora de sistemas, donde se estudia cómo dividir un software, en un conjunto de elementos arquitectónicos, que interactúen entre si, y que aporten propiedades específicas al sistema. La arquitectura de software estudia las estructuras de dichos elementos, y no se interesa en la implementación de ellas; en lugar, investiga cómo deben ser sus relaciones para aportar las cualidades especificadas.

Algunos autores, ([6] y [2] entre otros) proponen estudiar por separado tres grupos de elementos, otros, proponen otras cantidades. Pero todos concluyen en que los elementos deben de tener propiedades y relaciones diferentes, de manera que provean cualidades ortogonales. Para esta introducción, presentaré los tres elementos de la referencias [6] y [2].

El primer tipo es conocido como módulo, que es todo software visto como funcionalidad, cuya principal característica es que se puede descomponer en pedazos más pequeños, que un componente puede heredar de uno o más padres[10], y que puede existir interdependencia entre dos o más componentes[11]. Por ejemplo, en un sistema de navegación para aviones puede estar conformado por el módulo que toma decisiones de navegación, que a su vez se descompone, por

[10] Se hace referencia al concepto de herencia definido en el paradigma de programación orientada a objetos. Es decir, que un módulo puede extender las características de funcionalidad o interfaz de otro, agregando sus propias

[11] El concepto de dependencia que se manejara es el siguiente: Un módulo tiene dependencias, si requiere de la presencia y correctés de otro u otros.

un lado, en un módulo de inteligencia artificial, y por otro, en uno de control de navegación.

De esta forma, una estructura modular reflejará las relaciones de composición, dependencia y tipo entre los elementos de un sistema, y puede responder a preguntas como las siguientes: ¿Qué subsistemas son parte de otros? ¿Cuáles dependen de un subsistema en particular? o ¿A que tipo de subsistema pertenece un determinado elemento?, etc.

El segundo grupo de elemento está conformado por los conectados y conectores, que es todo el software que en ejecución puede comunicarse con otros, por ejemplo: procesos paralelos comunicantes, procesos secuenciales comunicantes, subsistemas de red, etc. En un software de navegación para aviones, se encuentra el subsistema que comunica el estatus del avión al aeropuerto, el que se encarga de procesar y comunicar la información de los sensores al panel de la cabina, el que se encarga de almacenar en una base de datos las estadísticas de vuelo del avión, etc.

De esta forma, una estructura de conectados y conectores, reflejará cómo funciona la comunicación de los conectados, y puede responder a las siguientes preguntas: ¿Los elementos del sistema están comunicados en fila?; es decir, la salida de información de cada componente alimenta la entrada de exactamente un sucesor. ¿El elemento X transmite periódicamente su información?, o ¿Es que solo la transmite cuando otro elemento se lo pide?, etc.

El tercer grupo de elemento lo conforman las unidades localizable y los elementos de alojamiento, cuya principal propiedad es que la unidad pueda ser almacenada en uno o más elementos de alojamiento.

Una estructura de alojamiento, modelará en cuáles elementos están, (o deben estar) localizados las unidades. Los elementos de alojamiento pueden ser: procesadores, archivos, grupos de desarrollo, computadoras, etc., y las unidades localizables, cualquier forma de software que se pueda asignar. De esta forma, una tarea de control de un avión, puede estar asignado a un grupo

de desarrollo, y replicado en tres computadoras, ya que los elementos de alojamiento no necesitan estar directamente relacionados.

La configuración y uso que se les de a los elementos anteriormente mencionados, proporciona cualidades especiales al sistema. Por ejemplo, si se elije que un determinado proceso de control esté replicado varias veces en diferentes computadoras, se favorecerá la escalabilidad del sistema, ya que mientras uno está siendo ocupado, el otro, localizado en otra computadora, podrá atender nuevas peticiones. Por otro lado, si se elige dividir un módulo en un elemento de hardware y uno de aplicación, habrá más posibilidades de cambiar la plataforma electrónica que si la aplicación y la interfaz con el hardware están mezcladas en un solo módulo.

Los estilos presentados en el presente capitulo, son algunas configuraciones de estos elementos básicos, para proveer características especiales al sistema. A continuación se explicarán algunas cualidades y aplicaciones de la arquitectura de software.

Como modelo de referencia, la arquitectura de software es muy útil, ya que mediante sus estructuras se puede modelar sistemas, desde puntos de vista complementarios, y a un nivel abstracto. Además, y a pesar de la pérdida de detalles, se posibilita la respuesta a preguntas como: ¿Dónde colocar un nuevo módulo? ¿Cuál protocolo de comunicación debe utilizarse?, ¿Cuáles módulos fallarían si el módulo X falla?, etc.

Por otro lado, una vez que se cuente con una arquitectura, ésta se puede reutilizar en otros proyectos, haciendo innecesario tomar decisiones críticas una y otra vez, y que puedan poner en riesgo el desarrollo exitoso del sistema.

La arquitectura también puede servir como un documento de comunicación efectivo entre el personal que desarrolla el sistema, y la gente encargada de hacer negocios, u otras personas con cargos no técnicos, que requieren tomar decisiones sobre el desarrollo del producto. Esto se logra porque las estructuras abstraen una enorme cantidad de información técnica relacionada con la

implementación, pero pueden modelar la totalidad del producto, en forma que éste pueda ser fácilmente comprendido.

Sin embargo, basar el desarrollo de un producto, en un modelo estructural requiere tomar decisiones críticas en un tiempo muy prematuro en el desarrollo del sistema, y abre la posibilidad, (como todos los desarrollos tipo top-down) de incluir errores catastróficos en el producto cuando su detección es muy tardía, causando retrasos de tiempo y gastos de dinero.

Por lo anterior, se recomienda que sea un conjunto de ingenieros experimentados en el diseño, implementación, y verificación de sistemas auto-contenidos, quienes tomen éstas primeras decisiones. Además, es recomendable, que éstos tengan, y utilicen un catálogo actualizado de estructuras que hayan sido probadas con anterioridad. De lo contrario, pueden seguir un método de desarrollo que comience el proceso de diseño con las partes conocidas, y retrase las decisiones importantes a un tiempo cuando éstas sean fáciles de tomar. A continuación, los expertos deberán extraer, del sistema terminado, su arquitectura y documentarla para poderla utilizar en un futuro.

2.4.1. Patrones Arquitectónicos, (Estilos)

Los estilos descritos a continuación indican estructuras de software que pueden ser utilizadas en etapas previas al diseño del sistema, como es el caso del estudio mostrado en el capítulo 3 de la referencia [2], que aunque no fue usado en el producto final, (o al menos no en la forma descrita), ilustra muy bien algunos beneficios alcanzados en el prototipo en el que se probó.

Por otro lado, es necesario mencionar que las estructuras aquí presentadas no obstruyen, y pueden enriquecer, el desarrollo de sistemas generados con herramientas guiadas por modelos, (*Model Driven Architecture*) como son: *Matlab*, CIP [11] o *MetaH* [12]. Por otro lado, al ser patrones estructurales y no presentar detalles de implementación, no se contraponen al desarrollo y verificación con métodos formales actuales o en desarrollo, (para una breve introducción sobre métodos formales ver [3] y [4]), y en caso que el sistema fuera suficientemente grande para programarlo con algún lenguaje orientado a objetos como Java, (ver algunas restricciones impuestas sobre uso de lenguajes como Java en sistemas con poca memoria en [27]), se contaría con un amplio acerbo bibliográfico sobre estructuras utilizadas con el paradigma de la programación orientada a objetos.[12] Finalmente, es importante mencionar que no se utilizaran el formato de presentación de patrones: contexto, problema, solución, ya que los estilos, aunque son una especie de patrones, generalmente son descritos alrededor de las propiedades que pueden aportar. Dicha forma, se reservara para presentar algunos patrones no estructurales, sobre los que escribiré más adelante.

[12] Puede examinar la bibliografía mencionada en [6] si desea referencias

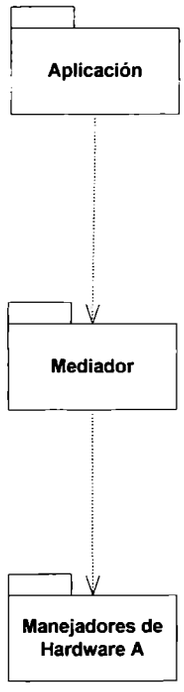
2.4.2. Estilo de Capas (Vista Modular)

Este estilo, facilita la realización de cambios en el software, además, promueve la portabilidad a diferentes plataformas de hardware. Esto se logra, agrupando el software en capas que tengan dependencias jerárquicas entre si; es decir, un grupo determinado, dependerá de su inmediato inferior, pero no de su superior.[13] Para cada capa, deberán establecerse los servicios que el grupo debe proveer, junto con sus reglas de acceso, de manera que el único requerimiento para cambiar cualquier capa por completo, sea el cumplimiento de dicha interfaz de servicios.

Éste, también permite organizar un sistema, de manera que exista una capa que amortigüe los cambios de interfaz de otras, (conocido en inglés como *middleware*). De esta forma, cada vez que las capas cambiantes modifiquen sus servicios, solamente se reemplaza el *midleware*, sin tener que reescribir las capas superiores a esta. Siendo el *midleware* la única capa que no puede cambiar los servicios que provee.

[13] Entendiéndose como dependencia, la cualidad de necesitar de la presencia y correcto funcionamiento de otros elementos.

Por ejemplo:



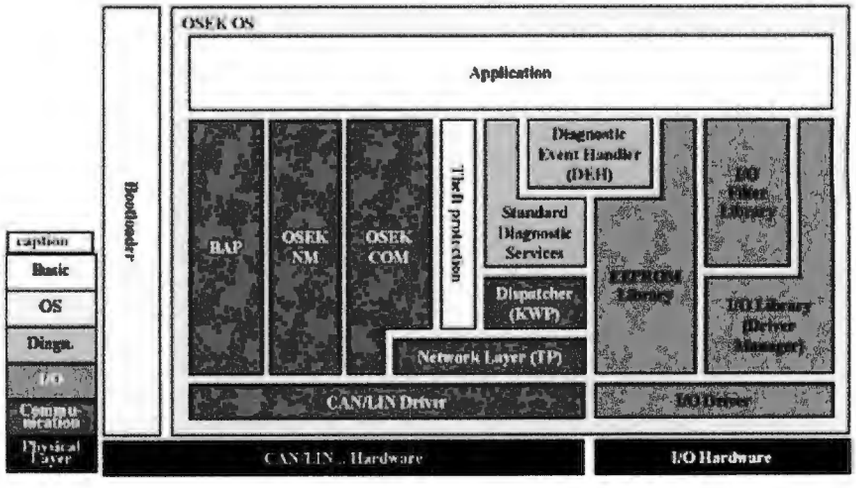
1. Representación del Estilo de Capas

En este estilo el mediador actúa como adaptador entre la plataforma de hardware y la aplicación.

Perforaciones entre capas, es decir que una capa requiera servicios de capas inferiores, pero no inmediata inferior, pueden ser aceptables en muchos sistemas, siempre y cuando no se abuse de ésta configuración. Una aplicación específica que muestra lo anterior, es el siguiente diagrama:[14]

[14] este modelo en el desarrollo de sistemas electrónicos automotrices es el siguiente, que fue tomado de la referencia [13] y que cuenta con una amplia explicación en [17]

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and or a fee.
EMSOFT'04 September 27–29, 2004, Pisa, Italy
Copyright 2004 ACM 1-58113-860-1/04/0009 ...$5.00.



2. Corazón Estándar de Software (*Standard Software Core-SSC-*)

A la izquierda se muestran los colores que corresponden a cada grupo de servicio. Así, se observan en negro los procesos, (del mundo real) con los que la capa de hardware interactúa. Dos colores hacia arriba de éste, etiquetado como I/O, se indica la capa que controla los dispositivos de entrada-salida del microcontrolador, como son: la memoria permanente (generalmente en forma de EEPROM), puertos de entrada-salida, convertidores analógicos digitales, y digitales analógicos, dispositivos de transmisión de datos síncronos y asíncronos, (como UARTs, USARTs, dispositivos *microwire*, etc.), etc. Un color hacia abajo del anterior, etiquetado como *communication*, se encuentra la capa de transmisión/recepción de red, que se encarga de manejar el hardware encargado de hacer conexiones entre computadoras. Finalmente, se encuentran el

sistema operativo, que se encarga de activar tareas, y los diagnósticos, que monitorean si los dispositivos del hardware operan correctamente. Juntos presentan sus servicios a la capa de aplicación.

Una descripción de una estructura no idéntica a la anterior, pero útil para sistemas con pocos dispositivos de hardware, es el siguiente:

1. Los servicios de transmisión/recepción de red, y entrada salida de la plataforma, mencionados con anterioridad, formarían una única capa de hardware.
2. Sobre la anterior capa, descansaran los servicios del sistema operativo, que atenderán las demandas de otras capas superiores, como la de diagnósticos y de aplicación.[15]

La combinación: sistema operativo – capa de hardware, generalmente ocupa de un 20% a un 25% del total de la memoria del sistema, y es conocida como *midleware*; Ésta generalmente se cambia junto con las plataformas de hardware, y tiene la cualidad de amortiguar el impacto de cambios en el hardware.

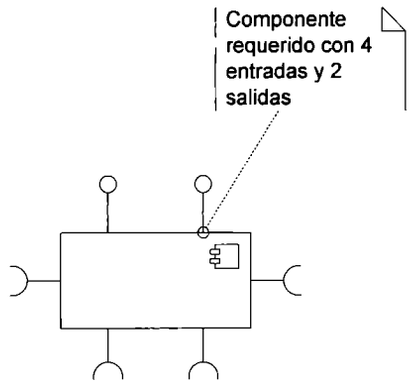
[15] Es importante mencionar que los grupos de aplicación y diagnósticos dependen del sistema operativo y no al revés, ya que este último, no requiere utilizar tareas específicas para funcionar correctamente, pero los primeros, si requieren del sistema operativo final para funcionar correctamente.

2.4.3. Adaptador (Elementos Modulares)

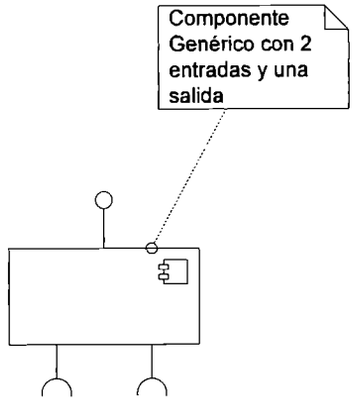
Este estilo es una modalidad del anterior, y está diseñado para facilitar la construcción de piezas reutilizables, separando las características generales, de las específicas.

Éste, regula que las piezas estén constituidas por los siguientes dos elementos:

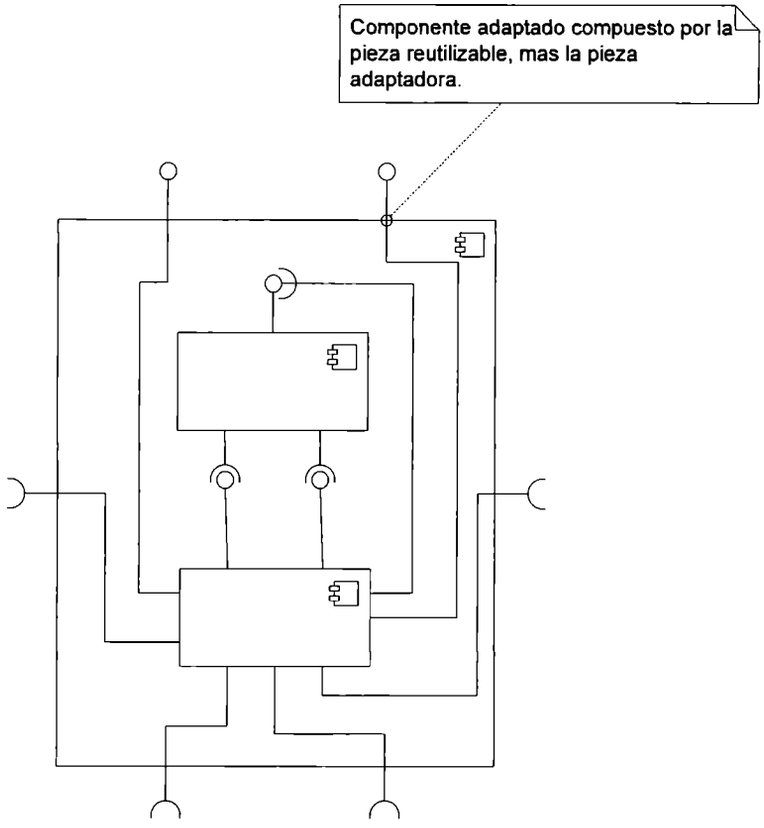
1) Un elemento genérico con la lógica a reutilizar, que será almacenado en la base de datos de piezas.
2) Un elemento específico, que será construido para adaptar la pieza genérica a un contexto particular. Siendo que ésta no necesita ser almacenada.



3. Componente Requerido



4. Componente Genérico



5. Componente Adaptado

Para ilustrar mejor la aplicación de este estilo se refiere al lector a la referencia [13], donde se utiliza este estilo para "factorizar" la funcionalidad del módulo de luz exterior.

Dicho modelo fue preparado teniendo en mente el costo para el productor de autos, y no para el proveedor de piezas electrónicas, y puede resultar en código y variables de operación duplicadas, lo que es poco deseable, porque el precio del producto se pudiera elevar. Sin embargo, una vez que el fabricante de autos ha mandado hacer los elementos, los podrá utilizar por mucho tiempo, y a través de muchas versiones y modelos, ahorrándose gastos de reingeniería de software.

Para lograr la separación entre la interfaz de comunicación y la lógica de la aplicación, es necesario utilizar un método de desarrollo como el de Jackson, (ver referencia [15] sobre el método Jackson).

Jackson propone comenzar el desarrollo analizando el entorno con el que el sistema computacional interactuará, que en el caso de sistemas auto-contenidos, son factores físicos como velocidad, aceleración, etc., o acciones del usuario. A partir de este estudio, se deberá obtener el elemento de software encargado de recibir y emitir eventos que interaccionen con el entorno real. Éste, tiene la función de conocer los protocolos de comunicación con el mundo exterior; sin embargo, su funcionalidad se limita a traducir eventos del entorno externo, y producir acciones de control sin aplicar lógica alguna.

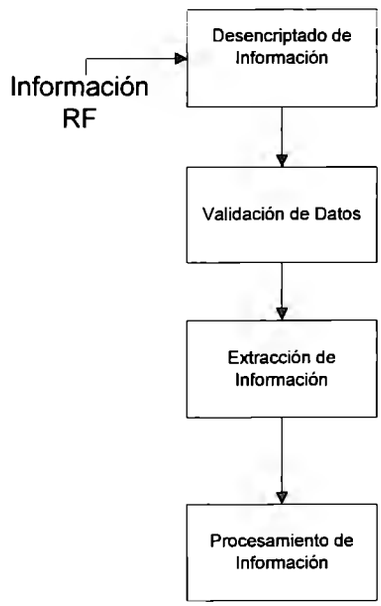
La lógica de control se obtiene en el siguiente paso del método; ésta observa los eventos del entorno real, y despacha acciones de control. Ésta es el resultado de un análisis de funcionalidad deseada, que deriva en un conjunto de procesos, que son simplificados para reducir su número.

2.4.4. Filtros y Tuberías (Elementos: Conectados y Conectores)

Éste esta pensado para ser aplicado en subsistemas que transforman datos, haciendo fácil la adición y substracción de elementos transformadores. Para lograr esto, cada lógica de transformación, se separa en elementos llamados filtros, y se crean piezas de comunicación, llamadas tuberías. Cada uno de los filtros sigue el mismo protocolo de comunicación, y las tuberías funcionan como medio de comunicación para los filtros.

Existen diferentes estructuras, por ejemplo las acíclicas, donde no existe ningún camino que comunique la salida de información de un filtro con su entrada.

Un ejemplo de tubería es el buffer, que almacena la información y permite su escritura y lectura de una manera estándar. Este modelo permite a los filtros trabajar a velocidades independientes.



6. Filtros y Tuberías

Una característica de este agrupamiento, es que facilita las adaptaciones requeridas para pasar de un procesador único, a una estructura de multiprocesadores, que generalmente es requerida cuando se necesita extender la respuesta a entradas más dinámicas; por ejemplo para ampliar la frecuencia de mediciones de un osciloscopio. Para lograr esto, se distribuyen las tareas en 3 filtros, de modo que cada uno pueda procesar su información en paralelo, y de forma más veloz que con un solo procesador. Reemplazando 1 súper microcontrolador por 3 de menor costo y capacidad.

Así, cada elemento podrá resumir sus operaciones de procesamiento, sin tener que esperar a que todo el ciclo:

1. adquisición
2. control
3. salida

haya terminado. Basta con que termine de procesar sus datos de entrada y los envié a la tubería, para poder reanudar sus operaciones específicas.[16]

[16] La referencia [30], trata de un dispositivo que automáticamente convierte el código de una estructura de filtros y tuberías uni-procesador en un sistema multi-procesador.

2.4.5. Información Persistente Compartida (Elementos Conectados y Conectores)

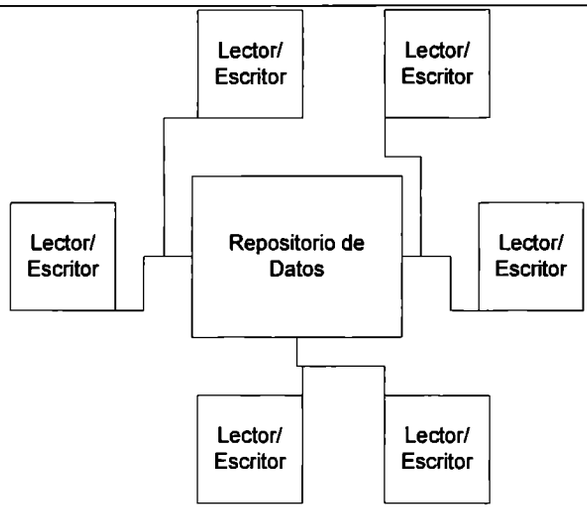
Esta estructura, facilita la construcción de subsistemas que accedan a datos persistentes. Para lograrlo, se centraliza la información, y se crea un controlador de ésta. Así, las lógicas de regulación de acceso, búsqueda y almacenamiento de datos, se factorizan y se mueven de las aplicaciones, al manejador central.

El manejador central puede comunicar información de dos formas diferentes con los elementos demandantes de información:

1) Los demandantes hacen una petición al controlador de información y éste responde con la información pertinente.
2) Cuando algún elemento actualiza los datos, el manejador se encarga de seleccionar demandantes, para avisarles que la información, de la que ellos dependen, ha cambiado.

La primera de ellas, es la configuración más sencilla, y puede ofrecer, más o menos, la misma funcionalidad que la segunda. La segunda, requiere incluir código que avise a los demandantes que su información ha cambiado; sin embargo, abre la posibilidad a nuevos usos, como la ejecución de tareas concurrentes, que funcionen en equipo, y que efectúen transformaciones sobre una única celda de datos. [29]

En los sistemas auto-contenidos, este estilo generalmente se aplica cuando se crea un controlador de memoria, que factoriza el código de lectura-escritura de los elementos demandantes.



7. Estructura para Compartir Información

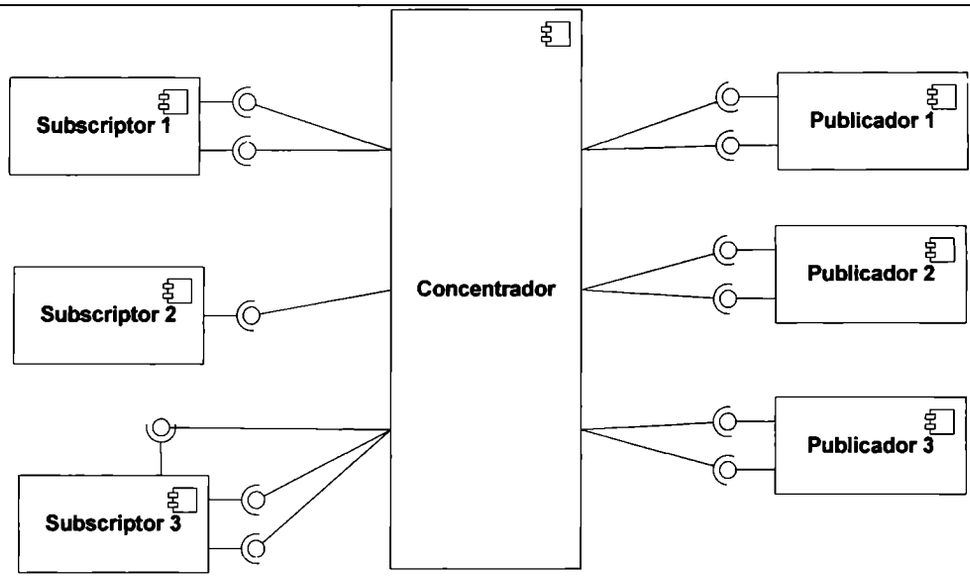
2.4.6. Publicador-Subscriptor (Elementos Conectados y Conectores)

Esta es una estructura de comunicación, que concentra las dependencias entre módulos, en un elemento central. De esta forma, dependencias complejas, como la que se puede dar cuando un elemento depende de otros 3 y estos 3 dependen a su vez de otros dos, pueden ser simplificadas a la dependencia a un solo elemento.

La estructura funciona de la manera siguiente:

1) Se agrega el controlador de dependencias al sistema, y se le añade una lista de evento-receptor, que es utilizada para disparar notificaciones de ocurrencia de eventos, a los elementos dependientes, cada vez que un evento relacionado con un dependiente, ocurra.
2) Cuando un módulo dependiente es agregado al sistema, éste añade nuevos elementos a la lista evento-receptor donde queda subscrito.
3) Cuando un evento en la lista ocurre, el controlador se encarga de notificar a sus dependientes. Esta notificación puede tomar la forma de un llamado a un procedimiento, o simplemente una indicación con una variable boleana.

Este estilo promueve la adición y substracción de elementos conectados a un sistema, ya que la forma de comunicación con los otros módulos, (interfaz), es estándar, y la dependencia es hacia un único elemento.



8. Estructura Publicador Subscriptor

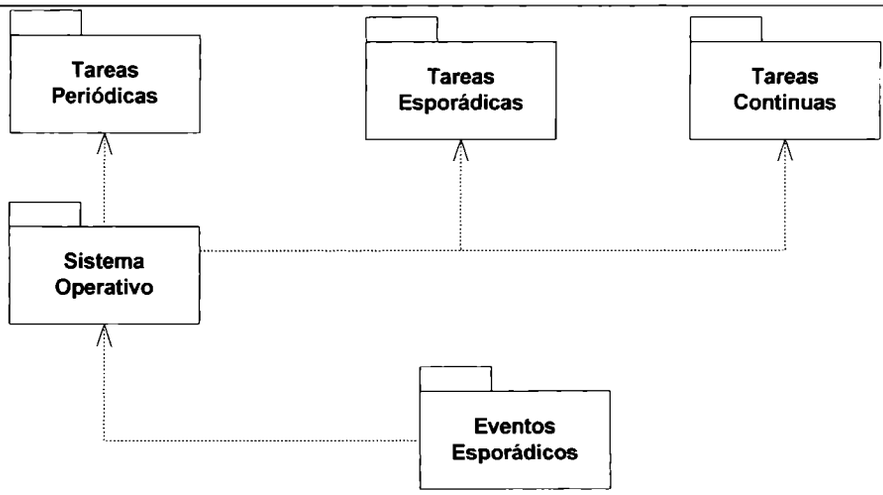
2.4.7. Despachador de Tareas con Monitor de Fallas (Elementos Modulares y Conectados y Conectores)

Anteriormente (2.4.2), se mostró un sistema de capas, donde el sistema operativo se encontraba sobre la capa de comunicaciones y hardware, sin embargo, el despachador de tareas puede incluir aquellas, e imponerles su propio contrato de servicios, que puede ser más flexible que otro que tenga que ajustarse a todas las situaciones posibles, como el que se vio en el segundo caso. A continuación una breve descripción de la comunicación de un despachador de tareas, simple, con los elementos que lo rodean.

Debido a que la discusión es a nivel estructural, no se requiere explicar el funcionamiento interno del despachador, haciendo posible que éste pueda cambiar, mientras cumpla con los requisitos de conectividad externa, y pueda activar los siguientes tipos de tareas:

1. Tareas periódicas que requieren ser activadas con frecuencias fijas.
2. Tareas esporádicas que tienen un tiempo mínimo de separación entre ejecución y ejecución. Para que el despachador se entere que debe ejecutar una de estas tareas, debe recibir una notificación de un evento esporádico, y debe responder programando su ejecución.
3. Tareas continuas, (normalmente conocidas como de background), que son ejecutadas cuando no hay tareas periódicas o esporádicas por realizar.

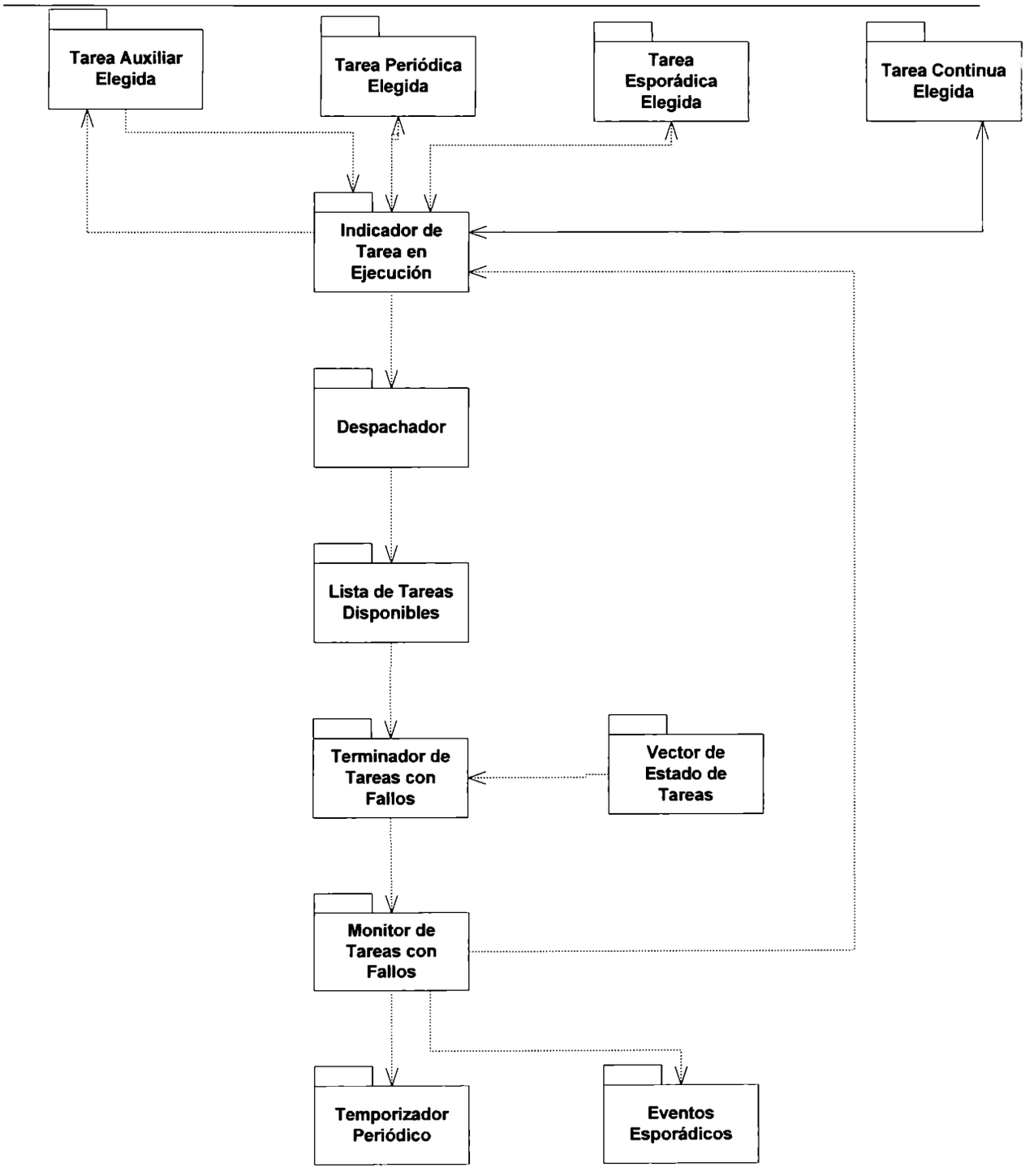
Entonces, la estructura que deberá tener, es la siguiente:



9. Modelo de un sistema operativo

A la que se le pueden agregar dependencias circulares para obtener un sistema capaz de detectar, y corregir tareas, que debido a alguna falla, no retornan el control al despachador, funcionando de la siguiente manera:

1. El despachador agrega el identificador de la tarea, que está a punto de ejecutar, en la zona de indicadores.
2. La tarea es ejecutada.
3. Cuando la tarea es terminada, se elimina el identificador agregado en el paso 1.
4. Una notificación periódica para ejecutar la siguiente tarea, interrumpe al CPU.
5. El monitor de tareas con fallas verifica que no existan identificadores que indiquen tareas sin terminar.
6. Si existe algún identificador, se pasa el control al terminador de tareas, que se encarga de restaurar el contexto de operación del CPU, tal y como se encontraba antes de ejecutar la tarea conflictiva.



10. Estructura de un Despachador de Tareas con Centinela de Fallas

Para más información sobre la implementación de un centinela de fallas, y sus pruebas, vea la referencia [14], que trata sobre la implementación de un sistema similar, utilizado un microcontrolador ARM.

2.4.8. Estilo Modelo Vista Controlador (Elementos Conectados y Conectores)

Una de las principales razones de su uso, es la separación de la presentación de información del resto de la aplicación, por lo que es útil para facilitar cambios en la presentación, sin afectar el resto del sistema. Para que esto sea posible, el sistema se divide en los siguientes 3 grupos:

1) Grupo de presentación, o vista del sistema, que contiene los servicios requeridos para mostrar información en un formato determinado.
2) Grupo de modelo, que contiene los servicios de la lógica de control, (o negocio), y que es el más grande de los tres. En su forma de operación más sencilla se prescinde del grupo de control, (tercer grupo), y este provee servicios de notificación al grupo de la vista; en su forma mas completa, únicamente se encarga de proveer servicios de control, (o negocio) a las otras gradas.
3) Grupo de control, que se encarga de recibir eventos externos al sistema, y mediar entre el modelo y la vista para mostrar los resultados.

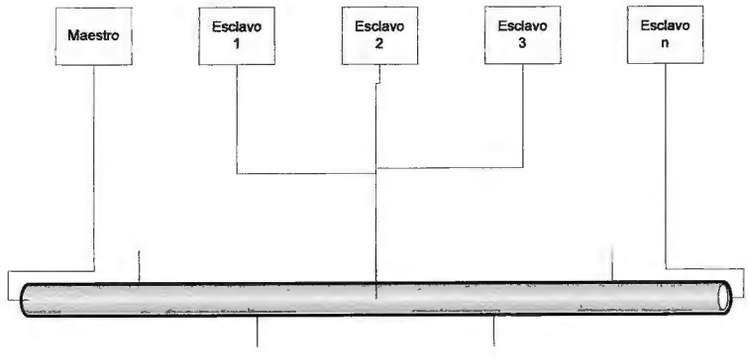
En los sistemas auto-contenidos, este estilo puede ser utilizado como modelo para dividir un sistema que requiera cambiar mucho la presentación de la información; por ejemplo en diferentes modelos de relojes, donde el dispositivo que mantiene el tiempo se mantiene constante, mientras que el elemento que despliega la hora, cambia constantemente. Esta división puede hacerse a nivel microcontrolador, cambiando únicamente el chip que controla la presentación, obteniendose nuevos relojes.

2.4.9. Estilo Cliente Servidor (Elementos Conectados y Conectores)

Es una modificación del estilo subsistema-subsistema; en este caso, algunos grupos se eligen servidores y otros clientes, y se reglamenta que la comunicación únicamente puede ser iniciada por el cliente. Los browsers de Internet, y los servidores WEB, son un ejemplo que utiliza esta arquitectura, donde el cliente siempre comienza la comunicación con el servidor, (a menos que el servidor funcione también como cliente de otra computadora).

Existen otras reglamentaciones que se pueden aplicar, como por ejemplo, que el cliente espere la respuesta del servidor antes de hacer otra petición, conocido como "invocación síncrona", y que es útil para algunos sistemas distribuidos, pero que mal empleado puede causar bloqueos del sistema cuando el cliente espera infinitamente la respuesta de un servidor con fallas.

Este estilo puede ser utilizado en aplicaciones embebidas que requieran un control central, (el cliente), y varios servidores de cálculo, como se muestra en la siguiente figura:



11. Sistema centralizada utilizando el estilo cliente servidor

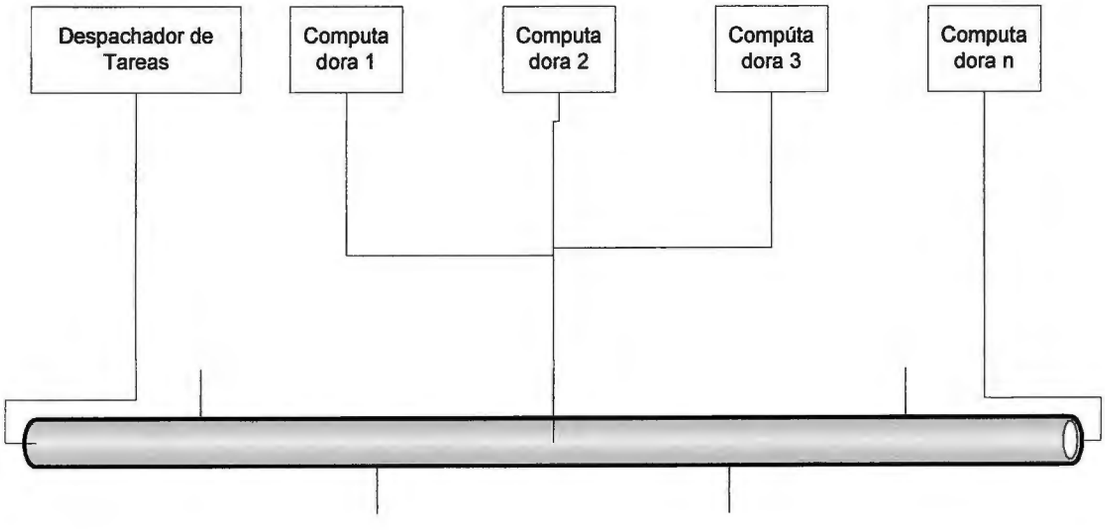
También puede ser utilizada en sistemas distribuidos que cumplen con las 2 siguientes características:

1. El tiempo de ejecución de cada etapa de un proceso tiene ciertos límites interior y superior conocido.

2. Cada mensaje transmitido sobre un canal se recibe en un tiempo limitado conocido.

En este caso, el maestro, no reparte tareas, solamente está encargado de optimizar el uso del ancho de banda del sistema de comunicación. Para lograr esto, el maestro va pasando una estafeta a las computadoras en la red del sistema, que da derecho a la computadora para transmitir su información, y solo cuando ésta sea recibida, y con un límite máximo de tiempo para responder[17], la computadora correspondiente podrá transmitir su réplica. El control maestro utilizará un patrón de envió de estafetas, que permita que todas las computadoras transmitan su información, dentro del espacio de tiempo requerido, minimizando los momentos donde todos quieran transmitir, o nadie utilice el bus del sistema.

Una parecida a las dos anteriores ha sido sugerida, ([1] y [26]); ésta utiliza el estilo cliente servidor para disparar tareas distribuidas en varias computadoras. De ésta forma, el despachador de tareas es externo, y las tareas quedan sincronizadas externamente.



12. Despachador de Tareas Externo, con el estilo Cliente-Servidor

Éste sistema puede ser utilizado en aplicaciones donde varias computadoras ejecuten tareas que requieran ser sincronizadas con respecto a un mismo cliente; por ejemplo, en un sistema de semáforos, donde la tarea es hacer cambios de luces, éstos, deben estar sincronizados a un mismo

[17] Si el tiempo límite para responder expira antes de que una computadora pueda transmitir su mensaje de réplica, el maestro asumirá que dicha computadora ha dejado de funcionar correctamente.

ritmo de cambio, de lo contrario podrían ocasionar un accidente.

2.5. Listado de Patrones No Arquitectónicos Útiles Para los Sistemas Auto-Contenidos

La siguiente lista es una pequeña colección de patrones no estructurales para resolver problemas comunes en el desarrollo de sistemas auto-contenidos. Éstos, no indican estructuras o relaciones entre elementos de software del sistema, ahora se trata de propuestas de implementación para solucionar problemas recurrentes en sistemas auto-contenidos, y pueden, por si solos, proveer una base de conocimiento suficiente para implementar un sistema[18]; sin embargo, si se les une con los estilos estructurales, se puede llegar a soluciones que además de satisfacer los requerimientos funcionales, facilitarán la capacidad para re-usar componentes.

Es necesario mencionar que existe una gran cantidad de patrones recopilados por Michael Pont[19], y que el objetivo del trabajo, no es mencionarlos todos aquí, más bien, resaltar algunos que se pueden utilizar en el ambiente de los sistemas automotrices, y modificar otros que sean buenos candidatos, además de complementar la lista con patrones novedosos que se hayan probado anteriormente,

En esta ocasión la presentación será contexto, problema y solución, que es como este tipo de patrones se documentan generalmente.

[18] Por ejemplo en la referencia [22], se describen los resultados de la implementación de un piloto automatico para autos, utilizando patrones no estructurales, y en la referencia [23] se trata sobre una herramienta de auto código que funciona integrando patrones no estructurales para producir código.
[19] En sus escritos hace mención a más de 70 patrones disponibles.[26]

2.5.1. Programación Defensiva para Reducir los Efectos del Ruido

ANTECEDENTES: Generalmente los circuitos de los sistemas auto-contenidos se encuentran inmersos en un mar de interferencia electromagnética. Ésta, puede ser causada por circuitos vecinos al sistema, o por el mismo sistema; ocasionando, interferencias en el flujo normal de entradas y salidas, haciendo que el microcontrolador confunda el ruido electromagnético, con eventos reales de dispositivos externos, (como interruptores y sensores)[20]. Además, en el interior de los microcontroladores, la interferencia puede cambiar el contenido de los registros y variables, aumentando las probabilidades de fallo del sistema.

CONTEXTO: Se desarrolla un sistema embebido que está expuesto a interferencia electromagnética que puede hacerlo fallar.

PROBLEMA: ¿Cómo reducir los efectos negativos de la interferencia electromagnética?

SOLUCIÓN: Debido a que la interferencia electromagnética causa problemas en diferentes lugares del sistema, es necesario responder con un conjunto de soluciones que engloben todos los puntos. Para este documento se presentarán patrones para las siguientes zonas afectadas.

1. Dispositivos de entrada, como sensores e interruptores
2. Registros de puertos de salida.
3. El registro apuntador de programa
4. Variables en RAM.

Para los dispositivos de entrada, se recomienda tomar muestras periódicamente[33], y después de recolectar un número adecuado, promediarlas y así obtener un resultado que atenúe los efectos de la interferencia electromagnética, (IEM). En el caso cuando la entrada es boleana, en lugar de

[20] Las referencias [34] y [35] pueden proveer al lector una buena introducción sobre los efectos y causas del ruido electromagnético, o EMI.

promediar los valores, se tomará una entrada como verdadera, si se han leído consecutivamente un número, constante de valores verdaderos; se entenderá que la entrada es falsa, si por el contrario, los valores consecutivos han sido falsos. Finalmente, no se detectaran cambios, hasta que se haya conseguido una muestra no ambigua de valores a la entrada.

Por otro lado, los registros de los puertos de salida, pueden ser afectados con la IEM, de manera que conviene reescribir el valor de éstos periódicamente, aún cuando no existan cambios en su salida. Una forma de lograr esto, es que las funciones que modifiquen las salidas, lo hagan alterando un banco de memoria RAM, en lugar de modificar directamente los registros y otro procedimiento, ejecutado periódicamente utilizará dicho banco para reescribir los registros periódicamente.

El apuntador del programa, puede ser alterado también, causando comportamientos erróneos en el sistema. En este caso, por lo menos es necesario cubrir dos áreas:

1. La primera, está conformada por los espacios de memoria ROM no programados, donde se debe de escribir la instrucción de interrupción de software, de manera que si el contador apunta a alguna de éstas direcciones, por error, se ejecute la interrupción, que se encargara de tomar acciones correctivas.
2. La segunda área donde existe la posibilidad de errar, está en las funciones y procedimientos almacenados en memoria, donde se debe asegurar que la tarea despachada por el sistema operativo sea precisamente, el procedimiento o función requerida por el despachador. Para lograr esto, basta que el despachador escriba en una variable el identificador de la tarea que desea ejecutar; dicho identificador es comparado al principio y al final de los procedimientos y funciones, y en caso de encontrar que no son iguales, se deberá manejar la excepción causada cuando el apuntador de programa falla su función.

Cualquier variable en memoria RAM tiene cierta probabilidad de corromperse con la IEM, por lo que es adecuado implementar una verificación de corrupción en bancos importantes. Ésta verificación puede ser tan sencilla como llevar la suma de los bytes que son agregados al banco, y compararla periódicamente con la suma obtenida de los bytes almacenados.

Por otro lado, habrá valores cuya importancia justifique la implementación de algoritmos de recuperación. Éstos, funcionan duplicando los valores importantes, y decidiendo, por mayoría, el valor recuperado.

2.5.2. Reducción del *Bit Stuffing*

ANTECEDENTES: *Bit stuffing* es una técnica utilizada para evitar la corrupción en la recepción de datos durante una transmisión sobre un solo hilo. Consiste en buscar grupos de más de cuatro bits iguales, por ejemplo 1111111, y agregar un bit en la sexta posición, que ayude a re sincronizar los dispositivos. El bit agregado es el negativo de los primeros 5 bits encontrados; para el ejemplo anterior: 11111011.

El *bit stuffing* incrementa la longitud de las cadenas transmitidas, por lo que el disminuirlo, además de reducir la utilización del bus, disminuye la variación de las longitudes de los mensajes enviados en el bus, dificultando la predicción del tráfico de información.

CONTEXTO: Se requiere que el sistema embebido reduzca la cantidad de Bit stuffing durante la transmisión de un mensaje.

PROBLEMA: ¿Cómo reducir la cantidad de *bit stuffing*?

SOLUCIÓN: Convertir cada byte en el mensaje original, que requiera *bit stuffing*, utilizando la operación byte XOR 0xAA.

Se deberá agregar en el encabezado del mensaje, un bit que indique que el mensaje tiene formato especial; además, al final del mensaje original, un conjunto de bytes, cuyos bits, indiquen con su posición, cuales partes del mensaje fueron modificados. Éstos, se separan por un bit que evita que a dicha información, se le aplique el *bit stuffing*.[21] Por ejemplo:

[21] Para mayor información, ver referencia [21].

Bit Ajustable Basado en el bit vecino	Bit de Formato del Byte 1	Bit de Formato del Byte 2	Bit de Formato del Byte 3	Bit Ajustable Basado en el bit vecino	Bit de Formato del Byte 4	Bit de Formato del Byte 5	Bit Ajustable Basado en el bit vecino
Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8

13. Bit de Formato

Si los mensajes tienden a requerir en gran número el formateo XOR, se pueden eliminar los bits del final del mensaje, y aplicar el formato a todo el mensaje original[22].

Tecnológico de Monterrey, Campus Ciudad de México
Biblioteca

[22] Ver referencia [24] para mayor información.

2.5.3. Antecedentes de los Despachadores de Tareas

El despachador de tareas es el software que se encarga de seleccionarle las tareas a ejecutar al CPU. Como este solo pude ejecutar una tarea a la vez, (en microcontroladores con un solo CPU), es necesario crear la lógica para que más de una tarea se complete en determinado tiempo, y las opciones son:

1) Ejecutar tareas cooperativas: donde una de las tareas espera que la otra termine de ser ejecutada.
2) Ejecutar tareas interrumpibles: donde la segunda tarea interrumpe la ejecución de la primera, se guardan los avances de la tarea pausada, y se ejecuta la segunda tarea. Cuando ésta termina, se retoma a la primera tarea, restaurando su estado de ejecución con la información almacenada.

Sin embargo, no todas las tareas pueden ser interrumpidas; por ejemplo, considere que el objetivo de la primera tarea es obtener el promedio de los valores almacenados en las localidades de memoria 0 a100, y que la segunda tarea actualiza dichos valores; entonces, si es interrumpida la tarea de obtención del promedio por la otra, se obtendrá un resultado equivocado.

La ejecución de una tarea puede ser iniciada a partir de un evento externo, como la recepción de un byte de información en el sistema de comunicaciones; o por otro lado, por un evento periódico disparado por un temporizador. En el primer caso, se dice que el sistema es activado por eventos, y en el segundo, por tiempo.

2.5.4. Micro Despachador Cooperativo

ANTECEDENTES: Ver antecedentes de los despachadores de tareas.

CONTEXTO: Se desea un sistema cooperativo que tenga capacidad de ejecutar tareas pequeñas, y con periodos iguales a: 1, 2, 4, 8, 16, 32, 64, 128, o 256 veces el valor de ciclo. Además, el despachador debe ocupar el menor espacio en memoria y tiempo del CPU. En resumen, se trata de un despachador muy pequeño.

SOULUCIÓN: Se puede utilizar un temporizador para incrementar el valor de un contador de ciclos. Por otro lado, desde la función principal, (*main*) se despacharán las tareas utilizando dicho contador:

```
void main(void){
  inicializacionTemporizador();
  while (true){
    //Ejecuta tareas con periodo igual a 1 vez el ciclo
    RunPeriodicTasks1();
     if (!(contadorCiclos & 0x01))  {
      //Ejecuta tareas con periodo igual a 2 veces el ciclo
      RunPeridicTasks2();
    }
    if (!(contadorCiclos & 0x03)){
      //Ejecuta tareas con periodo igual a 4 veces el ciclo
      RunPeridicTasks4();
    }
    if (!(contadorCiclos & 0x07)){
      //Ejecuta tareas con periodo igual a 8 veces el ciclo
      RunPeridicTasks8();
    }
    InactivaCPU();
}
```

Mientras que el servicio periódico de interrupción será:

```
void SCH_Update(void) interrupt INTERRUPT_Timer_2_Overflow
{
  contadorCiclos++;
  TF2 = 0;
}
```

Generalmente, cada una de las condiciones if, ocupará únicamente tres instrucciones del

procesador:

Por ejemplo, para el 8051, y el condicional if que ejecuta tareas cada 8 veces el ciclo, las instrucciones serán:

```
MOV A, Contador; //Mueve el valor de la variable contador al registro A
ANL A, #0x07; //Ejecuta la operación AND
JNZ: Continuar // Si el resultado es diferente de cero, salta fuera del if, (indicado por la etiqueta Continuar), de lo contrario, ejecuta las tareas periódicas dentro del if.
```

Únicamente se debe tener cuidado que las tareas ejecutadas durante el ciclo, no rebasen el periodo mismo, de lo contrario los periodos de otras tareas se verán afectadas.

2.5.5. Despachador Cooperativo

ANTECEDENTES: Ver antecedentes de los despachadores de tareas.

CONTEXTO: Se desea un sistema cooperativo, que tenga capacidad de ejecutar diversas tareas de mediana duración, y con periodos múltiplos a un valor fijo; además, se desean ejecutar tareas continuas (o de *background*) durante el tiempo que el microcontrolador espera por la siguiente tarea periódica.

Dicho despachador puede ser ampliamente utilizado en sistemas que controlen más de un dispositivo; por ejemplo, un sistema que además de controlar las luces exteriores de un automóvil, controle las interiores, y la campanilla de puerta abierta.

Para estos dispositivos de control, es posible que cada una de las tareas, requiera ser disparada a una frecuencia diferente; por ejemplo, el sistema que controla las luces exteriores, puede requerir leer foto-sensores cada 50milisegundos (ms), mientras que el sistema que controla la campanilla de puerta abierta, puede requerir leer un sensor, colocado en la puerta, cada 20ms. De esta forma, se requiere ejecutar diferentes tareas, con diferentes frecuencias múltiplas a un valor, en este caso a 10ms.

SOLUCIÓN: Esta solución es una modificación al patrón *Co-operative Scheduler* de la referencia [26]; el que funciona de la manera siguiente:

1. La función principal, (*main*) manda llamar a un despachador de tareas en un ciclo infinito:

```
while(1)
  {
  SCH_Dispatch_Tasks();
  }
```

14. Despachador Cooperativo Tomado de un Patrón de Pont

2. El despachador de tareas se encarga de seleccionar las tareas más necesitadas de

activación, y después de ejecutarla la elimina, en caso de que ésta esté marcada como tarea de una sola activación.

```
// Dispatches (runs) the next task (if one is ready)
for (Index = 0; Index < SCH_MAX_TASKS; Index++)
  {
  if (SCH_tasks_G[Index].RunMe > 0)
    {
    (*SCH_tasks_G[Index].pTask)(); // Run the task

    SCH_tasks_G[Index].RunMe -= 1;  // Reset / reduce RunMe

    // Periodic tasks will automatically run again
    // - if this is a 'one shot' task, remove it from the array
    if (SCH_tasks_G[Index].Period == 0)
      {
      SCH_Delete_Task(Index);
      }
    }
  }
```

3. Terminada la ejecución de las tareas, escribe un reporte de errores, y pone al CPU en estado inactivo.

```
// Report system status
SCH_Report_Status();

// The scheduler enters idle mode at this point
SCH_Go_To_Sleep();
```

4. A continuación, el CPU es despertado por una interrupción periódica, que se encarga de actualizar la lista de activación de tareas a ejecutar. La actualización consiste en decrementar una variable contadora que lleva la cuenta del tiempo esperado.

```
void SCH_Update(void) interrupt INTERRUPT_Timer_2_Overflow
  {
  tByte Index;

  TF2 = 0; // Have to manually clear this.

  // NOTE: calculations are in *TICKS* (not milliseconds)
  for (Index = 0; Index < SCH_MAX_TASKS; Index++)
    {
    // Check if there is a task at this location
    if (SCH_tasks_G[Index].pTask)
      {
      if (SCH_tasks_G[Index].Delay == 0)
        {
        // The task is due to run
        SCH_tasks_G[Index].RunMe += 1;  // Incr. the run flag

        if (SCH_tasks_G[Index].Period)
```

```
            {
            // Schedule periodic tasks to run again
            SCH_tasks_G[Index].Delay = SCH_tasks_G[Index].Period;
            }
          }
        else
          {
          // Not yet ready to run: just decrement the delay
          SCH_tasks_G[Index].Delay -= 1;
          }
        }
      }
    }
```

5. Después que esto ha ocurrido, el ciclo reinicia.

Como esta solución no está diseñada para ejecutar tareas continuas, (o de *background*), es necesario incluir dicha lógica. Que consistirá en las siguientes modificaciones:

1. El temporizador, que anteriormente bastaba que fuese de 16 bits auto recargable, ahora deberá tener, también, un registro de comparación. Dicho registro es utilizado para generar una interrupción cada vez que el valor del registro, y el temporizador, son iguales, y será usado para alertar al manejador de tareas continuas, que el tiempo de ciclo está por terminar, y que finalizando la ejecución de las últimas instrucciones, éste deberá despachar la instrucción para poner en estado inactivo al CPU[23].
2. Se agregará un ejecutor de tareas continuas, cuya función será activar tareas de background cuando el despachador de tareas periódicas haya terminado su labor. Éste, ejecutará sus tareas hasta recibir una notificación que indica que el fin del ciclo se aproxima, entonces, esperará finalizar su última tarea, y pondrá al CPU en estado inactivo. Cuando el ciclo haya terminado, se reestablecerá la bandera de notificación, (*EndOfCicleApproaching*), para indicar que una vez más se pueden ejecutar tareas continuas.

```
static tByte bkgndTsk=BKG_MAX_TASK;
while (EndOfCicleApproaching==0){
  if (++bkgndTsk>=BKG_MAX_TASK){
```

[23] En la actual implementación se utilizaron dos temporizadores con el mismo periodo, pero desfasados, lo que simula la falta de un registro de comparación en el microcontrolador utilizado para probar los cambios.

```
        bkgndTsk=0;
      }
      if (BKG_tasks[bkgndTsk]!=0) {
              (*BKG_tasks[bkgndTsk])();//Ejecuta la tarea continua
      }
    }
```

3. Para evitar que las tareas continuas ocupen espacio del ciclo siguiente, es necesario que todas ellas tengan una duración menor al tiempo que resta antes de comenzar el siguiente ciclo.
4. Finalmente, se agrega lógica para ejecutar tareas periódicas en cada ciclo, ya que anteriormente solo se podían ejecutar tareas cada dos ciclos, y se utilizaba el código 0, para indicar que la tarea se debía ejecutar una sola vez, y después debía ser borrada. Ahora, se utilizará el código 0xFFFF para indicar que una tarea se desea ejecutar en cada ciclo, y no se desea borrarla. (Comparar con referencia [26])

```
    if (SCH_tasks_G[Index].Period!=0xFFFF && SCH_tasks_G[Index].Period!=0)
      {
      // Schedule periodic tasks to run again
      SCH_tasks_G[Index].Delay = SCH_tasks_G[Index].Period;
      }
    }
```

```
/* Parámetros de la función Add Task
* Tarea a Ejecutar: La dirección de la tarea a ejecutar
* Espera antes de la primera ejecución de la tarea
* Periodo de la tarea, si el periodo es 0, se indica que la tarea se ejecuta una sola vez, si es 0xFFFF, se
* indica que la tarea se ejecuta en cada ciclo, y si es entre 1 y 0xFFFE, es el múltiplo de veces más uno,
* por ejemplo: 1, quiere decir un múltiplo de dos.
*/
SCH_Add_Task(LED_Flash_Update, 0, 0xffff);
```

El resto del sistema está constituido como se describe a continuación:

1. La agenda de activación de tareas periódicas, es una estructura que guarda la siguiente información:

- Un contador de ciclos, que indica cuantos ciclos faltan para que la tarea sea activada.
- Una variable que indica con que valor recargar el contador, antes descrito.
- Un apuntador que indique la tarea a ejecutar.
- Un contador que se incrementa cada vez que la tarea ha esperado el tiempo especificado.

```
// Store in DATA area, if possible, for rapid access
// Total memory per task is 7 bytes
typedef data struct
  {
  // Pointer to the task (must be a 'void (void)' function)
  void (code * pTask)(void);

  // Delay (ticks) until the function will (next) be run
  // - see SCH_Add_Task() for further details
  tWord Delay;

  // Interval (ticks) between subsequent runs.
  // - see SCH_Add_Task() for further details
  tWord Period;

  // Incremented (by scheduler) when task is due to execute
  tByte RunMe;
  } sTask;
```

2. Un procedimiento para verificar los errores del despachador:

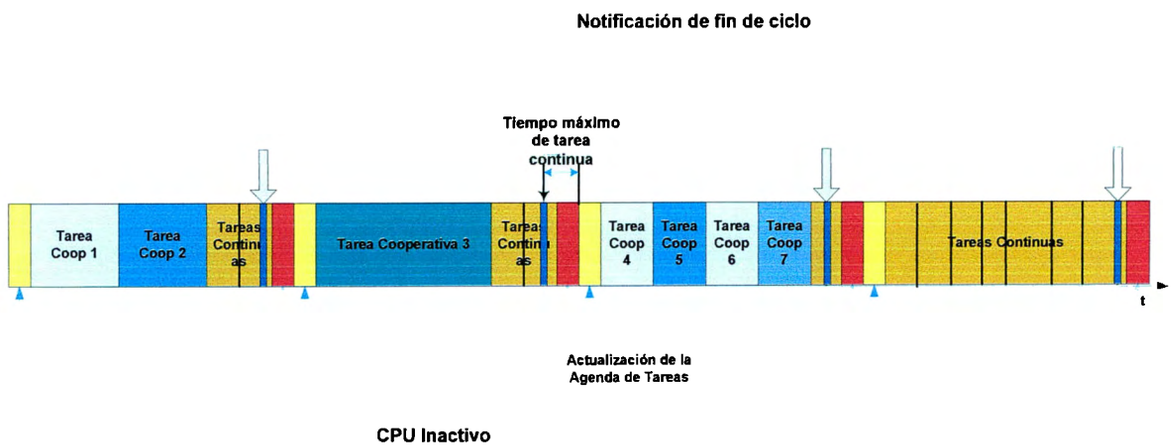
```
void SCH_Report_Status(void)
  {
#ifdef SCH_REPORT_ERRORS
  // ONLY APPLIES IF WE ARE REPORTING ERRORS
  // Check for a new error code
  if (Error_code_G != Last_error_code_G)
    {
    // Negative logic on LEDs assumed
    Error_port = 255 - Error_code_G;

    Last_error_code_G = Error_code_G;

    if (Error_code_G != 0)
      {
      Error_tick_count_G = 60000;
      }
    else
      {
      Error_tick_count_G = 0;
      }
    }
  else
```

```
        {
        if (Error_tick_count_G != 0)
          {
          if (--Error_tick_count_G == 0)
            {
            Error_code_G = 0; // Reset error code
            }
          }
        }
    #endif
    }
```

3. Un procedimiento para poner al CPU en estado inactivo:

```
void SCH_Go_To_Sleep(void)
  {
  PCON |= 0x01;   // Enter idle mode (generic 8051 version)

  // Entering idle mode requires TWO consecutive instructions
  // on 80c515 / 80c505 - to avoid accidental triggering
  //PCON |= 0x01;   // Enter idle mode (#1)
  //PCON |= 0x20;   // Enter idle mode (#2)
  }
```

En este sistema se debe tener mucho cuidado que la suma de la duración de las tareas no exceda el tiempo de periodo de ciclo, de lo contrario, ocurrirán retrasos en ciclos posteriores.



15. Esquema de Ejecución Cooperativa

En el diagrama anterior se resalta en amarillo las interrupciones generadas para actualizar la lista de activación de tareas periódicas.

En color rojo, se muestran las áreas donde el CPU es apagado mientras se espera la siguiente interrupción de actualización, con el que un nuevo ciclo comenzará.

En azul marino se muestran las interrupciones que sirven para notificar que el tiempo de fin de ciclo se aproxima, y finalmente, en café, se muestra la ejecución de las tareas continuas.

Debido a que este sistema incluye lógica de búsqueda de candidatos a ejecutar, el promedio de efectividad puede variar de sistema a sistema. Por lo anterior, este despachador debe ser preferido cuando se tenga poca cantidad de tareas periódicas por ciclo, (1 a 3), y que éstas, ocupen la mayor cantidad del mismo, (arriba del 50% por ciclo). Además, es preferible que las tareas continuas sean de mediano tamaño, (300 microsegundos –us- a 700us), de manera que el tiempo de ejecución de éstas, sea mucho mayor que el tiempo tomado para despacharlas.

Por ejemplo, este sistema fue probado en el simulador de hardware para 8051, de Keil, con las siguientes características:

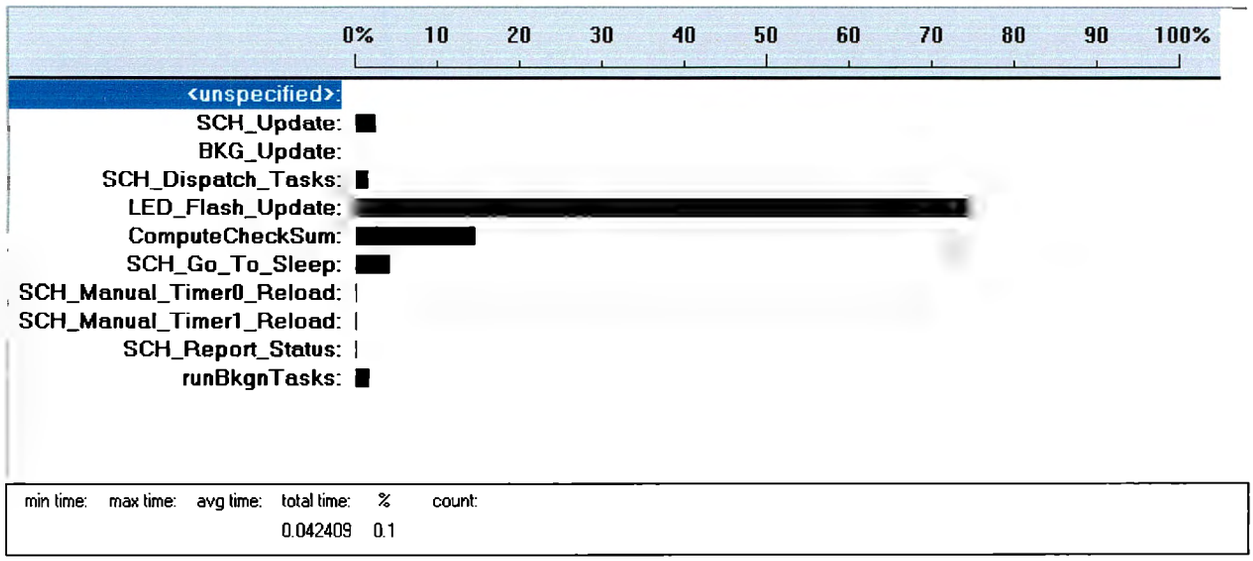
1. Se eligió un ciclo de 5ms.
2. Se ejecuta una sola tarea periódica por ciclo; *LED_Flash_Update*, de 3.792ms
3. Una sola tarea cooperativa, *ComputeCheckSum*, de 388us
4. Se avisa con 500us de anticipación, el fin de ciclo. Debido a que la plataforma donde se llevaron a cabo las pruebas, no cuenta con registro de captura, se utilizaron los temporizadores 0 y 1 con un periodo de 5ms, desfasados para lograr el aviso de fin de ciclo.

```
void SCH_Manual_Timer1_Reload()
  {
  // Stop Timer 1
  TR1 = 0;
  TR0 = 0;
  // 8051, 12 MHz
  // The Timer 1 resolution is 5000 µs
  // We set the timer to generate interrupt after 5 ms
  // -> we are generating timer ticks at ~5 ms intervals
  TH1  = 0xEC;
  TL1  = 0x78;
  TH0  = 0xFE;
  TL0  = 0x0C;

  //  Start Timer 0
  TR1  = 1;
  TR0  = 1;
```

```
}
```

```
void SCH_Manual_Timer0_Reload()
  {
  // Stop Timer 0
  TR0 = 0;
  TR1 = 0;

  // 8051, 12 MHz
  // The Timer 0 resolution is 1.000 µs
  // We set the timer at 64336 to generate interrupt after 1 ms
  // -> we are generating timer ticks at ~1 ms intervals
  TH0  = 0xEC;
  TL0  = 0x78;
  TH1  = 0xEE;
  TL1 = 0x6B;
  //  Start Timer 0
  TR0  = 1;
  TR1  = 1;
  }
```

Con las características anteriores, se obtuvo la siguiente gráfica de porcentaje de aprovechamiento del CPU.



16. Porcentaje de Aprovechamiento del CPU del Despachador Cooperativo

En ésta gráfica se observa que más del 75% del tiempo, el CPU es utilizado para ejecutar la tarea periódica, (que corresponde a 3.792ms de duración de la tarea, entre 5ms del ciclo). También se observa que la tarea continua, (*ComputeCheckSum*), utiliza más del 12% del CPU,

que es proporcional al número de veces que se repite la tarea por ciclo. El CPU, pasa un poco más del 3% del ciclo inactivo, y el resto del tiempo es utilizado por el sistema operativo, aproximadamente de un 5% a un 7%, lo que lo convierte en un excelente despachador, bajo estas características.

2.5.6. Despachador Hibrido de Tareas

ANTECEDENTES: Ver antecedentes de los despachadores de tareas.

CONTEXTO: Se desea un sistema operativo hibrido, que despache tareas con las siguientes características:

1) Una tarea periódica larga que se pueda interrumpir.
2) Una tarea periódica corta, que no se pueda interrumpir, y cuyo periodo de activación sea un múltiplo de la larga.
3) Se deberán ejecutar tareas *continuas* mientras el CPU espera ejecutar una tarea, larga o corta.

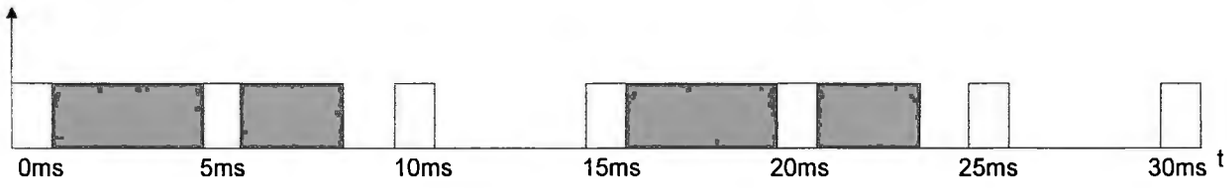
El despachador de tareas descrito anteriormente, puede ser utilizado en sistemas de control que estén divididos en las siguientes áreas:

1. Un área de traducción de eventos del entorno real en eventos del sistema.
2. Un área con la lógica de control, que se encargue de observar los eventos externos y aplicar acciones de control; ésta debe ser más larga que la duración de las tareas cortas, e incluso, debe tardar más tiempo en ejecutarse, que el tiempo comprendido entre dos tareas cortas ejecutadas periódicamente.
3. Un área de traducción de acciones del sistema, a control del entorno real.

La característica necesaria en esta implementación, es que el entorno real requiera de atención continua del sistema, es decir, que aunque no existan eventos a reaccionar, el sistema deberá continuar emitiendo acciones periódicas. De esta forma, la tarea corta, ejecutaría las funciones 1 y 3, descritas anteriormente, las que gozarán de continua atención, mientras que la tarea larga, ejecutaría la lógica de control. Para ver más usos del despachador descrito, vea la referencia [18]

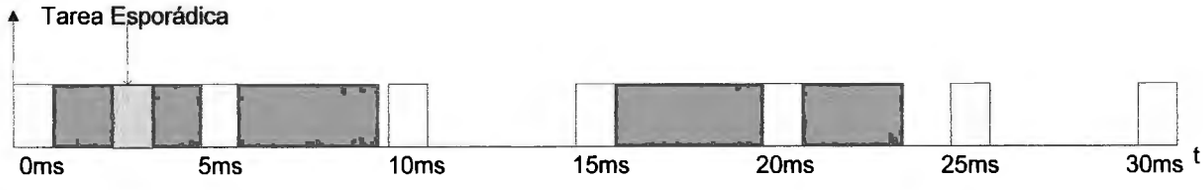
PROBLEMA: Cómo implementar un despachador hibrido con las características anteriores.

SOLUCIÓN: Utilizar un temporizador con registro de captura para llevar el ritmo del despachador. El temporizador se utilizará para activar directamente las tareas cooperativas, que tienen el menor de los periodos. El registro de captura, se utilizará para avisar al procedimiento principal, (*main*), que el fin del ciclo se aproxima, y que deberá terminar de ejecutar las tareas continuas, (si es que en ese momento activa ese tipo de tareas). La tarea de larga duración es activada por el procedimiento principal, (*main*), que la activa cuando el valor de un contador de ciclos, incrementado por la interrupción periódica del temporizador, es igual al múltiplo de un número de ciclos.



17. Ejecución del Despachador Híbrido

En el dibujo anterior, se muestra un despachador con una tarea corta, de periodo de ejecución igual a 5ms, y una tarea larga y divisible, con periodo igual a 15ms. Tareas esporádicas se podrían incluir durante la ejecución de la tarea larga; por ejemplo:

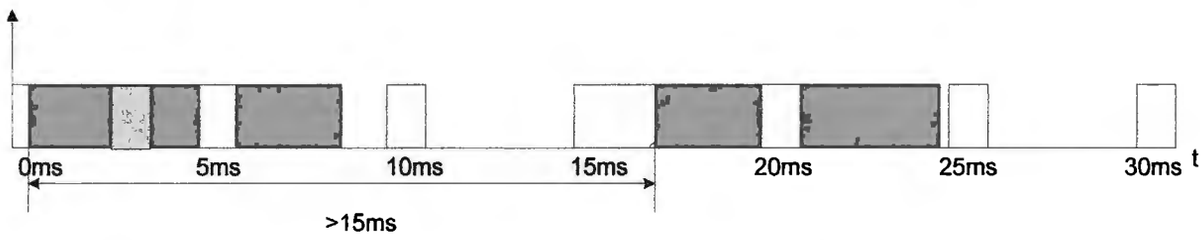


18. Despachador Hibrido con Tarea Esporádica

La inclusión de la tarea esporádica, alargará la ejecución de la tarea interrumpida, y se deberá tener cuidado que el final de ésta, no alcance el comienzo de su siguiente ciclo. Este problema puede ser detectado durante el desarrollo del sistema, si se suman los máximos tiempos

ejecutados dentro de cada ciclo. Para este caso, se sumará la longitud de la tarea larga, más tres veces la longitud de la tarea corta, ya que ésta es ejecutada 3 veces durante el ciclo de la tarea larga, y se incluirá el máximo tiempo ocupado por las tareas esporádicas.

Otro punto en el que se debe tener cuidado, es en la exactitud del periodo de la tarea alargada, ya que éste, depende del tiempo de ejecución de la tarea cooperativa, y si esta se alarga, o se acorta, causará imprecisiones en el disparo de la tarea.



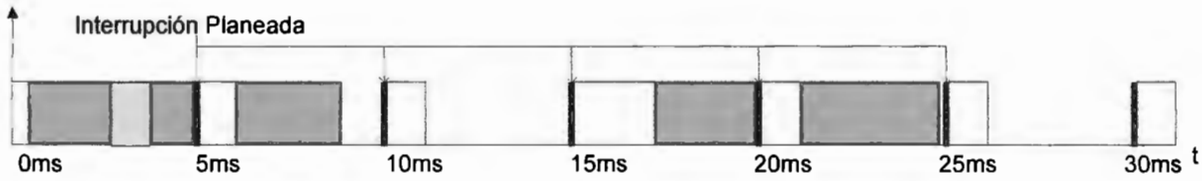
19. Imprecisión en el Periodo de un Despachador Híbrido

No obstante de la inexactitud del periodo de la tarea, existen muchos usos para este despachador.[24] Además, la tarea corta, en este caso, la que se ejecuta cada 5ms, gozará de un periodo muy estable, que por si fuera poco, se puede ajustar mas, utilizando una técnica descrita en la referencia [19], que evita tener diferentes tiempos de reacción a la notificación del temporizador; diferencias causadas porque cuando el CPU es interrumpido, para ejecutar las tareas cooperativas, éste tiene que terminar de ejecutar su última instrucción, (que puede encontrarse en su inicio, mitad, o fin), y guardar el vector de estados, para retomar en el futuro las actividades que éste hacía antes de ser interrumpido, lo que ocasiona diferentes tiempos de respuesta de acuerdo al momento de la interrupción, (inicio, medio, o fin de la ejecución de una tarea).

El ajuste fino, se logra con otro temporizador, que tiene el mismo periodo que el temporizador

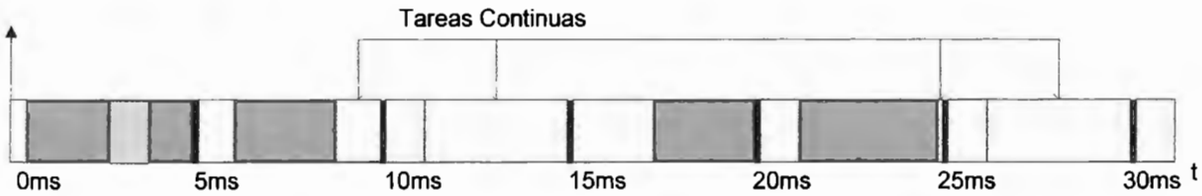
[24] Ver la referencia [18] para saber más.

que activa las tareas cooperativas, pero que está defasado unos cuantos microsegundos del primero, y que pone en estado de inactividad al CPU cada vez que éste es llamado; de ésta manera el CPU siempre será sorprendido en estado inactivo, tardando siempre el mismo tiempo en atender la interrupción.



20. Despachador Híbrido con Interrupción Planeada

En los espacios donde el CPU espera la siguiente tarea, se pueden añadir las tareas *continuas* (o de background), que puedan ser interrumpidas en cualquier momento por las esporádicas, y que ponen en uso al CPU:



21.* Despachador de Tareas con Ejecución en *Background

Dichas tareas deberán tener una longitud máxima, igual al tiempo restante después del aviso de fin de ciclo.

Finalmente, hay que mencionar que la utilización de este patrón puede ser considerada riesgosa en sistemas de alta seguridad. Este inconveniente es causado debido a que la tarea corta, al tener la mayor prioridad de ejecución, puede no regresar el control al despachador durante una

falla no prevista.[25] Dado a lo anterior, en esos sistemas se recomienda utilizar una estructura modificada del centinela de fallos, descrito en la sección 2.11

SEUDO CÓDIGO:

```
void main(){
inicializaTimers();
ajustaPrioridadInterrupcion(CONTINUA);
comienzaTimers();
while (true){
if (Need2RunLongTask){
RunLongTask();
}
while (EndOfCycleApproaching!=0){
RunBackgroundTask();
}
deshabilitaTareasEsporadicas();
}
}
void servicioInterrupcionCooperativa(){ //Tiene la mayor prioridad de las interrupciones
tareaCooperativa1();
tareaCooperativa2();
...
EndOfCycleApproaching=0;
LongTaskCntr++;
HabilitaTareasEsporadicas();
}
void servicioInterrupcionFinCicloAcerca(){//Con mayor prioridad que las tarea esporádicas
EndOfCycleApproaching=1;
}
void servicioInterrupcionDormir(){//Con menor prioridad que la interrupción de las tareas cooperativas
  inactivaCPU();
}
```

[25] Las fallas más comunes son en hardware, debido a eventos inesperados, (como la falla del contador de programa debido a emisiones electromagnéticas excesivas) o en el software, (por una espera infinita no detectada).

2.6. Las Bases Existentes y el Control Remoto

El control remoto forzosamente requerirá un despachador de tareas; sin embargo, debido a su forma de operar, ninguno de los patrones presentados por Pont (Capítulo 2.5) es directamente utilizable. Esto es debido a las siguientes razones:

1. El despachador deberá de ser capaz de "auto apagarse[26]" cuando se haya confirmado que no hay actividad en las entradas del sistema, para después reactivarlo cuando una interrupción detecte posible actividad en los puertos. Este tipo de retroalimentación no está soportada en ninguno de los patrones propuestos por los autores revisados.
2. El control remoto tiene, por un lado, tareas que requieren ser ejecutadas con una frecuencia muy alta, pues la transmisión del mensaje de RF es de un baudio cada 208 microsegundos, y por otro, la lógica de control requiere mucho más tiempo que 208us para completar su ejecución. Por lo anterior, no son directamente utilizables los patrones híbridos y cooperativos de Pont, requiriendo un despachador totalmente *preemptivo*. Sin embargo, un despachador *preemptivo* es un sistema más complejo de lo necesario, como se vera en las siguientes secciones.

En los capítulos siguientes se describirá un despachador totalmente nuevo, que ayudará a facilitar los cambios de software en el sistema y promoverá la reutilización de componentes.

[26] El sentido de apagar, es dejar de ejecutar el despachador. El borrar las tareas a ser ejecutadas por el despachador, pero continuar ejecutándolo, no es la meta buscada.

3. Producto de la Investigación

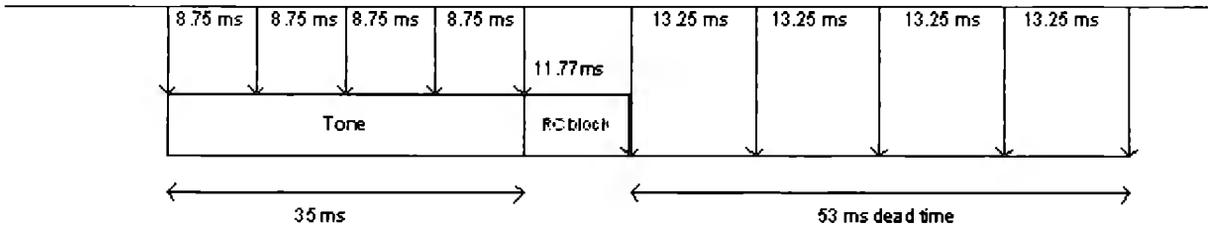
El presente capítulo es el más importante del documento, ya que contiene los resultados obtenidos de la investigación, así como la metodología utilizada para llegar a ellos.

Comienza con una breve descripción sobre el diseño del sistema hecho en Siemens[27]. Sigue con un capítulo dedicado a la metodología de desarrollo de sistemas Jackson, que fue utilizada para obtener el sistema. Finalmente concluye con los resultados obtenidos de la investigación. El código del control remoto se anexa a este trabajo en un CD, y como se recordará en la introducción, los diagramas de los algoritmos están escritos utilizando lenguaje UML.

3.1. Diseño Utilizado por la Empresa

Actualmente se mantiene al sistema en estado *idle* hasta que se detecta, (mediante una interrupción), la posibilidad de la ocurrencia de un nuevo evento externo. A continuación, se inicializa una tarea que es activada cada 2ms para confirmar el evento. Si en 5 ocasiones consecutivas se detecta el mismo estímulo externo, se reconoce un nuevo evento externo y se procede a transmitir un comando. Durante la transmisión del comando, se cambia la periodicidad de la tarea que realiza los muestreos, tomando 5 valores cada 8.75ms y después uno de 11.7ms. Una vez transmitido el comando, se vuelve a modificar la periodicidad de la tarea que muestrea, y se ajusta a 13.25ms. Durante este lapso, basta con obtener 3 muestras iguales, para confirmar un nuevo evento externo.

[27] Ver e-mails en el apéndice A



22. Tarea que Muestrea Durante la Transmisión de un Comando

La lógica descrita arriba, es más compleja que otra que tome muestreos constantes, por decir cada 4ms. Esto es debido, a que por cada variación de frecuencia, se requiere una lógica diferente para confirmar eventos externos. Por ejemplo, cuando muestrea cada 2ms, requiere tener 5 muestras iguales, mientras que cuando muestrea cada 8.75ms, requiere una nueva lógica, que ahora se conforme con detectar 3 muestras iguales para disparar un evento externo.

Además, si la lógica de detección no está completamente aislada de la lógica de reconocimiento de secuencias externas, entonces, por cada lógica de detección se necesitará adaptar la lógica de reconocimiento, haciendo aún más complejo el software.

En este caso, la complejidad está vinculada al aumento de dependencias entre módulos, ya que cada lógica de reconocimiento de secuencias, dependerá de una diferente lógica de detección, lo que ocasiona dificultades en la reutilización de código por componentes, pues como se dijo en el marco teórico, es preferible tener un nivel bajo de dependencias que faciliten el reemplazo entre módulos.

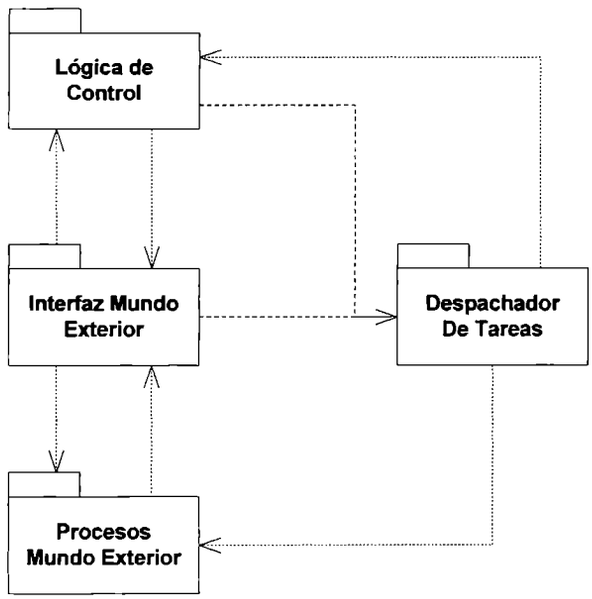
Por otro lado, la capacidad para cambiar de un muestreo variante, a un muestreo constante, radica en las limitaciones y capacidades del despachador de tareas del sistema, que como se verá más adelante es un elemento fundamental en esta investigación.

3.2. Metodología Utilizada para Obtener la Nueva Propuesta

3.2.1. J.S.D. (*Jackson System Development*)

Es un método de diseño de sistemas, que comienza con un análisis de los procesos externos, con los que el sistema deberá interactuar para producir cambios en su ambiente. A continuación, deduce una interfaz con el mundo externo, que está encargada de acoplar el mundo externo, con el sistema de control y finaliza el desarrollo, creando la lógica de control necesaria para que el sistema modifique su ambiente como es requerido.

La partición del sistema en interfaz del mundo real, y lógica de control, da origen al primer estilo arquitectónico a utilizar.



23. División por Gradas del Sistema

Como se ve en el diagrama, el despachador de tareas juega un papel muy importante en el

sistema, ya que tanto la interfaz con el mundo exterior, como la lógica de control dependen de él para ser ejecutados a un ritmo constante; sin embargo, el despachador también depende de los eventos en el mundo real y de la lógica de control, que son dependencias no mostradas antes en los patrones mencionados en el marco teórico.

3.2.2. Aplicando JSD al Sistema

Antes de comenzar a aplicar el método de desarrollo Jackson, es necesario seleccionar un microcontrolador que sirva como controlador único. Para este caso, Siemens utiliza el microcontrolador PCF7941 de *Philips Semiconductors*, que cuenta con 4KB de memoria ROM, 128B de memoria RAM, y dos temporizadores. Sin embargo, al no poseer las herramientas de desarrollo, se utilizará un microcontrolador semejante, el 89C51, que cuenta con las siguientes características:

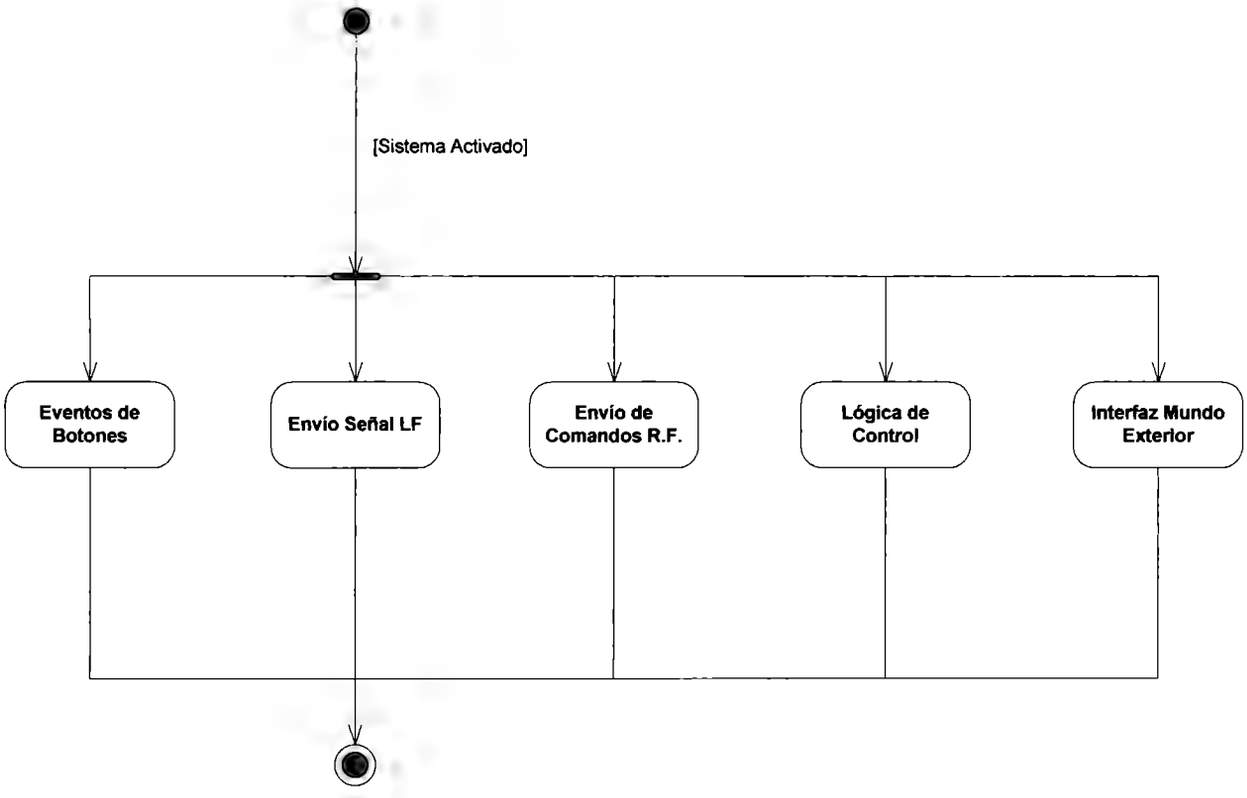
(Propuesto) **89C51**	**(Utilizado)** **PCF7941**
ROM:4k	ROM:4k
RAM:128B	RAM:128B
IE0, (Utilizada como ILF) TF0, (Utilizada como IT0) IE1,(Utilizada como IPort) TF1, (Utilizada como IT1)	ILF IT0 IPort IT1

Como se describe en la tabla anterior, los dos microcontroladores son semejantes, y la implementación en el 89C51, puede ser aplicada al PCF7941.

El método Jackson comienza con la identificación de los procesos externos que interactuarán

con el sistema; en este caso:

1. Eventos en los botones del control remoto
2. Eventos de recepción de campo LF que indica transición a modo transmisor receptor
3. El sistema interactuará con el mundo real enviando señales electromagnéticas que representan un comando para el automóvil.



24. Abstracción del Sistema

El diagrama anterior fue deducido de las especificaciones en 1.3 más atrásantes mencionadas. A continuación se caracteriza el comportamiento de dichos procesos:

1. Los Eventos de Botones: Son los eventos provocados por el usuario al presionar y dejar de presionar botones en el control remoto. Éstos, deben ser confirmados mediante muestreos consecutivos, ya que son producidos con señales eléctricas al cerrar y abrir interruptores, mismos que pueden, por si solos o debido al ruido

electromagnético, ocasionar falsos eventos.

2. Envío Señal LF: Este proceso se encarga de enviar al control remoto una señal de campo LF, que le indica que el modo control remoto debe ser terminado, y que el modo transmisor receptor debe ser iniciado. Dicho evento sucede cuando el control remoto se encuentra cerca de una central transmisor receptor, (en este caso cuando el control remoto esta insertado en el zócalo de ignición del automóvil, y el control debe transmitir la señal para dar marcha al motor del automóvil)
3. Envío de Comandos RF: Es el medio de comunicación entre el control remoto y el automóvil. De acuerdo a las especificaciones recibidas, se transmitirán señales electromagnéticas moduladas con el protocolo ASK y codificadas en Manchester a 2.4Kbps. Que significa que se deberán transmitir 2 baudios cada 1 / 2.4k segundos, es decir, cada 416 microsegundos, por lo que se deberá enviar cada baudio aproximadamente, cada 208us.

El siguiente paso, determina la forma de activar los procesos del sistema, no está incluido en el método Jackson; sin embargo, es necesario para asignar el hardware que será responsable de ejecutar las diferentes tareas del sistema.

Para este caso, se ha propuesto crear un **nuevo despachador de tareas,** basado en el patrón "*cooperative scheduler*" de Pont[28], donde se tomarán en cuenta las siguientes restricciones:

1. El microcontrolador solamente cuenta con dos temporizadores T0 y T1.
2. El temporizador T0 tiene un evento que puede interrumpir al servicio de interrupción del temporizador T1, pero T1 no tiene eventos que puedan interrumpir el servicio de interrupción del temporizador T0.
3. La transmisión de comandos de R.F. no debe ser retardada o interrumpida, ya que se

[28] Ver marco teórico para mas información sobre el patrón *Cooperative Scheduler* de Pont

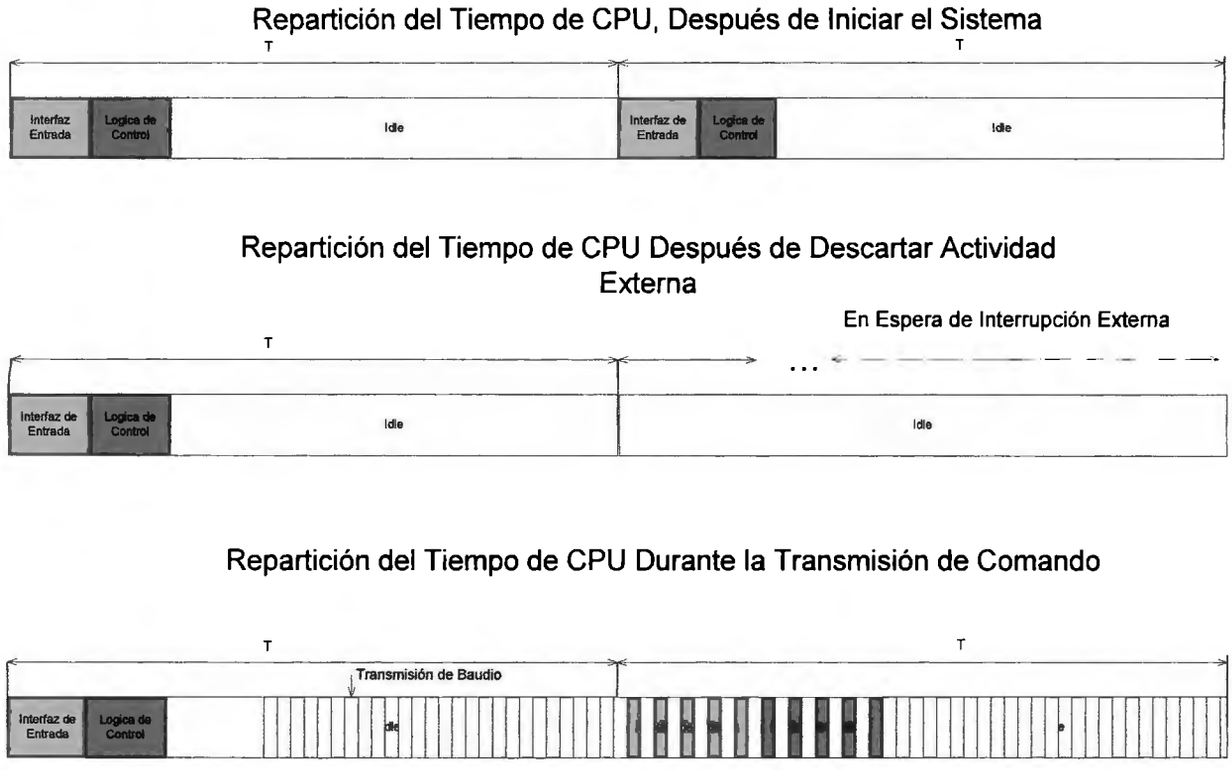
afectará la calidad de los datos transmitidos.

Por lo anterior, se ha elegido utilizar el temporizador T0 para transmitir los comandos de RF, y al temporizador T1 para despachar la lógica de control y la interfaz con el mundo exterior. T1, tendrá un rol similar que el temporizador utilizado en el patrón *Co-operative Scheduler* de Pont [26], pero T0, tendrá un rol totalmente nuevo.

Este despachador es una de las principales contribuciones de la investigación, ya que es capaz de ejecutar tareas hibridas, como el despachador de Pont, y además cuenta con las siguientes nuevas capacidades:

1. Puede despachar tareas esporádicas y de periodicidad menor que las tareas de la lógica de control y la interfaz con el mundo exterior.
2. Recibe retroalimentación de la lógica de control, y de los eventos externos, para modificar su comportamiento. De esta forma, a la orden de la lógica de control, el despachador puede suspender la ejecución de la interfaz con el mundo exterior, así como la lógica de control, reanudando su operación periódica al detectar la posibilidad de un nuevo evento externo, (mediante una interrupción).

En el siguiente diagrama se muestra la funcionalidad pretendida de dicho despachador:



25. Repartición de los Procesos del Sistema en el CPU

El anterior diagrama se muestra cómo el despachador cooperativo de Pont, se mezcla con otro despachador de tareas de alta frecuencia, y cómo es que a la orden de la lógica de control, el despachador deja de operar periódicamente.

Es importante notar, que se requiere considerar durante el diseño, el alargamiento de las tareas comunes, y el acortamiento del tiempo en *idle* durante la presencia del despachador de tareas de alta frecuencia. Además, es necesario verificar que la suma de los tiempos de ejecución de las siguientes tareas:

1. Interfaz de Entrada
2. Lógica de Control
3. Hilo de Transmisión

Sean menores que el periodo de las tareas de baja frecuencia, de manera que aún durante la transmisión del comando, la periodicidad de las tareas de baja frecuencia, no se afecte.

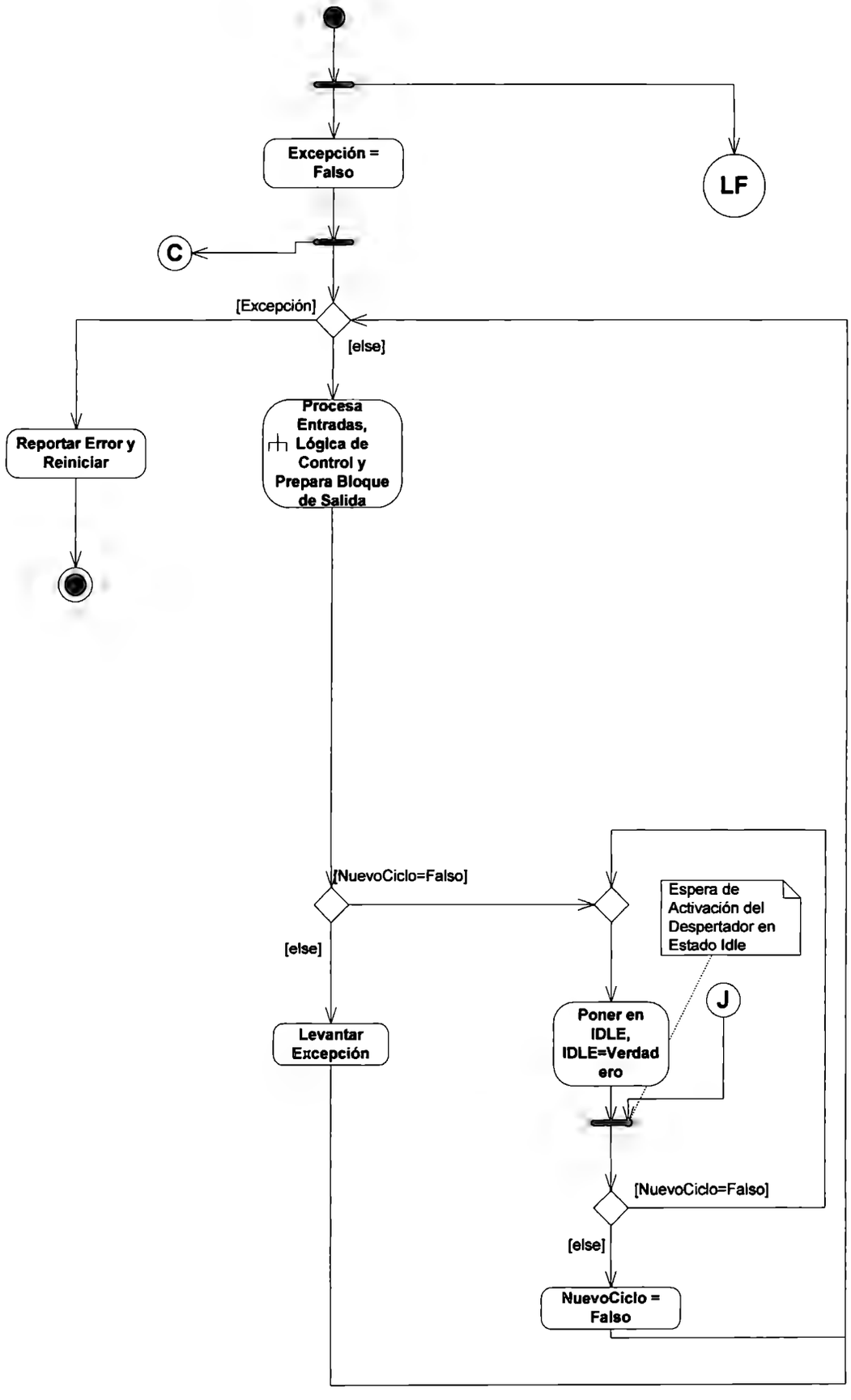
En el siguiente capítulo se examinará más a detalle el nuevo patrón obtenido, y los últimos pasos del método Jackson: la implementación de la interfaz con el mundo exterior y la lógica de control.

3.3. El Despachador

Como se mencionó en el análisis de activación de procesos, se utilizarán los dos temporizadores T0 y T1 para despachar las tareas del sistema. El temporizador T0, que tiene mayor prioridad de interrupción, se utilizará para activar la transmisión de comandos, mientras que el temporizador T1 se encargará de activar las siguientes tareas:

1. Interfaz con el Mundo Exterior, (detección de activación de botones)
2. Lógica de Control

A continuación se describirán los *hilos* con los que se propone construir el sistema:



26. Hilo Principal del Sistema

En el anterior diagrama se muestra el hilo principal, o *main* del sistema, que es donde se encuentra la parte análoga del despachador cooperativo de Pont.

Como se mencionó, el temporizador T1, sacará del estado IDLE al hilo main, haciendo que la secuencia de activación se repita. En el diagrama el conector C, inicializa dicho hilo despertador.

El conector LF, es utilizado para inicializar el hilo que se encarga de detectar una señal *transponder* en las cercanías.

El conector J indica que el hilo principal ha sido interrumpido por otro hilo, ya sea por el hilo de transmisión, o cualquiera de los hilos despertadores.

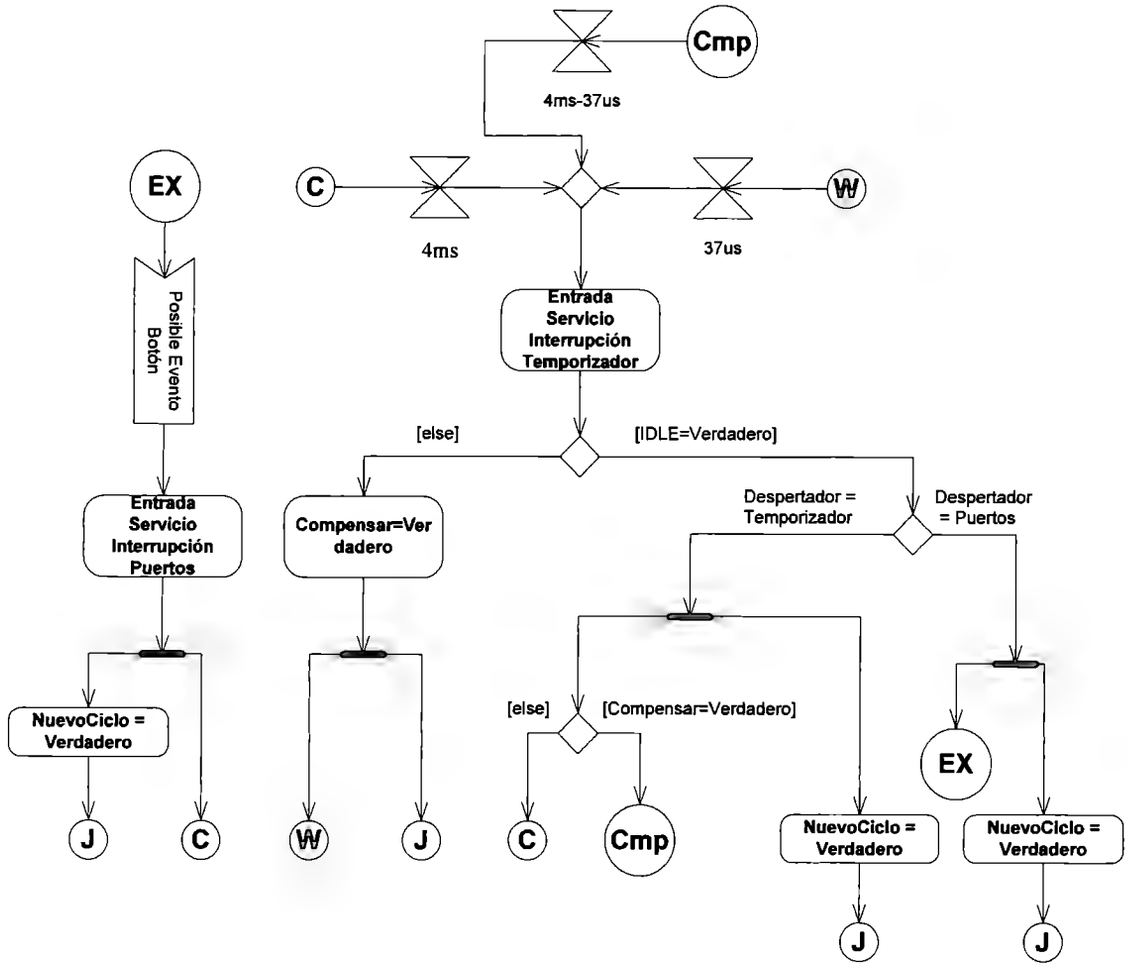
Su equivalente en código es el siguiente:

```
void main(void){
  isAwakerRinging=FALSE;
  //Inicializa hilo que detecta el campo LF
  (*ptrEnableLFInterrupt)();
  //Pone en falso la bandera de compensación
  (*ptrInitWakeUpThread)();
  //Inicializa las tareas a ser despachadas
  (*ptrInitCLogic)();
  (*ptrInitDebouncer)();
  //Comienza el temporizador que despertará al hilo main
  MAC_Start_AWAKER(SLEEP_4MS_HI,SLEEP_4MS_LO);
  //Ciclo del despachador
  while(1){
    //Ejecuta las tareas
    (*ptrBEventFilter)();
    (*ptrCLogic)();
    //Duerme hasta que el hilo de despertador lo indique
    while (!isAwakerRinging){
      MAC_Main_Go_2_Sleep;
    }
    //Borra la bandera de despertado
    isAwakerRinging=FALSE;
  }
}
```

27. Código Equivalente

Debido a que el espacio de memoria en el sistema es muy limitado, se omitió la funcionalidad encargada de detectar excepciones y reiniciar el sistema, que de cualquier forma no estaba dentro de los requerimientos funcionales.

A continuación se muestran los hilos despertadores:



28. Hilos Despertadores

La figura de la derecha muestra el hilo del temporizador T1, que se encarga de despertar al CPU con una periodicidad estable. Mientras que el diagrama del lado izquierdo representa la lógica que se encarga de despertar al sistema, cuando el despertador periódico ha sido apagado por orden de la lógica de control.

Como se observa en la figura, una vez que el hilo de la izquierda se ha ejecutado, se reactiva el despertador periódico, regresando a la ejecución periódica de las tareas y evitando que el microcontrolador se quede dormido esperando otra interrupción externa.

Su equivalente en código es la siguiente:

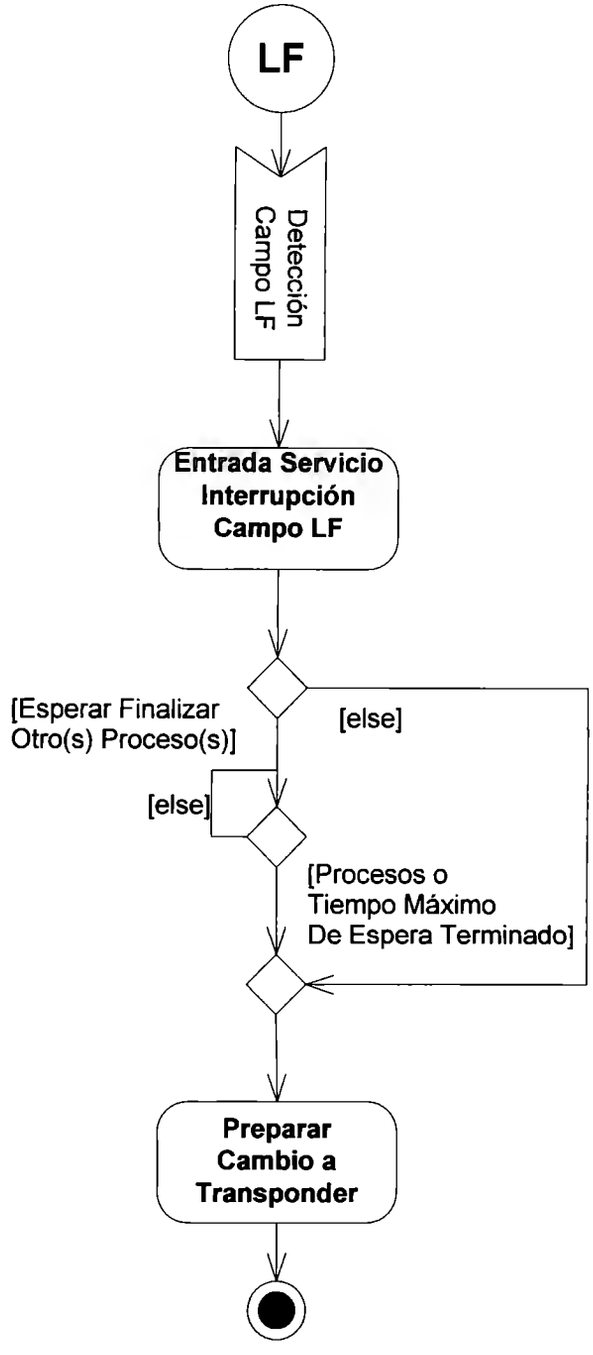
```
void wakeUpByTimer(void) interrupt CLogicINTERRUPTCODE {
  //Primero detecta si el hilo principal estaba durmiendo, o no
```

```
    if (MC_Was_Main_Sleeping){
      //Determina si el control hadado la orden de apagar el
      //Despertador periódico
      if (MC_Is_Timer_AWAKER){
        //Determina si el hilo despertador ha esperado 37us a que
        //el hilo main se durmiera.
        if (waitingCycles){
          MC_Reload_Timer(SLEEP_4MS_MINUS_37US_HI, SLEEP_4MS_MINUS_37US_LO);
        } else {
          MC_Reload_Timer(SLEEP_4MS_HI, SLEEP_4MS_LO);
        }
        //Pone la bandera que indica, al hilo principal, que debe
        //hacer otra ronda activando sus tareas
        MC_Turn_On_Awaker_Bell;
      } else {
        //Apaga el despertador periódico y cambia al despertador
        //por eventos
        (*ptrResetDebouncer)();
        MC_Switch_Waker_2_Ports;
      }
      waitingCycles=0;
    } else {
      //Si el hilo principal estaba despierto cuando se activo el
      //despertador, espera 37us a que se duerma.
      waitingCycles=1;
     //Espera 37us e intenta nuevamente
      MC_Reload_Timer(SLEEP_37uS_HI,SLEEP_37uS_LO);
    }
  }

  void wakeUpByPort(void) interrupt PortINTERRUPTCODE {
    //Enciende el despertador periódico
    MC_Set_Timer_As_Awaker;
    MC_Reload_Timer(SLEEP_4MS_HI,SLEEP_4MS_LO);
    //Pone la bandera que indica, al hilo principal, que debe
    //hacer otra ronda activando sus tareas
    MC_Turn_On_Awaker_Bell;
  }
```

29. Código de los Hilos Despertadores

A continuación, se muestra el hilo encargado de terminar el modo control remoto, e iniciar el modo *transponder*. Este hilo surge como una necesidad en el análisis de procesos externos, ya que cuando el control remoto es insertado en el zócalo del interruptor de ignición del automóvil, se deberá terminar el modo control remoto, y pasar al modo transmisor receptor, que indicará al automóvil que el motor puede ser encendido al girar el interruptor.

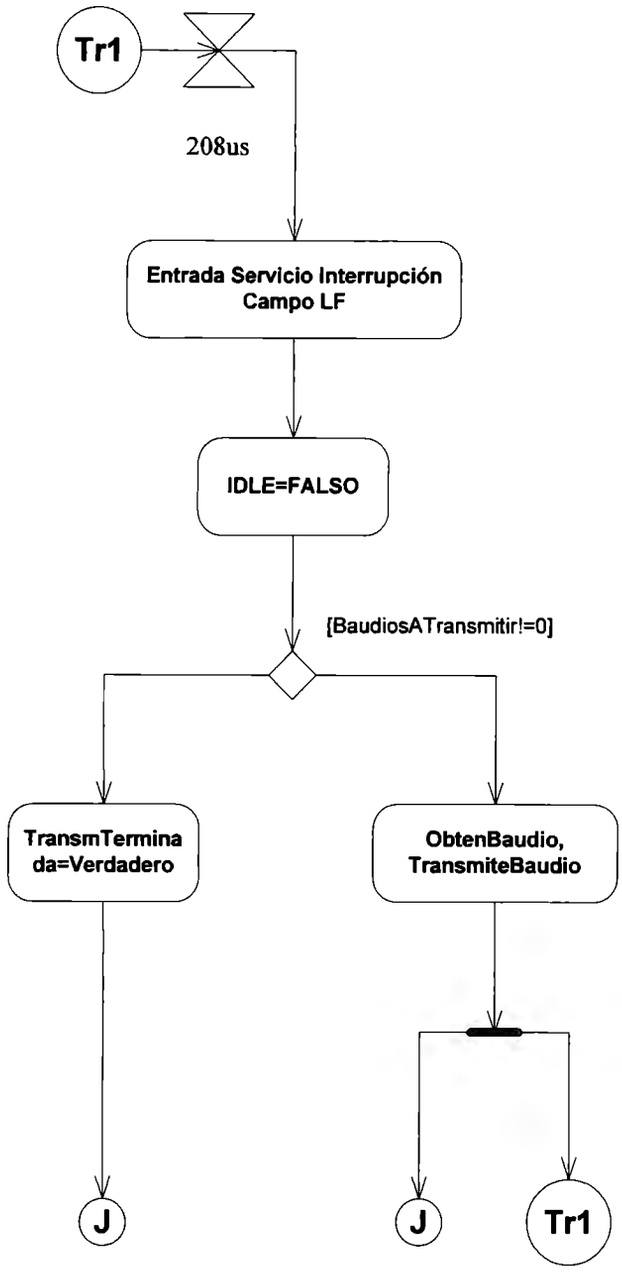
Esta tarea espera a que los procesos importantes hayan terminado, y a continuación reinicia el microcontrolador, que cambiará a modo trasmisor receptor automáticamente.



30. Hilo de Detección de *Transponder* en las Cercanías

Finalmente, se muestra el hilo de transmisión, que es encargado de cambiar el valor de un puerto del microcontrolador para transmitir la señal de radiofrecuencia. Este hilo es la innovación al despachador de Pont, ya que permite tener dos despachadores de tareas, relativamente

independientes.



31. Hilo de Transmisión de Comandos

Este hilo se encarga de transmitir los baudios del comando, *encriptado* y traducidos en código Manchester. Además, se auto termina cuando ha finalizado la transmisión. Debido a que tiene mayor prioridad que el hilo T1, su periodo de activación es más constante que la del despachador

cooperativo.

Su equivalente en código es el siguiente:

```
void TransmitCommand(void) interrupt TransmissionINTERRUPTCODE using 3{
  //Borra la bandera que indica que el hilo main está dormido
  MAC_Erase_Flag;
  //Verifica que existan más baudios a transmitir
  if (allocatedBytes!=0){
    //Cambia el estado del pin
    if ((dataArray[byteIndex] & FIRST_BIT) == FIRST_BIT){
      ML_SET_OUT_BIT;
    } else {
      ML_CLEAR_OUT_BIT;
    }
    //Prepara próxima transmisión
    dataArray[byteIndex]<<=1;
    //Calcula la cantidad de baudios restantes
    bits2Extract--;
    if (!bits2Extract){
      bits2Extract=8;
      byteIndex++;
      allocatedBytes--;
    }
  } else {
    //Detiene transmisión
    MAC_STOP_TRANS;
    //Levanta la bandera que indica que la transmisión ha terminado
    MAC_SET_TRANSMITION_FINISHED;
  }
}
```

32. Código del Hilo de Transmisión

Es necesario mencionar que la codificación de este hilo requiere de procedimientos y funciones para soportar el llenado del buffer de datos a transmitir. Además, como se desea que la duración de la ejecución, del hilo transmisor, tarde lo menos posible, es necesario hacer la codificación Manchester durante el insertado de datos.

A continuación se muestran dichas funciones de soporte:

```
/*****************************************************************
 * Este procedimiento debe ser llamada antes de comenzar el      *
 * llenado del buffer                                            *
 *****************************************************************/
void prep2Fill(void){
  byteIndex = 0;
}

/*****************************************************************
 * Este procedimiento debe ser llamada antes de comenzar la rutina *
 * de transmisión de información                                 *
```

```
 ****************************************************************/
void prep2Read(void){
  bits2Extract=8;
  allocatedBytes=byteIndex;
  byteIndex=0;
}

/****************************************************************
 * Este procedimiento inserta un byte en el buffer de transmisión                *
 ****************************************************************/
void insertByte (tByte d){
  tByte i;
  dataArray[byteIndex]=0;
  //Convierte el byte a insertar, a código Manchester
  //1. Convierte los primeros 4 bits
  for (i=0;i<3;i++){
    if (d & FIRST_BIT){
      dataArray[byteIndex]|=0x02;
    } else {
      dataArray[byteIndex]|=0x01;
    }
    dataArray[byteIndex]<<=2;
    d<<=1;
  }
  if (d & FIRST_BIT){
    dataArray[byteIndex]|=0x02;
  } else {
    dataArray[byteIndex]|=0x01;
  }
  if (MANCHESTER_TYPE){
    dataArray[byteIndex]=~dataArray[byteIndex];
  }
  byteIndex++;
  dataArray[byteIndex]=0;
  //2. Convierte los siguientes 4 bits
  for (i=0;i<3;i++){
    if (d & FIRST_BIT){
      dataArray[byteIndex]|=0x02;
    } else {
      dataArray[byteIndex]|=0x01;
    }
    dataArray[byteIndex]<<=2;
    d<<=1;
  }
  if (d & FIRST_BIT){
    dataArray[byteIndex]|=0x02;
  } else {
    dataArray[byteIndex]|=0x01;
  }
  if (MANCHESTER_TYPE){
    dataArray[byteIndex]=~dataArray[byteIndex];
  }
  byteIndex++;
}
```

33. Funciones de Soporte para el Llenado del *Buffer* de Transmisión

Ahora, se describirá cómo fue implementada la lógica de control y la interfaz de entrada.

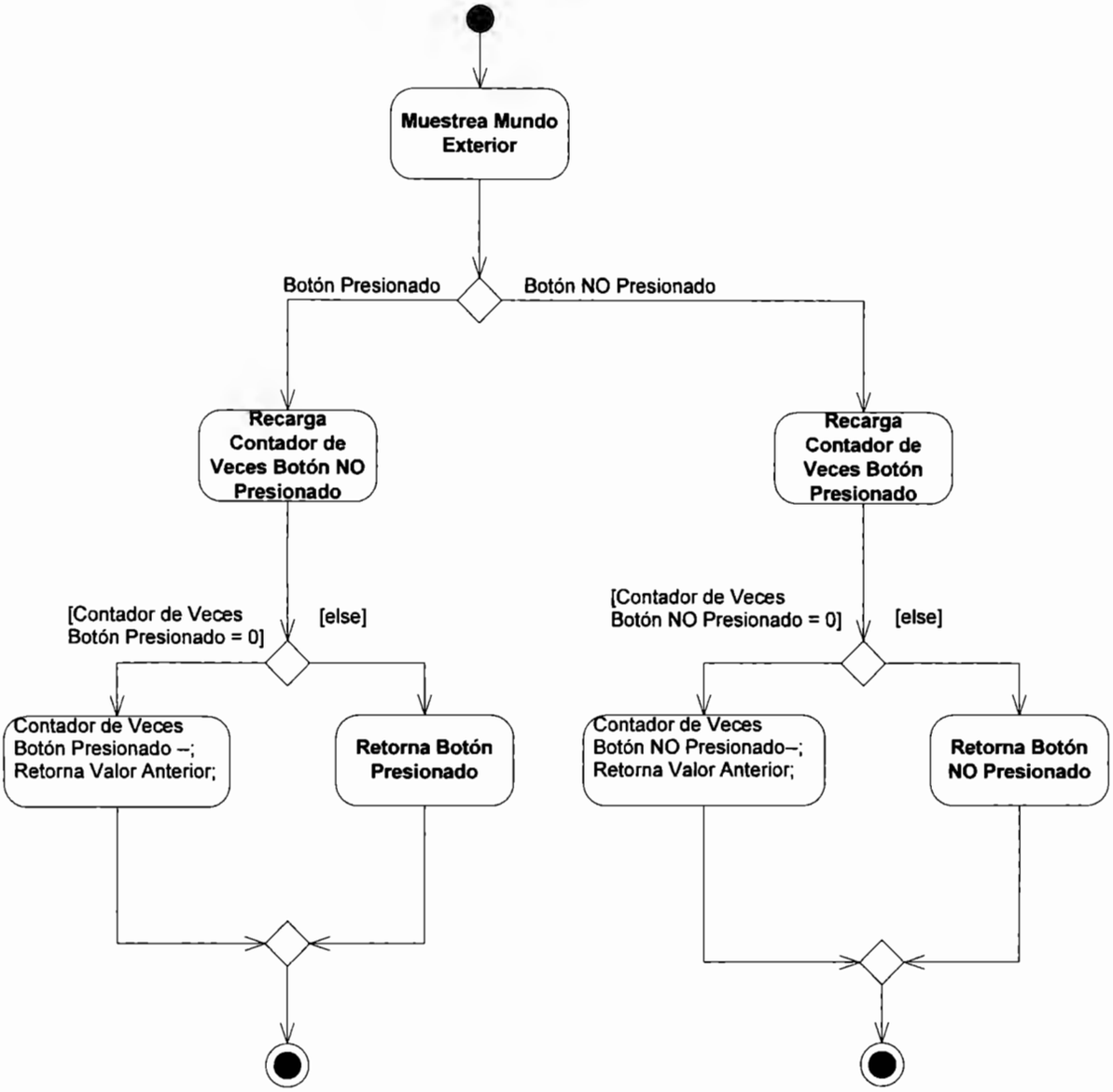
3.4. Interfaz con el Mundo Exterior y Lógica de Control

Como ya se ha mencionado anteriormente, la interfaz con el mundo real y la lógica de control, son el último paso de la metodología Jackson; éstos están encargados de implementar la funcionalidad del sistema. La interfaz del mundo real, se encarga de traducir los eventos del mundo real en eventos del sistema, y la lógica de control emite acciones de acuerdo a los eventos del sistema.

3.4.1. Interfaz con el Mundo Real

La función principal de la interfaz con el mundo exterior, es confirmar la existencia de eventos externos, y filtrar eventos ficticios, (causados por ruido electromagnético o rebotes en el interruptor). Para lograr lo anterior, se utilizan contadores de estado, que registran la cantidad de veces, que consecutivamente, el estado del evento ha sido detectado por el sistema.

A continuación se muestra el diagrama de actividad de dicha lógica y su traducción a código:



34. Diagrama de Actividad de Rutina "*Debounce*"

Traducción a código de la rutina de filtrado:

```
static bit debounceButton(tByte* onCntr, tByte* offCntr, bit buttonStat, const tByte RELOAD_VAL,
bit currVal){
    bit retVal = currVal;

    if (buttonStat){
      //Botón NO presionado
      (*onCntr) = RELOAD_VAL;
      if((*offCntr)!=0){
        (*offCntr)--;
      } else {
        //button debounced
        retVal=0;
```

```
      }
    } else {
      //Botón presionado
      (*offCntr)=RELOAD_VAL;
      if ((*onCntr)!=0){
        (*onCntr)--;
      } else {
        retVal=1;
      }
    }
    return retVal;
  }
```

35. Código que Efectúa el "Debounce" de 1 Botón

Este código puede ser optimizado por una función "*inline*", o un macro, cuando la velocidad de ejecución sea una meta importante a alcanzar.

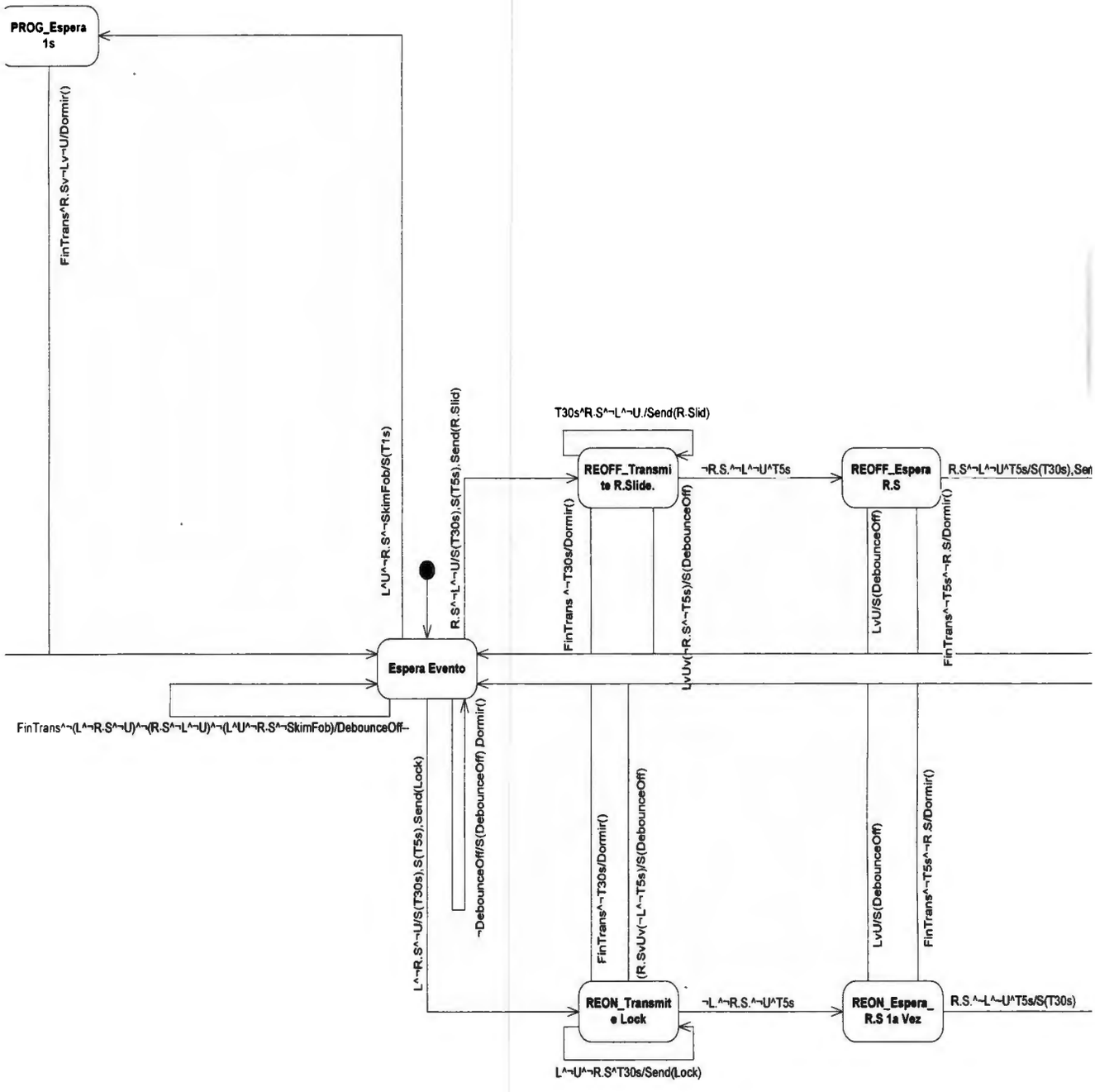
La rutina que hace el filtrado para cada botón, es la siguiente:

```
void debounceButtons(void){
  tButtons readVals;
  //Valores instantáneos
  readVals = (*ptrGetButtonsStatus)();
  //Efectúa Debounce Lock
  bEvents.butonsStruct.lock=debounceButton(&lockOnCntr,&lockOffCntr,
            readVals.butonsStruct.lock,
            LOCK_DEBOUNCE-1,
            bEvents.butonsStruct.lock);
  //Efectúa Debounce Unlock
  bEvents.butonsStruct.unlock=debounceButton(&unlockOnCntr,&unlockOffCntr,
            readVals.butonsStruct.unlock,
            UNLOCK_DEBOUNCE-1,
            bEvents.butonsStruct.unlock);
  //Efectúa Debounce Trunk
  bEvents.butonsStruct.trunk=debounceButton(&trunkOnCntr,&trunkOffCntr,
            readVals.butonsStruct.trunk,
            TRUNK_DEBOUNCE-1,
            bEvents.butonsStruct.trunk);
  //Efectúa Debounce Panic
  bEvents.butonsStruct.panic=debounceButton(&panicOnCntr,&panicOffCntr,
            readVals.butonsStruct.panic,
            PANIC_DEBOUNCE-1,
            bEvents.butonsStruct.panic);
  //Efectúa Debounce Remote Start
  bEvents.butonsStruct.remote=debounceButton(&remoteOnCntr,&remoteOffCntr,
            readVals.butonsStruct.remote,
            REMOTE_DEBOUNCE-1,
            bEvents.butonsStruct.remote);
  //Efectúa Debounce Lift Gate
  bEvents.butonsStruct.liftgate=debounceButton(&liftgateOnCntr,&liftgateOffCntr,
            readVals.butonsStruct.liftgate,
            LIFTGATE_DEBOUNCE-1,
            bEvents.butonsStruct.liftgate);
}
```

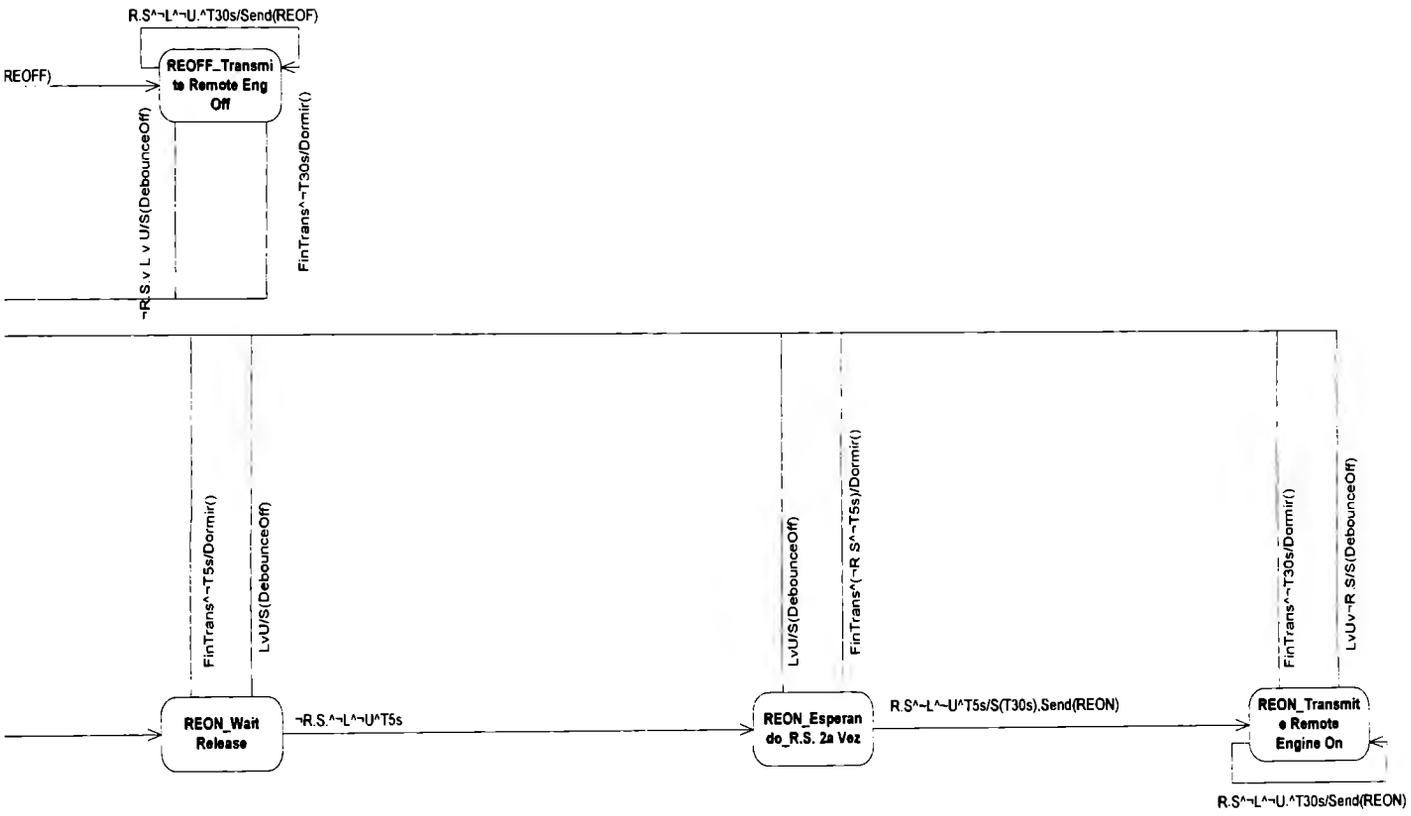
36. Código que Efectúa el "*Debounce*" de Cada Botón

3.4.2. Lógica de Control

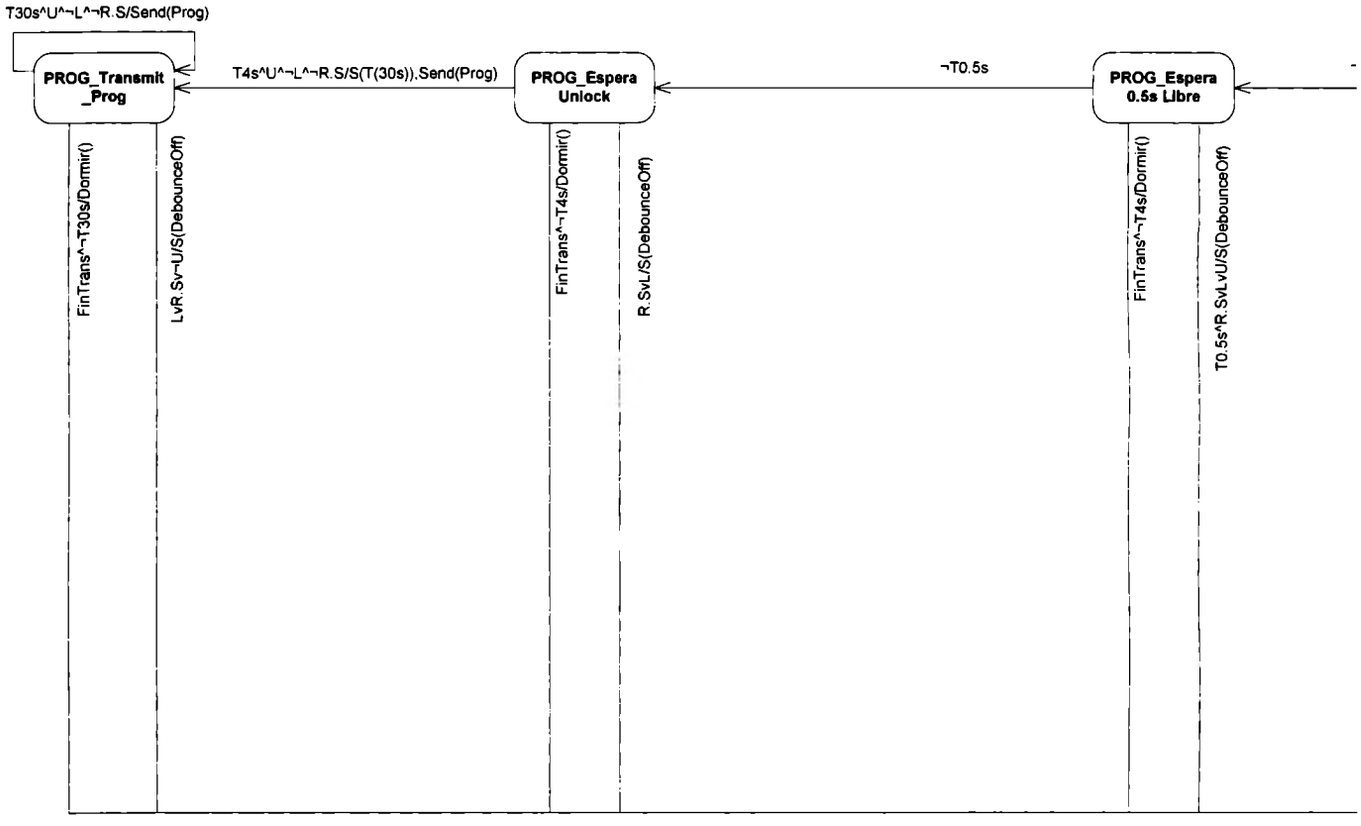
La lógica de control esta implementada por máquinas de estados que se comunican entre si utilizando variables compartidas. De esta forma, la funcionalidad compleja es dividida en secciones de poca complejidad. A continuación se muestran los diagramas de las máquinas de estados:



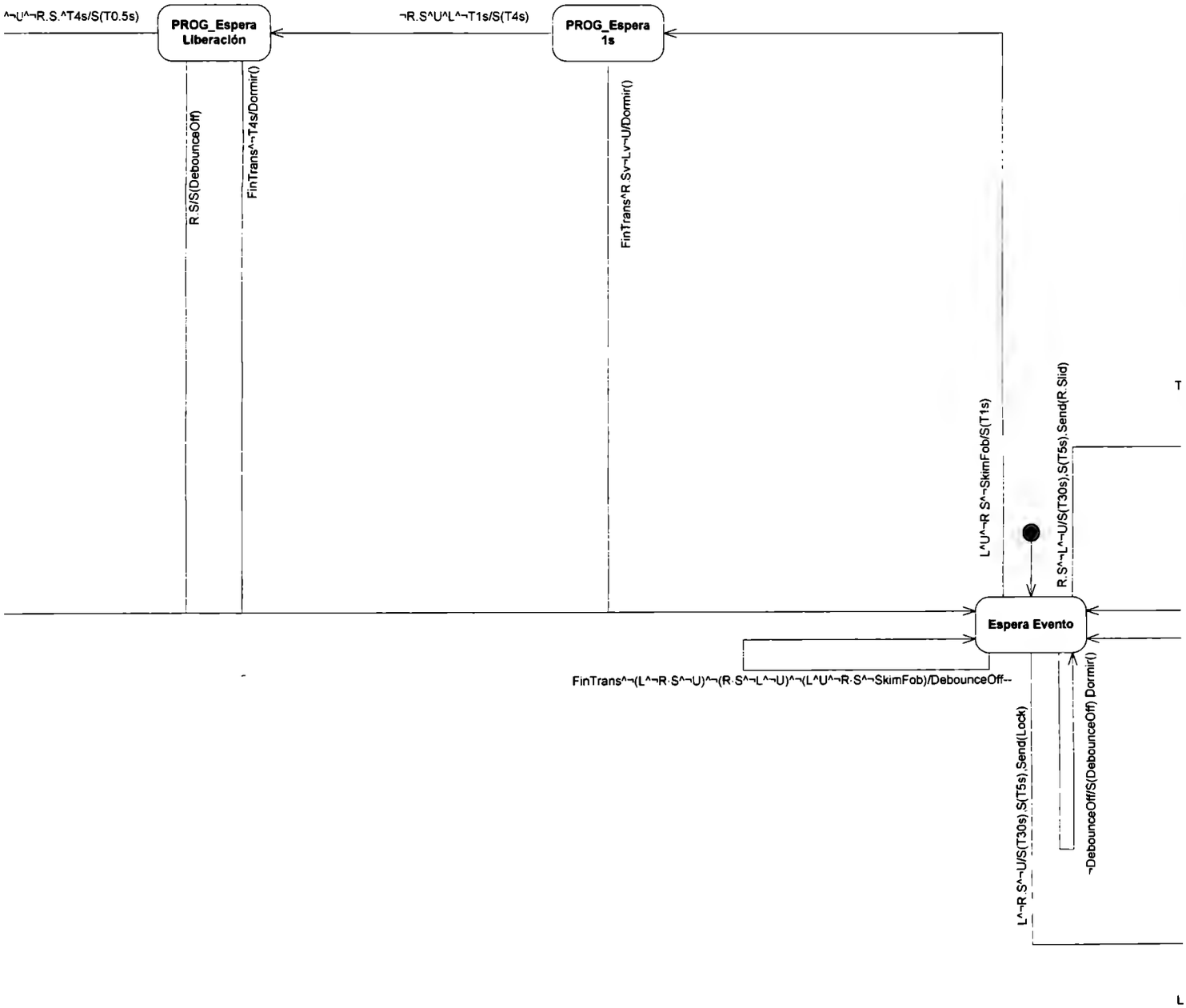
37. Diagrama de Estados



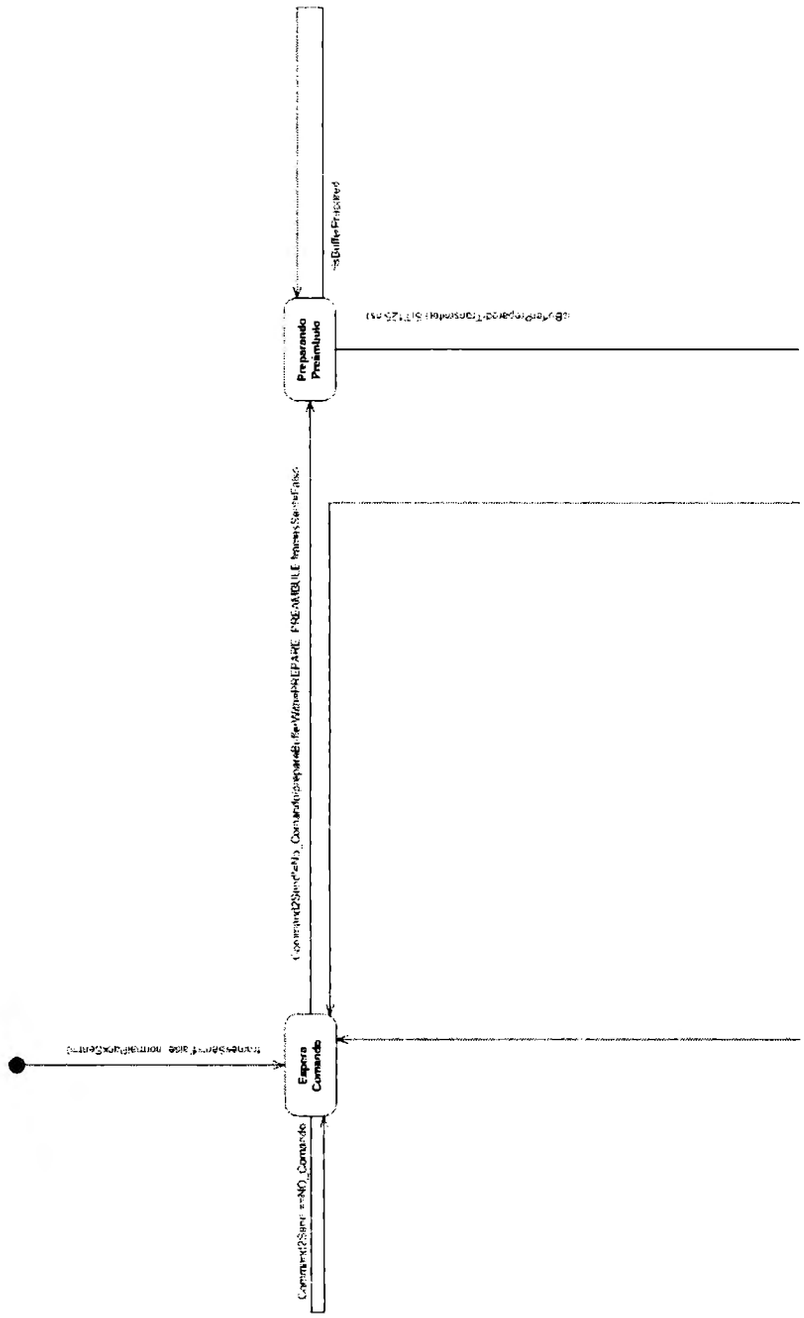
de la Funcionalidad "*Remote Engine On y OFF*"



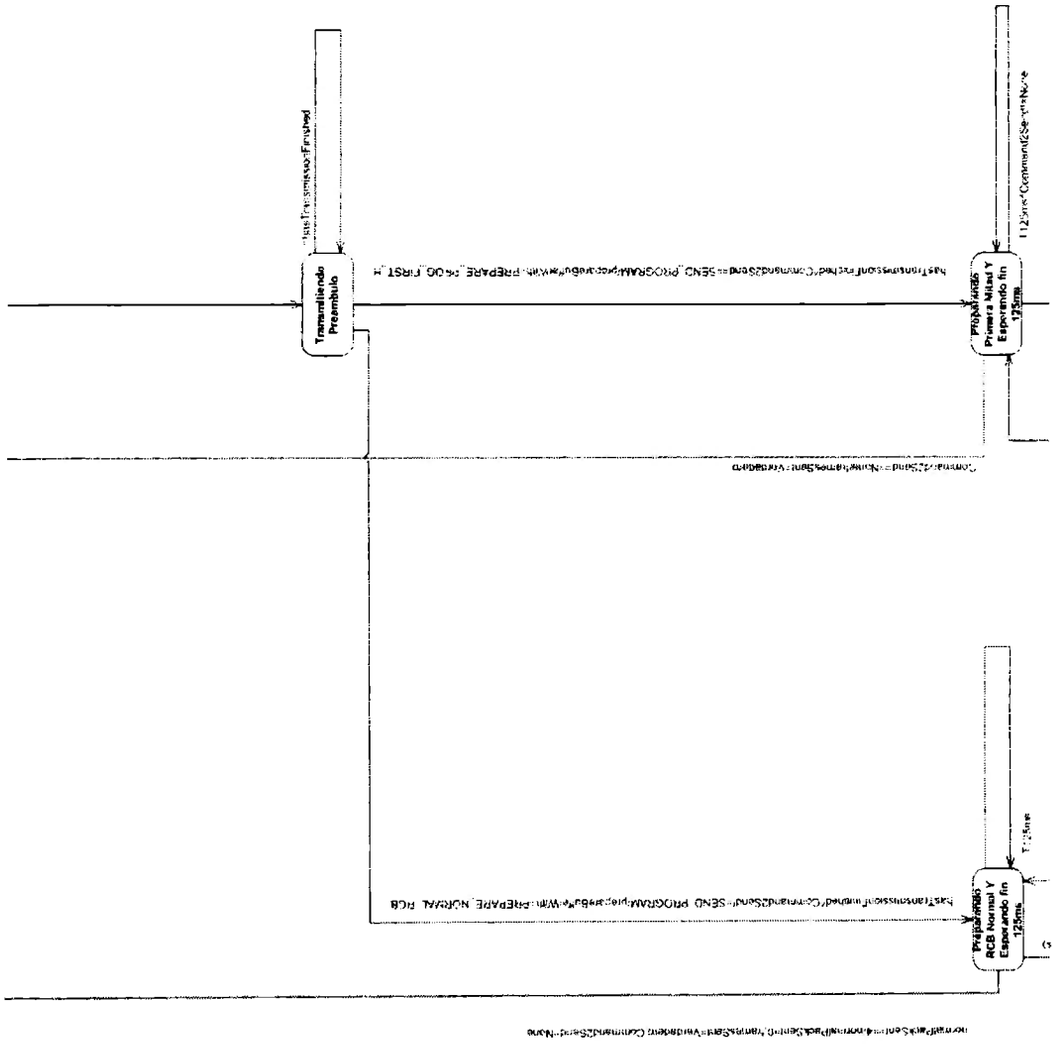
38. Decodificador d



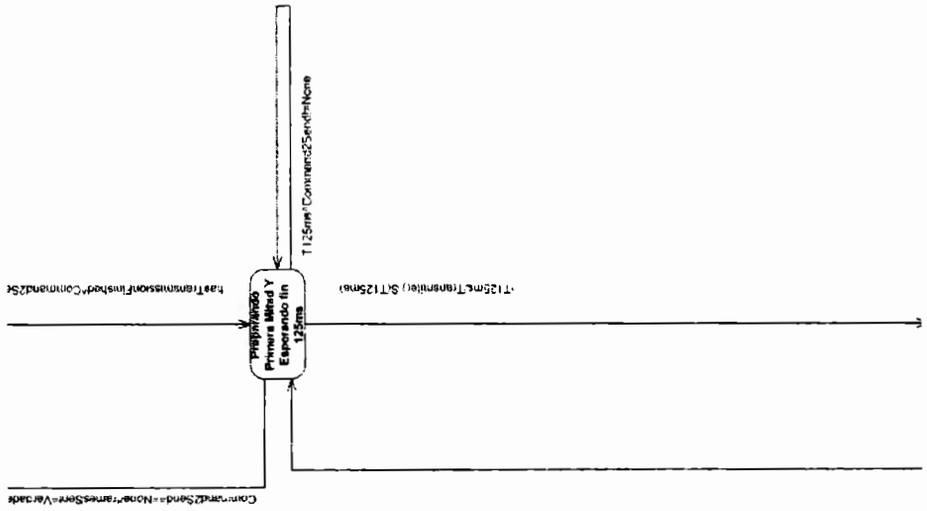
Secuencia de Programación

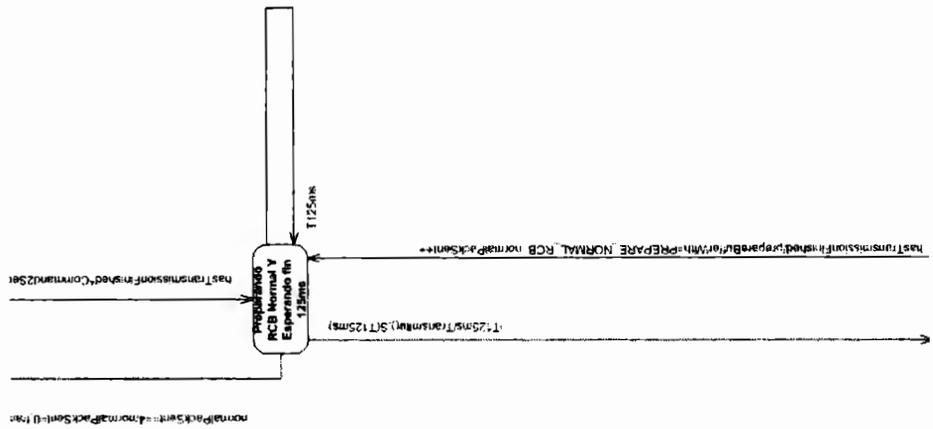


39. Maquina de E:

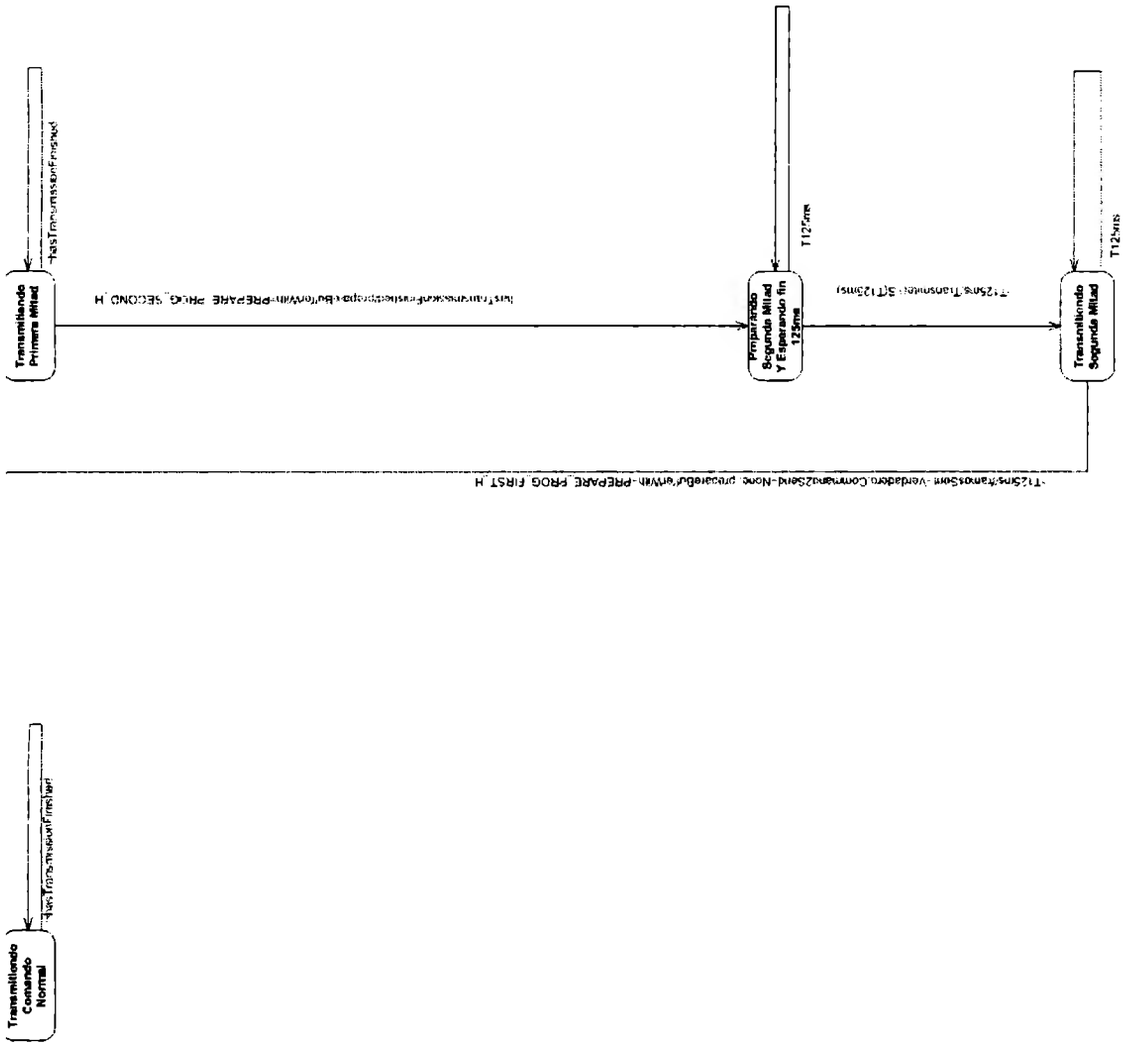


tados para Formar Mensaje a Transmitir





40. Maquina de Est



ados para Formar Mensaje a Transmitir

En el diagrama 37 más atrás, se muestra la lógica de decodificación, para detectar la secuencia que debe activar y desactivar, remotamente, el motor del automóvil.

En el diagrama 38 más atrás, se muestra la lógica de decodificación para detectar la secuencia de programación del control remoto.

Los dos diagramas, (37 y 38) representan la máquina de estados reconocedora de secuencias, y que ha sido traducida a código como sigue:

```
static void decodeButtons(void){
//Eventos del Sistema
  tButtons events;
  events=(*ptrGetBEvents)();//Lee Eventos  Externos
  switch (decoderState){
    case WAITING_FOR_EVENT:
      if (events.byte == REMOTE_EVENT){
        _30SecTimer=_30_SEC_RELOAD;
        _5SecTimer = _5_SEC_RELOAD;
        command2Send = SEND_R_SLIDE;
        decoderState = REOFF_TRANSMIT_R_SLIDE;
      } else if (events.byte == LOCK_EVENT){
        _30SecTimer=_30_SEC_RELOAD;
        _5SecTimer = _5_SEC_RELOAD;
        command2Send = SEND_LOCK;
        decoderState = REON_TRANSMIT_LOCK;
      } else if (events.byte == LOCK_UNLOCK_EVENT && !skimFOB){
        _1SecTimer=_1_SEC_RELOAD;
        decoderState = PROG_WAIT_1S;
      } else if (debounceFinished){
        debounceFinished--;
      } else {
        debounceFinished=DEBOUNCERELOAD;
        wakeBy=W_PORTS;
      }
    break;
    case REOFF_TRANSMIT_R_SLIDE:
      if ((events.byte == REMOTE_EVENT) && _30SecTimer){
        command2Send = SEND_R_SLIDE;
      } else if (!_30SecTimer && (framesSent == 1)){
        wakeBy=W_PORTS;
        decoderState = WAITING_FOR_EVENT;
      } else if

            // 1. Verifica que ningún botón, fuera de Remote
            //  sea presionado, o que se libere Remote
            //  después de 5s
            //
            (
            events.byte & (~REMOTE_EVENT)
          || (
```

```
                !events.butonsStruct.remote
              && !(_5SecTimer)
              )
            ){
        decoderState = WAITING_FOR_EVENT;
        debounceFinished=DEBOUNCERELOAD;
      } else if
              //Si se sueltan los botones antes de los 5s.
              ((events.byte == NO_EVENTS) && _5SecTimer) {
        decoderState = REOFF_WAIT_REMOTE;
      }
    break;
    case REOFF_WAIT_REMOTE:
      if //Si presionas Remote dentro de los 5s
        ((events.byte == REMOTE_EVENT)&&_5SecTimer){
        _30SecTimer=_30_SEC_RELOAD;
        command2Send = SEND_REMOTE_OFF;
        decoderState = REOFF_TRANSMIT_REMOTE_OFF;
      } else if //Si se presiona cualquier boton que no sea remote
              (events.byte & (~REMOTE_EVENT)){
        decoderState = WAITING_FOR_EVENT;
        debounceFinished=DEBOUNCERELOAD;
      } else if //Si no se presiono remote, y se terminaron los 5s
              (!_5SecTimer && !events.butonsStruct.remote && (framesSent == 1)){
        wakeBy=W_PORTS;
        decoderState = WAITING_FOR_EVENT;
      }
    break;
    case REOFF_TRANSMIT_REMOTE_OFF:
      if //Si se presiona remote dentro de los 30s
        ((events.byte == REMOTE_EVENT) && _30SecTimer){
        command2Send = SEND_REMOTE_OFF;
      } else if //Se presionó cualquier otro botón, o se liberó
              //remote
        (
          (events.byte & (~REMOTE_EVENT))
        || !events.butonsStruct.remote
        ){
        decoderState = WAITING_FOR_EVENT;
        debounceFinished=DEBOUNCERELOAD;
      } else if //Si se terminaron los 30s y se terminó de transmitir.
              (!_30SecTimer&&(framesSent == 1)){
        wakeBy=W_PORTS;
        decoderState = WAITING_FOR_EVENT;
      }
    break;
    case REON_TRANSMIT_LOCK:
      if //Se presiona lock y no se han terminado los 30 segundos
        ((events.byte == LOCK_EVENT) && _30SecTimer){
        command2Send = SEND_LOCK;
      } else if //Si se presiona cualquier botón, que no sea lock
              //o, si se suelta el botón lock después de
              //transcurridos 5s.
              (
                events.byte & (~LOCK_EVENT)
              || (
                  !events.butonsStruct.lock
                && !(_5SecTimer)
```

```
            )
           ){
              decoderState = WAITING_FOR_EVENT;
              debounceFinished=DEBOUNCERELOAD;
        //Si se liberaron los botones dentro de los 5s
      } else if ((events.byte == NO_EVENTS) && _5SecTimer) {
        decoderState=REON_WAIT_REMOTE_1;
      } else if //Si se terminaron los 30s
            (!_30SecTimer && (framesSent == 1)){
        wakeBy=W_PORTS;
        decoderState = WAITING_FOR_EVENT;
      }
    break;
    case REON_WAIT_REMOTE_1:
      if //Si presiona Remote dentro de los 5s
        ((events.byte == REMOTE_EVENT)&&_5SecTimer){
        _30SecTimer=_30_SEC_RELOAD;
        decoderState = REON_WAIT_RELEASE_ALL;
      } else if //Si presionas cualquier boton que no sea remote
              (events.byte & (~REMOTE_EVENT)){
        decoderState = WAITING_FOR_EVENT;
        debounceFinished=DEBOUNCERELOAD;
      } else if //Si no se presionó remote, y se terminaron los 5s
            //y la transmisión terminó.
              (!_5SecTimer && !events.butonsStruct.remote && (framesSent == 1)){
        wakeBy=W_PORTS;
        decoderState = WAITING_FOR_EVENT;
      }
    break;
    case REON_WAIT_RELEASE_ALL:
      //Si libero todos los botones dentro de los 5s
      if((events.byte == NO_EVENTS) && _5SecTimer){
        decoderState = REON_WAIT_REMOTE_2;
      } else if //Se se presionó cualquier botón que no es Remote
            (events.byte & (~REMOTE_EVENT)) {
        decoderState = WAITING_FOR_EVENT;
        debounceFinished=DEBOUNCERELOAD;
      } else if //Si se acabaron los 5 segudos, se terminó la transmisión
            //y no se liberó remote
            (!_5SecTimer && (framesSent == 1)){
        wakeBy=W_PORTS;
        decoderState = WAITING_FOR_EVENT;
      }
    break;
    case REON_WAIT_REMOTE_2:
      //Si se presiona el botón remote start dentro de los
      //5s.
      if ((events.byte == REMOTE_EVENT)&&_5SecTimer){
        _30SecTimer=_30_SEC_RELOAD;
        command2Send = SEND_REMOTE_ON;
        decoderState = REON_TRANSMIT_REMOTE_ON;
      } else if //Si se presiona algún botón que no sea Remote
              (events.byte & (~REMOTE_EVENT)){
        decoderState = WAITING_FOR_EVENT;
        debounceFinished=DEBOUNCERELOAD;
      } else if //Si no se presiona Remote y los 5s se terminan
              (!_5SecTimer && !events.butonsStruct.remote && (framesSent == 1)){
        wakeBy=W_PORTS;
```

```
        decoderState = WAITING_FOR_EVENT;
      }
    break;
    case REON_TRANSMIT_REMOTE_ON:
      if //Si se continúa presionando Remote y no se han terminado
        //los 30s
        ((events.byte == REMOTE_EVENT) && _30SecTimer){
        command2Send = SEND_REMOTE_ON;
      } else if //Si se terminaron los 30s y se terminó de transmitir.
            (!_30SecTimer&&(framesSent == 1)){
        wakeBy=W_PORTS;
        decoderState = WAITING_FOR_EVENT;
      } else if //Se presionó, adicionalmente a remote, otro botón o se liberó
            //Remote
            ((events.byte & (~REMOTE_EVENT))
          || !events.butonsStruct.remote
        ){
          decoderState = WAITING_FOR_EVENT;
          debounceFinished=DEBOUNCERELOAD;
      }
    break;
    case PROG_WAIT_1S:
      //Si se terminó el segundo, presionando lock y unlock
      if (!_1SecTimer && (events.byte == LOCK_UNLOCK_EVENT)){
        _4SecTimer= _4_SEC_RELOAD;
        decoderState = PROG_WAIT_RELEASE_ALL;
      } else if //Si se liberaron los botones antes de tiempo,o se presionó
            //otro botón.
             (
               (framesSent == 1)
             &&(events.byte != LOCK_UNLOCK_EVENT)
             ){
        wakeBy=W_PORTS;
        decoderState = WAITING_FOR_EVENT;
      }
    break;
    case PROG_WAIT_RELEASE_ALL:
      if //Si se liberaron todos los botones dentro de los 4s
        (events.byte == NO_EVENTS && _4SecTimer){
          _HalfSecTimer = HALF_SEC_RELOAD;
          decoderState = PROG_WAIT_HALF_SEC;
      } else if //Si se terminaron los 4s antes de liberar los botones
            (!_4SecTimer && (framesSent == 1)){
        wakeBy=W_PORTS;
        decoderState = WAITING_FOR_EVENT;
      } else if //Si se presionó cualquier otro botón que no sea
            //lock unlock
            (events.byte & (~LOCK_UNLOCK_EVENT)){
          decoderState = WAITING_FOR_EVENT;
          debounceFinished=DEBOUNCERELOAD;
      }
    break;
    case PROG_WAIT_HALF_SEC:
      if //Si se presionó algún botón antes de terminar los 0.5s
        (events.byte != NO_EVENTS && _HalfSecTimer){
          decoderState = WAITING_FOR_EVENT;
          debounceFinished=DEBOUNCERELOAD;
      } else if
```

```
            //Si se terminaron los 4s para terminar el código y
            //no se presionó ningún botón
            (!_4SecTimer&&(framesSent == 1)){
            wakeBy=W_PORTS;
            decoderState = WAITING_FOR_EVENT;
          } else if
            //Si se terminaron los 0.5s y no se presionó ningún botón
            (!_HalfSecTimer){
            decoderState = PROG_WAIT_UNLOCK;
          }
        break;
        case PROG_WAIT_UNLOCK:
          if //Si se presionó unlock antes de 4s
            (_4SecTimer && events.byte == UNLOCK_EVENT){
            _30SecTimer=_30_SEC_RELOAD;
            command2Send = SEND_PROGRAM;
            decoderState = PROG_TRANSMIT_PROG;
          } else if //Si se presinó cualquier otro botón que no sea
                //unlock
              (events.byte & (~UNLOCK_EVENT)){
              decoderState = WAITING_FOR_EVENT;
              debounceFinished=DEBOUNCERELOAD;
          } else if //Si se terminaron los 4s para cmpletar el
                //código
              (!_4SecTimer&&(framesSent == 1)){
            wakeBy=W_PORTS;
            decoderState = WAITING_FOR_EVENT;
          }
        break;
        case PROG_TRANSMIT_PROG:
          if //Si se mantiene presionado unlock y no se han terminado los 30s
            ((events.byte == UNLOCK_EVENT)&&_30SecTimer){
            command2Send = SEND_PROGRAM;
          } else if //Se terminaron los 30s
            (!_30SecTimer&&(framesSent == 1)){
            wakeBy=W_PORTS;
            decoderState = WAITING_FOR_EVENT;
          } else if //Se preionó cualquier otro botón o se liberó unlock
            ((events.byte & (~UNLOCK_EVENT)) || !events.butonsStruct.unlock){
              decoderState = WAITING_FOR_EVENT;
              debounceFinished=DEBOUNCERELOAD;
          }
        break;
        default:
            decoderState = WAITING_FOR_EVENT;
            debounceFinished=DEBOUNCERELOAD;
        break;
      }
    }
```

41. Código del Decodificador de Eventos Externos

Esta máquina de estados utiliza un conjunto de temporizadores virtuales cuyo reloj tiene una frecuencia de 4ms. Dichos temporizadores están construidos con variables de 8 y 16 bits, que se decrementan cuando la lógica de control es llamada, hasta que su valor llega a cero. Siendo el

código correspondiente a estos temporizadores virtuales, el siguiente:

```
static void updateTimers(void){
  if (_30SecTimer!=0){
    _30SecTimer--;
  }
  if (_5SecTimer!=0){
    _5SecTimer--;
  }
  if (_4SecTimer!=0){
    _4SecTimer--;
  }
  if (_1SecTimer!=0){
    _1SecTimer--;
  }
  if (_HalfSecTimer!=0){
    _HalfSecTimer--;
  }
  if (_124mSecTimer!=0){
    _124mSecTimer--;
  }
}
```

42. Decremento de Temporizadores Virtuales

Una vez que se ha reconocido una secuencia de eventos externos, es necesario crear un paquete de información a transmitir, que cumpla con los requerimientos establecidos en 1.3 más atrás. Para este caso, el diagrama 39 representa la máquina de estados utilizada para implementar la funcionalidad antes mencionada. Ésta, se comunica con la máquina reconocedora de secuencias mediante la variable "*command2Send*".

Su traducción a código es la siguiente:

```
static void composeFrame(void){
  switch (frameComposerState){
    case S_WAITTING:
      if (command2Send!=SEND_NONE){
        prepareBufferWith=PREPARE_PREAMBULE;
        framesSent=FALSE;
        frameComposerState=S_PREP_PREAMBULO;
      }
    break;
    case S_PREP_PREAMBULO:
      if (isBufferPrepared){
        _124mSecTimer=_124_M_SEC_RELOAD;
        (*ptrStartTransmition)();
        frameComposerState=S_TRANSMITTING_PREAMB;
      }
    break;
    case S_TRANSMITTING_PREAMB:
      if (hasTransmissionFinished && command2Send==SEND_PROGRAM){
```

```
        prepareBufferWith=PREPARE_PROG_FIRST_H;
        frameComposerState = S_PREP_PROG_1_HALF;
      } else if (command2Send!=SEND_PROGRAM){
        prepareBufferWith=PREPARE_NORMAL_RCBP;
        frameComposerState = S_PREP_RCBN;
      }
    break;
    case S_PREP_PROG_1_HALF:
      if (command2Send==SEND_NONE){
        framesSent=TRUE;
        frameComposerState=S_WAITTING;
      } else if (!_124mSecTimer){
        (*ptrStartTransmition)();
        _124mSecTimer=_124_M_SEC_RELOAD;
        frameComposerState=S_TRANS_PROG_1_HALF;
      }
    break;
    case S_TRANS_PROG_1_HALF:
      if (hasTransmissionFinished){
        prepareBufferWith=PREPARE_PROG_SECOND_H;
        frameComposerState=S_PREP_PROG_2_HALF;
      }
    break;
    case S_PREP_PROG_2_HALF:
      if (!_124mSecTimer){
        (*ptrStartTransmition)();
        _124mSecTimer=_124_M_SEC_RELOAD;
        frameComposerState=S_TRANS_PROG_2_HALF;
      }
    break;
    case S_TRANS_PROG_2_HALF:
      if (!_124mSecTimer){
        framesSent=TRUE;
        command2Send=SEND_NONE;
        prepareBufferWith=PREPARE_PROG_FIRST_H;
        frameComposerState=S_PREP_PROG_1_HALF;
      }
    break;
    case S_PREP_RCBN:
      if (normalPackSent==4){
        normalPackSent=0;
        framesSent=TRUE;
        command2Send=SEND_NONE;
        frameComposerState=S_WAITTING;
      } else if (!_124mSecTimer){
        (*ptrStartTransmition)();
        _124mSecTimer=_124_M_SEC_RELOAD;
        frameComposerState=S_TRANS_RCBN;
      }
    break;
    case S_TRANS_RCBN:
      if (hasTransmissionFinished){
        prepareBufferWith=PREPARE_NORMAL_RCBP;
        normalPackSent++;
        frameComposerState=S_PREP_RCBN;
      }
    break;
```

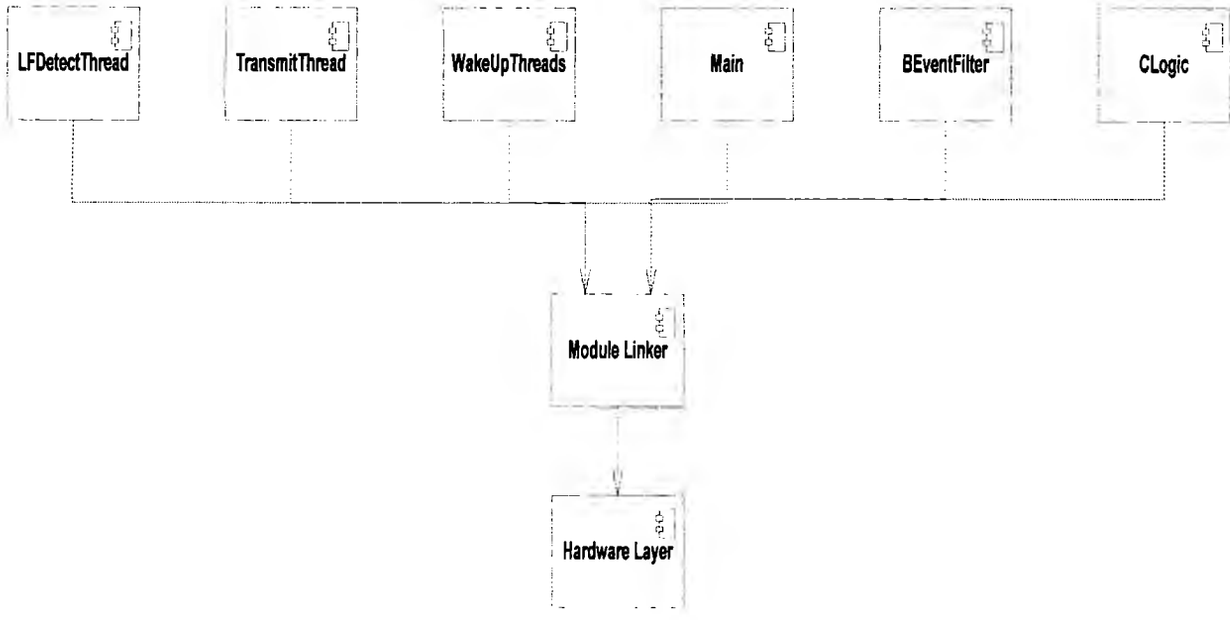
```
        default:
        break;
      }
    }
```

43. Código de la Máquina de Estados que Crea los Paquetes a Enviar

3.5. Arquitectura del Sistema

Una vez terminado el método de desarrollo Jackson, es necesario documentar los estilos utilizados en el sistema, y su forma de implementarlos.

Para el efecto se comenzará a describir el sistema, desde el punte de vista modular:



44. Vista Modular del Sistema

En el anterior diagrama se muestran las dependencias permitidas entre los componentes del sistema. En este estilo se concentran las dependencias de todos los módulos en el módulo *Hardware Layer*, que está conformado por: *hardware services* y *uC Registers*. Estos módulos determinan las prestaciones del microcontrolador utilizado, de manera que la migración entre plataformas de hardware se facilita, ya que los cambios se concentran en esos dos módulos.

Por otro lado, se observa que los componentes que le dan la funcionalidad al sistema:

1. *LFDetectThread*: Que es la parte de la interfaz con el mundo exterior, encargada de cambiar al modo transmisor receptor.
2. *TransmitThread*: Que es la parte de la interfaz con el mundo exterior, encargada de transmitir los comandos de radiofrecuencia.
3. *WakeUpThreads*: Que es la parte del despachador, encargada de despertar las tareas cooperativas del sistema.
4. El despachador *Main*: Que es la parte del despachador, encargada de despachar las tareas cooperativas del sistema.
5. *BEventFilter*: Que es la parte de la interfaz con el mundo exterior, encargada de confirmar la existencia de eventos externos.
6. *CLogic*: Que es la lógica de control del sistema.

No tienen dependencias entre ellos, a pesar de que se comunican entre si. Todos ellos dependen de *Module's Linker*, que es el componente encargado de describir las conexiones entre-componentes. Lo anterior, facilita la actualización de componentes, ya que para reemplazar un módulo por otro, solo es necesario modificar el descriptor de conexiones *Module's Linker*, en lugar de modificar todos los archivos que dependen del componente intercambiado.

Para lograr lo anterior, se utilizó la técnica de inyección de dependencias, que requiere que el lenguaje de implementación soporte paso de referencias a funciones. De esta manera las referencias se pasan de un componente al módulo *ModulesLinker*, y de *ModulesLinker* a otro componente.

A continuación se muestra el archivo descriptor de conexiones entre módulos: (Para saber más sobre el paso de referencias a funciones en el compilador utilizado ver la referencia [39]):

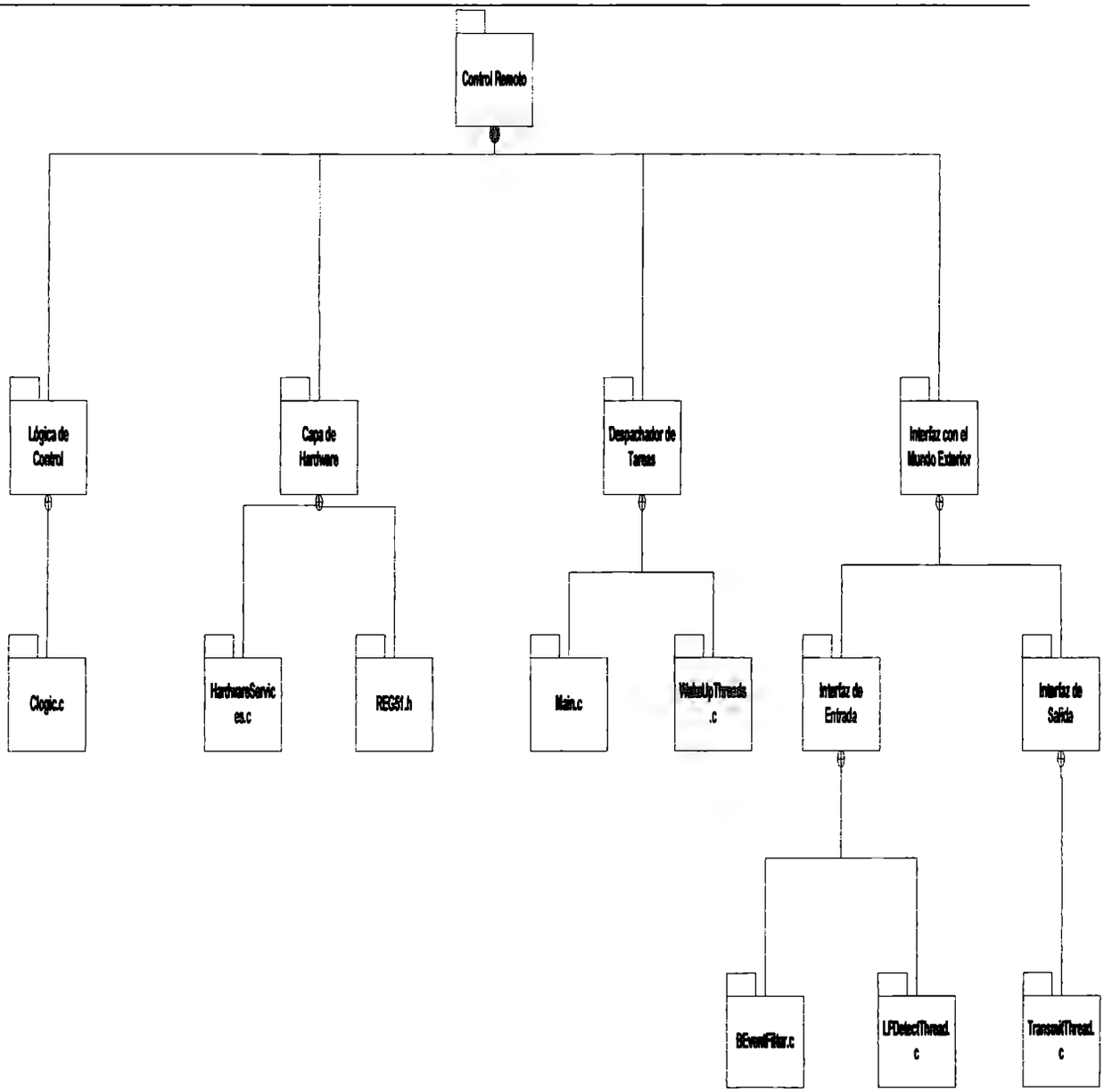
```
/*****************************************************************
 * Archivo: ModulesLinker.h                                      *
```

```
 * Función: Descriptor de las conexiones entre módulos                    *
 ************************************************************************/

#ifndef _MODULESLINKER_H
#define _MODULESLINKER_H

#include "..\HardwareLayer\REG51.H"
#include "..\HardwareLayer\HardwareServices.h"

#include "..\InputInterface\LFDetectThread.h"
#include "..\InputInterface\BEventFilter.h"

#include "..\OutputInterface\TransmitThread.h"

#include "..\TasksDispatcher\WakeUpThreads.h"
#include "..\TasksDispatcher\Main.h"

#include "..\ControlLogic\CLogic.h"
/*****************************************************
 * Conexiones de Reg51
 *****************************************************/

// Con:  LFDetectThread
#define LFDetINTERRUPTCODE       INTERRUPT_EXTERNAL_0

// Con: TransmitThread
#define TransmissionINTERRUPTCODE INTERRUPT_Timer_0_Overflow

// Con: WakeUpThreads
#define CLogicINTERRUPTCODE      INTERRUPT_Timer_1_Overflow
#define PortINTERRUPTCODE        INTERRUPT_EXTERNAL_1

/*****************************************************
 * Conexiones de HardwareServices
 *****************************************************/



//Con WakeUpThreads
#define ML_SW_CT_2_PT DEF_switchTimer12Ex1
#define ML_SW_PT_2_CT( hi, lo) DEF_switchExt12Timer1( hi,lo)
#define ML_WAS_MAIN_SLEEPING DEF_isSleepingSignalSet


//Con LFDetectThread
#define ML_RESET resetUc
#define ML_ENABLE_LF_INT EnableEx0
#define ML_DISABLE_LF_INT DisableEx0

//Con BEventFilter
#define ML_GET_BUTTONS GetButtons

//Con Main
#define ML_ENTER_IDLE DEF_go2Sleep
#define ML_START_AWAKER( hi, lo) DEF_switchExt12Timer1( hi,lo)

//Con TransmitThread
#define ML_ERASE_SLEEPING_SIGNAL DEF_eraseSleepFlag
```

```
#define ML_SET_OUT_BIT DEF_setTransmitBit
#define ML_CLEAR_OUT_BIT DEF_clearTransmitBit
#define ML_STOP_TRANSMISSION DEF_stopT0

//Con CLogic
#define ML_START_TRANSMITION Restart_T0

/*****************************************************
 * Conexiones de BEventFilter
 *****************************************************/
//Con Main
#define ML_BOTTON_E_FILTER debounceButtons
#define ML_INIT_DEB initDebouncer
//Con CLogic
#define ML_GET_B_EVENTS getBEvents
//Con WakeUpThreads
#define ML_RESET_DEBOUNCER initDebouncer

/*****************************************************
 * Conexiones de CLogic
 *****************************************************/
//Con Main
#define ML_CONTROL_L CLogic
#define ML_INIT_CLOGIC initCLogic
//Con WakeUpThreads
#define ML_IS_TIMER_AWAKER IS_TIMER_AWAKER
#define ML_SET_TIMER_AS_AWAKER SET_TIMER_AS_AWAKER
//Con TransmitThread
#define ML_SET_TRANSMITION_FINISHED HAS_TRANS_FINISHED

/*****************************************************
 * Conexiones de TransmitThread
 *****************************************************/

//Con CLogic
#define ML_PREP_2_FILL prep2Fill
#define ML_INSERT_BYTE insertByte
#define ML_SET_BUFFER_READY prep2Read

/*****************************************************
 * Conexiones de WakeUpThreads
 *****************************************************/
//Con Main
#define ML_INIT_WUT initWakeUpThread

/*****************************************************
 * Conexiones de Main
 *****************************************************/
//Con WakeUpThread
#define ML_TURN_ON_AWAKER_BELL DEF_Turn_On_Awaker_Bell

#endif
```

45. Módulo Descriptor de Conexiones

Otra vista modular del sistema, muestra los subsistemas del control remoto:



46. Descomposición en Subsistemas

3.6. Validación

1. **Validación del nuevo despachador**: Tal como Pont lo menciona en la página 263 de [26], el despachador cooperativo debe seguir la regla de una sola interrupción por microcontrolador; sin embargo, el nuevo despachador requiere más de una interrupción habilitada para prestar sus nuevos servicios. Debido a lo anterior, es necesario verificar que las interrupciones adicionales no afecten al sistema.
2. **Validación del sistema**: Dado que UML es un lenguaje formal y se presume como no ambiguo [37], se traducirá el patrón de diseño en código que sea capaz de implementar la funcionalidad descrita en 1.3 más atrás, y se efectuarán pruebas de caja cerrada, (*black box testing*), para comprobar que el diseño lleva a la implementación correcta.

3.7. Pruebas Análisis e Interpretación de Resultados

3.7.1. Validación del Nuevo Despachador

Como se mencionó en el capítulo anterior, es necesario probar que la interrupción del transmisor no afecte negativamente el desempeño del despachador, al provocar estados inconsistentes en el sistema. Para lo cual es requerido un análisis de concurrencia de tres hilos de ejecución:

1. El hilo *main*, que se encarga de despachar la interfaz con el mundo exterior y la lógica de control.
2. El hilo despertador.
3. El hilo de transmisión

Después de despachar las tareas correspondientes, el hilo *main* pone en estado IDLE al microcontrolador, y entra en espera de reactivación, para que alguno de los hilos despertadores, ya sea el periódico o el asíncrono, (que es el que espera por un evento externo), lo active, y reinicie el ciclo de ejecución de tareas. Sin embargo, no solo la activación del hilo despertador despierta al hilo *main*; el hilo de transmisión también lo reactiva. Por lo anterior, es necesario incluir el patrón *guarded wait* de la referencia [43] durante la espera, quedando el código como sigue:

```
while (!isAwakerRinging){
  MAC_Main_Go_2_Sleep;
}
```

47. Aplicación del patrón *Guarded Wait*

Dado lo anterior, los hilos despertadores se encargarán de modificar la bandera de condición *isAwakerRinging*:

```
void wakeUpByTimer(void) interrupt CLogicINTERRUPTCODE {
  //Primero detecta si el hilo principal estaba durmiendo, o no
  if (MC_Was_Main_Sleeping){
    //Determina si el control hadado la orden de apagar el
    //Despertador periódico
    if (MC_Is_Timer_AWAKER){
      //Determina si el hilo despertador ha esperado 37us a que
      //el hilo main se durmiera.
      if (waitingCycles){
        MC_Reload_Timer(SLEEP_4MS_MINUS_37US_HI, SLEEP_4MS_MINUS_37US_LO);
      } else {
        MC_Reload_Timer(SLEEP_4MS_HI, SLEEP_4MS_LO);
      }
      //Pone la bandera que indica, al hilo principal, que debe
      //hacer otra ronda activando sus tareas
      MC_Turn_On_Awaker_Bell;
    } else {
      //Apaga el despertador periódico y cambia al despertador
      //por eventos
      (*ptrResetDebouncer)();
      MC_Switch_Waker_2_Ports;
    }
    waitingCycles=0;
  } else {
    //Si el hilo principal estaba despierto cuando se activo el
    //despertador, espera 37us a que se duerma.
    waitingCycles=1;
   //Espera 37us e intenta nuevamente
    MC_Reload_Timer(SLEEP_37uS_HI,SLEEP_37uS_LO);
  }
}

void wakeUpByPort(void) interrupt PortINTERRUPTCODE {
  //Enciende el despertador periódico
  MC_Set_Timer_As_Awaker;
  MC_Reload_Timer(SLEEP_4MS_HI,SLEEP_4MS_LO);
  //Pone la bandera que indica, al hilo principal, que debe
  //hacer otra ronda activando sus tareas
  MC_Turn_On_Awaker_Bell;
}
```

48. Poniendo la Bandera del *Guarded Wait*

Solucionado este problema, es necesario considerar el caso cuando la interrupción del despertador periódico ocurre inmediatamente después de hacer la verificación de la condición de ciclo *while*, pero antes de poner en estado IDLE al microcontrolador. Esto puede ocurrir si se activa primero la interrupción de transmisión, y luego la interrupción periódica del despertador, pero antes de que el hilo *main* ponga a dormir al CPU.

La solución a este problema, es sincronizar la tarea que despierta al microcontrolador, con el

hilo *main*, de manera que se evite, que el hilo principal reciba la notificación para despertarse, después de haber verificado dicha condición, causando que las tareas dejen de ejecutarse por un ciclo.

Para resolver el problema, se utiliza una bandera que indica el momento cuando el microcontrolador cambia de estado funcional, (entra en IDLE) y el código utilizado para este caso, es el siguiente:

```
//Lo pone a dormir y pone la bandera de propósito
//general en 1. Por motivos de "performance" se ha
//dejado este código cómo una macro

#define DEF_go2Sleep PCON |= 0x05
```

49. Código que Pone a Dormir el Microcontrolador e Indica que está Dormido

En el anterior macro, se pone a dormir el microcontrolador, y simultáneamente se levanta la bandera de propósito general.

Aunque que el microcontrolador de *Philips* no cuenta con banderas de propósito general, como el 89C51, esta función si es sujeta a implementación.

Antes de atender la interrupción que despierta al microcontrolador *Philips*, el chip siempre ejecuta la instrucción inmediata a la que puso al microcontrolador en modo IDLE. De esta forma, basta con hacer lo siguiente:

```
SetIdleFlag()//Macro para entrar en modo IDLE
SetSleepingFlag()//Macro para activar la bandera de propósito general
```

50. Opción para Entrar en IDLE con el PCF7941

En otras palabras, poner inmediatamente después del código que pone a dormir al CPU, el código que pone la bandera indicadora, resolviéndose así la incompatibilidad entre microcontroladores.

Hecho esto, ahora es necesario borrar la bandera desde el hilo de transmisión, de manera que el hilo despertador se de cuente que el hilo *main* está en un estado inconsistente. Para lograr esto, se ha llamando al siguiente macro desde el hilo de transmisión:

```
//Borra la bandera que indica que el microcontrolador duerme
#define DEF_eraseSleepFlag PCON &= 0xFB
```

51. Macro para Borrar la Bandera de Estado IDLE

Que es una macro que borra la bandera que indica que el hilo *main* está en estado consistente.

En el hilo despertador se incluye la lógica que verifica dicha condición.

```
void wakeUpByTimer(void) interrupt CLogicINTERRUPTCODE {
  //Primero detecta si el hilo principal estaba durmiendo, o no
  if (MC_Was_Main_Sleeping){//Detecta si el CPU está en IDLE
    //Determina si el control hadado la orden de apagar el
    //Despertador periódico
    if (MC_Is_Timer_AWAKER){
      //Determina si el hilo despertador ha esperado 37us a que
      //el hilo main se durmiera.
      if (waitingCycles){ //Reajusta el tiempo de sincronización
        MC_Reload_Timer(SLEEP_4MS_MINUS_37US_HI, SLEEP_4MS_MINUS_37US_LO);
      } else {
        MC_Reload_Timer(SLEEP_4MS_HI, SLEEP_4MS_LO);
      }
      //Pone la bandera que indica, al hilo principal, que debe
      //hacer otra ronda activando sus tareas
      MC_Turn_On_Awaker_Bell;
    } else {
      //Apaga el despertador periódico y cambia al despertador
      //por eventos
      (*ptrResetDebouncer)();
      MC_Switch_Waker_2_Ports;
    }
    waitingCycles=0;
  } else {
      //Si el hilo principal estaba despierto cuando se activo el
      //despertador, espera 37us a que se duerma.
    waitingCycles=1;
   //Espera 37us e intenta nuevamente
    MC_Reload_Timer(SLEEP_37uS_HI,SLEEP_37uS_LO);
  }
}
```

52. Hilo Despertador

En el código anterior, se observa que el despertador entra en estado de espera hasta que el hilo main se ponga en un estado consistente, y luego lo despierta. Evitando que la condición para despertar al CPU sea ignorada por haberse recibido fuera de tiempo.

Dicha espera, debe ser lo suficientemente corta, como para impedir que el hilo transmisor despierte nuevamente al microcontrolador, y el problema se repita, pero lo suficientemente grande, como para que el CPU realice el cambio de contexto entre el hilo despertador y el hilo

main, y ponga al CPU en estado IDLE.

Como se recordará en el capítulo 1.3 más atrás, se deberá activar el hilo de transmisión cada 208us para transmitir un baudio de información, por lo que el límite superior debe ser menor que ese límite, y para medir el límite inferior, se ha desensamblado el código que incluye el cambio de contexto, y la instrucción que pone en IDLE al CPU, y se han calculado 37 microsegundos.

Con esta información el código del despachador se ha probado, y no se ha encontrado ningún estado inconsistente.

3.7.2. Validación del Sistema

La funcionalidad del sistema ha sido ejecutada en tres partes: validando la interfaz de entrada con el mundo exterior, la interfaz de salida con el mundo exterior, y la lógica de control.

1. Para probar la interfaz con el mundo exterior, se verificó que los eventos externos se hayan producido después de registrar consecutivamente la cantidad de pre-eventos requeridos. Por ejemplo para el botón Remote, 3 eventos. (Ver apéndice B para mayor información)
2. La interfaz de envío de mensajes ha sido probada por separado. Se ha llenado el buffer de transmisión con datos arbitrarios, y se ha verificado que estos hayan sido transmitidos por la terminal seleccionada.
3. La lógica de control ha sido probada por partes. Primero se verificó que la máquina de estados que se encarga de encontrar los códigos de los eventos externos, funcionara de acuerdo con las especificaciones en 1.3 más atrás. A continuación, se verificó que la máquina de estados, encargada de construir el paquete de los mensajes a enviar funcionara de acuerdo a las especificaciones en el apéndice B.

4. Conclusiones

4.1. Recapitulación

Como se estableció en la introducción, este trabajo está enfocado a encontrar formas nuevas de diseñar el sistema de control remoto; de manera que se facilite la reutilización de piezas de software. Por lo que se han revisado patrones y estilos, actualmente utilizados para el desarrollo de sistemas embebidos.

Se ha determinado que un factor primordial para facilitar la reutilización de componentes está en el nuevo despachador cooperativo. Además, que la inyección de dependencias reduce significativamente la cantidad de conexiones entre módulos, facilitando la escritura y reutilización de componentes.

Se ha descrito con diagramas de actividad UML y código, el nuevo despachador, además que se ha probado que el nuevo patrón no conduce a estados inconsistentes al sistema.

Aún cuando no se ha realizado una verificación que integre al mismo tiempo todos los componentes del sistema, se ha probado la lógica de control y la interfaz con el mundo real, mostrando que el nuevo despachador puede ser utilizado en sistemas como el que ha sido tratado en esta investigación.

4.2. Principales Contribuciones de la Investigación

La primera contribución de esta investigación, es el nuevo patrón del despachador de tareas, que es un software encargado de activar componentes del sistema. Éste, utiliza el temporizador T0 para activar la lógica de transmisión de mensajes, mientras que usa el temporizador T1 para activar indirectamente las tareas de la lógica de control y la interfaz con el mundo externo en el hilo main.

La segunda contribución de esta investigación, es la aplicación de la inyección de dependencias para concentrar las conexiones entre módulos en un solo archivo. Esto permite realizar actualizaciones en los componentes, modificando únicamente un archivo.

La tercera contribución de esta investigación, radica en la implementación del sistema de control remoto, que podrá ser tomado como referencia en investigaciones posteriores.

4.3. Limitaciones

1. Debido a que la tesis se debió terminar en el periodo de un año, solamente se examinó un problema real en esta investigación
2. Debido a que no se contó con el material necesario para desarrollar el sistema con el microcontrolador usado en Siemens, este, se probó en un simulador de un microcontrolador con características similares al real.
3. Debido a que el compilador elegido solamente soporta la creación y simulación de código de hasta 2KB[29], no se realizó una prueba integradora.

[29] En la realidad parece que el simulador deja de funcionar después de los 1.4KB

4.4. Trabajo a Futuro

Se sugiere adaptar el código presentado aquí al lenguaje del compilador sdcc, ("*Small Device C Compiler*"), que es un compilador gratuito, cuyo código fuente esta abierto y que cumple con muchas de las especificaciones de ANSI C y del compilador actualmente utilizado.

Por otro lado, se pienza que se deben invertir esfuerzos en la investigación, desarrollo y aplicación de herramientas que traduzcan de un modelo formal de sistema, a código ejecutable directamente en diferentes plataformas. Esto traería la ventaja de ocultar los detalles de implementación del sistema, y facilitaría, al diseñador, el concentrarse en la arquitectura del sistema.

Bibliografía

[1] AYAVOO D., PONT M.J., SHORT M, PARKER S., Two novel shared-clock scheduling algorithms for use with CAN-based distributed systems en Proceedings of the Second UK Embedded Forum, Noviembre 2005.

[2] BAS, Len y KAZMAN, Rick, Sofware Architecture in Practice, Addison Wesley, Segunda edición, EUA, 2004

[3] BOWEN, Jonathan, HINCHEY, Michael, Seven More Myths of Formal Methods, en Software, IEEE Volumen 12, Issue 4, Julio 1995 Páginas: 34 – 41

[4] BOWEN, Jonathan, HINCHEY, Michael, Ten Commandments of Formal Methods... Ten Years Later, en IEE, Volumen 28, Issue 4, 995 Páginas:56 – 63

[5] CHENG, Albert M, Real-Time Systems Scheduling and Verification, Editorial Wiley-InterScience, Primera edición, 2002.

[6] CLEMENTS, Paul, Bechman Felix, Documenting Software Architecture, Addison Wesley, primera edición, sexta reimpresión, 2005

[7] CORTES Luis Alejandro; ELES, Petra; PENG, Zebo, Verification of Embedded Systems Using a Petri Net Based Representation, en 13th International Symposium on System Synthesis (ISSS'00) p. 149

[8] COULOURIS G., DOLLIMORE J., KINDBERG T.; Sistemas Distribuidos, Conceptos y Diseño, Addison Wesley, tercera edición, 2001.

[9] ECO, Humberto, Cómo se hace una tesis, Limpergraf, Tercera reimpresión, España, 2003

[10] EDWARDS T., PONT M.J., SHORT M., SCOTSON P., CRUMPLER S.; A comparison of synchronous and asynchronous network architectures for use in embedded control systems with duplicate processor nodes en Proceedings of the Second UK Embedded Forum, Noviembre 2005.

[11] FEILER, P, LEWIS B., y VESTAL S., Improving Predictability in Embedded Read-Time Systems, en la Conferencia Life Cycle Software Engineering, Redstone Arsenal, AL. Disponible como el reporte técnico 00-SR-011 del SEI

[12] FIERZ, Hugo, The CIP Method: Component and Model-Based Construction of Embedded Systems, en Foundations of Software Engineering Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering, ISSN:0163-5948, 1999.

[13] HARDUNG B., KÖLZOW T., KRÜGER A.: Reuse of Software in Distributed Embedded Automotive Systems. En la Cuarta Conferencia Internacional ACM en Software Embebido (EMSOFT 2004), PISA, Italia, Septiembre 27-29, 2004.

[14] HUGHES Z.M, PONT M.J., Design and test of a task guardian for use in TTCS embedded systems en: Proceedings of the 1st UK Embedded Forum, pp. 16-25, University of Newcastle upon Tyne, Birmingham, UK, 2004.

[15] JACKSON, Michael A, A System Development Method, en Tools and notions for program construction: An advanced course; Nice 1981; pages 1-25; Cambridge University Press, 1982.

[16] KAMAKURA, T.; SHIMODA, T.; DOHI, Y.; MURAKOSHI, H.; Implementation of a large Petri net by a group of Petri net controller en Conferencia Internacional en Electrónica Industrial, Control e Instrumentación, 1997. IECON 97.

23vo, Volumen 3, 9-14 Nov. 1997 Página(s):1210 - 1215 vol.3

[17] KRÜGER, A., WAGNER, G.,EHMKE, N.,PROKOP, S. "Wirtschaftliche Betrachtungen und mögliche Geschäftsmodelle für Standard-Software / Economic Considerations and Business Models for Automotive Standard Software Components" En: Electronic Systems for Vehicles", Baden-Baden, Germany, VDI Berichte 1789, Septiembre 2003.

[18] KURIAN S., PONT M.J., Building reliable embedded systems using Abstract Patterns, Patterns, and Pattern Implementation Examples, en Proceedings of the Second UK Embedded Forum, Noviembre 2005.

[19] MAAITA A., PONT M.J, Using "planned pre-emption" to reduce levels of task jitter in a time-triggered hybrid scheduler, en Proceedings of the Second UK Embedded Forum, Noviembre 2005.

[20] MARCOS María, Manual para la elaboración de tesis: Tesis 1: MATI-PGIT, Editorial Trillas SA de CV, Segunda edición, México, 1998

[21] MOUAAZ, Nahas, y PONT, Michael, Using XOR operations to reduce variations in the transmission time of CAN messages: A pilot study, en Proceedings of the Second UK Embedded Forum, Noviembre 2005.

[22] MWELWA C., PONT M.J., WARD D; Developing reliable embedded systems using a pattern-based code generation tool: A case study, University of Leicester, MIRA Ltd en Proceedings of the Second UK Embedded Forum, Noviembre 2005.

[23] MWELWA C., PONT, M.J. and WARD, D.; Code generation supported by a pattern-based design methodology, en: Proceedings of the 1st UK Embedded Forum, pp. 36-55, University of Newcastle upon Tyne, Birmingham, UK, 2004.

[24] NOLTE, T.; HANSSON, H.; NORSTRÖM, C. y PUNNEKKAT, S. "Using Bit-

stuffing Distributions in CAN Analysis", en IEEE/IEE Real-Time Embedded Systems Workshop (Satellite of the IEEE Real-Time Systems Symposium) Londres 2001.

[25] PONT, M. J., Designing and Implementing Reliable Embedded Systems Using Patterns, en EUROPLOP 99, 1999.

[26] PONT, M.J., Patterns for Time-Triggered Embedded Systems, Addison Wesley, primera edición, 2001.

[27] SCHULTZ, Ulrik, BURGAARD, Kim, y CHRISTENSEN, Gram, Compiling Java for Low-End Embedded Systems, en ACM SIGPLAN Notices, Volumen 38 , Issue 7 (Julio 2003), Special Issue: Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool support for embedded systems (San Diego, CA).

SESSION: Efficient Java systems, Páginas: 42 – 50, ISSN:0362-1340, 2003

[28] SOMMERVILLE, Ian, Software Engineering, Addison Wesley, Séptima edición, EUA, 2004

[29] SUN, Microsystems, Build a Compute Gris with Jini™ Technology, en www.jini.org/whitepapers/JINI_ComputeGrid_WP_FINAL.pdf, Diciembre 2004, tomado el 22 de junio del 2006

[30] VIDLER P. J., PONT M.J.; Automatic conversion from 'single processor' to 'multi-processor' software architectures for embedded control systems en Proceedings of the Second UK Embedded Forum, Noviembre 2005.

[31] WOLF, Alexander L., PERRY, D. Foundations for the Study of Software Architecture, en ACM Sigsoft, Software Engineering Notes. Vol 17 no 4, 1992

[32] GARLAN, D., SHAW M., An Introduction to Software Architecture, en Advances in Software Engineering, Vol. 1, World Scientific Publishing Company,

1993.

[33] CAMPBELL D., Designing for Electromagnetic Compatibility with Single-Chip Microcontroller, en la nota de aplicación AN1263 de Motorota, 1995.

[34] GLENEWINKEL M., System Design and Layout Techniques for Noise Reduction in MCU-Based Systems, en AN1259 de Motorota, 1995.

[35] KOBEISSI I., Noise Reduction Techniques for Microcontroller-Based Systems, en AN1705, 1999.

[36] PHILIPS Semiconductors, Security Transponder and RISC Controller (STARC 2XLite), esta versión es confidencial, y es accesible únicamente por requisición a: sales.addresses@www.semiconductors.philips.com, Mayo 2006.

[37] FOWLER, Martin, UML Distilled, A Brief Guide to the Standard Object Modeling Language, Addison-Wesley Object Technology Series, Tercera Edición, 2004

[38] JACKSON, Michael, System Development, Prentice Hall, 1983

[39] KEIL, Application Note 129

[40] PHILIPS Semiconductors, 89C51/89C52/89C54/89C58 80C51 8-bit microcontroller family 4K/8K/16K/32K Flash, 2002

[41] ATMEL, 8-Bit Microcontroller with 4K Bytes Flash AT89C51, 2000

[41] ATMEL, Instruction Set 0509B-B–12/97, 1997

[42] ATMEL, Atmel 8051 Microcontrollers Hardware Manual, 2005

[43] LEA, Doug, Concurrent Programming in Java Second Edition Design Principles and Patterns, Addison Wesley, 1999

Apéndice A:
Cartas de Relación con Siemens

-----Original Message-----
From: Vazquez-Murillo, Jorge [mailto:jorge.vazquez-murillo@siemens.com]

De hecho es configurable, no recuerdo si habíamos hablado de esto, pero hablemos del tiempo de debounce en el primer button press.

Cuando se presiona por primera vez se escanea cada 2ms y si tienes 5 muestras con la misma lectura entonces decimos que el botón se presiono y asignamos un button code.

Durante la transmisión de los RC blocks, se realizan escaneos cada determinado tiempo (como lo especifica la figura) y en un escaneo basta con tres hits (tres lecturas con el mismo valor) para indicar un nuevo botón presionado.

Si tienes dudas me dices.

Saludos,

José Luís

From: Carlos Aceff [mailto:carlos.aceff@gmail.com]
Sent: Monday, September 04, 2006 2:55 PM
To: Herrera-Martinez, Jose
Subject: Tiempos de Debounce = 8ms?

Otra pregunta,

El periodo de muestreo de los puertos de entrada, (debounce) puede ser de 8ms?

Es decir: En el peor de los casos detectar por primera vez que un botón se presiono 8ms después del evento, y 8ms mas tarde (16ms en total), si el puerto sigue en estado activo, iniciar la creación del bloque a transmitir, e iniciar la transmisión?

Saludos

Carlos

-----Original Message-----

From: Vazquez-Murillo, Jorge [mailto:jorge.vazquez-murillo@siemens.com]

Sent: Tuesday, April 27, 2006 1:44 PM

To: Carlos Aceff

Subject: RE: Ofrecimiento de investigación

Carlos,

con gusto podemos participar contigo de la forma que esto te genere los menores gastos.

Te pongo en contacto con Alberto Romero que es jefe de grupo de desarrollo y quien tiene

amplia experiencia en embedded software. De parte de SiemensVDO, Alberto sería tu contacto principal para el desarrollo de este trabajo de investigación, por lo que te pido que lo contactes por mail directamente y si es necesario hablar, se pongan de acuerdo para que el pueda llamarte a la ciudad de México.

Saludos,

Saludos cordiales / Best regards / Viele Grüsse

Jorge Vázquez

Siemens VDO S.A. de C.V.

Development Center Manager

Guadalajara/Mexico

Phone: +52 33 3283-4001

Mobile: +52 33 3662-1970

E-Mail: jorge.vazquez-murillo@siemens.com

Internet: http://www.siemensvdo.com

From: Carlos Aceff [mailto:carlos.aceff@gmail.com]

Sent: Wednesday, April 26, 2006 12:42 PM

To: Vazquez-Murillo, Jorge

Cc: jose.molina@itesm.mx

Subject: Ofrecimiento de investigación

Hola Jorge,

Mi objetivo es encontrar problemas específicos que sean reales, y aplicables.

La dinámica de la investigación que yo propongo es sencilla. Les voy a pedir problemas computacionales con cierto formato y tipo, después de procesarlos, les enviare una estructura para que la podamos discutir juntos para determinar el impacto que la solución tendría en ustedes.

Como yo habito en la ciudad de México, tratare de utilizar los recursos tecnológicos como el Internet y el teléfono para completar la investigación desde la ciudad de México. Sin embargo, al final de la investigación, me gustaría visitarlos para determinar el impacto de la investigación.

La investigación aun tiene algunos detalles que pueden cambiar, (pues depende de mi asesor de tesis el camino que tome), sin embargo, la idea general ya esta escrita.

Los siguientes pasos, aunque también pueden cambiar, serian: firmar un acuerdo de confidencialidad sobre la información que me provean, si es que están interesados.

Ponernos de acuerdo en el formato de problemas computacionales, y comenzar la investigación.

Para cualquier duda, por favor escriban a este correo, o si quieren hablar por teléfono, lo pueden hacer al (55)55944889

El nombre de mi asesor de tesis es Doctor Martín Molina jose.molina@itesm.mx

Si quieren compensarme por el producto de la investigación, me podrían ayudar con los gastos de movilización a Guadalajara, cuando sea necesario, y con las llamadas a la ciudad de México cuando discutamos las estructuras derivadas de la investigación, que tal vez sean

pequeños gastos para la empresa, pero me evitarían grandes gastos para mi.

Saludos

Carlos

-----Original Message-----

From: Vazquez-Murillo, Jorge [mailto:jorge.vazquez-murillo@siemens.com]

Sent: Tuesday, April 25, 2006 1:44 PM

To: Carlos Aceff

Subject: RE: Ofrecimiento de investigacion

Hola Carlos,

Claro que te recuerdo. Estamos interesados en ser receptores de esta investigación, por lo que me interesa saber que implicaría y cuales serían los siguientes pasos.

Saludos,

Jorge

-----Original Message-----

From: Carlos Aceff [mailto:carlos.aceff@gmail.com]

Sent: Monday, April 24, 2006 11:00 AM

To: Vazquez-Murillo, Jorge

Subject: FW: Ofrecimiento de investigacion

Hola Jorge,

Tal vez me recuerdes, me llamo Carlos Aceff, y fui a entrevista a Siemens hace ya casi un año para el trabajo de ingeniero en software.

Actualmente me encuentro haciendo mi tesis para graduarme de la maestría en Ciencias computacionales en el ITESM CCM. Como tal, estoy ofreciendo un trabajo de investigación sobre patrones de diseño en sistemas auto-contenidos, y el receptor del trabajo de investigación podrá utilizar los conocimientos resultantes en sus procesos habituales. Busco identificar, clasificar y documentar patrones como los que ha descrito Michael Pont en su trabajo Reliable Embedded Systems. (paper adjunto)

Si están interesados en ser receptores de esta investigación, por favor responder esta invitación lo más rápido posible.

Adjunto un documento con los datos generales del proyecto.

Saludos

Carlos Aceff

Apéndice B
Requerimientos Originales

Esta sección está compuesta por los siguientes tres documentos:

1. Una sección de requerimientos funcionales.
2. Una sección que especifica las características básicas de los paquetes a transmitir
3. Una sección sobre las calibraciones requeridas para quitar los rebotes de los interruptores

Estas secciones están puestas tal como me fueron proporcionadas, por lo que están en inglés y por razones de confidencialidad, no están completas.

Requerimientos Funcionales

KeyFOB Programming Message (Allowed only with non-SKIM Enabled keyFOBs)

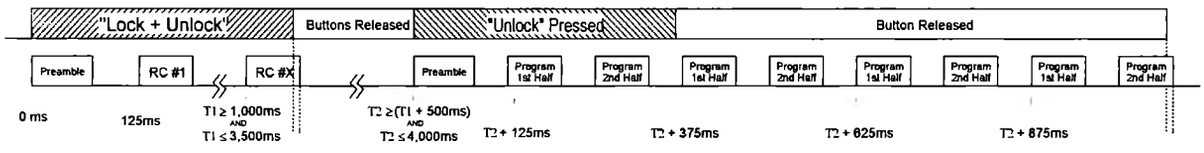


FIGURE 14: PROGRAMMING MESSAGE

KeyFOB Remote Start Message (Allowed only with Remote Start Enabled keyFOBs) ***

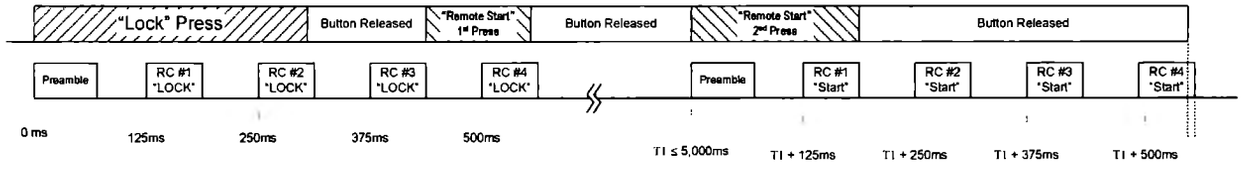


FIGURE 15: REMOTE START BUTTON SEQUENCE

G. The program mode is entered by a sequence of button activations (pushes). This sequence must be completed within 4 seconds of the start. The sequence and 4 second timer are started

when both unlock and lock buttons are active simultaneously (a staggered button activation causes the 4 second timer to start upon the activation of the second button). The first step of the sequence is completed by having both unlock and lock buttons active (pressed) for a minimum of 1 second (a staggered button release does not influence the programming sequence). The second step is to have at least 0.5 seconds of no buttons active (pressed). The third step (which must take place within 4 seconds of the start condition, i.e. both unlock and lock buttons active) is the reactivation of the unlock button. Upon the third step being met, the FOB will transmit the program sequence. NOTE: the programming mode shall only be accessible with non skim fobs. (Non SKIM FOBs are either an RKE only FOB or an unprogrammed SKIM FOB)

H. The programming message will consist of two data halves spaced 125 mS (+/- 12%) apart. The programming message will conform to the standard message group (see item #3) with the exception that both the first half and second half data packets represent one normal data packet (i.e. there will actually be eight data packets in between preambles). The programming message loop will be active as long as the single button activation after the unlock/lock combination stays true (up to the 30 second button hold timer expires).

I. Remote Engine Off is activated by a sequence of button activations (pushes). This sequence must be completed within 5 seconds of the start. The sequence and 5 second Remote Start timer are started when the remote start button is activated and the Remote start timer is not running. The first step of the sequence is completed by having a remote start button activated with the remote start timer not running. The FOB will transmit the Right Slider command message. The second step is to have no buttons active (pressed). The third step (which must take place within 5 seconds of the start condition, is the reactivation of the remote start button. Upon the third step being met, the FOB will transmit the Remote off message, the Remote Start timer is forced to be timed out.

J. Remote Start is activated by a sequence of button activations (pushes). This sequence must be completed within 5 seconds of the start. The sequence and 5 second timer are started when the lock button is activated and the remote start timer is not running. The first step of the sequence is completed by having a lock button activated with the remote start timer not running. The FOB will not transmit any command message. The second step is to have no buttons active (pressed). The third step is the activation of the remote start button. The fourth step is to have no buttons active (pressed). The fifth step within 5 seconds of the start condition, is the reactivation of the remote start button. Upon the fifth step being met, the FOB will transmit the Remote Start message, and the Remote Start timer is forced to be timed out.

Características de los Paquetes a Transmitir

2.0 GENERAL PROTCOL REQUIREMENTS ***

2.1 Timing and Envelope Requirements

Message timing and envelope requirements are specified in Table 1.

TABLE 1: GENERAL PROTOCOL REQUIREMENTS		
Parameter	Requirement	Units
TIMING		
Data Rate	2.4	Kbps
Timing Tolerance	+/- 10	%
MESSAGE STRUCTURE		
Packet Spacing (Start-to-Start)	125	Msec
Maximum Message Length	30.175	Sec
ENVELOPE		
Encoding	Manchester	-
Modulation Type	ASK (All Markets)	-

2.2 Uniqueness and Security Requirements

The device uniqueness and security requirements are specified in Table 2.

TABLE 2: UNIQUENESS AND SECURITY REQUIREMENTS		
Parameter	Requirement	Units
DEVICE ID (DID)		
Length	32	Bits
Encryption	None	-

TABLE 2: UNIQUENESS AND SECURITY REQUIREMENTS		
Parameter	Requirement	Units
Invalid ID's	All "1"'s	-
	All "0"'s	-
	All "0101 ... "	-
	All "1010 ... "	-
VARIABLE SEQUENCE CODE (AKA: ROLL COUNT)		
Length	32	Bits
Encryption	None	-
Value	Per Philips HITAG2	-
SEQUENCE RESPONSE		
Length	32	Bits
Encryption	Required	-
Value	Per Philips HITAG2	-

Message Structure ***

See Section 7.0 and 9.0 for details of RF messages based on user actuation of the FOB buttons.

Packet Structure

The RKE message consists of several packets, defined below.

4.1 Preamble Packet Defined

Time (msec)	0 47
Length (bits)	112
Position	111 0
Section	Preamble
Value	'11....1'

FIGURE 1: PREAMBLE PACKET

4.2 Remote Control (RC) Packet Defined

4.2.1 Normal RC Packet ***

Time (msec)	0 47									
Length (bits)	15	1	3	32	8	10	32	1	8	2
Position	111 97	96	95 93	92 61	60 53	52 43	42 11	10	9 2	1 0
Section	Run-In	Start Bit	SOF	DID	CID	SI	SR	BAT	PAR	EOD

Value	'11....1'	'0'	'001'	ID	Table 4, 5	SI	SR	Table 7	Section 8.1	'10'

FIGURE 2: NORMAL RC PACKET

4.2.2 Programming RC Packets ***

When Customer RKE Programming is invoked, two programming specific packets are sent to the receiver. Both packets must be received to transfer the all keyFOB programming information.

Time (msec)	0 40							
Length (bits)	15	1	3	32	32	3	8	2
Position	95 81	80	79 77	76 45	44 13	12 10	9 2	1 0
Section	Run-In	Start Bit	SOF	DID	SI	Filler	PAR	EOD
Value	'11....1'	'0'	'010'	ID	SI	'000'	Section 8.2	'10'

FIGURE 3: PROGRAMMING PACKET 1

Time (msec)	0 40							
Length (bits)	15	1	3	16	48	3	8	2
Position	95 81	80	79 77	76 61	60 13	12 10	9 2	1 0
Section	Run-In	Start Bit	SOF	DID	Secret Key	Filler	PAR	EOD
Value	'11....1'	'0'	'100'	ID	SK	'000'	Section 8.3	"10'

FIGURE 4: PROGRAMMING PACKET 2

Calibraciones Requeridas Durante el "*Debounce*"

Calibratable button debounce values for all buttons

As the buttons are very sensitive, it is necessary that some have a longer debounce time than others to avoid inadvertent actuations. A byte should be implemented per **FUNCTION** in EEPROM. This byte can vary between 0x00 and 0xFF with each count representing 4ms for a maximum debounce time of 1020ms.
As in the current CID, the default value in EEPROM should be set to 0x03 to create the 12ms minimum debounce per function.

Example – **used as default setting:**

Function	**EEPROM location**	**Contents**	**Resultant debounce time**
Lock	Page x	0x03	12ms
Unlock	Page x + 1	0x03	12ms
Trunk	Page x + 2	0x4B	300ms

Panic	Page x + 3	0x4B	300ms
Remote	Page x + 4	0x03	12ms
Liftgate	Page x + 5	0x4B	300ms

Country code configuration

The configuration for the country code should be as follows in page 4 byte 0 of the EEPROM:

EEPROM Page 4 – Byte 0								
Country	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
North America	0	0	0	0	0	0	0	0
Europe	0	0	0	0	0	0	1	1
Japan	0	0	0	0	0	0	1	0

Bit 0: used for frequency selection as follows – North America and Japan 315Mhz ; Europe 433Mhz
Bit 1: used for high power remote start – North America only

Apéndice C
Metodología de la Investigación

El presente trabajo se realizará en la empresa Siemens VDO, ubicada en la ciudad de Guadalajara Jalisco. Ésta es una empresa dedicada a la producción y desarrollo de sistemas auto-contenidos automotrices, donde se realizarán visitas durante el desarrollo de la tesis.

Metodología

Debido a que la investigación pretende inducir de la información existente sobre patrones de sistemas auto-contenidos, una solución que promueva la reutilización, se utilizará un método cualitativo de investigación, cuyo principal tema de estudio, será la información y documentación existente sobre sistemas auto-contenidos.

Además, se utilizará un método de análisis de problema, (mundo real), análogo al propuesto por Jackson [15], para iniciar la búsqueda de posibles alternativas.

La figura 25, que adelante se explica, muestra de manera abstracta los pasos a seguir para completar esta investigación:

La Definición de la Aplicación del Sistema Auto-Contenido: Consiste en acotar el área de los sistemas embebidos al sector automotriz.

Definición del Problema Recurrente a Estudiar: Consiste en definir cual será el problema real y recurrente que se quiere solucionar. Para este caso se seleccionó el problema explicado en 1.3.

Definición de Documentos a Investigar: En este caso se han investigado los patrones para sistemas auto-contenidos (soluciones documentadas a problemas recurrentes), y el método de diseño de sistemas Jackson [15], para analizar el problema.

Investigación: Consistirá en utilizar el método Jackson (*J.S.D.*), junto con la lista de patrones

investigados para crear un nuevo conjunto de patrones que resuelvan el problema descrito en 1.3 más atrás.

Proceso de Mejoramiento de las Estructuras: Consiste en modificar las estructuras para promover la capacidad de re-uso. Esto en caso que la capacidad de re uso no se haya alcanzado en el paso anterior.

Análisis de Datos y Creación de un Conjunto de Estructuras Útiles: Una vez obtenido el patrón de diseño, se le extraerá un conjunto de estructuras arquitectónicas, que sean útiles para poder comunicar la información del diseño, a una amplia gama de personas con diferentes niveles de conocimientos técnicos.

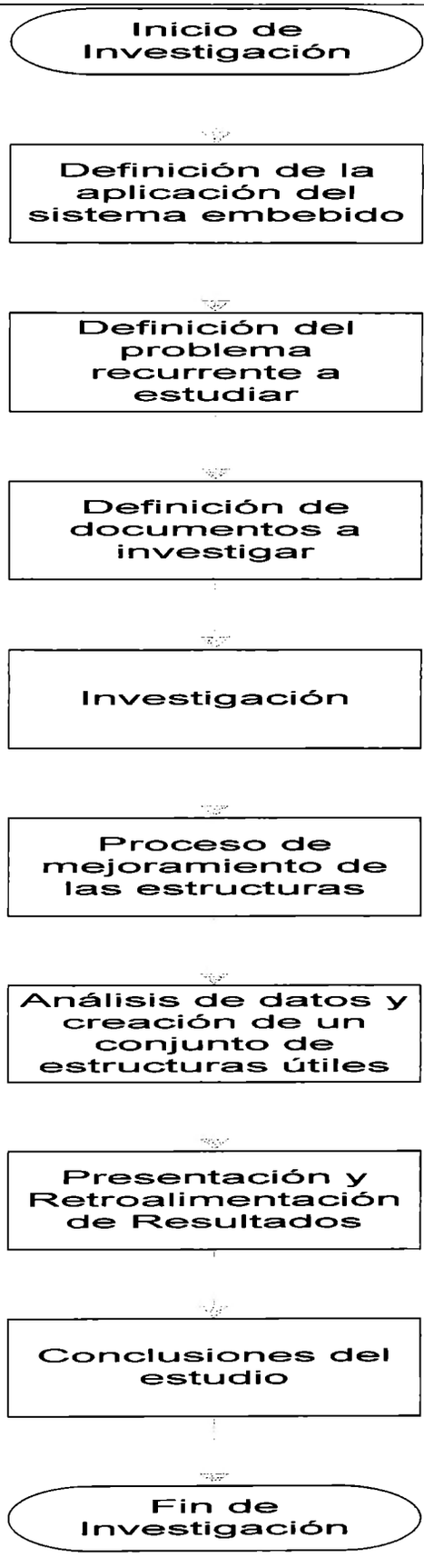
Presentación y Retroalimentación de Resultados: Consistirá en una visita a Siemens VDO, para presentar los patrones obtenidos en esta investigación.

Conclusiones del Estudio: Serán mis impresiones sobre el éxito del trabajo.

Presupuesto

El principal beneficiado del trabajo de tesis será Siemens VDO, y, esta ha acordado ayudarme con los gastos de la investigación, aun cuando no ha especificado montos ni rubros a cubrir.

Por mi parte cuento con un reducido presupuesto, y me encuentro limitado a utilizar un simulador para microcontroladores para realizar mis pruebas y validación de software.



53. Metodología de investigación