INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS ESTADO DE MÉXICO

SUBSISTEMA DE COMUNICACIÓN PARA UNA CELDA FLEXIBLE DE MANUFACTURA

TESIS QUE PARA OPTAR POR EL GRADO DE MAESTRO EN CIENCIAS COMPUTACIONALES PRESENTA

MIGUEL ANGEL URIBE LEYVA

Asesores:	Dr. Jesús Sánchez Velázquez	
	Dr. Fabian García Nocetti	
Comité de Tesis:	Dr. Eduardo de Jesús García García	
	Dr. José de Jesús Vázquez Gómez	
Jurado:	Dr. Eduardo de Jesús García García	Presidente
	Dr José de Jesús Vázquez Gómez	Secretario
	Dr Jesús Sánchez Velázquez	Vocal
	Dr Fabian García Nocetti	Vocal

Atizapán de Zaragoza, Edo. Méx., Enero 1999

Contenido

Indice de Figuras

Indice de Tablas

1 INTRODUCCION

El mundo actual no se podría concebir sin la existencia de las máquinas que se utilizan para la generación de bienes y servicios tan necesarios para la vida cotidiana. El éxito de las industrias de manufactura depende en muy buena medida en la eficiencia de estas máquinas y del control que se tiene sobre ellas y automatizar su control y su operación es la tendencia mundial actual en esta rama de la producción. La automatización requiere que estas máquinas y robots sean interconectados y que se establezca un lenguaje de comunicación entre ellos. Este arreglo para la intercomunicación requiere de interfaces y establecimientos de reglas que se tienen que implementar en todas las máquinas a conectar, lo que ha generado que se diseñen e implementen un sin número de redes propietarias que inherentemente implican una gran inversión al agregar cada miembro a la red. Esto se soluciona estableciendo acuerdos entre productores de dispositivos inteligentes y consumidores de éstos, orientados siempre a mejorar el desempeño de las unidades de manufactura. Uno de estos esfuerzos lo constituye la especificación de mensajes de manufactura MMS, (Manufacturing Message Specification).

En el proyecto de celda de manufactura del ITESM-CEM, se desarrolla un controlador de celda de manufactura basado en procesadores operando en paralelo. Este controlador tiene entre otros objetivos, el de aprovechar todas las características que el paralelismo brinda para poder establecer una celda flexible, reconfigurable y tolerante a fallas. Esta red tiene que contar con un protocolo de comunicación que permita su expansión natural a bajo costo por lo que debe ser compatible con los estándares de manufactura existentes.

El objetivo principal del presente trabajo es proponer un protocolo validado para el intercambio de mensajes de manufactura en un controlador de celda. Dicho protocolo está basado en la especificación de mensajes de manufactura MMS, por lo mantiene a la celda dentro de los estándares internacionales de manufactura. Además, en este trabajo se propone un esquema del subsistema general de comunicaciones de esta celda.

En el Capítulo 2 de esta tesis se destacan los antecedentes que originan este trabajo. Se aborda el tema de las celdas de manufactura en lo general y de la del ITESM-CEM en particular, describiendo sus componentes y su sistema de control actual. Se menciona el proyecto Conacyt-NSF, en el cual se suscribe la tesis, se presenta el nuevo controlador para la celda que en éste se propone, los trabajos ya terminados y los que están actualmente en desarrollo y, finalmente, se destaca la importancia que tiene la implementación de un buen protocolo para este nuevo controlador.

El estado del Arte del entorno del presente trabajo se describe en el Capítulo 3. Se deja en claro la importancia de los estándares y de los sistemas abiertos mencionando, en especial, los relacionados a la rama de la manufactura: el protocolo de la automatización de manufactura MAP (Manufacturing Automation Protocol) y MMS. Así mismo se presenta el concepto de los protocolos y su papel dentro de los estándares, remarcando la importancia que tiene hacer una buena validación de ellos antes de implementarlos. Acerca de la validación, se introducen los métodos y herramientas actuales para llevar a cabo esta tarea. Por último se muestra la arquitectura del nuevo controlador para celda flexible de manufactura, destacando las características de la tarjeta madre utilizada, de los procesadores, del lenguaje de programación, de la interfaz de este controlador con los robots y de la topología de red que se usa.

El Capítulo 4 tiene como objetivo describir en forma general el subsistema de comunicación para la celda y, en detalle, el diseño del protocolo propuesto en este trabajo. Además, se muestran los protocolos desarrollados en otros trabajos y que forma parte integral subsistema propuesto.

Las bases establecidas en el Capítulo 4 para el protocolo propuesto se utilizan en el Capítulo 5 para describir como fue hecha la validación del mismo. Se detallan en el modelado de validación, todos los procesos participantes y la secuencia de interacción entre ellos, puntualizando la manera en que fue modelado cada requerimiento de corrección utilizando la herramienta Xspin. Este capítulo concluye con una sección de los resultados obtenidos.

El Capítulo 6 se dedica a la implementación del protocolo. Se dan a conocer los detalles de cómo fue implementado el modelo de validación del Capítulo 5 en la plataforma de los transputers. Así mismo, se describen los procesos y programas generados para la implementación.

Finalmente, en el Capítulo 7, se enumeran las conclusiones del presente trabajo y los trabajos a futuro que se pueden realizar para complementar el proyecto general.

2 ANTECEDENTES

En el presente capítulo se introducen los conceptos de la manufactura integrada por computadora, las celdas de manufactura y sus controladores. En particular, se describe la celda existente en el ITESM-CEM y se hace referencia al Proyecto de Investigación Conacyt-NSF, en el cual se subscribe el presente trabajo, y a los trabajos desarrollados y en desarrollo dentro de éste. Al final del capítulo se justifica la importancia de la investigación desarrollada en esta tesis.

2.1 MANUFACTURA INTEGRADA POR COMPUTADORA

La manufactura actualmente se enfrenta a cambios muy sustanciales debido a dos grandes factores, el crecimiento de los mercados altamente competitivos y una rápida evolución de las tecnologías. Esta situación requiere ser analizada detalladamente para la toma de decisiones de las empresas de manufactura en las áreas de costo, producción y planeación de productos. El concepto de la Manufactura Integrada por Computadora, CIM (Computer Integrated Manufacturing), nace para mejorar esta situación permitiendo el control y el análisis de los negocios actuales y de la información tecnológica, diseño más rápido y ciclos de desarrollo para productos más sofisticados y sistemas de manufactura flexible [11].

El objetivo principal de la CIM es utilizar las computadoras como herramienta básica para el diseño, la planeación, el control y el despacho de las distintas etapas de los procesos industriales [8], ayudando a reducir tiempos e inventarios, incrementando la flexibilidad en la manufactura y la calidad del producto. Además de la integración de equipos de cómputo, la CIM implica la integración de gente, organizaciones y procedimientos, abarcando las etapas de diseño, producción, mercadeo, entrega y soporte de producto [9]. Aunque este concepto es relativamente reciente, desde hace ya varios años, las computadoras se han ido introduciendo en los ambientes de manufactura a través de aplicaciones independientes como:

- Diseño Asistido por Computadora, CAD (Computer Aided Design).
- Planeación de Procesos Asistido por Computadora, CAPP (Computer Aided Process Planning).

- Planeación de requerimientos de Material, MRP (Material Requirement Planning).
- Máquinas de control numérico y Robots.

Sin embargo, los desarrollos basados solo en alguna de estas aplicaciones tienden a quedarse aislados por la incapacidad de interconectarse con otros sistemas. La filosofía CIM trata de dar una idea globalizada de los distintos niveles que se presentan en el ambiente de la manufactura para poder homogeneizar las aplicaciones. Existen varios modelos, llamados de información jerárquica, para este propósito.

Los modelos de información jerárquica de la CIM muestran los distintos niveles de control encontrados en los sistemas CIM, que van desde los controles de tiempo real, incluyendo sensores y actuadores, hasta los sistemas de control de aplicación ejecutiva. En estos modelos la información fluye en dos direcciones. La información de monitoreo, de estado de la producción, indicadores de excepción y otros datos operativos son pasados hacia el nivel más alto y en cada segmento se procesan algunos datos para su uso. En sentido contrario, se transfieren comandos, puntos de prueba y calendario de operación. Algunos de estos modelos jerárquicos son:

- Sistemas de Administración de Fabricación Avanzada, AFMS (Advanced Factory Management Systems), desarrollado por Computer Aided Manufacturing Inc.
- Facilidad de Investigación de Manufactura Avanzada, AMRF (Advanced Manufacturing Research Facility), de National Bureau of Standards.
- Modelos de Comunicación en Fábricas, FCM (Factory Communications Models), de la Sppan-Borowka.

En el modelo AMRF, las funciones de control y planeación han sido divididas en una serie de niveles que son: Facilidad, Tienda (Shop), Celda, Estación de Trabajo y Equipo [17].

El nivel de Facilidad incluye planeación de procesos, administración de producción, programación a largo plazo y administración de la información con ligas a funciones administrativas y financieras. Debajo de este nivel se encuentra el nivel Tienda, que administra la coordinación de recursos y trabajos a nivel piso.

Debajo del nivel Tienda se encuentra el nivel Celda el cual contiene los sistemas de control de la celda para programar y controlar los trabajos. En este punto, los trabajos se dividen en grupos y se alojan en celdas de acuerdo a sus semejanzas. Se incluyen funciones adicionales para la programación de manejo de material y herramientas en la celda.

El nivel Estación de trabajo coordina las actividades de una estación de trabajo. Desde el punto de vista de AMRF, una estación de trabajo consiste de un robot, una máquina herramienta, un almacenador de material y una computadora de control. Por lo que, la estación de trabajo arregla la secuencia de operaciones necesarias para que los trabajos alojados en una celda en particular sean realizados. El nivel más bajo de esta jerarquía de control y planeación es el nivel Equipo, el cual consiste en controladores para recursos individuales como: máquinas herramientas, robots o sistemas de manejo de materiales

En este modelo, el nivel celda es el que tiene una mayor relevancia en el diseño de la CIM pues es a este nivel es donde se tienen que comunicar los equipos que generalmente son de distintos proveedores.

2.2 CELDAS DE MANUFACTURA

Una celda de manufactura es la unidad básica en una planta automatizada desde el punto de vista del control de procesos.

2.2.1 LA CELDA DE MANUFACTURA Y SU CONTROLADOR

Actualmente no todos los dispositivos de la CIM se comunican con un mismo protocolo. Los controladores de celda son usados para integrar dispositivos de manufactura de proveedores distintos y para coordinar las operaciones en una misma celda y entre celdas a un mismo nivel. Las tareas de un controlador de celda son las siguientes.

- **Dispositivo de Adquisición de Datos**.- El controlador de celda es el siguiente nivel después de los dispositivos de piso (Equipo). Es responsable de la recolección de datos de los controladores programables, PLCs (Programmable Controllers), robots y dispositivos de adquisición de datos en general. Algunas veces, en el caso de dispositivos más avanzados como los PLCs, los controladores pueden ayudarse de los dispositivos de piso para obtener resúmenes o solo resultados. Además, los controladores de celda son un medio entre el piso de fábrica y el operador para la inserción de datos manuales. La colección de datos es generalmente hecha en dos bases:
  - Por frecuencia, donde los puntos son monitoreados a una velocidad de rastreo preestablecida.
  - Por evento, donde los puntos son monitoreados cuando cierto evento sucede. La señal puede venir de un dispositivo terminal o de cualquier otro sistema en la red. La entrada de datos manuales se considera dentro esta categoría.
- **Administración de Datos**.- Los datos recolectados de los dispositivos terminales son almacenados en áreas de memoria de tiempo real, y procesados por una serie de procedimientos establecidos por el usuario. Se pueden realizar diferentes funciones usando esos procedimientos, tales como manejo de alarmas, escalamiento, temporizadores, chequeo de rangos, reducción de datos y chequeo de estado, entre otros. Los datos de tiempo real son almacenados en áreas de acceso disponibles para la mayoría de los módulos. En aplicaciones flexibles cualquier programa de usuario debe ser capaz de escribir o leer datos en o desde estas áreas.
- **Acumulación de Datos**.- El propósito del controlador de celda es ayudar a un mejor control de los procesos de manufactura y esto se alcanza con una análisis histórico de datos. No todos los datos colectados deben registrarse; las condiciones, la frecuencia y el tipo de datos que deben almacenarse generalmente se dejan al usuario. Generalmente se usan bases de datos relacionales porque es el mejor modelo adaptado a reconfiguraciones dinámicas del sistema.

En un sistema de controlador de celda fuertemente cargado, la base histórica puede residir en una computadora separada.

- **Reportes de información y Análisis**.- Con el controlador de celda se alcanza un mejor control del piso de manufactura por la información que brinda a supervisores y operadores. Por ello, y para alcanzar una mejor utilidad, la información debe ser temporalizada, precisa y en la cantidad de detalle necesario. También debe ser presentada en la mejor forma posible para que el operador pueda tomar decisiones más fácilmente.
- **Comunicación de Datos**.- El controlador debe ser capaz de comunicarse con otros nodos del mismo nivel y nivel mayor. Al mismo nivel el controlador debe poder accesar valores o puntos de otras celdas, lo que brinda la oportunidad de tomar decisiones basadas en las actividades de éstas, que es el primer paso para el genuino control integrado. Al mismo tiempo el controlador debe poder enviar información y responder a señales de sistemas de mayor nivel.
- **Otras funciones**.- Los controladores deben ser construidos para coexistir con aplicaciones de usuario. Debido a que están directamente arriba de dispositivos de piso, son usados para controlar información desde y para tales dispositivos. Esto incluye datos de especificaciones de trabajos que son usados para construir una unidad del producto. Los controladores de celda también se usan para desarrollar programas de PLCs que son cargados en tiempo real.

Hay módulos globales que deben incluirse en un controlador de celda; por ejemplo, un configurador para las funciones, un sistema monitor que facilite el control de los módulos de la celda, sistemas de respaldo y recuperación y sistemas de seguridad, entre otros.

2.2.2 LA CELDA ITESM-CEM

El ITESM Campus Estado de México cuenta con un prototipo de una celda de manufactura [2]. Los elementos de esta celda (ver Figura 1) se describen a continuación.

- **Robot AS/RS.-** Es un robot de fabricación norteamericana que se encarga de mover la materia prima y/o producto terminado desde y hacia los anaqueles provistos para este fin. Todos sus movimientos son servocontrolados y pueden ser descritos en el plano (x,y); donde "x" es distancia horizontal entre compartimentos, y "y" es la distancia vertical entre ellos. Otros ejes que usa son el "z" y el "c"; el primero indica movimientos lineales para desplazarse hacia el interior y exterior del compartimento, mientras que el segundo es indica un movimiento de rotación para poner o quitar paletas del compartimento.
- **Bandas transportadoras.-** Son servomecanismos que sirven para llevar los componentes del producto de un punto a otro dentro de la celda; cuentan con unas bases para colocar la materia prima llamadas paletas. A lo largo de cada banda hay sensores de presencia que sirven al control central para evaluar la posición de cada paletas montada en la banda.
- **Robot Mitsubishi.-** Es un robot de fabricación japonesa de tipo esférico. Cuenta con cinco ejes servo-controlados y un actuador eléctrico.
- **Robot IBM 7575.-** Es un robot de fabricación norteamericana con 4 grados de libertad servocontrolados, un actuador final eléctrico y cuatro puntos con diferentes funciones.

- **Robot Cincinnati Milacron 7374.-** Es robot de fabricación norteamericana con un sistema de comunicaciones soportado por una tarjeta de cuatro canales seriales.
- **Robot Puma serie 500.-** Este robot es de origen sueco y consta de 5 ejes o grados de libertad controlados por servomotores. Su actuador es una tenaza de accionamiento neumático. Este robot es de tipo esférico y su función principal es interactuar con el torno y la fresadora.
- **Robot Júpiter Amatrol.-** Es un robot norteamericano con cuatro grados de libertad servo-controlados; cada eje cuenta con sensores inductivos para detectar metales ferrosos y no ferrosos hasta una distancia de 50 mm.
- **Torno y Fresadora.-** Son máquinas herramientas controladas por una máquina de control numérico.

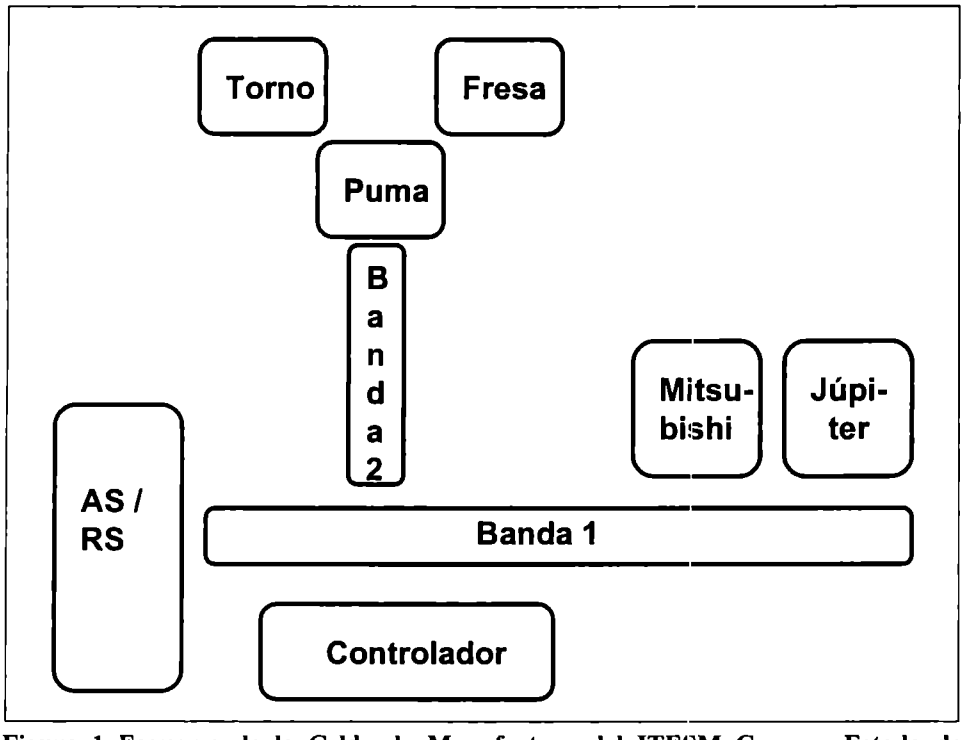


Figura 1 Esquema de la Celda de Manufactura del ITESM Campus Estado de México.

Todos estos elementos de la celda son controlados actualmente por una misma computadora que coordina los trabajos dando instrucciones directas y montando programas en los diversos robots. Este sistema de control tiene como desventaja principal que presenta una reducida integración del sistema. Evolucionar el control de esta celda para la optimización de sus componentes es un objetivo del proyecto de investigación llamado "Tecnologías de Comunicación Avanzadas en Robótica y Celdas Flexibles" [1], en el cual se suscribe la presente tesis y se aborda en la siguiente sección.

2.3 EL PROYECTO DE CONACYT – NSF

El proyecto mencionado es financiado por el Consejo Nacional de Ciencia y Tecnología (CONACyT) México, y la National Science Foundation (NSF) E.U.A. Será realizado en México por el ITESM-CEM y en E.U.A por la Universidad de Texas en Austin (UTA). En una etapa primaria de este proyecto se evaluó el tipo de procesadores que deberían controlar la celda, tomando en consideración los siguientes requerimientos:

- **Tiempo Real.-** El sistema de control debe proporcionar una capacidad de respuesta en tiempo real para la optimización de la producción de la celda.
- **Tolerancia a Fallas.** El sistema debe contar con mecanismos de recuperación contra fallas de hardware y/o software del controlador.
- **Reconfigurabilidad.** El sistema de ser capaz de cambiar su configuración para manufacturar nuevos productos o tolerar fallas.

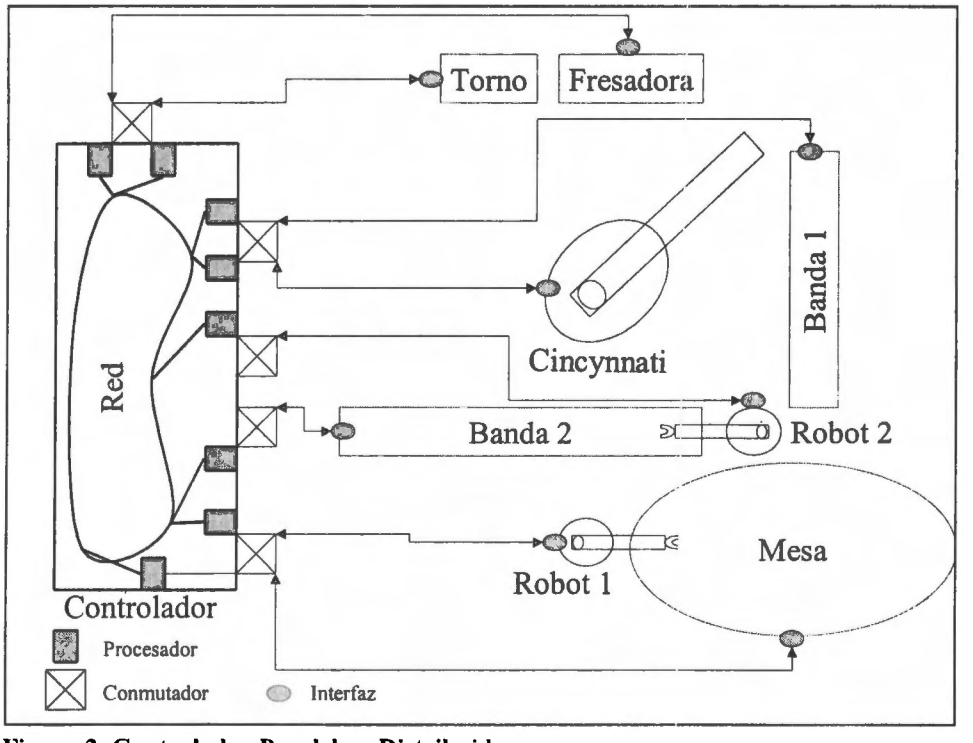


Figura 2. Controlador Paralelo y Distribuido.

En esa fase del proyecto se está diseñando y desarrollando un controlador para celda llamado PARDICO (PARallel and Distribuited Intelligent Controller) el cual es mostrado en la Figura 2. Este controlador [12] está basado en una red de procesadores llamados transputers (sección 3.3) los cuales tienen características óptimas para cubrir los requerimientos de la celda [13] y cuenta con interfaces conmutables de estado sólido [14] que permiten que cada robot esté conectado a la red a través de más de un punto de acceso.

Las tareas de este proyecto están siendo realizadas por el ITESM-CEM y por la Universidad de Texas en Austin de acuerdo a los objetivos que tiene cada institución dentro del referente proyecto de investigación.

2.3.1 OBJETIVOS ITESM

Dentro de este proyecto el ITESM se encarga del estudio las comunicaciones entre los componentes de la celda de manufactura, específicamente se concentra en el estudio de:

- Arquitecturas del controlador de celda.
- Tecnologías de comunicación en redes configurables.
- Interfaces con robots.
- Software de comunicaciones.
- Detección e identificación de fallas en la celda.
- Simulación visual.
- Planificación flexible.

2.3.2 OBJETIVOS UTA

Por su parte la NSF se encarga del problema de las comunicaciones dentro del robot inteligente, centrando su atención en:

- Arquitectura del controlador del robot.
- Tecnologías de comunicación en buses.
- Interfaces con actuadores.
- Controladores de módulos.
- Detección e identificación de fallas en el robot.

2.3.3 NUEVO CONTROLADOR DE LA CELDA ITESM –CEM

Los transputers (sección 3.3) son procesadores que soportan el procesamiento en paralelo y que pueden ser usados individualmente o se pueden conectar varios a través de sus enlaces (puertos) para formar redes [16]. Los enlaces del transputer son conexiones seriales de comunicación que constan de 2 hilos para transmitir información punto-a-punto, uno en cada sentido, a una velocidad de hasta 20Mb/s [11]. Existen instrucciones nativas de los transputers para enviar y recibir mensajes a través de sus enlaces, las cuales minimizan retardos en la comunicación inter-transputer [17]. La familia elegida de transputers para este proyecto es la T800 y la tarjeta madre es una tarjeta IMS B008 [13].

La tarjeta IMS B008 (ver Figura 3) es una tarjeta madre de módulos de transputers, TRAMs (TRAnsputer Modules). Un TRAM es un módulo electrónico pequeño que consta básicamente de un transputer y memoria RAM aunque pueden contener otros dispositivos electrónicos. El TRAM más pequeño, conocido como TRAM tamaño 1, tiene 8 pines y se monta en una de las ranuras de

la tarjeta. Existen TRAMs más grandes que son múltiplos en dimensiones del TRAM tamaño 1. Por ejemplo, el TRAM de tamaño 2 ocupa dos ranuras de la tarjeta. Algunos TRAMs tienen conectores por encima de ellos de tal forma que se les puede apilar otro TRAM.

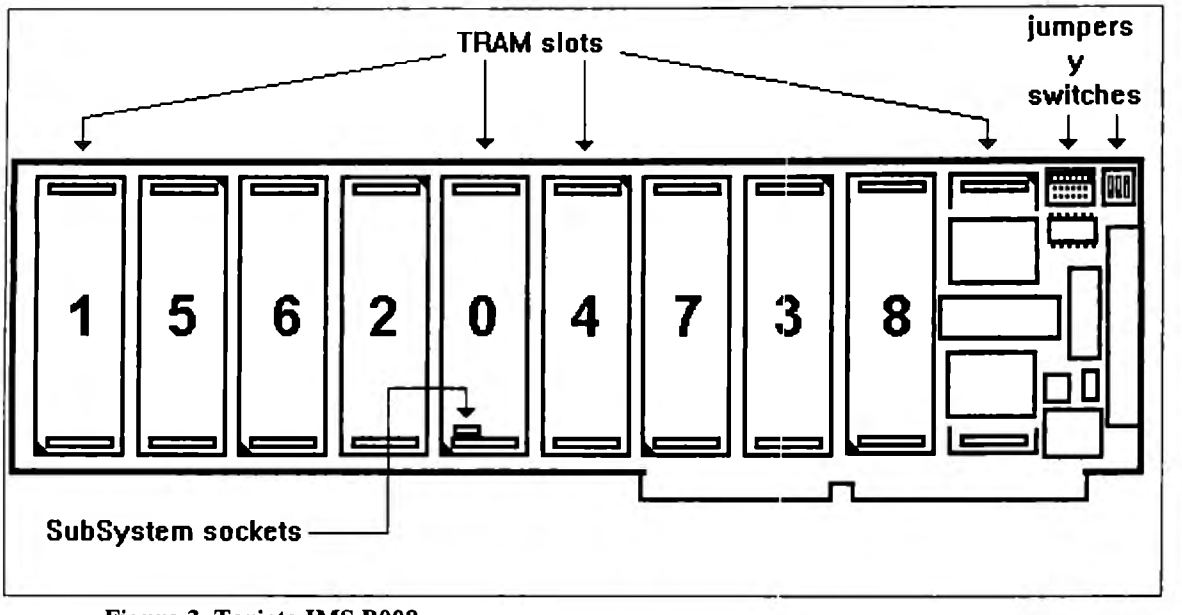


Figura 3. Tarjeta IMS B008.

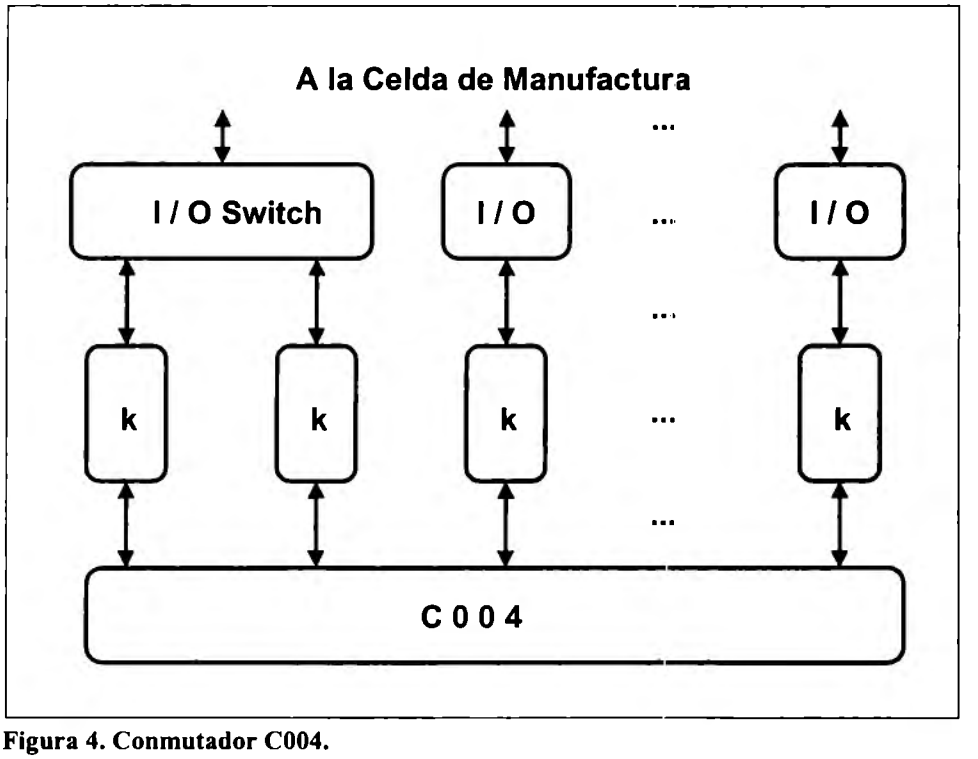


Figura 4. Conmutador C004.

La IMS B008 está diseñada para conectarse a un bus de PC y consta de 10 ranuras TRAM, una interfaz al bus de la PC (Ver Figura 3) y un conmutador de enlaces IMSC004 (Ver Figura 4), el cual permite reconfigurar dinámicamente la red a través de software. Las ranuras TRAM de la IMS B008 están conectados en línea (pipeline) usando 2 de los 4 enlaces de cada ranura. Los dos enlaces restantes de las ranuras 1 a 9 están conectadas al conmutador C004 para poder formar diferentes topologías de red.

El conmutador C004 consta de 32 puertos disponibles para conectar enlaces de transputers. Las conexiones son llevadas a cabo cuando llega un mensaje de configuración a través de un enlace especial del C004 llamado *ConfigLink*. Un transputer IMS T22 adicional está conectado al *ConfigLink* con el propósito de controlar y configurar al C004.

2.3.4 TRABAJOS YA HECHOS

Previamente a este trabajo se han desarrollado otras investigaciones dentro del proyecto, las cuales se describen a continuación.

2.3.4.1 Protocolo de Comunicación en una Arquitectura de Transputers para Controlar una Celda Flexible de Manufactura. [18]

Este trabajo se desarrolló en el Campus Toluca del ITESM por la M. en C. Sara del Socorro Mota, con el objetivo principal de implementar un protocolo orientado a byte para una arquitectura similar a la del controlador de la celda del ITESM-CEM. Esta investigación sirve como antecedente al desarrollo del presente trabajo por lo que se describe más detalladamente en el punto 0 de este documento.

2.3.4.2 Diseño de la Arquitectura de Hardware utilizando Transputers para el Control de una Celda Flexible de Manufactura [13]

Esta investigación fue desarrollada por la M. en C. Mirna Salmerón durante los años 1996 y 1997. Salmerón propone en este trabajo una topología de red de transputers (ver Figura 5) que permite la recuperación de fallas que pudiera tener algún procesador de la red, asegurando que siempre habrá un segundo que lo sustituya en su trabajo para no interrumpir el proceso de la celda.

La topología es formada utilizando el interruptor IMS CO04 para interconectar los enlaces de los transputers. Este interruptor es programado antes de cargar el software de aplicación que correrá en los diferentes procesadores utilizando una herramienta del C-tools de INMOS.

En esta arquitectura se contemplaron además los módulos SIO-232 y SIO-234, (refiérase al punto 3.4) que son las interfaces del controlador propuesto con los robots.

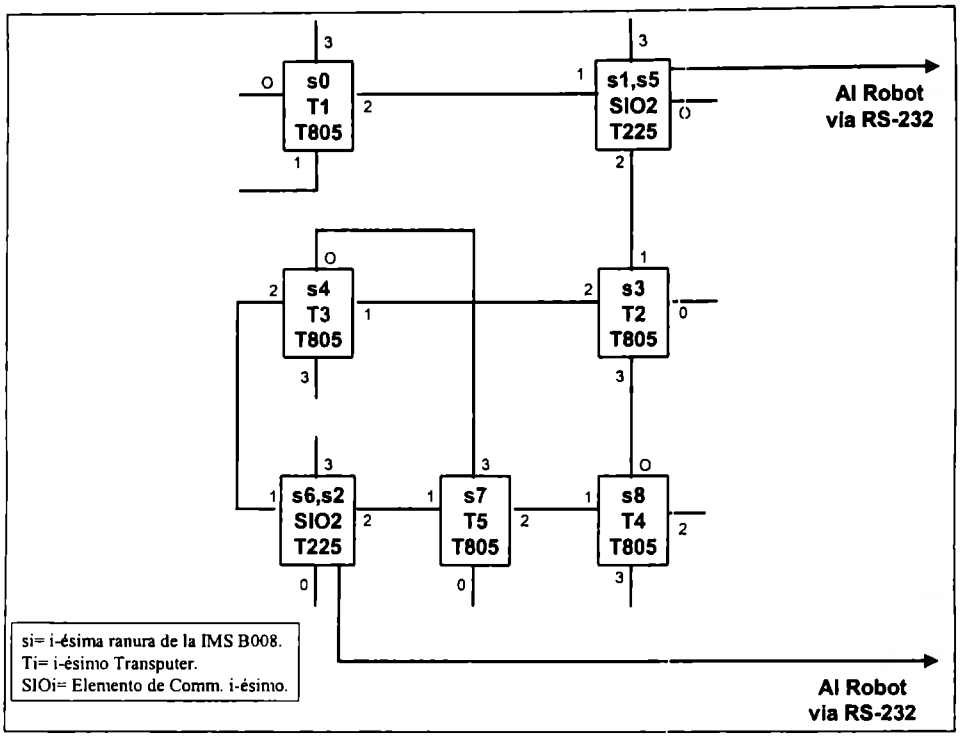


Figura 5Topología de Red propuesta por Salmerón para una Celda Flexible.

2.3.5 TRABAJOS EN CURSO

En paralelo a este trabajo se están desarrollando otros trabajos que complementan el proyecto general [1]. La importancia de conocer estos trabajos radica en tener la idea general de los requerimientos y medio ambiente en el que se implementará el protocolo objeto del presente trabajo.

2.3.5.1 Diseño de Interfaz y Conmutador de Comunicación entre Controlador y Elementos de una Celda de Manufactura [14]

Esta investigación está siendo desarrollada por el Ing. Edgar Morales en el ITESM-CEM. El objetivo de esa tesis es desarrollar una interfaz transputers - robots que permita redundancia para la tolerancia a fallas en una celda flexible de manufactura. Un diagrama general de esta interfaz se muestra en la Figura 6.

Los transputers del controlador procesan información e intercambian datos entre sí a través de sus enlaces, sin embargo, el objetivo fundamental del controlador es enviar comandos a los robots y obtener información de ellos para el monitoreo y control de actividades; debido a que los robots no cuentan con una interfaz propia para enlaces de transputers es mandatorio su desarrollo. La interfaz que está en investigación está basada en un módulo TRAM desarrollado por Transtech Parallel Systems, modelos SIO232 y SIO422 que se detallan más adelante en el punto 3.4 por ser tema básico para la implementación del protocolo de comunicación propuesto en el presente trabajo.

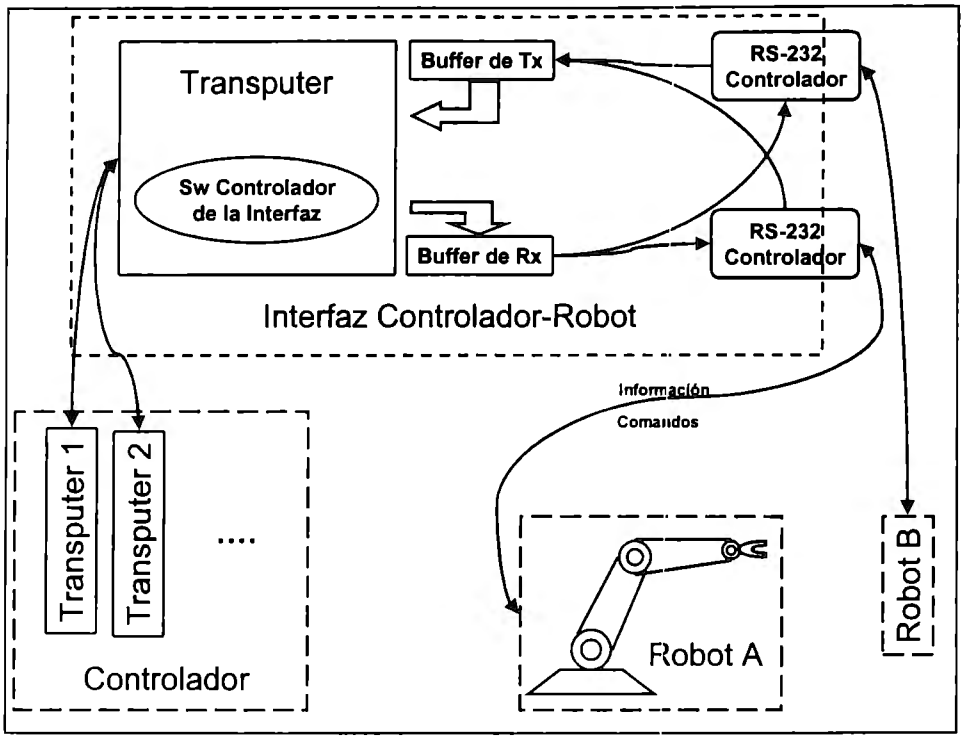


Figura 6 Interfaz Controlador-Robot.

2.3.5.2 Algoritmo de Tolerancia a Fallas para una celda flexible de Manufactura [15]

El diseño e implementación de un algoritmo de Tolerancia a fallas para la celda del ITESM - CEM está a cargo de la Ing. Laura Zavala. Con éste, Zavala propone un mecanismo de detección y recuperación de fallas para la celda, considerando tantos fallas de software como de hardware. Las bases de este algoritmo son:

- Cada elemento de la celda está controlado por al menos un transputer.
- El procesamiento debe ser en tiempo real.
- La configuración de la red puede cambiar ya sea debido a una falla o por que los requerimientos de la producción lo requieran.

Para ello propone un proceso llamado "Monitor" que es el encargado de la señalización de la falla; gracias a este se puede llevar un estado global del sistema ya que la actividad de todos los procesos es supervisada por él. El algoritmo permite la recuperación de fallas de enlace de comunicación, fallas de algún procesador y/o falla de algún proceso [15]. Al inicio de la

operación, el controlador se lanza con una topología de red inicial que es la que mantendrá en condiciones normales, solo en caso necesario está cambiará. La configuración de la red esta basada en lo propuesto por Salmerón [13] (ver Figura 5) en la que se requiere:

- Cuatro transputers pueden manejar cuatro elementos de la celda; la interfaz SIO sólo puede manejar dos canales externos (dos elementos de la celda) y cada elemento requiere de al menos un procesador, por lo que para usar los dos SIOs disponibles se requiere de cuatro transputers de la red.
- Un transputer para el monitor. Debido a que el proceso Monitor debe ser altamente confiable, el procesador no debe compartirse con otros procesos de producción.
- Un transputer para la comunicación con la computadora huésped.

Por lo anterior, la red de transputer queda formada por cinco o seis procesadores además de los SIOs como se muestra en la figura antes mencionada. Los procesadores T1 a T4 de tal esquema se encarga de controlar, cada uno, a un elemento de la celda. T5 se destina para el proceso principal del Monitor, éste sólo puede intercambiar mensajes directamente con los procesos que corren en los procesadores T4 y T2; para coordinar al resto utiliza canales virtuales (refiérase al punto 3.3.8.2 del presente trabajo). Sin embrago, para tener una red más robusta que coadyuve en la recuperación de la falla se hace uso del interruptor digital múltiple CO04 que esta integrado en la tarjeta IMS B008 (refiérase al punto 2.3.3) quedando una red totalmente conectada (ver Figura 7)

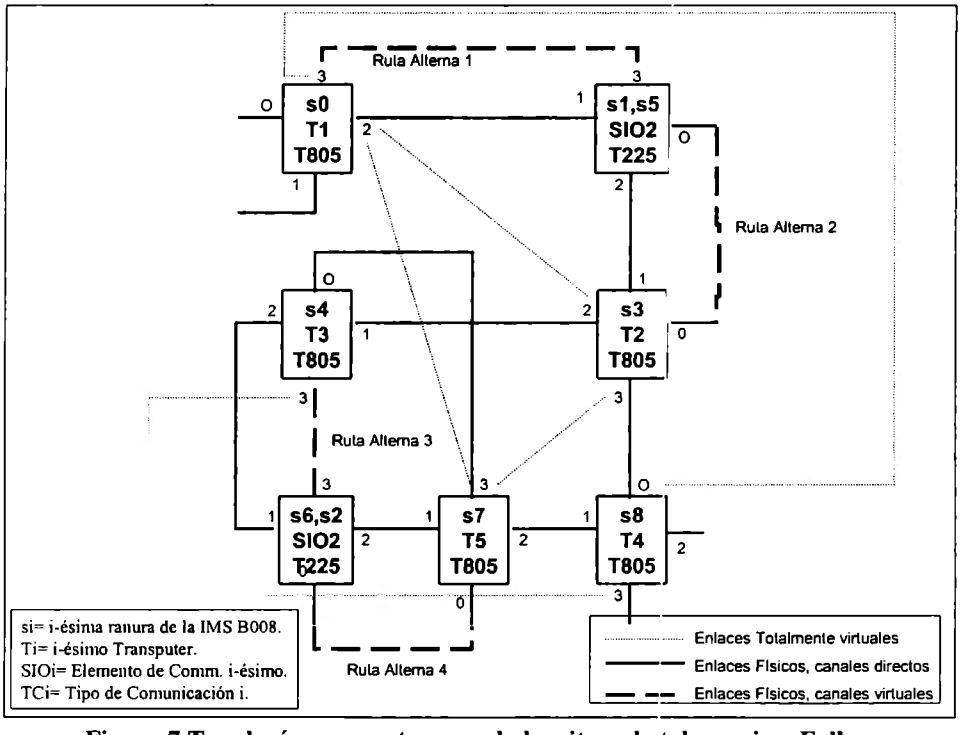


Figura 7 Topología porpuesta para el algoritmo de tolerancia a Fallas.

El algoritmo propuesto por Zavala se fundamente en las técnicas de redundancia en hardware y en software. Desde el punto de vista del hardware se tiene redundancia en los enlaces de comunicación ya que siempre habrá una ruta alterna para seguir comunicando al procesador con su interfaz gracias a la red totalmente conectada; también se tiene redundancia en los procesadores, ya que en caso de que falle un procesador, el procesador que esta conectado a la misma interfaz puede asumir el lugar del primero [15].

La redundancia en software consiste en replicar el código que se ejecuta en un procesador en su procesador de respaldo correspondiente; cada procesador cuenta con código y datos principales y código y datos auxiliares. En caso de que un procesador falle, el sistema del Monitor detectará la falla y se encargará de que el procesador de respaldo ejecute un nuevo proceso con el código y con los datos de respaldo; el Monitor indica al proceso de respaldo el último estado estable en el que se quedo, el proceso fallido.

Este trabajo esta en la etapa final de la implementación y pruebas y forma parte integral de este proyecto.

2.4 IMPORTANCIA DEL PRESENTE TRABAJO

El nuevo hardware propuesto en [12] es la base para una explotación óptima de la Celda de Manufactura, pero no es suficiente por sí misma; la eficiencia y efectividad de un sistema distribuido depende de su sistema de comunicación [8], por lo que se hace necesario implementar un subsistema que garantice la rapidez y confiabilidad de comunicación que todo sistema industrial requiere. Es imprescindible, además, que este subsistema siga lineamientos de estandarización internacional para que sea abierto, siendo éste el principal objetivo del presente trabajo.

3 ESTADO DEL ARTE

En el presente capítulo se abordan tópicos relevantes que conforman el ambiente de conocimientos en el cual se desarrolla el presente trabajo. Primeramente se introduce el concepto de los estándares y la importancia que tienen en el desarrollo de todo trabajo; siguiendo por esta línea se presentan los estándares que existen en el ambiente de manufactura, especialmente se habla de MAP y de MMS. Se presenta también el tema de los protocolos, modelos de validación y herramientas para validar protocolos, haciendo énfasis en el lenguaje Promela ya que este se utiliza en la validación del protocolo propuesto en este trabajo. Por último, se dedica una sección a los procesadores que se utilizan como plataforma para el controlador, los transputers.

3.1 LOS ESTÁNDARES

Uno de los principales problemas que aparecen en la implantación de un sistema es la integración de éste a los sistemas existentes. Esta integración es requerida debido a que habrá datos que intercambiar entre sistemas. En muchas ocasiones las computadoras y dispositivos a comunicar son de diferentes fabricantes y no hay una forma directa de comunicarlos. Una solución es tener equipo sólo de un fabricante; usualmente un solo vendedor provee todas las facilidades y servicios de comunicación de red, sin embargo, debido a los requerimientos de las diferentes aplicaciones, se requiere de dispositivos inteligentes hechos por diversos fabricantes. Para poder enlazar los equipos de los distintos vendedores en la red se invierte bastante tiempo para obtener las especificaciones del proveedor y, ni aún teniendo esta información, se garantiza la interoperabilidad.

Una solución más adecuada es reemplazar las redes propietarias por redes que reconozcan estándares como, por ejemplo, el modelo de referencia ISO-OSI (International Standardization Organization - Open System Interconnection). OSI es un conjunto de reglas que especifican ciertos requerimientos en el diseño de LANs (Local Area Networks) agrupados como protocolos de capas [4]. Además de ISO, existen muchas organizaciones y grupos cuyos esfuerzos se centralizan en la estandarización, algunas de las más relevantes son: el Comité Consultivo Internacional en Telegrafía y Telefonía CCITT, el Instituto Americano de Estándares, ANSI

(American National Standards Institute), el NBS (National Bureau of Standards), el Instituto de Ingenieros Eléctricos y Electrónicos, IEEE (Institute of Electrical and Electronics Engineers) y el grupo para los protocolos MAP/TOP [10]

3.1.1 LOS ESTÁNDARES EN LA MANUFACTURA AUTOMATIZADA

El costo y esfuerzo para tener un sistema de manufactura automatizada apropiado es alto debido a que en el mercado hay esquemas incompatibles de vendedores de equipos automáticos, compañías de computadoras y proveedores de sistemas de comunicación. Las soluciones anteriormente practicadas [8] eran:

1. Los mismos usuarios diseñaban interfaces propias para interconectar los equipos de diferentes marcas, dando como resultado las redes conocidas como de usuario o redes propietarias. El costo de esta "solución" es alto, ya que para agregar un nuevo dispositivo o computadora a la red, implicaba el diseño particular, en muchas ocasiones, e implementación de una interfaz, es decir, este forma de habilitar el intercambio de información con los dispositivos de diferentes productores no es ni flexible ni abierta.

2. El usuario adoptaba y compraba máquinas y productos de un solo proveedor. En este caso ya no se presentaban problemas para el crecimiento del sistema, ya que el mismo proveedor vendía los nuevos dispositivos con sus interfaces necesarias; además al tener productos de una misma marca se disminuían los problemas de funcionamiento del sistema, sin embargo, esta solución obligaba al usuario a mantenerse permanentemente con un mismo proveedor; el nivel de flexibilidad y apertura no es muy completo.

Se requería de una solución que garantizara la interoperabilidad de los equipos de los diferentes proveedores, es decir, una estandarización.

En los años 80's se generó un trabajo para la estandarización de las comunicaciones en las fábricas llamado MAP, así como un protocolo similar para oficinas (TOP) [9]. El objetivo principal de MAP y TOP es especificar técnicas para la implementación de las comunicaciones en la industria de la manufactura. En la siguiente sección se describe el protocolo MAP y TOP.

3.1.2 MAP Y TOP (MANUFACTURING AUTOMATION PROTOCOL AND TECHNICAL AND OFFICE PROTOCOL)

Las especificaciones MAP & TOP comprenden una familia de protocolos capaces de soportar un amplio rango de necesidades de cómputo distribuido [5], incluyendo, para el caso de TOP, estándares de aplicación en administración, acceso y transferencia de archivos entre máquinas del ambiente de producción y máquinas del ambiente de oficina.

MAP ha sido orientado específicamente a las necesidades de la CIM desde el punto de vista operación. Esto es evidente en los estándares de la capa de aplicación de MAP, tal como la

especificación de mensajes de manufactura MMS (Manufacturing Message Specification) [5]. La primera versión de MAP fue sólo una especificación de procedimientos, adoptada en el otoño de 1982. La última versión publicada de MAP (MAP 3.0) data de Junio de 1988.

Las características del protocolo de MAP 3.0 están basadas en los siguientes criterios:

- Adopción del modelo de referencia OSI de ISO, con una selección para cada capa, de protocolos estándares existentes que se adecuaban a los requerimientos de la automatización industrial.
- Adopción de nuevos protocolos para estandarización en las áreas donde no existían protocolos satisfactorios.

Hay tres tipos diferentes de sistemas finales incluidos en la especificación MAP 3.0, de acuerdo a los roles que juegan en la red, FullMAP, EPA y MiniMAP, los cuales son explicados a continuación.

3.1.2.1 FullMAP

File transfer and Management (ISO 8571) Manufacturing message specification (ISO 9506) MAP network management Direct service (ISO 9594) Association control service element(ISO 8649)	CAPA OSI 7
Presentation Kernel (ISO 8822)	6
Sesion Kernel (ISO 8326)	5
Transport class 4 service (ISO 8072)	4
Connectionless network service (ISO 8348)	3
Logical link control classes 1 and 3 (ISO 8802/2) Token-bus medium access control (ISO 8802/4)	2
Token-bus 10 Mbps broadband (ISO 8802/4) Token-bus 5 Mbps carrierband (ISO 8802/4)	1

Figura 8 Perfil de protocolos para estaciones Full MAP.

El primer tipo de sistemas MAP es llamado FullMAP ya que incluye las siete capas completas de OSI (ver Figura 8). La estación FullMAP no es adecuada para aplicaciones de tiempo real ya que el número y complejidad de los protocolos usados reducen notablemente la velocidad en la comunicación. Este tipo de estaciones son útiles para aplicaciones relacionadas a la administración de la corporación y/o de la planta.

Las estaciones MAP de este tipo son las más apegadas al modelo OSI, sin embargo, las capas 3-6 están desarrolladas teniendo en mente aplicaciones de procesamiento de datos en general más que aplicaciones industriales y el uso de protocolos de propósitos generales no es por sí mismo una característica a favor de implementaciones de tiempo real. FullMAP contiene un complemento muy rico de servicios, probablemente más rico que lo que requiere la mayoría de las aplicaciones [9], pero implica un procesamiento adicional muy sustancial por lo que los dispositivos de piso no deben ser de este perfil.

3.1.2.2 MiniMAP

La arquitectura de protocolo de los sistemas MiniMAP representa una subdivisión de los estándares de OSI, ya que las capas 3 a 6 no están incluidas. Esta reducción garantiza mejoras en tiempos de respuesta de los mensajes transmitidos en la red, pero se tienen las siguientes pérdidas de funcionalidades de comunicación con respecto a FullMAP:

- Por la ausencia de la capa de transporte, no se lleva a cabo la fragmentación, por lo que el tamaño de los mensajes de la aplicación está restringido al tamaño máximo permitido por el canal.
- No es posible el ruteo, ya que no existe capa de red. Las redes de MiniMAP están limitadas a un solo segmento.
- Solo se puede usar comunicación fue dúplex, ya que no hay capa de sesión.
- La confiabilidad de entrega de mensajes, según OSI garantizadas por las capas de red y transporte, es implementada al introducir reconocimiento de la capa de enlace de datos, usando LLC clase 3.
- Los formatos de los mensajes son fijos, ya que no hay capa de presentación.

Se debe señalar que MiniMAP ha iniciado el trabajo de los organismos internacionales de estándares en redes con requerimientos de comunicación de tiempo crítico. En la Figura 9 se observan las características de protocolos en estaciones MiniMAP.

Los protocolos en la capa de aplicación sólo incluyen servicios de administración y servicios de directorio, debido a que la estación MiniMAP es esencial para el control de dispositivos de manufactura y no para la administración de la planta.

MMS es el componente principal de MiniMAP para el trabajo en tiempo real de dispositivos de manufactura [9] y es un proveedor de servicios que simula en software un dispositivo general de

manufactura que reemplaza a una máquina dada. MMS es un lenguaje de comandos que elimina la necesidad de procedimientos propietarios y la conversión de lenguajes particulares, además provee la oportunidad de programar de una forma orientada a tareas y de reusar los programas de aplicación; esta especificación es detallada en el punto 3.1.3.

	CAPA OSI
Manufacturing Message Specification (MMS) directory, network management	**7**
N U L L	**6 - 3**
802 class 3 802.4 intermediate response	**2**
802.4 carrierband (5 Mbps)	**1**

Figura 9 Perfil de protocolos de las estaciones MiniMAP.

3.1.2.3 EPA

Los nodos MiniMAP y FullMAP no pueden comunicarse directamente entre sí debido a sus diferencias en cuanto a protocolos. La conexión de subredes MiniMAP al backbone FullMAP bien puede hacerse por medio de un gateway pero implica costo e induce retardo en las transacciones; otra solución es el sistema EPA (Enhanced Performance Architecture, Figura 10), que provee protocolos de 3 y de 7 capas al mismo tiempo [9], distinguiendo los modos de servicio por sus puntos de acceso a éste.

Los nodos EPA son los idóneos para implementar la interfaz entre el ambiente de producción y el ambiente de administración de la producción y muy eficientes para celdas de manufactura donde el 70 al 95% del tráfico de las comunicaciones esta localizado dentro de ella [22]. Un elemento importante en la evolución de MAP es el incluir la especificación de opciones para redes que soportan comunicaciones "tiempo real".

Las ventajas generados por este tipo de arquitectura son los siguientes:

- Bajos tiempos de respuesta para mensajes cortos de alta prioridad.
- Alta confiabilidad del medio y del método de señalización.
- Configuración sencilla.
- Conexión sencilla al backbone principal.

- Control de acceso para evitar conexiones no autorizadas.

Algunas restricciones generadas por la simplificación en la arquitectura que deben ser observadas son:

- Perdida de la garantía de la entrega del mensaje.
- Limitación en el tamaño máximo de los mensajes restringido por el tamaño máximo soportado por la capa de enlace.
- Imposibilidad de sincronizar diálogos entre aplicaciones o de proveer puntos de chequeo debido a la inexistencia de la capa de sesión.
- La sintaxis de presentación debe ser conocida a priori por las aplicaciones que utilizan EPA.

		CAPA
ACSE, FTAM, MMS, Directory and Management Protocolo selector		7
Presentation		6
Session		5
Transport		4
Network		3
802.2 class 1 802.4 (Immediate response)	802.2 class 3	2
802.4 broadband (10 Mbps)	802.4 carrierband (5 Mbps)	1

Figura 10 Perfil de los protocolos de las estaciones EPA.

3.1.2.4 Por qué no se implementa MiniMAP en la celda del ITESM-CEM

El modelo más adecuado para una celda como la del ITESM-CEM, es el MiniMAP, ya que sólo comunica dispositivos del nivel tienda (shop) y la información intercambiada no abandona nunca la red. Sin embargo, la plataforma elegida para el controlador implica inherentemente la capa física entre nodos de la red, los enlaces de transputers, los cuales se describen en el punto 3.3; para la capa de enlace se presenta una situación análoga, ya que los transputers cuentan con un mecanismo de comunicación muy eficiente a través de canales (refiérase al punto 3.3.2). Por

supuesto, se podría emular la comunicación a través de los protocolos de la familia 802.4 para estas capas, pero al hacerlo se atentaría contra las ventajas decisivas que motivaron la elección de estas herramientas para implementar el controlador.

La importancia de MiniMAP para este proyecto no disminuye por lo comentado en el párrafo anterior, ya que se puede usar este mismo modelo adaptándolo a la estructura en las comunicaciones en los transputers para contar, por un lado, de las bondades que está brinda y, por otro, de las ventajas que tiene el modelo de tres capas.

Por lo que respecta a la capa de aplicación de MiniMAP, MMS, es totalmente adaptable al controlador de la celda ITESM-CEM y es su implementación la meta principal de este trabajo.

3.1.3 LA ESPECIFICACIÓN DE MENSAJES DE MANUFACTURA MMS

MMS es un estándar de comunicación de alto nivel desarrollado por EIA (Electronics Industries Association) para el intercambio de datos en tiempo real y la supervisión del control de la información entre dispositivos de red y/o aplicaciones de manufactura, de tal forma que sea independiente de la aplicación y del creador del dispositivo. Esta especificación elimina la necesidad de procedimientos propietarios de la aplicación para la interpretación de datos transferidos a través de la red y permite una programación orientada a la tarea al hacerse cargo de lo referente a la comunicación de la información. MMS es un estándar internacional (ISO 9506) que es desarrollado y mantenido por el Comité Técnico Número 184 (TC184) Automatización Industrial de ISO y su se encuentra en dos documentos:

- ISO/IEC 9506-1. Define los dispositivos virtuales de manufactura VMD (Virtual Manufacturing Device); los servicios que son ofrecidos remotamente para la manipulación de los VMDs y los atributos y parámetros asociados con el VMD y sus servicios.
- ISO/IEC 9506-2. Especifica el protocolo MMS en términos de PDUs, es decir, las reglas de la comunicación quedan especificadas a través del formato de los mensajes; tal especificación es descrita utilizando una representación estándar llamada Notación de Sintaxis Abstracta No. 1 (ASN.1 - ISO 8824). En [27] se encuentra el formato de los mensajes MMS según ASN.1.

Además de los servicios generales MMS, que aplican para una amplia variedad de dispositivos de manufactura, existen extensiones de especificaciones para dispositivos más especializados conocidos como Anexos MMS y que detallan cómo deben ejecutarse e interpretarse los mensajes en tales dispositivos y particularidades adicionales de MMS para ellos. Algunos de estos anexos son:

- Especificación de Mensajes para Robots (Robot Message Specification) desarrollado por RIA.
- Especificación de Mensajes para Control de Procesos (Process Control Message Specification) desarrollado por ISA.
- Especificación de Mensajes para Control Numérico (Numerical Control Message Specification) desarrollado por EIA.

- Especificación de Mensajes para Controladores Programables (Programmable Controller Message Specification) desarrollado por IEC.
- Especificación de Mensajes para Administración de la Producción (Production Manager Message Specification) desarrollado por ISO.

Existen ya muchas aplicaciones en las industrias de energía eléctrica que están siendo implementadas adicionadas con MMS, tales como en Unidades Terminales Remotas (RTU), Sistemas de Administración de Energía (EMS) y otros dispositivos electrónicos inteligentes (IED) como interruptores inteligentes [23]. La plataforma computacional MMS más popular tiene conectividad disponible, ya sea implementada por el fabricante directamente o por un tercer elemento; algunas aplicaciones ya disponibles son Interfaces Programables de Aplicación (API), sistemas de monitoreo gráfico, compuertas, procesadores de palabras, habilitadores de aplicación (A/Es) y sistemas de administración de bases de datos relacionales (RDBMS). Las implementaciones MMS soportan una variedad de enlaces de comunicaciones incluyendo Ethernet, Token bus, RS-323C, OSI, TCP/IP, MiniMAP, FAIS, etc. y pueden conectarse a varios otros tipos de sistemas usando puentes de red, ruteadores y compuertas [23].

3.1.3.1 Intercambio de Información MMS

En general una entidad MMS describe, en software, un dispositivo general de manufactura y el servicio que éste ofrece; este elemento puede ser mapeado a un dispositivo real por un usuario o por un vendedor, lo que permite que al momento que el usuario manipule este modelo realmente este teniendo el control del dispositivo [11]. Todas las interacciones entre dos entidades MMS están basadas en el modelo Cliente-Servidor. La Figura 11 representa las interacciones entre entidades MMS; cada entidad contiene una máquina de estados llamada MMSPM (MMS Protocolo Machine), la cual tiene secciones de envío y recepción. El cliente puede enviar una petición de servicio confirmado al servidor, el cual es responsable de generarle una respuesta, positiva o negativa, ó puede generar peticiones de servicio no confirmado en las que no se requiere respuesta del servidor. Cualquiera que sea el tipo de servicio, el intercambio de información se llama a cabo en unidades de datos del protocolo llamadas PDUs (Protocol Data Unit).

MMSPM solo implementa las funciones que se requieren para el intercambio PDUs entre las dos unidades comunicantes y no provee ninguna función para la ejecución del servicio; para ello el servidor debe incluir una tarea o un grupo de tareas de aplicación, referidas como dispositivo virtual de manufactura VMD (Virtual Manufacturing Device), el cual está encargado de ejecutar la función solicitada en la petición y preparar una respuesta, que eventualmente envía al cliente a través a través de las dos MMSPMs.

De acuerdo al modelo Cliente-Servidor, se puede tener una configuración Solo-Servidor en algunas máquinas y Solo-Cliente en otras, de tal manera que el cliente puede solicitar a cada servidor su estado actual; pero también es posible que una entidad MMS pueda actuar como servidor y como cliente al mismo tiempo, es decir, esta entidad puede enviar peticiones de servicio hacia servidores, mientras procesa peticiones que le llegaron de otras entidades; una

situación donde se requiere esta configuración es en sistemas compuestos de máquinas equivalentes que necesitan coordinar sus operaciones con un control no centralizado.

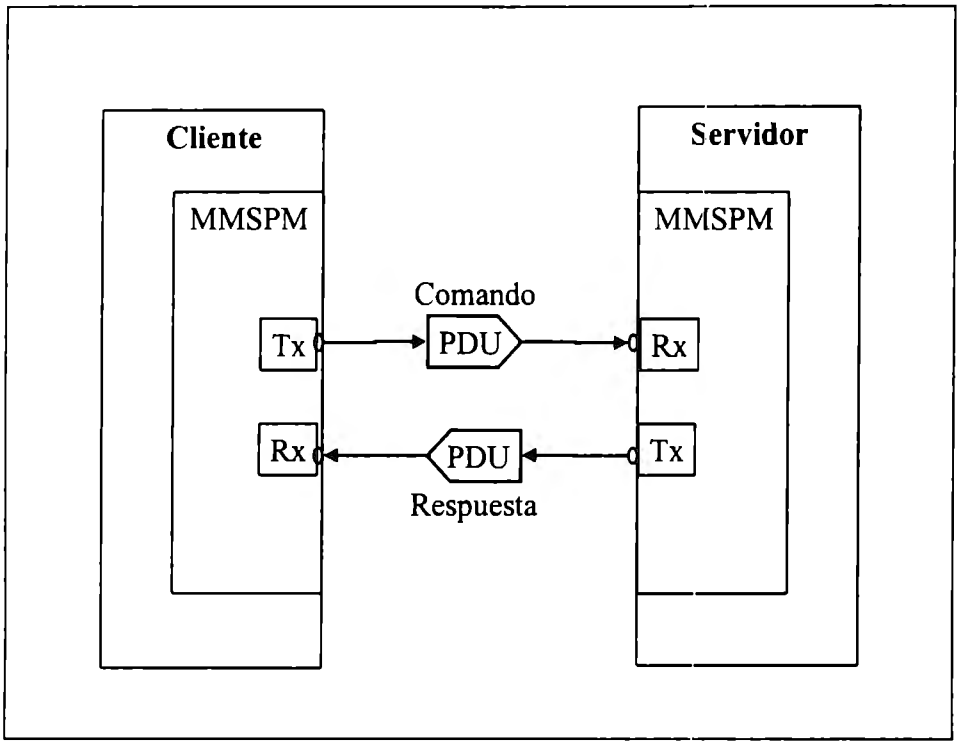


Figura 11 Esquema abstracto del intercambio de información entre un Cliente y un Servidor MMS.

Las transiciones en las MMSPM para una operación de servicio confirmado se observan en la Figura 12; el evento mostrado sobre la línea horizontal de cada transición es una operación de entrada y el que está abajo es operación de salida, las operaciones denotadas con iniciales mayúsculas corresponden a interacciones con otras MMSPMs mientras las otras se refieren a interacciones con el usuario local a la MMSPM.

En MMS también existen servicios no confirmados, aunque sólo son un pequeño grupo; en el caso de éstos, el diagrama de estados de la MMSPM transmisora se reduce a un solo estado con una transición a sí mismo, que corresponde a las acciones de recepción de la petición y de la transmisión de los PDUs correspondientes. Un esquema similar se presenta para la MMSPM receptora.

Cuando un servidor MMS recibe de un cliente una petición de servicio confirmado se crea un objeto llamado "objeto de transacción", el cual contiene una estructura de datos con toda la información de la petición y todos los modificadores pre-ejecucion y post-ejecución. El objeto de transición se destruye cuando se responde a la petición.

Los formatos de los PDUs y los servicios que se ofrecen según MMS son analizados más a detalle en la sección 4.3.4 debido a su importancia en este trabajo.

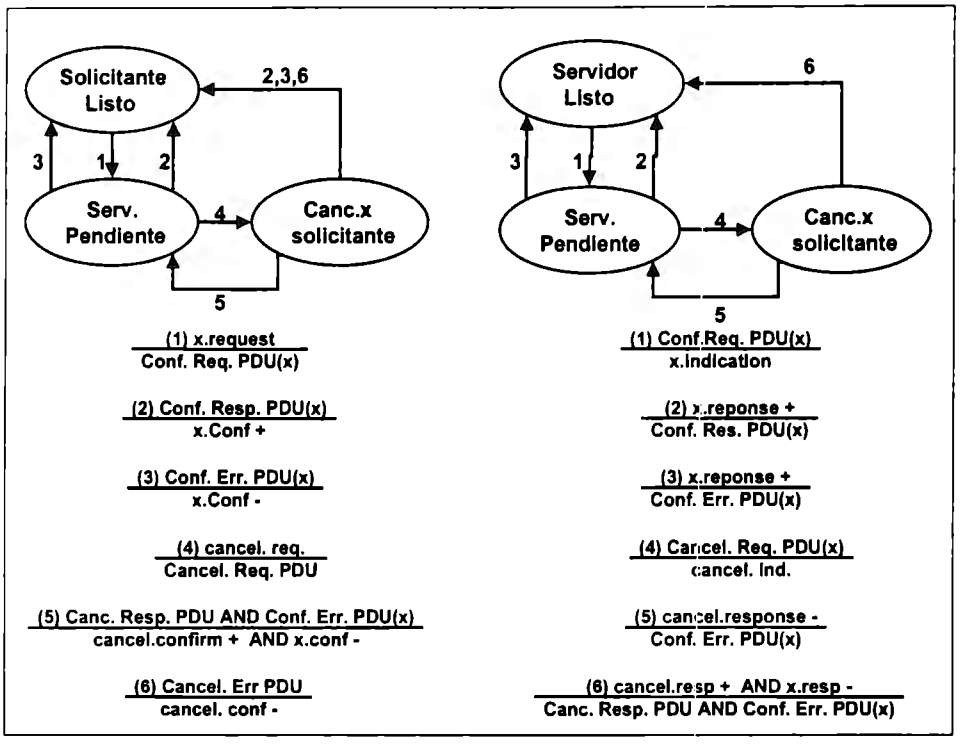


Figura 12 Diagrama de Estados para la ejecución de una invocación de servicio confirmado.

3.1.3.2 Asociación de Aplicación

MMS hace uso de un protocolo orientado a conexión, por lo que se requiere establecer un ambiente entre el cliente y el servidor antes de cualquier intercambio de información. Este ambiente es conocido como asociación de la aplicación AA (Application Association). La AA puede ser requerida tanto por el servidor como por el cliente y es en esta fase cuando se negocian los parámetros con los que trabajaran las MMSPMs de cada entidad. Los parámetros a establecer al inicio de la AA son:

- Número máximo de peticiones pendientes.- Número máximo de objetos de transacción que puede coexistir en cualquier momento. Este mecanismo puede ser visto como la implementación de un control de flujo a nivel aplicación, que es muy útil para protocolos reducidos como MiniMAP, ya que no cuentan con mecanismos implementados por capas intermedias OSI. Es posible negociar dos valores distintos de Número máximo de peticiones pendientes, uno para cada extremo del AA.
- Límite superior del nivel de anidación en la definición de variables.- Como es posible manipular variables, es necesario definir el límite, pues a mayor nivel de anidación se requiere de una mayor capacidad de memoria. Se determina un mismo valor para cada extremo.
- Número de versión.- La versión MMS compatible de ambas entidades.

- Bloque de construcción de conformidad.- El conjunto más grande de bloques de construcción de conformidad soportado por ambas entidades.
- Servicios soportados.- El conjunto mayor de servicios soportados por ambas entidades.

3.1.3.3 Dispositivo Virtual de Manufactura VMD

A diferencia de otros protocolos OSI, MMS no establece solamente el comportamiento de las dos entidades apareadas por nivel envueltas en el intercambio, sino que además, define parcialmente el comportamiento externo de las tareas de aplicación colocadas en un nivel más alto a la entidad MMS. La definición del protocolo sólo cubre las operaciones de transferencia de mensajes, mientras que la definición del servicio está basada en la descripción del comportamiento de objetos (como variables, semáforos, eventos y trabajos), que son implementados por las tareas de aplicación del lado del servidor y agrupados en una abstracción conocida como DVM. Al establecer el perfil general del DVM se obliga a las aplicaciones a una estandarización en su parte básica. En la Figura 13 se esquematiza la relación de la máquina del protocolo MMS con sus objetos locales.

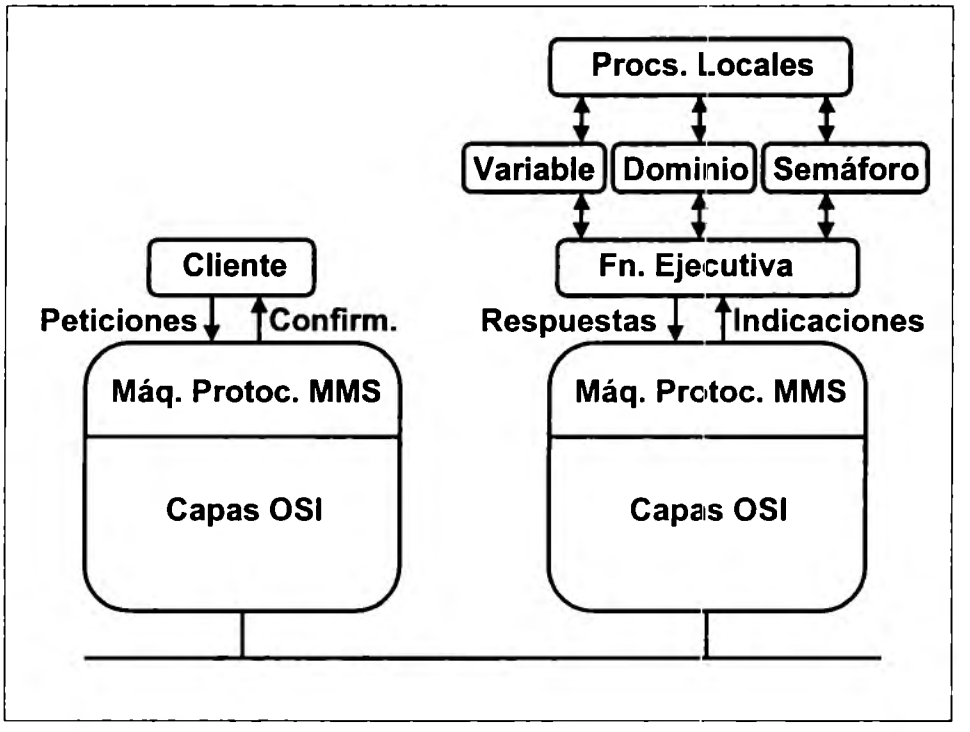


Figura 13 La MMSPM y su relación con los objetos MMS y capas inferiores OSI.

El VMD juega el rol principal en la definición del servicio MMS debido a que la implementación de un servidor MMS debe mapear el comportamiento externo de un VMD a las funcionalidades de un dispositivo real de manufactura. Para tener un mapeo fidedigno, el VDM cuenta con un estado lógico y un estado físico; el primero relacionado con el modelo de software y el segundo

con el hardware de la unidad real. En la Tabla 1 y en Tabla 2 se muestran los valores posibles para estos estados.

Estado	Significado
STATE-CHANGES-ALLOWED	El VMD está disponible para ejecutar todos los servicios definidos.
NO-STATE-CHANGES-ALLOWED	El VMD sólo está disponible para aceptar peticiones de servicio relacionadas a la iniciación y cierre de AA y aquellas usadas para obtener los atributos del objeto o leer su contenido; Sólo se pueden llevar a cabo operaciones de monitoreo pero no es posible controlar el estado de los objetos MMS.
LIMITED-SERVICE-PERMITTED	El VMD sólo soporta los servicios usados para cerrar AA's y para obtener el estado y la información de identificación del VMD. El VMD no está disponible para operaciones de monitoreo.
SUPPORT-SERVICE-ALLOWED	Todos los servicios del VMD pueden ejecutarse excepto los servicios de control de las invocaciones de programas; El VMD puede participar en aplicaciones de control y monitoreo con la limitante de no permitir a estaciones remotas activar/desactivar partes de los programas.

Tabla 1 Estados Lógicos de un VMD.

Estado	Significado
OPERATIONAL	El VMD es capaz de ejecutar todas las tareas.
PARTIALLY-OPERATIONAL	Existen problemas en el hardware que limitan el número de operaciones que pueden ejecutarse.
INOPERABLE	Los problemas son tan severos que se evita la ejecución de cualquier trabajo real.
NEEDS-COMMISSIONING	Se requiere una operación local (y probablemente manual) antes de ejecutar cualquier tarea.

Tabla 2 Estados Físicos de un VMD.

Además de un estado físico y un estado lógico, todo VMD cuenta con un grupo de atributos que ayudan a los clientes solicitantes referirse a éste y sus objetos subordinados; Estos atributos se enlistan en la Tabla 3.

Es preciso mencionar que MMS no contempla la creación ni destrucción de VMD, por ello la activación y desactivación de los procesos que implementan un VMD debe llevarse a cabo por operaciones de administración del sistema sobre el cuál se implemente MMS.

Atributo	Descripción
Función Ejecutiva.	Nombre de la función ejecutiva.
Nombre del vendedor.	Nombre del desarrollador del software del dispositivo virtual.
Nombre del modelo.	Nombre lógico del modelo virtual.
Revisión.	Número de revisión del software.
Sintaxis abstracta soportada.	Este parámetro se refiere a las sintaxis abstracta definida en ASN.1 (refiérase a [27])
Estado lógico.	Describe el nivel real de funcionalidad de la función ejecutiva (ver Tabla 1).
Lista de capacidades.	Capacidades incluidas en este dispositivos.
Estado Físico.	Describe el estado de las capacidades en el VMD (ver Tabla 2).
Lista de invocaciones de programas.	Descripción de todas las invocaciones a programas definidas para el VMD.
Lista de dominios.	Incluye todos los dominios definidos en el VMD.
Lista de objetos de transacción.	Esta lista es dinámica y crece a medida que lleguen indicaciones de servicio confirmado.
Lista de máquinas de estado de carga.	Incluye el estado de todas las máquinas de estado que se están actualmente cargando.
Información adicional.	Atributos definidos en los estándares adicionales

Tabla 3 Atributos de un MVD.

Cada dispositivo real puede requerir implementar más de un VMD según la complejidad de su operación, sin embargo los dispositivos virtuales siempre se consideran lógicamente independientes uno del otro y sus direcciones deben estar separadas debido a que cada VMD está asociado con distintos grupos de puntos de acceso en la capa de presentación; La relación entre el dispositivo real, el VMD y los objetos de éste se representan en la Figura 14, en la que se observa que la estructura de un VMD incluye:

- Exactamente una función ejecutiva.
- Cero o más invocaciones a programas.
- Uno o más dominios relacionados con una o más invocaciones a programas.
- Objetos locales a cada dominio, tales como tipos, variables, semáforos y trabajos.

En los siguientes párrafos se detallan estos componentes del DVM.

3.1.3.3.1 La Función Ejecutiva

La función ejecutiva es la parte del VMD que realmente ejecuta las acciones en los objetos del VMD relacionados con la petición entrante; la relación entre la función ejecutiva, la MMSPM y los procesos de aplicación locales se muestra en la Figura 13. La función ejecutiva manipula los objetos VMD a petición del cliente remoto mientras que los procesos locales cambian el valor de estos objetos de acuerdo a operaciones locales. El acceso concurrente de la función ejecutiva y de los procesos locales a estos objetos es controlado generalmente por el sistema operativo local.

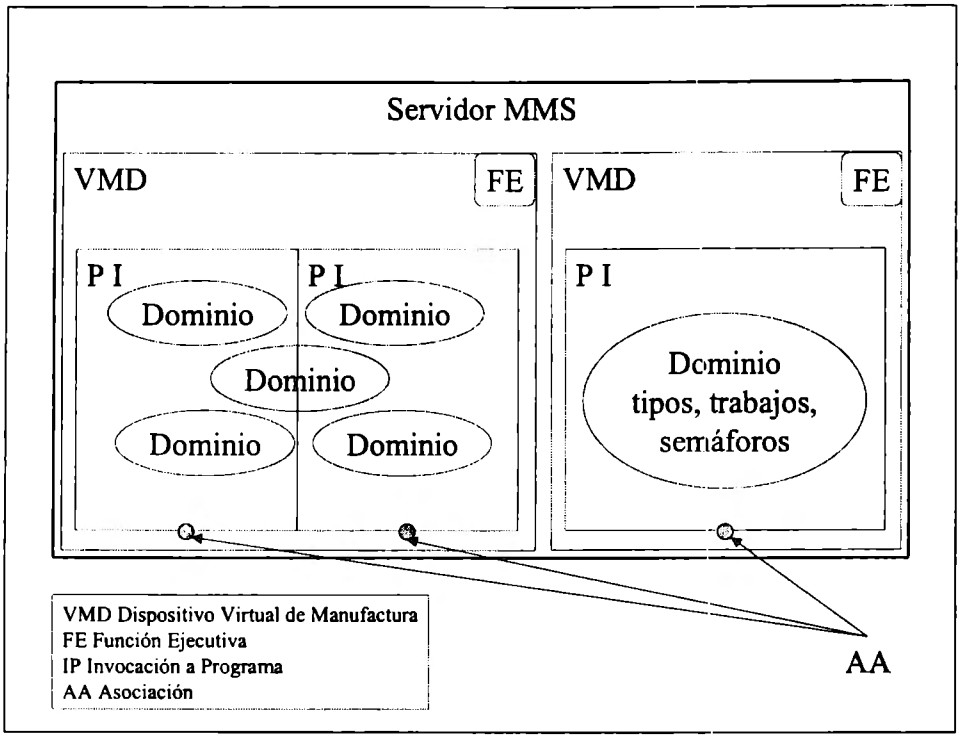


Figura 14 Dispositivo real, VMDs y sus componentes.

3.1.3.3.2 Capacidades

Una capacidad es un recurso o grupo de recursos de un dispositivo real, que puede asociarse a una cadena de caracteres. No se da una definición estándar para el comportamiento de las capacidades ni hay administración ni reglas de definición. Se asume que la función ejecutiva está alerta de los recursos que le corresponden a cada capacidad y tiene información suficiente para su administración. Un ejemplo de capacidad es la variable que contiene el valor actual de las coordenadas (x, y, z) que representan la posición real de un brazo de robot con respecto a su origen; dicha capacidad puede referirse por el nombre lógico *RealPos*.

Es posible tener capacidades compartidas y no compartidas de acuerdo a reglas locales, ya que no se cuenta con un servicio MMS para administrar las capacidades.

3.1.3.3.3 Dominios

Un dominio es un objeto que incluye todos los recursos requeridos para ejecutar un aspecto particular de un control coordinado o una aplicación de monitoreo. Un ejemplo de dominio en un robot es el brazo, que está compuesto por todos los recursos para la operación del brazo real del robot, incluyendo los datos y código para el programa de control.

Un dominio puede incluir:

- Un grupo de capacidades.

- Objetos MMS subordinados que no existen si el dominio no existe, que pueden ser variables, semáforos o tareas. Por ejemplo el semáforo de control sobre el brazo.
- Información; áreas de código y datos que pueden usarse para la ejecución de un programa.

Los dominios pueden ser compartidos por diferentes invocaciones a programas pero sus objetos subordinados no pueden ser compartidos con otros VMDs. Por su ciclo de vida se pueden clasificar como estáticos, si siempre están presentes en el VMD, o dinámicos, si se pueden crear y destruir. Para este último caso es posible cargar/descargar el contenido de dominios desde y hacia otra estación de trabajo. Cuando se crea un dominio, se crean todos los objetos MMS subordinados y cuando se destruye, se destruyen con él todos esos objetos.

En la Tabla 4 se pueden observar los atributos más relevantes de un dominio y en la Tabla 5 sus estados posibles.

Atributo	Descripción
Nombre	Identificador del dominio
Lista de capacidades:	Si cuenta con capacidades, estas están registradas en este atributo.
Estado	El estado actual de este dominio ().
Borrable MMS	Indica si el dominio es dinámico
Compartir	Indica si se puede compartir el dominio entre invocaciones de programas.
Lista de objetos subordinados	Esta lista incluye todos los trabajos actualmente activos definidos dentro del alcance del dominio.
Lista de las invocaciones de programas	Esta lista incluye las invocaciones a programas definidas en el dominio.

Tabla 4 Atributos de un Dominio.

Estado	Significado
LOADING	Indica que el dominio está siendo cargado.
COMPLETE	Ya se cargó hasta el último segmento del dominio pero aún no se recibe el comando del cierre de carga.
READY	Se ha cerrado apropiadamente la carga del dominio y esta listo para operar.
INCOMPLETE	Se genera cuando se ha cerrado la carga sin tener todos los segmentos completos.
IN-USE	Indica si alguna invocación de programa está usando el dominio.

Tabla 5 Estados posibles de un Dominio.

3.1.3.3.4 **Invocaciones de Programas**

Las invocaciones de programas no están relacionadas a la ejecución de un sólo programa secuencial en particular como su nombre lo puede sugerir, sino que corresponden a la ejecución de tareas cooperativas en un ambiente multitarea; Las tareas asociadas con una misma invocación de programa son dedicadas a la implementación de una función de control específica.

Para ejecutar una invocación de programa es necesario que esté relacionada con uno o más dominios, los cuales deben contener toda la información requerida por la invocación con excepción de los parámetros de la ejecución que se dan en forma separada. Los segmentos de código y datos para las tareas que componen la invocación están almacenados en áreas de memoria asociadas con los dominios.

Los estados posibles de una indicación de programas y los servicios que provocan las transiciones a esos estados se presentan en la Tabla 6 y Tabla 7 respectivamente.

Estado	Significado
NON-EXISTENT	Estado incluido para indicar que la invocación no ha sido aún creada.
IDLE	La indicación ha sido creada pero aún no esta activa.
RUNNING	La indicación se está ejecutando; la ejecución de una invocación está asociada con cambios en los dominios subordinados causados por ella y no implica necesariamente que los programas asociados se están ejecutando.
STOPPED	La invocación ha terminado su ejecución.
UNRUNABLE	La invocación ya no puede ejecutarse más; este estado precede a la destrucción de la invocación.
STARTING	Estado transitorio que indica que se ha recibido una *Start.indication* pero que no ha llagado el reconocimiento.
STOPPING	Estado transitorio que indica que se ha recibido una *Stop.indication* pero que no se ha dado el reconocimiento de éste mensaje.
RESETTING	Estado transitorio entre STOPPED e IDLE, alcanzado por las invocaciones especificadas como *reusables* en su creación. Una invocación reusable puede ir de STOPPED a IDLE por medio de RESETTING.
RESUMING	La invocación espera cierto tiempo en STOPPED para recibir un *Resume.indication* que puede ser generada por el VMD para continuar con su ejecución.

Tabla 6 Estados posibles de una Invocación a Programas.

Servicio	Descripción
CreateProgramInvocation	Crea una nueva invocación.
DeleteProgramInvocation	Elimina una invocación existente, si ésta es borrable; para ejecutar este servicio se requiere que la invocación se encuentre en estado

	STOPPED o UNRUNNABLE.
Start	Provoca la transición de IDLE a RUNNING de una invocación existente.
Stop	Provoca la transición de RUNNING a STOPPED de una invocación existente. El resultado negativo a este servicio puede ser no-destructivo (la ejecución de Stop no se pudo completar pero la invocación es resumible), o destructiva (cuando la invocación no queda resumible). En el primer caso la invocación se va a RUNNING y en el segundo se va a UNRUNNABLE.
Resume	Provoca la transición de STOPPED a RUNNING a través de RESUMING.
Kill	Causa la transición a UNRUNNABLE de cualquier otro estado (excepto de NON-EXISTENT y P1-P4), indica una terminación anormal de la invocación.
GetProgramInvocationAttributes	Provee al cliente los valores de los atributos de la invocación.

Tabla 7 Servicios asociados a una Indicación a Programas.

3.1.3.3.5 Variables

Son objetos subordinados a un domino de un VMD que pueden ser de dos tipos básicos, Nombrada y Sin Nombre, o de una combinación de ellos. Los objetos relacionados con las variables describen cómo son accesadas, pero ninguno de estos objetos contiene el valor real de la variable; para accesar el valor real el VMD usa dos operaciones *V-PUT* y *V_GET,* que utilizan los objetos MMS para escribir o leer la variable respectivamente.

Es posible que una variable MMS no corresponda a una variable real sino que su valor se obtenga como resultado de un procedimiento (esta variable es sólo de lectura). También es posible que la escritura a una variable corresponda a la activación de una función.

Debido a que es probable tener estructuras complejas de variables, se hace necesario que el acceso a la variable se declare como completo o fallo; para evitar que la lectura de una parte de la variable compleja haya cambiado cuando se termino de leerla, es indispensable forzarla a un acceso de tipo no-interrumpible, característica opcional en MMS y que el fabricante establece si está o no implementada y bajo qué condiciones.

3.1.3.3.6 Semáforos

Es una entidad MMS que puede usarse para el control de acceso a algún objeto subordinado del VMD, así como para sincronizar tareas de aplicación.

EL VMD no puede establecer la relación entre un semáforo específico y el conjunto de recursos controlados, como lo hace un sistema operativo; por lo anterior, se debe forzar al cliente a nivel capa de aplicación para que use el recurso sólo a través del semáforo correspondiente.

Cada semáforo tiene cierto número de tokens que están relacionados directamente con el número de recursos que se pueden accesar a un mismo tiempo. Cada petición para tomar el control del semáforo se registra y si el proceso que requiere el recurso no encuentra un token vacío espera hasta que lo haya.

Existen dos tipos de semáforos definidos por MMS: Semáforos de Tokens y Semáforos de Poll; los tokens de los semáforos del primer tipo son idénticos por lo que las peticiones esperan por cualquiera de ellos, y los del segundo tipo son nombrados, por lo que las peticiones esperan por la liberación de un token en particular.

Las peticiones registradas solicitando un token de semáforo tienen parámetros que influyen en el comportamiento del mismo; estos parámetros se pueden observar en la Tabla 8 y los estados posibles de una petición en la Tabla 9.

Servicio	Descripción
Prioridad	La lista de espera está organizada por clase de prioridad y, en la misma clase, el primero en llegar es el primero en ser atendido. (Las reglas reales del ordenamiento dependen del VMD)
Tiempo máximo de espera	Parámetro opcional que indica el tiempo máximo de espera después del cual la petición es cancelada y el cliente notificado.
Tiempo máximo de control	Parámetro opcional que especifica el tiempo máximo que una petición puede tener el token y se utiliza para prevenir el monopolio del token.
Restablecer.	Parámetro booleano que indica si la petición pierde el token al perderse la AA involucrada.
Abortar al expirar.	Parámetro booleano que indica si se debe abortar la AA cuando expira el tiempo máximo de control del token.

Tabla 8 Parámetros de una petición de Semáforo.

Estado	Significado
NON-EXISTENT	Sólo sirve para indicar que la entrada no existe.
QUEUED	Si el control del semáforo no se le puede dar a la petición entrante entonces se genera un registro con este estado.
OWNER	Este estado se asigna cuando el token se otorga a la petición.
HUNG	Si estando la petición en OWNER ocurre un evento especial se marca como HUNG, donde no se suelta token. Con una petición prioritaria *TakeControl* se regresa la petición a OWNER.

Tabla 9 Estados posibles de una Petición de Semáforo.

Se permiten dos tipos de entradas de semáforo: simple y modificadora. Las entradas simples son generadas por peticiones normales que solicitan el control del semáforo y las modificadoras por peticiones de algún servicio adicionado con un conjunto de parámetros modificadores. Estas últimas peticiones no son atendidas cuando llegan ya que su ejecución depende del estado del semáforo; cuando una entrada modificadora toma el control del semáforo, se ejecuta la petición de servicio y, una vez que la respuesta está lista, se libera el token sin necesidad de alguna acción adicional del cliente. Para las entradas simples se genera una respuesta positiva cuando se otorga el control del token y se requiere de una petición explícita del cliente para liberarlo.

El mecanismo de entradas modificadoras permite al cliente programar un número predefinido de acciones, cuya ejecución es retrasada hasta que el semáforo esté disponible. Esta es la forma directa de sincronizar operaciones entre diferentes tareas.

3.1.4 ESTÁNDARES ASOCIADOS MMS

El modelo provisto por el núcleo MMS es muy general; las características de MMS la hacen muy flexible para usarse en muchos tipos de aplicaciones pero esto representa una limitante para la interoperabilidad de los dispositivos de los diferentes fabricantes ya que una parte de la especificación queda libre. Para solucionar lo anterior surgen los estándares asociados CS (Companion Standards) cuyos objetivos principales son:

- Definir un modelo para la comunicación y para la estructura lógica interna de un VMD.
- Definir atributos específicos de la aplicación y su semántica.
- Definir servicios específicos de la aplicación, o modificación de aquellos definidos por el núcleo MMS.
- Definir nuevas clases de objetos.
- Definir configuraciones estándares de objetos MMS o CS y su semántica para un campo de aplicación específico.
- Definir nuevas clases de coincidencia y bloques constructores relevantes al campo específico.

Debido a que cada área de la manufactura tiene su propia cultura y terminología, los CSs se han estado desarrollando por distintas organizaciones en coordinación con ISO tal y como se mencionó en el punto 3.1.3.

A manera de ejemplo se ha incluido en el siguiente punto el CS para robots ya que es el más utilizado en las celdas de manufactura.

3.1.4.1 Estándar Asociado para Robots

3.1.4.1.1 El modelo robot

Este modelo es la representación de un robot actuando como un servidor MMS, modelando el robot real. La comunicación con este estándar de robot puede llevarse a cabo por 4 configuraciones que se muestran en la Figura 15. En la primera, sólo un usuario de la red puede

ser cliente del robot, así que la implementación de la MMSPM puede simplificarse, debido a que solo una asociación ocupará el robot.

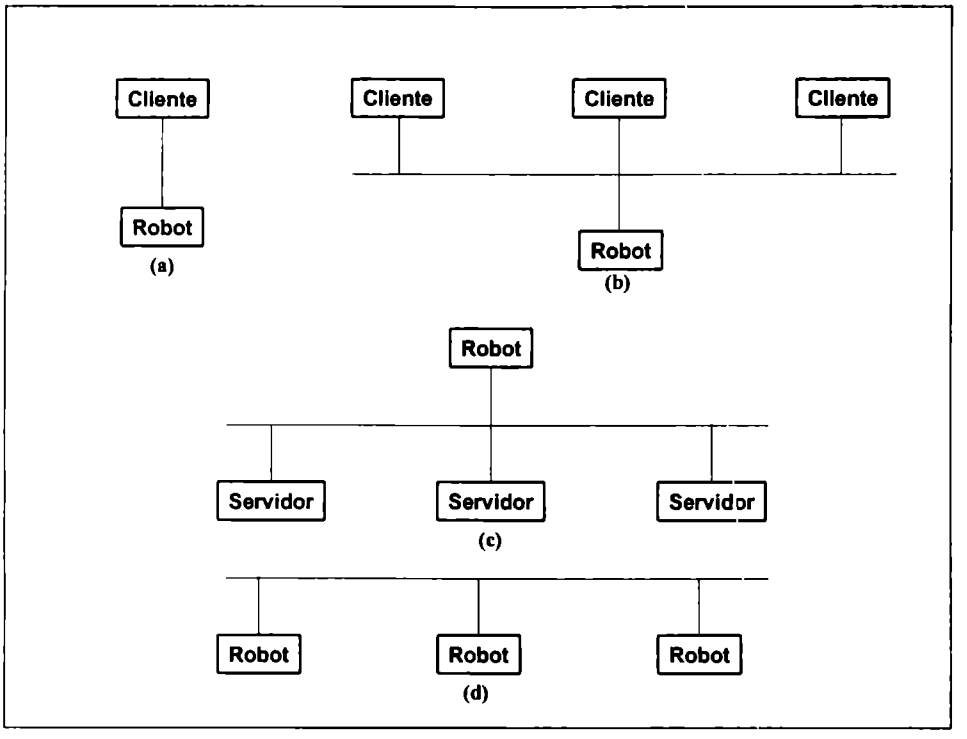


Figura 15 Posibles Configuraciones: a) Cliente Unico (b) Clientes Múltiples (c) Cliente Robot. (d) Robots al mismo nivel.

La segunda está hecha para manejar varios clientes para un mismo robot (ver Figura 15(b)); en este caso, se permiten operaciones paralelas de monitoreo (lecturas), y es necesario coordinar el control de operaciones de los distintos clientes, para que sólo uno de ellos pueda tener el control del robot a la vez a través de semáforos MMS. Adicionalmente a lo anterior, el servidor debe ser capaz de mantener varias AAs abiertas al mismo tiempo.

En la tercera configuración, Figura 15(c), el robot es el cliente mientras que los otros dispositivos (tales como dispositivos de carga/descarga) son los servidores. Esta configuración es posible pero no se considerará ya que el CS solo trata a los robots actuando como servidores.

En la configuración cuatro se muestra un sistema con varios robots actuando en diferentes tiempos como clientes o servidores unos de otros (ver Figura 15(d)). Esta configuración tiene el mismo problema de sincronización que la segunda, debido a que cada robot tiene varios clientes; la única diferencia es que, en este caso, las partes de cliente y servidor deben implementarse en la misma máquina.

Un robot se compone de uno o más brazos, un controlador y un grupo (opcional) de dispositivos separados, los que son independientes de cualquier brazo del sistema.

El brazo del robot puede estar compuesto por varios tipos de dispositivos. El brazo físico se modela como un grupo de uniones y ligas mecánicas, un par de ligas y su unión asociada es un eje del robot. Cada unión es manejada por un actuador. Un controlador se asocia con cada brazo para formar un brazo de robot lógico; de aquí en adelante sólo se hará referencia al brazo lógico del robot. El controlador incluye dos tipos de dispositivos: el servomecanismo y el planeador de ruta. Su relación con el brazo físico se muestra en la Figura 16.

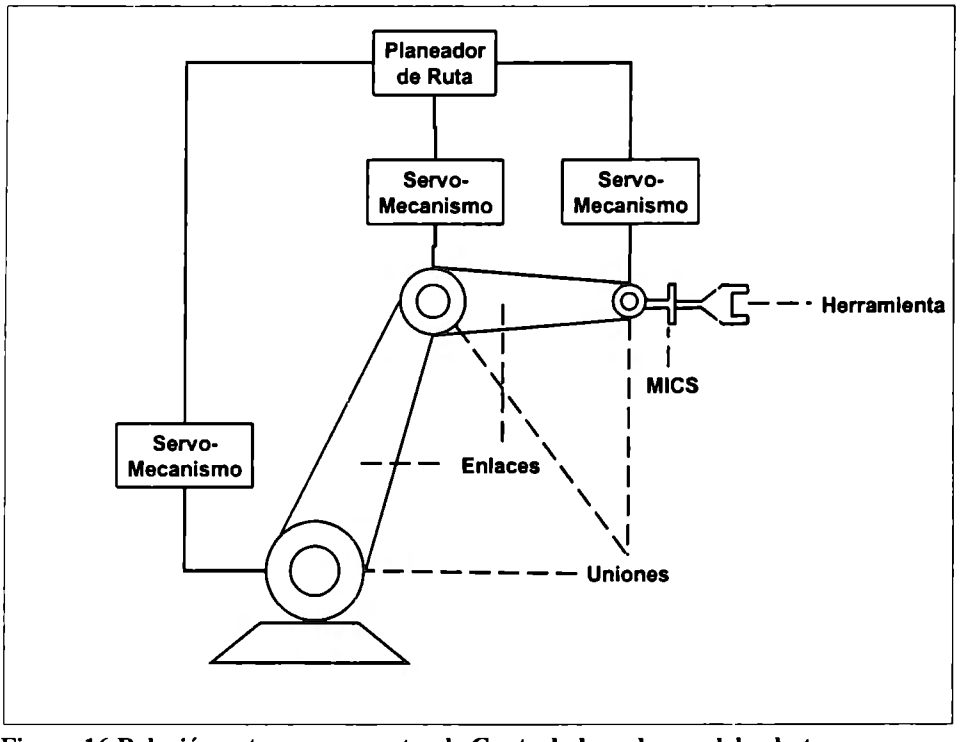


Figura 16 Relación entre componentes de Controlador y brazo del robot.

Existe un servomecanismo asociado con cada unión, el cual controla el movimiento de la ésta mandando un comando al actuador y recibiendo información de los sensores asociados a la unión. Las actividades de control de más alto nivel se llevan a cabo por un planeador de trayectoria, el cual es responsable de traducir una trayectoria final a comandos del servomecanismo. Note que el planeador controla todo movimiento que se requiera del brazo, así que debe obtener los comandos específicos para cada servomecanismo. La velocidad y aceleración del efecto final también son controladas por el planeador, usando la información recibida por los diferentes servomecanismos.

La velocidad y aceleración son indicadas, en el modelo del robot, especificando valores programables de la velocidad y aceleración por omisión y de un factor cambiante que se relaciona directamente con los valores de default. Los parámetros de velocidad y aceleración son entradas para el planeador de trayectoria.

3.1.4.1.2 Mapeo a los objetos MMS

La estructura de un VMD robot sigue estrictamente el modelo definido en la sección anterior. Cada brazo es mapeado a un dominio diferente, con los nombres estándares R_ARM_1, R_ARM_2, etc.

Se requiere agregar más atributos a los definidos en el núcleo MMS para poder describir las uniones y la configuración de los servomecanismos. La lista de los atributos específicos del robot es la siguiente:

Attribute: Local control (TRUE,FALSE)
Attribute: Device power on (TRUE,FALSE)
Attribute: Device calibrated (CALIBRATED, NOTCALIBRATED, CALIBRATING)
Attribute: Number of joints -integer
Attribute: Base world -pose
Attribute: Servomecanismo
  Attribute: MICS-base - pose
  Attribute: List of joints
    Attribute: Joint type (REVOLUTE, PRISMATIC)
    Attribute: Calibrated (CALIBRATED, NOTCALIBRATED, CALIBRATING)
    Attribute: Joint brakes (TRUE,FALSE)
    Constraint: Joint brakes = TRUE
      Attribute: Brakes on (TRUE,FALSE)
    Attribute: Upper bound -floating point
    Attribute: Lower bound -floating point
    Attribute: Joint servo
      Attribute: Actual joint value -floating point
      Attribute: Moving enable
    Attribute: End effector
      Attribute: ID number
      Attribute: Tool descriptor
      Attribute: Tool-MICS - pose
    Attribute: Path planner
      Attribute: User-base - pose
      Attribute: Desired tool-user - pose
      Attribute: Speed factor – floating point
      Attribute: Programmed speed – floating point
      Attribute: Acceleration factor – floating point
      Attribute: Programmed acceleration - floating point

Además de la información general, como el estado de calibración, los atributos describen los parámetros del controlador y el brazo físico agrupados en el atributo servomecanismo. Algunos atributos pueden ser todo un conjunto de parámetros, esto se indica en la lista anterior por la indentación.

Se asocia un semáforo tipo token con un solo token a cada dominio de brazo, para que solo uno pueda llevar a cabo las operaciones que generan las transiciones de estado del VMD o de las invocaciones asociadas con el dominio. Exclusivamente a un usuario se le permiten las peticiones de servicios como *Start, Stop, Resume y Kill.* Para obtener esta característica, CS extiende la semántica de algunos servicios del núcleo MMS, como obligar que la implementación del estándar MMS cheque sí es el propietario del token el generador de la petición antes de servirla.

El semáforo de control asociado con cada dominio de brazo tiene el nombre estándar R_CONTROL. Además de este objeto, el CS robot define otros con nombres estándares que pueden observarse en la Tabla 10.

Dominios

Nombre	Significado
R_ARM	Modelo de brazo
R_CAL	Asociado con el procedimiento de calibración
R_SAFE	Asociado con el equipo de seguridad

Invocaciones a Programas

Nombre	Significado
R_ARM	Manipula el brazo del robot
R_CAL	Ejecuta el procedimiento de Calibración

Tipos

Nombre	Tipo	Significado
R_PIS	Arreglo	(x,y,z) posición en el espacio
R_OS	Arreglo	(a,b,c) rotación en el espacio
R_PSE	Estructura	Posición dada como (R_PIS, R_OS)
R_TEF	Estructura	Descripción de una herramienta

Semáforos

Nombre	Tipo	Significado
R_CTRL	Token	Semáforo de control del robot

Condiciones de Evento

Nombre	Significado
R_RVS	Cambio de estado VMD
R_SIV	Candados de seguridad violados
R_RLC	Control de robot cambiado
R_ARM	Brazo operando

Acciones de Evento

Nombre	Significado
R_STC	Estado de Robot cambiado
R_ARM	Brazo de robot operando

Variables Nombradas

Nombre	Tipo	Significado
R_VPWR	Booleano	Robot encendido
R_VUOM	Booleano	La unidad es el milímetro
R_CAL	Booleano	Recursos calibrados
R_VLOCAL	Booleano	Control local
R_VSAFEC	Booleano	Candados de seguridad violados
R_DPWR	Booleano	Dispositivo encendido (dominio específico)
R_DCAL	Integer8	Dispositivo calibrado (dominio específico)
R-DLOCAL	Booleano	Control local del dispositvo (dominio específico)
R_NJ	Integer8	Numero de uniones (R_ARM específico)
R_JT	Array	Descripción de uniones (R_ARM específico)
R_JCAL	Array	Uniones calibradas (R_ARM específico)
R_JBK	Array	Frenos de unión activados (R_ARM específico)
R_JBD	Array	Límites de unión (R_ARM específico)
R_JAV	Array	Valores actuales de la unión (R_ARM específico)
R_TUB	R_PSE	Transformación de usuario a base (R_ARM específico)
R_TTM	R_PSE	Transformación Tool-a-MICS (R_ARM específico)
R_TBW	R_PSE	Transformación de base a universo (R_ARM específico)
R_ATUP	R_PSE	Punto actual de usuario a herramienta (R_ARM específico)

R_CTUP	R_PSE	Posición deseada de herramienta a usuario (R_ARM específico)
R_AMBP	R_PSE	Transformación de MICS a base (R_ARM específico)
R_SF	Punto flotante	factor de velocidad (R_ARM específico)
R_PSTU	Punto flotante	velocidad programada (R_ARM específico)
R_EEFU	R_EEF	Herramienta en uso

Tabla 10 Lista de Objetos Estándares para un robot.

Además del dominio del brazo R_ARM, se presentan otros dos dominios estándares: uno relativo a la calibración y otro a las operaciones y equipo de seguridad. Aunque se pueden adicionar otros que cuenten con la descripción de sus dispositivos asociados (sí los hay), tal como un equipo de carga y descarga.

3.2 PROTOCOLOS

En esta sección se introducen los protocolos, la importancia de un buen diseño y de una verificación precisa de las funciones que debe desempeñar. Adicionalmente se describen algunas herramientas comunes empleadas en la validación de los protocolos, especialmente Promela, el lenguaje empleado para validar el protocolo propuesto en esta tesis.

3.2.1 DEFINICIÓN

Un protocolo es un conjunto preciso de reglas que definen la interacción entre los elementos de un sistema [20]. Es análogo a un lenguaje, ya que:

- Define un formato conciso para los mensajes válidos (sintaxis).
- Define reglas procedurales para el intercambio de información (gramática).
- Define el vocabulario de mensajes válidos que se intercambian, con su significado (semántica).

Los diseñadores de redes de los años 60's aprendieron que el camino difícil de las secuencias muy improbables de eventos realmente pasan y pueden arruinar los mejores diseños y que redes complejas se pueden paralizar por protocolos incompletos o erróneos [20], por lo que es importante considerar desde el diseño todos los comportamientos posibles de las entidades comunicantes para garantizar un correcto funcionamiento del protocolo. Un requerimiento escondido en los protocolos es que "No solo existen reglas para el intercambio de información, debe haber un acuerdo entre el transmisor y el receptor de estas reglas.", según [20].

La especificación del protocolo consiste de 5 partes:

1. El servicio que debe proveer el protocolo.
2. Las hipótesis sobre el ambiente en el cual se ejecutará el protocolo.
3. El vocabulario de los mensajes usados para implementar el protocolo.
4. La codificación (formato) de cada mensaje en el vocabulario.
5. Las reglas procedurales para garantizar la consistencia en el intercambio de mensajes.

Estos cinco elementos son las tareas más difíciles de diseñar y de verificar, por lo que se sugiere seguir siempre un diseño bien estructurado.

3.2.2 DISEÑO ESTRUCTURADO DE LOS PROTOCOLOS

Las principales reglas de implementación de protocolos son las capas de software de control y las estructuras de datos, las cuales deben guardar los siguientes conceptos.

- **Simplicidad**. Se debe construir un protocolo ligero. Un protocolo correctamente estructurado puede ser construido con un pequeño número de piezas bien diseñadas y bien pensadas. Cada pieza lleva a cabo una función simple. Para entender cómo trabaja un protocolo es suficiente conocer cómo trabaja cada pieza y cómo se intercalan entre sí. Un protocolo ligero es simple, robusto y eficiente.
- **Modularidad**. Jerarquía de funciones. Un protocolo que lleva a cabo funciones complejas puede construirse por piezas que interactúan en una forma simple y bien definida. Cada pieza es un protocolo ligero que puede desarrollarse, verificarse, implementarse y mantenerse separadamente. Las funciones ortogonales no se mezclan, son diseñadas como entidades distintas. Los módulos individuales no deben hacer hipótesis acerca de que el otro esté trabajando y algunas veces, ni siquiera que existan.
- **Protocolos bien formados**.- Para que un protocolo se considere de este tipo debe contar con las siguientes características:
  - No contiene código inalcanzable o no ejecutable.
  - No está inespecificado o incompleto.
  - Está acotado. No puede sobrepasar límites del sistema, como el tamaño de canales.
  - Es auto-estabilizado. Después de un error el protocolo cambia de estado, pero regresa a un estado deseado dentro de un número finito de transiciones.
  - Es auto-adaptable. Se puede adaptar a los cambios en la velocidad de transmisión/recepción.
- **Robustez**.- El protocolo debe estar preparado para reaccionar adecuadamente ante eventos posibles y cualquier secuencia de estados en cualquier condición. Los protocolos sólo deben asumir lo mínimo acerca del ambiente y deben evitar características particulares. La mayor robustez se alcanza cuando no se agregan funciones anticipándose a nuevas condiciones, y se obtiene el diseño mínimo sin suposiciones no esenciales.
- **Consistencia**.- Algunas formas estándares en las que fallan los protocolos son:
  - Deadlocks.- Estados en los que no se puede ejecutar más el protocolo, porque todos los procesos están esperando por condiciones que nunca se cumplirán.

- Livelocks.- Secuencia de ejecución que puede repetirse indefinidamente sin que exista un progreso efectivo.
- Terminaciones Impropias.- El término de la ejecución del protocolo sin satisfacer las condiciones de terminación.

Los principios descritos se pueden lograr si se tienen en consideración las siguientes 10 reglas [20]:

1. Asegurar que el problema está bien definido. Todos los criterios, requerimientos y restricciones deben enumerarse antes de iniciar el diseño.
2. Definir el servicio que llevará a cabo cada nivel de abstracción antes de decidir qué estructuras debe usar para realizar esos servicios.
3. Diseñar las funcionalidades externas antes de las internas. Primero se debe considerar la solución como una caja negra y decidir cómo debe interactuar con el ambiente, para después decidir cómo debe funcionar internamente.
4. Mantener la simplicidad. Los protocolos de moda son más difíciles de implementar y de verificar, pero generalmente son menos eficientes que los simples.
5. No conectar lo que es independiente.
6. No introducir lo inmaterial. No se debe restringir lo que es irrelevante. Un buen diseño es abierto y resuelve una clase de problemas, no un problema en particular.
7. Antes de implementar un diseño, se debe construir un prototipo de alto nivel y verificar que los criterios de corrección se alcanzan.
8. Implementar el diseño, medir su desempeño y, si es necesario, optimizarlo.
9. Checar que la implementación optimizada sea equivalente al diseño que fue verificado.
10. No omitir ninguna de regla de la 1 a la 7.

3.2.3 VALIDACIÓN DE LOS PROTOCOLOS

Análogamente a los programas, los protocolos se necesitan probar, ya que al inicio existe una probabilidad muy alta de que contengan errores; a las pruebas requeridas se les llama validación y son similares a las pruebas aplicadas a los procesos concurrentes [6]. La validación es una operación interna en el sentido de encontrar inconsistencias de la especificación y una operación externa en el sentido de verificar que se cumplan las especificaciones del usuario.

3.2.3.1 Clases de Validación

Es necesario distinguir entre dos clases de validación, la exhaustiva y la no-exhaustiva. A la primera clase pertenecen los algoritmos que pueden ser aplicados a la especificación para hacer un análisis completo del sistema y así determinar sus propiedades. Tales algoritmos pueden ser usados para establecer, por ejemplo, que un sistema está libre de deadlock. Por otro lado, es probable que los algoritmos no-exhaustivos no terminen así que no pueden usarse para hacer aseveraciones acerca del sistema; sin embargo, son muy útiles en la búsqueda de errores. La metodología más significativa de la validación exhaustiva, debido a su poder, facilidad y

simplicidad de automatización, es el análisis de alcanzabilidad, cuya ayuda consiste en calcular el grafo de todos los estados del sistema que pueden ser alcanzados partiendo de un estado inicial; cada estado es analizado para determinar que todas las transiciones que salen de él van a ser usadas y que cumple con las aserciones e invariantes del sistema. Un análisis completo de un protocolo es un cálculo complejo de un grafo de alcanzabilidad que normalmente contiene un número excesivo de estados (este problema es conocido como explosión de estados) [7].

Si el análisis completo no es aplicable por la complejidad del protocolo, se deben usar técnicas de validación no-exhaustiva. Estos métodos son muy efectivos para localizar errores en sistemas complejos que pueden ser analizados en subconjuntos tomados al azar, pero la desventaja radica en que no es completo y, por lo tanto, no es posible demostrar que el sistema está libre de error.

Las dos metodologías mencionadas no pueden ser aplicadas directamente al protocolo, se deben aplicar a una especificación formal de éste. Estas representaciones son conocidas como Modelos de Validación de protocolos.

3.2.4 MODELOS DE VALIDACIÓN DE PROTOCOLOS

Un modelo de validación es una descripción usualmente formal de un sistema que está sujeto a validación [7] y define las interacciones de procesos en un sistema distribuido. No resuelve detalles de implementación [20], no muestra cómo se transmite, codifica o almacena un mensaje. Para facilitar el problema se aísla y se concentra en la parte más difícil: el diseño de un conjunto completo y consistente de reglas que gobiernan las interacciones en un sistema distribuido.

A estos modelos son representados con una técnica de descripción formal para después demostrar sus características usando alguna herramienta de validación. En el siguiente punto se introduce el concepto de las técnicas de descripción formal y más adelante se detalla la herramienta de validación usada para verificar el protocolo diseñado en esta investigación.

3.2.5 TÉCNICAS DE DESCRIPCIÓN FORMAL

Las técnicas de descripción formal FDTs (Formal Description Techniques) son herramientas importantes para el diseño, análisis y especificación de sistemas de precesamiento de información. A través de estas técnicas se pueden producir descripciones de sistemas completas, consistentes, concisas y precisas. Estos es posible solo si la FDT se contiene a sí misma, de tal manera que la descripción en una FDT no necesita referirse a ningún conocimiento informal del sistema que se está describiendo. Un aspecto importante de un sistema formal es que permite el análisis por medio de métodos matemáticos. Una FDT que tiene una base matemática formal puede usarse para probar el nivel de corrección de especificaciones.

Las principales características que deben satisfacer un FDT son:

- Expresiva: una FDT debe ser capaz de definir las especificaciones del protocolo y las definiciones de servicio de las siete capas del modelo OSI.
- Bien definida: una FDT debe ser tener un modelo matemático formal que sea adecuado para el análisis de estas especificaciones y definiciones.
- Bien estructurada: una FDT debe ofrecer los medios para una correcta estructura de la descripción de una especificación o definición de manera que sea útil e intuitiva. Una buena estructura incrementa la asimilación, flexibilidad y reduce el mantenimiento de las descripciones de sistemas.
- Abstracta: hay dos aspectos de abstracción que una FDT debe ofrecer:
  - Una FDT debe ser completamente independiente de los métodos de implementación, de tal manera que la técnica no debe restringir en absoluto a los implementadores.
  - Una FDT debe ofrecer los medios de abstracción para los detalles irrelevantes con respecto al contexto en cualquier punto de la descripción.

Algunas técnicas de descripción formal son:

- **Lotos.** (Language Of Temporal Ordering Specification) es una técnica de descripción para la especificación formal de sistemas abiertos distribuidos, y en particular para aquellos relacionados con la arquitectura de red de OSI [24]. Al contrario de lo que su nombre puede sugerir, no está relacionada con la lógica temporal sin que está basada en métodos algebraicos de procesos, que fueron introducidos por los trabajos de Milner en CCS (Calculus of Communicating Systems) [25]; Algunos conceptos y notación fueron introducidos más tarde inspirados por el modelo CSP (Communicating Sequential Processes).
- **SDL.** (Specification and Description Language) estandarizada por ITU-T (International Telecommunications Union), es un lenguaje de descripción de propósito general para los sistemas de comunicación. La base para la descripción del comportamiento son máquinas de estados representadas por procesos. La comunicación está representada por señales y puede llevarse a cabo entre procesos o entre procesos y ambiente del modelo del sistema [26]. Muchas herramientas se han desarrollado para SDL: Open Site, SDL Integrated Tool Environment (aún no comercial), SDT (la herramienta de diseño SDL para Telelogic).
- **Estelle.** Es un lenguaje de especificación basado en máquinas de estados finitas estandarizado por ISO y CCITT (International Telephone and Telegraph Consultive Committee) ahora ITU-T. Existen dos herramientas en el mercado EDT (Estelle Development Toolset) y petdinngo.
- **SPIN.** Es un sistema de verificación y PROMELA es su lenguaje, el cual está basado en el modelo de Hoare CSP. Es un software gratuito de propósito educacional el cual fue seleccionado para la validación del protocolo desarrollado en el presente trabajo.
- **Z.** La especificación formal Z (pronunciada "zed") está basada en la teoría de conjuntos y la lógica de predicados de primer orden y esta siendo usada por la industria para el desarrollo de procesos de software y hardware.
- **Petri Net.** Es un lenguaje formal y gráfico que es muy apropiado para el modelado de sistemas con concurrencia [25].

3.2.6 PROMELA Y SPIN

Promela, Process Meta Language, es un lenguaje no determinístico usado para construir modelos formales de protocolos a validar; está inspirado en la notación de lenguaje de comandos de Dijkstra y en el lenguaje CSP de Hoare, adicionado con otras construcciones más poderosas Contiene las primitivas para especificar paso de mensajes asíncronos a través de canales. con un número arbitrario de parámetros. También permite la especificación de sistemas con paso de mensajes síncronos (rendez-vous) y sistemas que mezclan ambos tipos de comunicación.

El lenguaje puede modelar sistemas de crecimiento y/o contracción dinámica; se pueden crear y destruir nuevos procesos y canales en ejecución, los identificadores de los nuevos canales pueden pasarse de un proceso a otro a través de los canales existentes.

Las propiedades de corrección pueden especificarse como invariantes de sistema o proceso (usando aserciones), o como requerimientos temporales lineales LTL (Linear Temporal Logic requeriments), ya sea directamente en la sintaxis de LTL o indirectamente como un autómata Büchi (expresado en la sintaxis Promela como requerimientos tipo *Never*).

Por otro lado, SPIN es un software para verificar formalmente sistemas distribuidos y protocolos usando Promela como lenguaje de entrada; este software fue desarrollado en los laboratorios Bell por el grupo de métodos formales y verificación, y se ha usado para detectar errores de diseño en sistemas distribuidos, tales como sistemas operativos, protocolos de comunicaciones, algoritmos concurrentes, etc. Spin es la herramienta usada para validar el protocolo propuesto en esta tesis.

SPIN checa la consistencia de la especificación, reporta deadlocks, recepciones no especificadas, hipótesis no garantizadas acerca de las velocidades relativas de los procesos. En [28] se encuentran las instrucciones para la obtención e instalación de una versión gratuita de este software y de XSPIN, el cual es una interfaz gráfica para manejar SPIN (escrita en Tcl/Tk).

SPIN puede usarse en tres modos básicos:

- Como un simulador, permitiendo generar prototipos gracias a simulaciones interactivas, guiadas o aleatorias.
- Como un analizador exhaustivo de estados, es capaz de probar rigurosamente la validez los requerimientos de corrección especificados por el usuario (opcionalmente usando la teoría de reducción de orden parcial para optimizar la búsqueda).
- Como un analizador binario de estados, que puede validar protocolos muy extensos con una cobertura máxima.

En los siguientes puntos se detallan conceptos importantes de Promela útiles para la validación hecha al protocolo propuesto.

3.2.6.1 Sintaxis de Promela

La sintaxis de Promela es muy parecida a la del lenguaje C, un resumen de ésta se presenta en la Tabla 11.

Convenciones Léxicas

Hay 5 clases de Tokens: Identificadores, Palabras Reservadas, Constantes, Operadores y Separadores de enunciados.

Identificadores

Letra, coma o guion menor (_) seguido por cero o más letras, dígitos, comas o guiones bajos.

Palabras Reservadas

active	assert	atomic	bit	Bool	Break
byte	Chan	d_step	Dproctype	Do	Else
empty	Enabled	fi	full	Goto	Hidden
	Init	int	len	Mytype	Nempty
	Nfull	od	of	Pcvalue	Printf
priority	Proctype	provided	run	Short	Skip
timeout	Typedef	unless	unsigned	Xr	Xs

Constantes

Secuencia de dígitos representados por un entero decimal. Hay dos formas de definirlas, la segunda se menciona en el siguiente apartado.

define NAME 5

Constantes Simbólicas

Declaradas por una de las dos siguientes formas:

Mtype = {OK, READY, ACK};

Mtype = Status + 3;

Solo se permite una definición de mtype en el modelo y puede contener hasta 256 constantes. La ventaja de las constantes simbólicas radica en que son reconocidas por Spin y durante las simulaciones son usadas como los nombres simbólicos y no por los valores que representan.

Operadores

-	*		%						
!	&		&&						
		Len()	empty()	nempty()	nfull()	Full()			
run	eval()	enabled()	pc_value()						

Separadores

Blancos, tabuladores, nueva línea y comentarios son separadores de tokens. Los separadores de enunciados son ";" y "->"

Etiquetas

Identificador seguido por dos puntos (:). Se puede etiquetar cualquier enunciado. Las etiquetas con prefijo "end", "progress" y "accept" tienen un significado especial en la validación del modelo (Refiere se a puntos 0 y 3.2.6.6.4)

Comentarios

Inicia con "/*" y termina con "*/". Los comentarios no pueden anidarse.

Tabla 11 Generalidades en la Sintaxis Promela.

Una facilidad en la sintaxis Promela, muy útil para el análisis de un modelo, es que se permite hacer referencia remota a variables y etiquetas de un proceso para saber el valor que contienen las variables o para saber si el proceso está en el estado marcado por la etiqueta. Por ejemplo, en la Figura 17 se muestra el código de un proceso escritor, el cuál es instanciado dos veces en el modelo, y un proceso monitor, cuyo objetivo evaluar la condición invariante del sistema (tratada en el punto 3.2.6.6.2) de que no estén ambos procesos usando el recurso compartido a un mismo tiempo. La zona crítica esta marcada con la etiqueta "ZonaCrit" y se hace referencia remota desde el proceso monitor, la instancia del proceso escritor queda referida por el índice, 0 y 1, que esta dentro del los paréntesis "[]".

```
Proctype escritor()
{
....

  do
  ::(token==true)-> goto ZonaCrit
  ::(token!=true)->skip
  od

...
ZonaCrit:
/*Manejo del recurso compartido*/
...
}

Proctype monitor()
{
assert(!escritor[0]@ZonaCrit || !escritor[1]@ZonaCrit)
}
```

Figura 17 Ejemplo de la Referencia Remota permitida en Promela.

3.2.6.2 Procesos, Canales y Variables

En Promela se describen reglas procedurales como programas formales para un modelo abstracto de un sistema distribuido. Este modelo está compuesto de diferentes objetos.

Definimos un modelo de validación en función de tres tipos específicos de objetos: Procesos, Canales de Mensajes y Variables de Estado, que para propósitos de análisis, cada uno de ellos puede traducirse en una máquina de estados finita. Por definición todos los procesos son objetos globales, pero las variables y los canales pueden ser globales o locales a un proceso. Los procesos especifican el comportamiento, los canales y las variables definen el ambiente en el que corren los procesos.

Un ejemplo de declaración de mensaje es:

```
Chan Transfer = [2] of {mtype, bit, short, chan},
        Device[3] = [0] of byte,
        Channel
```

Aquí, el canal "Transfer" puede almacenar hasta dos mensajes en todo momento; el tipo de mensaje está indicado entre las llaves "{}" (en este caso cada mensaje consiste de cuatro partes). "Device" es un arreglo de canales; cada canal es síncrono, es decir, los envíos y transmisiones deben estar sincronizadas ya que no se pueden almacenar mensajes. Finalmente, Channel es un canal no inicializado que puede ser usado hasta que se le haya asignado un canal adecuadamente inicializado. Se debe notar que un canal puede ser parte de un mensaje.

Una declaración básica de un proceso tiene la siguiente forma:

```
Proctype pname ( chan In, Out; byte id )
{ sentencias }
```

y cada proceso debe ser instanciado por una operación *run*.

```
run pname (Transfer, Device[0], 0).
```

A cada instancia de proceso se le asigna un número positivo único (pid), se mantiene activa hasta que el cuerpo del proceso termina completamente (si termina) y tiene asociada una prioridad de ejecución, ya sea de forma explícita (usando la palabra reservada "priority" en el enunciado "run") o de forma implícita (en cuyo caso el valor es 1)

Existe un proceso especial, referido como "proceso Init", usado generalmente para la inicialización del sistema y que tiene la siguiente estructura:

```
Init {
  sentencias de inicialización de variables y/o para instanciar los procesos del sistema
}
```

3.2.6.3 Ejecutabilidad de las instrucciones

En Promela no hay diferencia entre condiciones e instrucciones; la ejecución de una instrucción está condicionada a su "ejecutabilidad", un concepto básico de sincronización en Promela; la ejecutabilidad es la capacidad que tiene la instrucción para ser ejecutada o no, de acuerdo al valor de las variables y/o el contenido de los canales involucrados en la instrucción, y por está propiedad las sentencias pueden ser ejecutables o bloqueantes; una instrucción se vuelve bloqueante cuando, por ejemplo, esta supeditada a que una variable o el contenido de un canal

tenga un valor distinto al que tiene, y el proceso que ejecuta esa instrucción espera hasta se vuelva ejecutable, es decir que la variable o canal contenga el valor esperado.

3.2.6.4 Ejemplo: Semáforo de Dijkstra

En la Figura 18 se muestra la implementación del semáforo de Dijkstra, usando una comunicación rendevouz. Al inicio del código se definen las variables "p" y "v" de forma global (usando la directiva #define) y el canal "sema", cuya longitud es cero y cuyo dato contenido es de tipo bit. La longitud cero tiene como fin forzar al escritor del canal a que espere a que haya un receptor del mensaje listo. Se declara un proceso "dijkstra", cuya función es emular el semáforo, el cuál garantiza que solo un proceso será habilitado para el uso de la región crítica resguardada por él. El proceso "user", es el que solicita el token al semáforo (esperando por "p" en la línea sema ? p) para poder entrar a la zona crítica y libera el token después de haber concluido su tarea, regresando el control al semáforo mediante el envío de "v".

```
#define p    0
#define v    1
chan sema = [0] of {bit};
proctype dijkstra ( )
{ end: do
   ::sema ! p -> sema ? v
  od
}
proctype user( )
{  sema ? p;
   /*zona critica*/
   sema ! v
   /*zona no critica*/
}

init
{ atomic {
    run dijkstra( );
    run user; run user; run user;
   }
}
```

Figura 18 Ejemplo: Implementación del Semáforo de Dijkstra

Finalmente, el proceso "init" instancia tres procesos de tipo "user" y un proceso "dijkstra" de manera armónica, es decir, no puede ejecutarse ninguna sentencia en el sistema global durante hasta que no se hayan lanzado los 4 procesos. Los tres procesos "user" son los que compiten por la zona crítica.

3.2.6.5 El Comportamiento del Modelo

El tipo de exigencia sobre el comportamiento de un modelo puede ser: Inevitable o Imposible. Es inevitable cuando se asegura que la exigencia siempre se cumplirá en todas las secuencias posibles del modelo, se dice imposible si se asegura que ninguna secuencia la contiene.

Debido a que el número de comportamientos de cualquier modelo en Promela es finito, la exigencia de un tipo implica una exigencia del tipo contrario. Para demostrar que un comportamiento es inevitable, debemos demostrar que todos los comportamientos alternos son imposibles.

Todos los criterios de corrección que se expresan en Promela definen comportamientos que son exigidos como imposibles. Por ejemplo, si una afirmación de corrección establece que una condición es invariablemente verdadera, la exigencia de corrección establece que es imposible violar la afirmación independientemente del comportamiento del sistema.

El comportamiento de un modelo en Promela está definido por el conjunto de todas las secuencias posibles de ejecución. Una secuencia de ejecución es un conjunto de estados finito y ordenado. Un estado está definido por todos los valores de variables locales y globales. Se puede llegar a un estado a través de la asignación apropiada de valores a variables, a puntos de control de flujo y a canales.

Un conjunto ordenado y finito de estados en el modelo M, es una secuencia de ejecución válida si satisface que:

1. El primer estado de la secuencia es el estado inicial del modelo M, con todas las variables inicializadas en cero, todos los canales vacíos, con un solo proceso inicial activo y que este proceso se encuentre en su estado inicial.
2. Si M se coloca en el estado i, hay al menos una instrucción ejecutable que lo pueda llevar al estado i+1.

Existen dos tipos especiales de secuencias:

- **Terminal.** Es una secuencia de ejecución donde cada estado se presenta sólo una vez en la secuencia y M no tiene instrucciones ejecutables al llegar al último estado de la secuencia.
- **Cíclica**. Es una secuencia de ejecución donde todos los estados, excepto el último, son distintos, siendo el último estado de la secuencia igual a uno de los estados anteriores. Las secuencias de este tipo definen ejecuciones potencialmente infinitas

3.2.6.6 Requerimientos de Corrección

Los tipos de requerimientos de corrección pueden ser diferentes para las secuencias terminales y para las cíclicas. Un requerimiento importante para una secuencia terminal es, por ejemplo, la ausencia de deadlocks, aunque no todas carecen de ellos y por ello se tiene que expresar qué

propiedades de los estado finales deben hacer que la secuencia sea aceptable como una secuencia terminal sin deadlock. Para secuencias cíclicas, se debe ser capaz de expresar condiciones generales como la ausencia de livelocks.

Los requerimientos de corrección en los modelos Promela pueden construirse con proposiciones simples, donde una proposición es una condición booleana de un estado del sistema. Las proposiciones definen implícitamente un etiquetado de los estados; en cualquier estado dado una proposición es verdadera o es falsa. Los criterios de corrección pueden expresarse en términos de los estados definiendo explícitamente en cuales de ellos se requiere mantener verdadera la proposición, a esto se le conoce como propiedades de los estados. Si se va a usar más de una proposición, se puede necesitar que los requerimientos de corrección sean expresados como un orden temporal de proposiciones. El formalismo para soportar esta característica del lenguaje es conocido como requerimiento temporal y es explicado más adelante en este mismo punto.

Existen varias propiedades de los protocolos que se deberían probar, sin embargo, el problema es muy complejo por lo que se deben seleccionar un grupo de ellas [20]. A continuación se mencionan los criterios de corrección que pueden ser expresados en Promela.

3.2.6.6.1 Aserciones

Los criterios de corrección pueden expresarse como condiciones booleanas que deben satisfacerse cuando el proceso alcance un estado dado. Para verificar que este criterio se cumpla, se utiliza la instrucción assert(condición), que siempre es ejecutable y puede aparecer en cualquier lugar del modelo Promela. La condición puede ser cualquier expresión booleana arbitraria; si es verdadera al evaluarse el enunciado entonces no existe efecto alguno, pero si es falsa la validez del enunciado es violada. Spin reporta automáticamente este error con solo incluir este tipo de enunciados en el modelo.

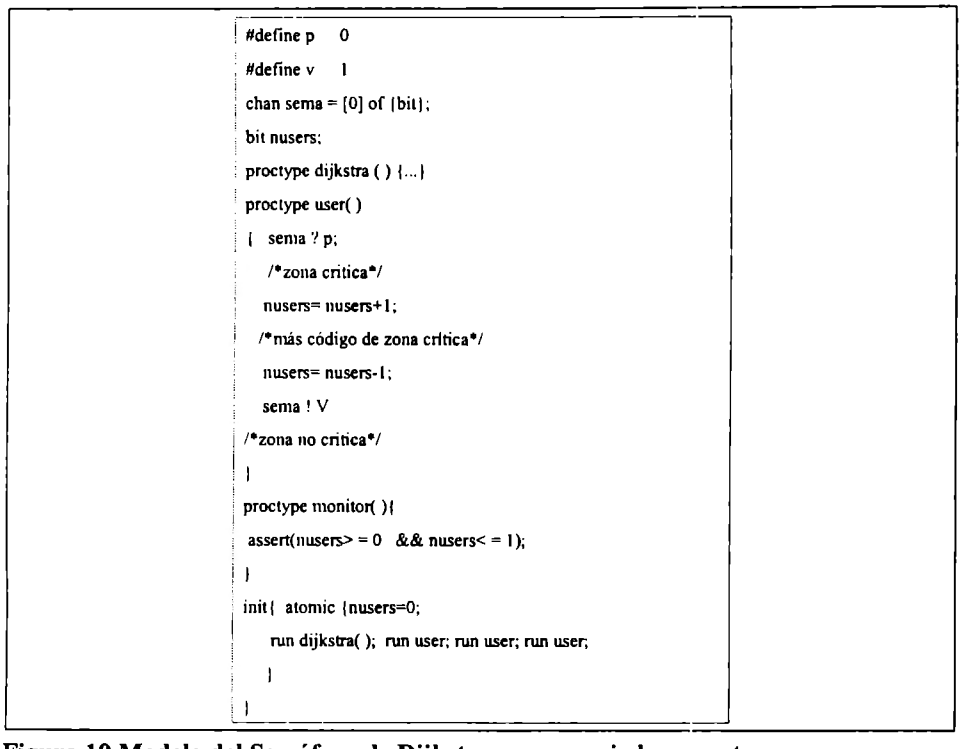
Para ejemplificar el uso de las aserciones se ha modificado el código de la implementación del semáforo Dijkstra mostrado en la Figura 18. Se ha agregado la variable global "nusers" que es incrementada por el proceso "user" cada vez que entra a la zona crítica y decrementada cada vez que sale de ella. Si el modelo cumple con sus expectativas, "nusuers" siempre oscilara entre 0 y 1. Se puede asegurar que en la zona crítica "nusuers" vale 1, después de que el proceso la ha incrementado (ver Figura 19).

Al correr el validador SPIN con este modelo, él buscará todas las secuencias posibles y si encuentra al menos una de ellas en que la condición no se cumpla lo reportará inmediatamente.

3.2.6.6.2 Invariantes del Sistema

Una aplicación más general del enunciado assert es formalizar invariantes del sistema, es decir, condiciones booleanas que, si son verdaderas en el estado inicial del sistema, deben permanecer

verdaderas en todo estado alcanzable del sistema. Para expresar esto en Promela, es suficiente colocar la invariante del sistema en un proceso monitor separado.



```
#define p   0
#define v   1
chan sema = [0] of {bit};
bit nusers;
proctype dijkstra ( ) {...}
proctype user( )
{  sema ? p;
   /*zona critica*/
   nusers= nusers+1;
  /*más código de zona crítica*/
   nusers= nusers-1;
   sema ! V
/*zona no critica*/
}
proctype monitor( ){
 assert(nusers> = 0  && nusers< = 1);
}
init{  atomic {nusers=0;
     run dijkstra( );  run user; run user; run user;
    }
}
```

Figura 19 Modelo del Semáforo de Dijkstra con enunciados assert.

proctype monitor() { assert(invariante)}

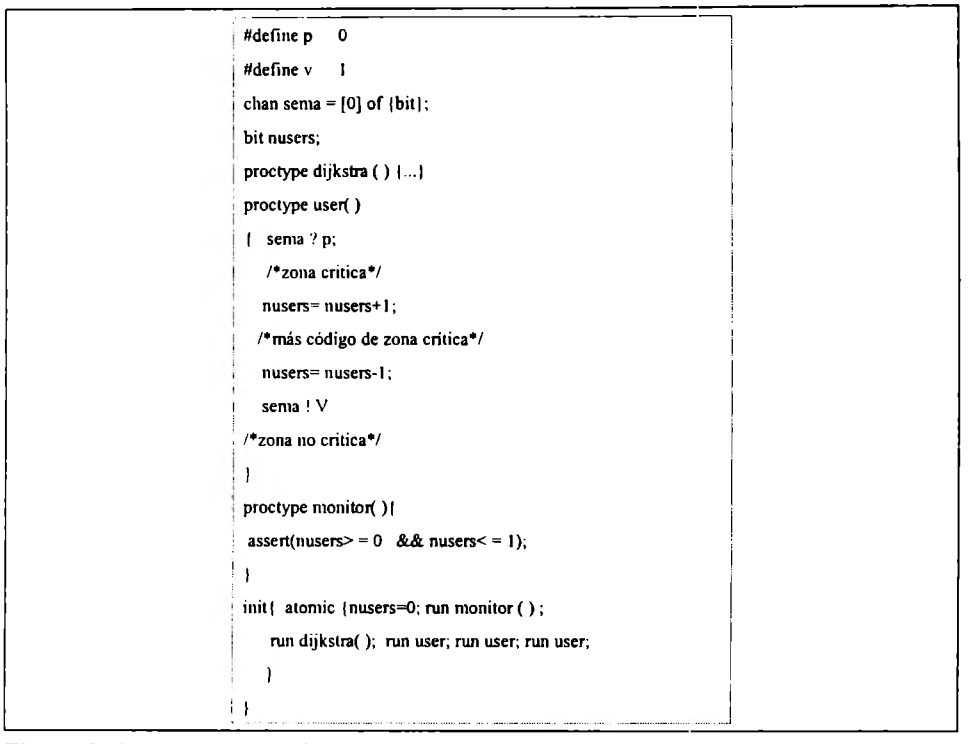
El nombre "monitor" no tiene relevancia alguna.

Como ejemplo, se toma nuevamente el modelo mencionado en los dos puntos anteriores donde la invariante del sistema puede expresarse como: "En todo estado del sistema nusers debe tener un valor de cero o de uno". El nuevo modelo queda como en la Figura 20.

SPIN se encargará de evaluar la sentencia assert del monitor una vez después de cada estado para toda secuencia posible del modelo.

3.2.6.6.3 Estados Finales Inválidos

En un sistema de estados finito, todas las secuencias de ejecución terminan después de un número finito de transiciones o se ciclan a un estado previamente visitado; no todas las secuencias cíclicas son necesariamente deadlocks. Para definir que es un deadlock en Promela, se debe distinguir primero los estados finales aceptable y los estados finales no esperados ó inválidos. Los estados finales no esperados no sólo incluirán deadlocks sino también muchos estados de error que pueden ser resultado de una especificación de protocolo incompleta lógicamente.



```
#define p    0
#define v    1
chan sema = [0] of {bit};
bit nusers;
proctype dijkstra ( ) {...}
proctype user( )
{  sema ? p;
   /*zona critica*/
   nusers= nusers+1;
  /*más código de zona critica*/
   nusers= nusers-1;
   sema ! V
/*zona no critica*/
}
proctype monitor( ){
 assert(nusers> = 0  && nusers< = 1);
}
init{  atomic {nusers=0; run monitor ( ) ;
    run dijkstra( );  run user; run user; run user;
   }
}
```

Figura 20 Modelo del Semáforo Dijkstra incluyendo una invariante del sistema.

El estado final en una secuencia de ejecución terminante debe satisfacer los siguientes dos criterios para considerarlos un estado final aceptable:

- Todo proceso que fue iniciado ha finalizado.
- Todos los canales de mensajes están vacíos.

Pero no todos los procesos terminan necesariamente; puede ser perfectamente válido que un proceso servidor permanezca vivo después de que los procesos del usuario ya han terminado. Se debe identificar a éstos como estados finales aceptables; en Promela esto se lleva a cabo anteponiendo a la sentencia terminal una etiqueta con prefijo "end". Puede existir más de un estado final aceptable en un mismo proceso pero todas las etiquetas deben ser únicas, por ello para el validador es suficiente que tales etiquetas inicien con las letras "end".

Con estas nuevas definiciones se puede modificar el primer criterio para un estado final aceptable:

- Todo proceso que fue iniciado ha finalizado o ha alcanzado un estado etiquetado como un estado final aceptable.

Cualquier estado final en una secuencia de ejecución que no cumpla los dos criterios mencionados es considerado como un estado final inválido.

Un criterio de corrección implícito en los modelos de validación es que no debe contener estados finales inválidos.

Retomando el modelo del semáforo de Dijkstra, un estado final aceptable es el del servidor del semáforo "dijkstra", ya que tal vez todos los procesos "users" terminen pero "dijkstra" puede quedarse ciclado. Para indicarle a Spin que este estado final es aceptable se le agrega una etiqueta "end" tal y como se muestra en el código de la Figura 18.

3.2.6.6.4 Ciclos Erróneos

En Promela se pueden expresar dos propiedades para las secuencias cíclicas, correspondientes a los dos tipos estándares de requerimientos de corrección. La primera propiedad especifica que:

- No existen comportamientos infinitos en estados no marcados.

Es decir, el sistema no puede ciclarse indefinidamente en estados no marcados. Los estados marcados son llamados estados de progreso, y las secuencias de ejecución que violan este criterio son llamadas ciclos sin progreso. La segunda propiedad es opuesta a la primera:

- No hay comportamientos finitos que incluyan estados marcados.

Las secuencias de ejecución que violan este requerimiento son llamadas ciclos de aceptación o livelocks.

Ciclos sin Progreso

Para asegurar la ausencia de ciclos sin progreso, debemos definir en Promela estados que denoten progreso. Los estados progreso se definen análogamente como los estados finales, una etiqueta con prefijo "progress" marca a un estado que debe ser ejecutado por el protocolo para que haya un progreso.

Un estado requerido para asegurar que el progreso del protocolo en la modelo del semáforo Dijkstra es el de la recuperación del token por parte del proceso controlador del semáforo; esto es mostrado en la Figura 21.

Ciclos de aceptación

Suponga que se requiere expresar lo opuesto a una condición de progreso, es decir, que se requiere formalizar que un comportamiento no puede pasar eternamente; para ello se usan etiquetas con prefijo "accept". Al checar ciclos de aceptación, el validador verifica que no exista una ejecución que visite indefinidamente un estado de aceptación.

En principio, se hace un mejor uso del estado aceptable si se utiliza para expresar comportamientos completos que se consideran imposibles, más que solamente la ausencia de un estado designado en todos los ciclos. Esto se hace precisamente usando las etiquetas para definir estados aceptables de un autómata especial que modela comportamientos erróneos. Estos autómatas expresan requerimientos temporales generales.

```
#define p    0
#define v    1
chan sema = [0] of {bit};
proctype dijkstra ( )
{do
            ::sema ! p ->
progress: sema ? v
  od
}
proctype user( )
{  sema ? p;
   /*zona critica*/
   sema ! v   /*zona no critica*/
}

init
{ atomic {
    run dijkstra( );
    run user; run user; run user;
    }
}
```

Figura 21 Ejemplo del uso de la etiqueta progress para validar el progreso en el modelo del Semáforo Dijkstra.

3.2.6.6.5 Requerimientos Temporales

Con las herramientas mencionadas en los puntos anteriores se pueden expresar criterios de corrección más potentes, simplemente utilizando las definiciones de procesos proctype. Suponga que se necesita expresar el siguiente requerimiento temporal:

"Todo estado donde la propiedad P es verdadera es seguido por un estado donde la propiedad Q es verdadera".

Hay dos interpretaciones para la palabra "seguido" dependiendo si los estados son "inmediatamente" o "eventualmente seguidos". Es básico en la semántica Promela que no debe hacerse asunciones relativas al tiempo del procesamiento así que la única interpretación legitima es "eventualmente seguido" uno del otro (que incluye el "inmediatamente seguido" como un caso particular). Esto genera un problema para expresar los otros tipos de propiedades, requiriéndose otro tipo de primitiva de validación.

La frase mencionada arriba se puede simbolizar como:

P -> Q

Debido a que todos los criterios de corrección están basados en las propiedades que se consideran imposibles, los requerimientos temporales deben ser expresados como ordenamientos de propiedades que deben ser imposibles.

Los requerimientos temporales son definidos en secuencias de ejecución completas. Aunque sí un prefijo de la secuencia es irrelevante, debe ser representado como una secuencia trivial de proposiciones.

De acuerdo a lo anterior podemos representar el requerimiento como:

never(do ::skip od ->P -> !Q)

es imposible que, independientemente de la secuencia inicial de eventos, para un estado donde la propiedad P se cumpla sea seguido por otro estado donde la propiedad Q no se cumpla. En la sintaxis Promela, lo anterior se expresaría de la siguiente forma:

```
never {do
        ::p -> break;
        ::skip;
        od;
accept:
        do
        :: !q
        od
        }
```

Los requerimientos temporales pueden ser enriquecidos con aserciones, etiquetas de progreso, de accept para poder obtener más tipos de errores. El requerimiento never puede expresarse como una máquina especial de estados finita que se cicla en un estado aceptable si el comportamiento no deseado es detectado.

El requerimiento en sí mismo es una máquina de estados finita, con una proposición definida en cada estado. Para cada transición en el modelo de validación, la máquina de requerimiento debe cambiar de estado y moverse a la siguiente proposición. Para completar el requerimiento temporal, en cada estado de la secuencia de estados, la proposición correspondiente al estado de la máquina de requerimiento debe ser verdadera.

Los requerimientos never en combinación con las etiquetas accept también pueden expresar la ausencia de ciclos sin progreso. Estos requerimientos son más genéricos que las etiquetas de progreso "progress". Por ejemplo, en la Figura 22 se muestra el mismo requerimiento de la

Figura 21 pero utilizando la estructura *never*; eventualmente después de que el proceso dijkstra otorga el token, y el valor de nusuers es actualizado a "1", éste es recuperado (nusers regresa a "0").

```
#define p    0
#define v    1
chan sema = [0] of {bit};
bit nusers;
proctype dijkstra ( ) {
do
                ::sema!v ->
libera:         sema?p;
od}
proctype user( ){   sema ?? p;    /*zona critica*/
   nusers= nusers+1;   /*más código de zona crítica*/
   nusers= nusers-1;    sema ! V; /*zona no critica*/
   ...}
never {do     ::(nusuers==1)-> break;
              ::(nusuers!=1);
od
do
accept:       ::(dijkstra@libera);
od}
init{ atomic {nusers=0; run dijkstra( ); run user(); run user();
run user();    }}
```

Figura 22 Ejemplo del uso de never para validar el progreso en el modelo del Semáforo Dijkstra.

3.2.6.7 Spin en el análisis del modelo

El proceso básico del validador es recursivo (aunque el algoritmo a validar no lo sea) y empieza en el estado inicial del sistema; en cada estado del sistema se procesan todas las verificaciones habilitadas al iniciar el análisis. Si se habilita el requerimiento *never* en la lista de verificaciones ésta se convierte en una lista de pares de acciones, una acción del requerimiento *never* junto con una del proceso activo. El proceso de recursión regresa, una vez de que ha recorrido toda una secuencia de estados, a cualquier estado visitado en el cual no esté habilitada la acción (par) o a alguno que no hayan sido seleccionado previamente. Así, el proceso básico es una generación en profundidad primero de todos los estados del sistema. Realmente el proceso es más complejo que un barrido simple de los estados posibles, ya que en cada estado se debe verificar la ausencia de violaciones a las correcciones.

3.2.6.7.1 Requerimientos de memoria

El tamaño del sistema que puede ser analizado por el validador SPIN está determinado directamente por la cantidad de memoria RAM disponible, debido a que ésta se requiere por las siguientes razones:

- Cuando se checan las propiedades de los estados se necesitan almacenar los estados visitados para poder decidir en cual de ellos continuará el análisis cada vez que la profundidad de una secuencia ha llegado al máximo, además de que estos estados son accesados aleatoriamente y no hay otra forma de saber cuales ya han sido visitados más que almacenándolos.
- En cada secuencia parcial se requiere guardar el cálculo actual del algoritmo que debe ser extendido en términos de la profundidad de la secuencia, lo que genera un crecimiento de la "pila".

Además, si en el análisis se tiene que verificar la ausencia (progreso) o presencia (aceptación) de secuencias cíclicas, el número de estados almacenados y el tamaño de la pila son de al menos el doble comparados al caso de la verificación sencilla de las propiedades de los estados. Nótese que al agregar un requerimiento no determinístico *never* al sistema se incrementa el espacio de estados por lo que es posible que el requerimiento en memoria sea más del doble cuando se busca un comportamiento que alcance el requerimiento *never* que contiene una etiqueta un estado de aceptación

3.2.6.7.2 Enunciados con la propiedad Atomic

Un enunciado *Atomic* o *d_step* causa que los enunciados dentro de su alcance se ejecuten como si se tratara de uno solo; al disminuir el número de estados en el sistema se reduce notablemente la memoria requerida. Esto es posible ya que se asegura que durante la ejecución de los enunciados lo cual puede hacerse debido a que ninguna instancia ha cambiado el estado global del sistema

3.2.6.7.3 Reducción de Orden Parcial

La reducción de orden parcial es una técnica de búsqueda optimizada que decrementa la memoria y tiempo necesario para el análisis en un orden de magnitud en la mayoría de los casos y algunas veces mucho más; nunca genera un incremento de memoria y, en el peor de los casos, sólo incrementa ligeramente el tiempo de análisis. Esta técnica es invocada al tiempo de compilación (-DREDUCE o la opción correspondiente en Xspin) y se garantiza que los resultados del análisis son los mismos que si no se activa la reducción. En forma general, la reducción de orden parcial trabaja durante el análisis decidiendo en cada estado donde existe no-determinismo, si la propiedad que esta siendo verificada depende del orden en que se ejecutan las sentencias de los procesos. Si es independiente de este orden, el validador establece uno y omite las ejecuciones en que estos pasos tomen diferente sucesión. El efecto de la aplicación de esta técnica es el mismo que si pequeñas secciones de código Promela se agruparan en sentencias *Atomic*, pero en forma dinámica.

La implementación de este método en Spin está estrechamente ligada con las variables globales compartidas y los canales de longitud mayor a cero. Si un sistema no tiene variables compartidas y sólo usa canales de longitud cero, no habrá disminución ni de memoria ni de tiempo.

El único requerimiento para usar la reducción de orden parcial es de que los requerimientos *never* sean oscilantes ("stutter-closed"). Esto significa que la propiedad expresada por el requerimiento no debe depender del número de pasos en que mantenga verdadera o falsa. Por ejemplo, el *never* de la Figura 22 es oscilante mientras que el siguiente no lo es (ya que depende de que p se mantenga verdadera por lo menos un paso en la ejecución del sistema):

Never{p;p}

3.3 TRANSPUTERS

En esta sección se presenta los transputers y los modelos de programación que pueden adoptarse en el diseño de programas para ellos; se describen las principales características de los transputers y de los sistemas basados en estos procesadores, y se introduce el concepto del modelo de procesos secuenciales comunicantes CSP (Communicating Sequential Process).

3.3.1 LOS TRANSPUTERS

Los transputers son microprocesadores de alto rendimiento que soportan procesamiento paralelo en el mismo circuito integrado y en canales de comunicación hacia el exterior. Pueden ser usados como procesadores poderosos dedicados o se pueden conectar uno a uno a través de sus enlaces seriales (links) en las formas requeridas por la aplicación, para usarse como bloques de redes complejas de procesamiento paralelo.

El transputer es una microcomputadora completa en un solo chip que además de soportar la programación concurrente en la comunicación entre procesos, contiene:

- Una memoria muy rápida (ciclo único).
- Una interfaz de memoria programable que permite agregar memoria externa y memoria mapeada a dispositivos con el mínimo soporte lógico.
- Los servicios para un sistema integrado de transputers.
- Relojes de tiempo real.
- Una unidad integral de punto flotante, en las series T8.

La Figura 23 muestra la arquitectura generalizada de la familia de los transputers INMOS de 32 bits.

3.3.2 LOS ENLACES DE LOS TRANSPUTERS

Los enlaces de los transputers (links) permiten a los procesos que corren en procesadores conectados intercambiar datos y sincronizar sus actividades. El soporte para las comunicaciones es implementado a nivel hardware en cada chip de transputer. Las comunicaciones en los enlaces

operan concurrentemente con la unidad de procesamiento y los datos pueden transmitirse simultáneamente a todos los enlaces. La mayoría de los transputers tiene cuatro enlaces excepto los IMS M212 y T400 que solo cuentan con dos. Las herramientas, como los programas de depuración, utilizan los enlaces para examinar la memoria directamente desde otro procesador remoto. Los enlaces también son el medio para carga de programas desde el host a un transputer.

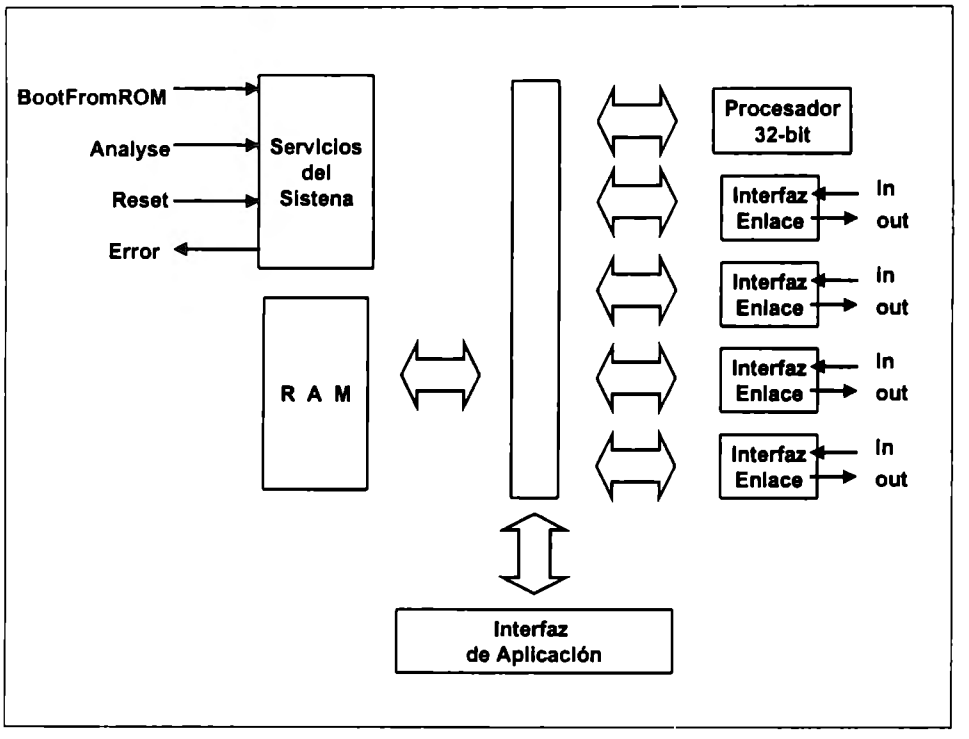


Figura 23 Arquitectura del procesador Transputer.

Cada proceso tiene un programador de tareas de alto rendimiento para los procesos de aplicaciones de usuario que corren en el mismo transputer. Los procesos que esperan por una entrada o salida, o por un tiempo de procesador, no consumen recursos de CPU, y el proceso de cambio de contexto es frecuentemente menor a un microsegundo.

3.3.3 PROGRAMACIÓN EN TIEMPO REAL

Las características de los transputers brindan soporte de hardware para la programación en tiempo real, algunas de éstas son:

- Implementación directa y eficiente de procesos paralelos en hardware.
- Implementación simple de manejo de interrupciones de software.
- Programación sencilla de temporizadores en software, permitiendo control de tiempo y poleo no saturado.
- Colocación de variables en direcciones específicas en memoria, para acceso a dispositivos mapeados a memoria.

3.3.4 SISTEMAS MULTITRANSPUTERS

Los sistemas multitransputers pueden construirse muy fácilmente usando los cuatro enlaces de alta velocidad; solo se requiere de dos líneas para conectar dos enlaces. Los circuitos para manejar cada enlace están en el mismo chip del transputer.

Los transputers pueden conectarse entre sí a través de sus enlaces para formar configuraciones de red distintas, dependiendo de las necesidades de las aplicaciones. Algunos de los arreglos posibles se muestran en la Figura 24.

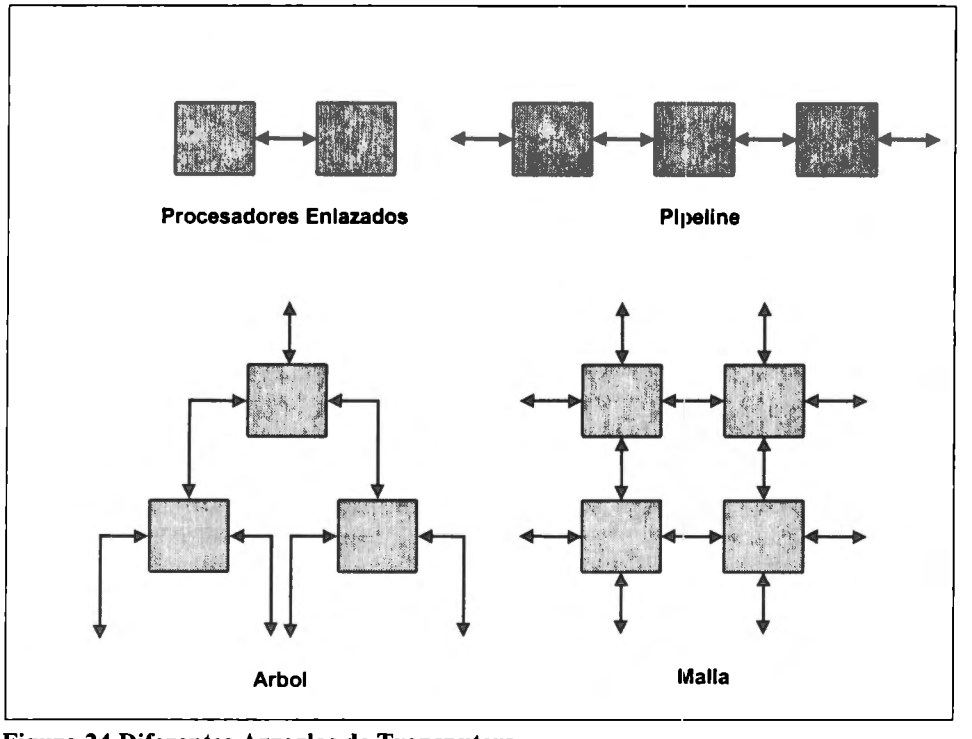


Figura 24 Diferentes Arreglos de Transputers.

3.3.5 TIPOS Y CLASES DE TRANSPUTERS

Existen varios modelos o tipos de transputers, los cuales se diferencian entre sí por su velocidad de procesamiento, número de enlaces y capacidad de memoria RAM. Estos tipos son agrupados en clases, concepto abstracto para clasificar los transputers que coinciden en su grupo de instrucciones. En la se listan todos los tipos de transputers y las clases a las cuales pertenecen, cabe notar que un mismo tipo de transputer se puede pertenecer a distintas clases.

El concepto de clases es muy útil para compilar y ligar los programas. A tiempo de compilación y ligado se indica para que transputers se va a generar el objeto; el objeto solo puede ser usado en el transputer para el cual fue generado. Se puede indicar la clase de transputer en lugar de indicar

el tipo, esto permite que los objetos sean útiles para un mayor número de transputers sin necesidad de volverlo a generar. Es importante mencionar que el código creado para una clase es frecuentemente menos eficiente que el que se crea para un tipo específico.

Clase	Procesadores
T2	T212, M212, T222, T225
T3	T225
T4	T414, T400, T425, T426
T5	T400, T425, T426
T8	T800, T801, T805
T9	T801, T805
TA	T400, T414, T425, T426, T801, T805
TB	T400, T414, T425, T426,

Tabla 12 Clases de Transputers.

Los conjuntos de instrucciones de las clases de los transputers se diferencian por lo siguiente:

- Las clases T2 y T3 soportan los transputers de 16 bits mientras que todas las clases restantes soportan transputer de 32 bits.
- La clase T3 es la misma que la clase T2 excepto que la T3 tiene algunas funciones extras para ejecutar CRC, operaciones de bits y de depuración.
- La clase T5 es la misma que la clase T4 excepto que la T5 tiene funciones extras para el CRC, operaciones de bits, de depuración e incluye la función *dup*.
- La clase T9 es la misma que la T8, excepto porque la T9 contiene funciones especiales para la depuración.
- Los procesadores T800, T801 y T805 usan un procesador de punto flotante en el mismo chip para ejecutar las operaciones aritméticas reales; por ello disponen de un mayor número de instrucciones.
- La implementación de las operaciones aritméticas se hacen vía software para las clases T4 y T5. Los transputers de estas clases solo cuentan con un número reducido de operaciones de punto flotante.

3.3.6 HERRAMIENTAS DE PROGRAMACIÓN

En el mercado existen dos herramientas que permiten el diseño, creación y montaje de programas en los transputers: ANSI C -Tool Set y Occam.

En ANSI C la programación paralela es soportada por librerías extras. Estas funciones permiten definir y crear procesos para comunicarse unos con otros a través de canales y sincronizarse usando semáforos.

Occam es el lenguaje nativo de los transputers y provee funciones sencillas para el manejo de canales y la creación de procesos.

En la Tabla 13 se muestra el contenido de las herramientas de ambos productos.

ANSI C	OCCAM	Herramienta	Descripción
X		*Icc*	Compilador de ANSI C.
X		*Icconf*	Cofigurador de ANSI C. Esta herramienta toma un archivo de configuración creado usando el lenguaje C de configuración de transputers y produce un archivo de datos de configuración que usa el colector de código para generar un archivo ejecutable.
X	X	*Oc*	Compilador Occam.
X	X	*Occof*	Configurador de Occam.
X	X	*Icollect*	Colector de código. Genera código ejecutable que puede cargarse a una red de transputers a través de un enlace.
X	X	*Idebug*	Depurador. Herramienta de depuración para programas de transputers. Puede correrse en modo post-mortem para determinar la causa de la falla en un programa detenido o en modo interactivo para ejecutar un programa sección por sección.
X	X	*Iemit*	Configurador de la interfaz de memoria. Sólo para los transputers T400, T414, T425, T800 y T805, los cuales cuentan con una interfaz de memoria externa configurable.
X	X	*Ieprom*	Convertidor de programas ROM. Se utiliza para convertir los programas que se han desarrollado y probado con las demás herramientas INMOS y colocarlos en ROM con mínimas modificaciones.
X	X	*Ilibr*	Manejador de bibliotecas. Construye bibliotecas tomando uno o más módulos compilados separadamente.
X	X	*Ilist*	Lector binario. Lee un archivo de código objeto, lo decodifica y despliega información útil del código en la pantalla.
X	X	*Imakef*	Generador de archivos make. Construye programas recompilando sólo los componentes que han cambiado desde su ultima modificación.
X	X	*Iserver*	Servidor de archivos del host. Carga código ejecutable a sistemas de transputers, permite a los programas tener acceso a los servicios del host y controla el acceso al sistema.

Tabla 13 Herramientas para el desarrollo de Programas para Transputers.

3.3.7 PROCESAMIENTO PARALELO

El procesamiento paralelo es soportado por el transputer incorporando en su diseño un programador de procesos, el cual es responsable de programar las tareas paralelas, y proporcionando los medios para conectar los procesos (enlaces) para crear una red de multiprocesadores.

3.3.7.1 Modelo de Programación

CSP (Communicating Sequential Processes) es un modelo abstracto generalizado de concurrencia basado en la idea de procesos ejecutándose independientemente intercambiando información a través de comunicación sincronizada.

En el C de SGS-Thompson los procesos concurrentes son independientes y pueden anidarse unos a otros, y son ligados a través de canales.

3.3.7.1.1 Procesos

Los procesos son los elementos principales del modelo CSP. Un proceso describe el comportamiento de un componente discreto, separable de una aplicación; puede consistir de otros procesos, operaciones secuenciales o combinaciones de estos. Las aplicaciones pueden dividirse en cualquier numero de procesos y cada uno de ellos puede mapearse a un procesador de la red a través de un archivo de configuración.

3.3.7.1.2 Canales

Los canales facilitan la comunicación entre procesos, son conexiones unidireccionales punto a punto y tienen dos funciones principales: proveen una ruta de comunicación independiente de los procesos ejecutándose y sincronizan sus tareas cooperativas.

3.3.7.1.3 Semáforos

El soporte de semáforos, que no es parte del modelo CSP, se provee en la herramienta C de SGS-Thompson para los que deseen desarrollar programas paralelos de la forma tradicional.

3.3.8 CONFIGURACIÓN DE LOS SISTEMAS DE TRANSPUTERS

Los transputers pueden ser unidos fácilmente para formar redes; cada transputer opera como nodo y se comunica con sus nodos vecinos a través de sus enlaces. La red más simple posible es la que sólo contiene un transputer que bien puede estar conectado al host o en un sistema aislado el cuál es iniciado desde una EPROM.

Para preparar una red para ejecutar una aplicación, es necesario definir que piezas de la aplicación van a ser colocadas en que nodos de la red. Este proceso se le conoce como "configuración" y se lleva a cabo por el configurador de Occam *(occonf)* y el de C (icconf), usando un archivo de configuración con terminación "pgm" y "cfs", respectivamente; éstos archivos contienen información de la red de hardware (descripción de la red), la descripción de los procesos (descripción del software) e información de cómo está mapeado el software al hardware.

El configurador lee el archivo de configuración y checa que los procesos han sido compilados para el tipo de transputer correcto según el nodo donde se colocará. También verifica que la configuración pueda llevarse a cabo en el hardware descrito y crea una descripción binaria de configuración, que puede ser leída por la herramienta *icollect* para generar un ejecutable.

Debido a que existen a lo más cuatro enlaces físicos por transputer, el hardware no puede configurarse para conectar directamente cada nodo con más de cuatro, sin embargo, este efecto puede llevarse a cabo agregando algunos procesos especiales para que un proceso en un nodo pueda comunicarse con varios procesos en otros nodos sin importar la topología de la red, de hecho no importa si los nodos a comunicar no son adyacentes. El configurador agrega estos procesos extras automáticamente pero el usuario puede especificar el grado en que esto debe hacerse; esta técnica es conocida como "ruteo de software".

3.3.8.1 Lenguaje de Configuración

Para el caso de los archivos de configuración "pgm" el lenguaje usado es una extensión de Occam y para los archivos "cfs" un lenguaje parecido a C. Para ambos casos, el lenguaje permite configurar módulos generados por cualquier compilador de INMOS TCOFF (Transputer Common Object File Format).

El lenguaje de configuración permite describir separadamente las redes de software y las de hardware y después unirlas usando una descripción de correspondencia del software al hardware. El lenguaje es simple e incorpora construcciones de alto nivel como los enunciados condicionales y las replicaciones.

Las partes de los archivos de configuración son detalladas en la presentación de los archivos para las aplicaciones desarrolladas en el presente trabajo (refiérase al punto 6.1.3).

3.3.8.2 Ruteo de Software y Multiplexaje

El configurador usa el ruteo de software y programas de multiplexaje para implementar la comunicación sobre enlaces virtuales; esto permite que muchos canales virtuales usen un mismo enlace físico entre transputers y permite a los procesos en procesadores no adyacentes comunicarse de una manera directa.

El ruteo de software y el multiplexaje es llevado a cabo automáticamente por el configurador y no requiere de la intervención del programador.

3.3.9 PROGRAMACIÓN EN VARIOS LENGUAJES

El uso del formato estándar TCOFF permite compilar y ligar módulos creados con distintos lenguajes fuentes, es decir, se puede mezclar C y Occam en el mismo sistema. Las unidades ligadas individualmente en formato TCOFF pueden ser mezcladas en cualquier combinación y colocadas en cualquier procesador de la red.

También es posible la llamada de módulos escritos en otro lenguaje. C puede llamar a Occam usando una directiva "nolink" que permite que el código Occam sea compilado sin la necesidad del parámetro *static* (o se puede declara en el código Occam un parámetro *static* de relleno). Occam puede llamar a C usando rutinas de librerías para inicializar y terminar las áreas de pila y *heap*.

Además se cuenta con un conjunto de interfaces que permiten llamar desde Occam programas escritos en C [16]. Esto puede Hacerse por varias razones, por ejemplo para usar programa C llamado desde el configurador *occonf*. Estas interfaces tienen como objetivo principal especificar adecuadamente el punto de entrada al programa C; el código producido con esta técnica es conocido como proceso equivalente Occam ya que hace parecer al programa como un proceso Occam con canales de entrada y salida.

Existen tres tipos de código de interfaz, conocidos como tipos 1, 2 y 3 (ver Figura 25). La primera interfaz es usada cuando el programa C corre en un único transputer y se comunica solo con el servidor de archivos del host. Esta interfaz es usada con la versión completa de la biblioteca C para ejecución (*runtime*).

El segundo tipo de interfaz se usa cuando el programa C además de comunicarse con el host, debe interactuar con otros procesos. Esta interfaz también usa con versión completa de la biblioteca C *runtime*.

La última interfaz es similar a la interfaz tipo 2, excepto que no requiere de acceso al host. Esta interfaz debe usarse con la versión reducida de la biblioteca C para tiempo de ejecución, la cual no contiene ninguna función que requiera acceso al sistema de archivos del host.

Como se puede apreciar en Figura 25, las interfaces del tipo 2 y 3 cuentan con un arreglo de canales que permiten al programa C comunicarse con otros procesos; estos arreglos son mapeados directamente a arreglos de canales que forman parte de la lista estándar de parámetros de la función *main* de C.

Tales arreglos son realmente arreglos de enteros en las listas de parámetros de Occam; esto permite pasar a apuntadores a canales al programa C, en el que hay más flexibilidad para mapear canales a los arreglos. Debido a que Occam no soporta apuntadores directamente, se cuenta con

dos procedimientos de biblioteca para asignar apuntadores de canal a los elementos del arreglo involucrando las direcciones directas de memoria.

La aplicación desarrollada en el presente trabajo utiliza las interfaces mencionadas por lo que en el punto 6.2.5 se detallan.

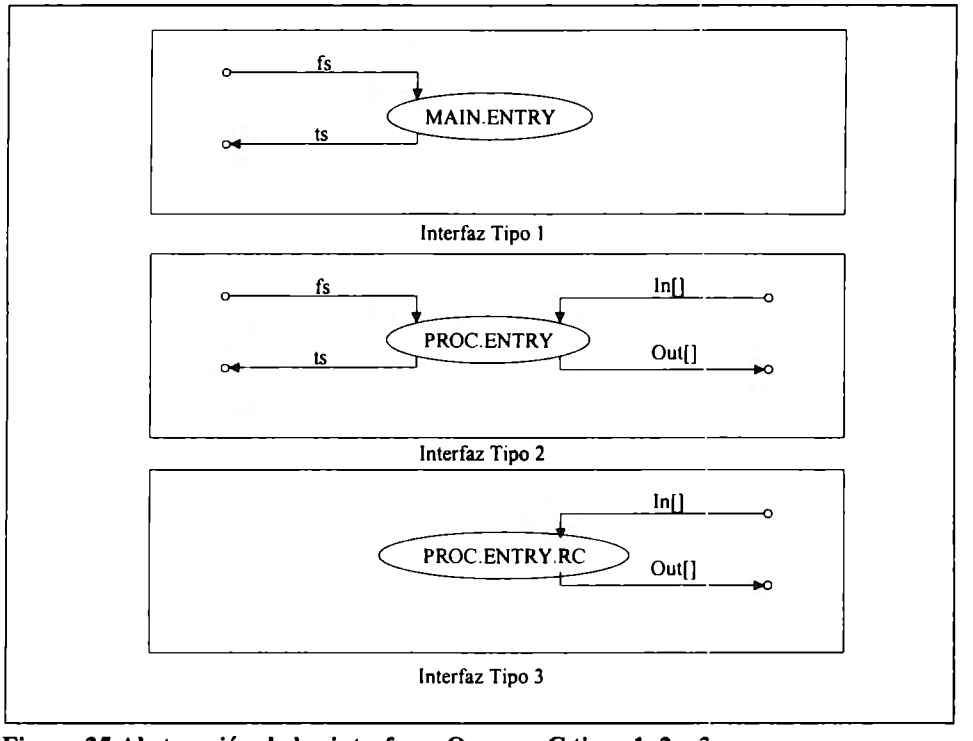


Figura 25 Abstracción de las interfaces Occam - C tipos 1, 2 y 3.

3.4 SIO 232 Y SIO 422

El SIO-232 y el SIO-422 son TRAMs (ver Figura 26) de tamaño 2 desarrollados por Transtech Parallel Systems con un transputer de 16 bits de la clase T2, una memoria SRAM de 64Kbytes, un receptor/transmisor asíncrono universal dual DUART (Dual universal asynchronous receiver/transmitter) de alto rendimiento, controladores RS-232 con sintetizadores de voltaje DC-DC y un controlador de interrupciones vía software configurable por el usuario. El DUART cuenta con diferentes opciones de configuración para la transmisión de datos, velocidad, bits de paro, paridad y tamaño de palabra [21].

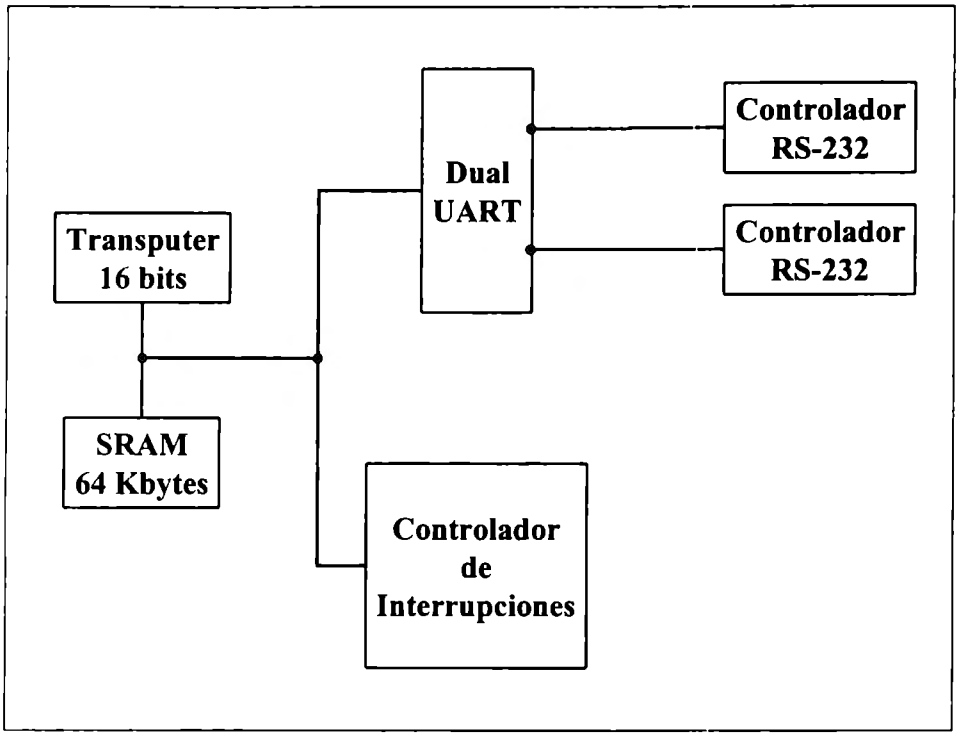


Figura 26 Esquema del SIO232.

Estos TRAMs proveen un método de alto rendimiento, flexible y económico para integrar un DUART y un controlador de dispositivo en una red de transputers sin la necesidad de hardware adicional.

El DUART provee dos canales asíncronos de recepción/transmisión full-duplex que se comunican directamente con cualquier dispositivo RS-232 o RS-422, como monitores, impresoras, módems, instrumentos de laboratorio y robots. La memoria SRAM permite alojar controladores y software del usuario en el mismo TRAM.

El modo de operación y el formato de datos de cada canal son programables e independientes para cada canal. Además, cada receptor y transmisor puede seleccionar su propia velocidad de operación de 18 velocidades fijas, un reloj 16X derivado de un temporizador/contador programable, o un reloj externo 1X o 16X. La capacidad para programar independientemente la velocidad de operación del receptor y transmisor hacen al DUART particularmente atractivo para aplicaciones de canales de dos velocidades como los sistemas de terminales encadenadas.

Cada receptor cuenta con un buffer de 8 caracteres para minimizar la perdida potencial de recepción por falta de espacio o para reducir la sobrecarga por interrupciones. Además, se provee una capacidad de control de flujo para deshabilitar un transmisor remoto cuando el receptor está lleno.

El DUART También provee un puerto de entrada de 7 bits y un puerto de salida de 8 bits, ambos de propósito general. Se pueden usar como puertos de entrada/salida o pueden asignarse a

funciones específicas (como entradas de reloj o como salidas de interrupciones/estado) por un programa de control.

Los registros del DUART están mapeados a direcciones de memoria del transputer en palabras de 16 bits. Sin embargo, sólo los 8 bits menos significativos de cada 16 son válidos. Los 8 bits de la parte alta de la palabra deben ignorarse o enmascarse dependiendo del uso específico.

Al momento de escritura del presente trabajo se están estudiando ambos TRAMs para el desarrollo de interfaces para el controlador de la celda de manufactura [14].

4 SUBSISTEMA DE COMUNICACIÓN

En este capítulo se plantea el esquema general del subsistema de comunicación para la celda de manufactura, así como el modelo de validación del protocolo orientado a MMS que propuesto en esta investigación, detallando las consideraciones de ambiente que se tomaron en cuenta para su diseño; este protocolo finalmente es la parte medular del subsisterna. Se describen además, otras partes de este subsistema que fueron desarrollados en otros trabajos.

4.1 ESQUEMA GENERAL DEL SUBSISTEMA DE COMUNICACIÓN

En la arquitectura propuesta por [13] para el controlador de la celda se pueden distinguir tres tipos de comunicación (ver Figura 27):

1. Comunicación nativa entre transputers del controlador a través de sus enlaces.
2. Interfaz entre los enlaces del transputer y el protocolo RS-232.
3. Comunicación entre el controlador y el robot a través del protocolo RS-232.

La comunicación natural del transputer ofrece grandes ventajas como se mencionó en la sección 3.3:

- Velocidad de comunicación.
- Paralelismo.
- Múltiples canales virtuales en un mismo enlace.

Sin embargo, se debe implementar un protocolo de comunicación que evite pérdidas de mensajes, duplicidad de mensajes, esperas indefinidas en caso de fallas de proceso y deadlocks, sin que esto impacte considerablemente en el desempeño de la comunicación del sistema. Estas características son cubiertas por el protocolo diseñado en este trabajo.

Los robots existentes en la celda cuentan con interfaz RS-232 y éste es un protocolo ampliamente probado en todo el mundo e inclusive es el más popular en términos de comunicación serial.

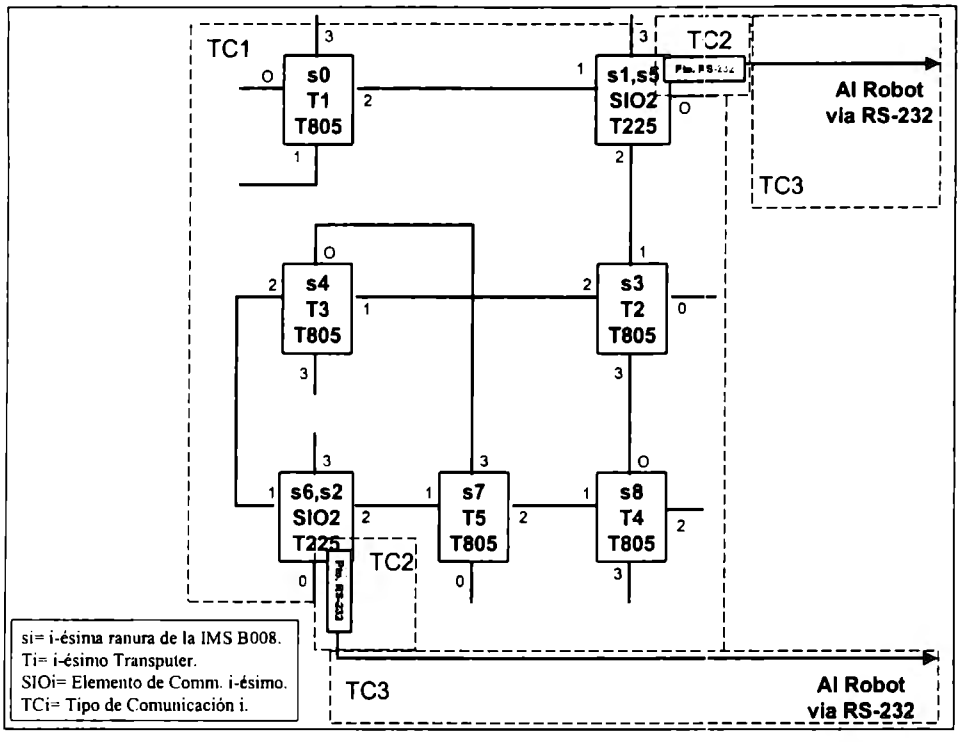


Figura 27 Tipos de comunicación en el controlador para la Celda.

Pero la comunicación entre los diferentes procesos del controlador no tendría sentido si no se pudiera comunicar con los robots de la celda. Para ello se hace necesario un protocolo para que el controlador pueda enviar comandos y datos al robot, y recibir información y estados de operación de él; objetivo cubierto con la interfaz que se está desarrollando dentro de este mismo proyecto [14] basada en el hardware SIO-232 y SIO-422 (refiérase a la sección 3.4). Esta interfaz utiliza un transputer IMS T212 que se diferencia del resto de los transputers utilizados en el controlador (IMS T805's) porque es un procesador de 16 bits y no de 32. El principal problema de utilizar un IMS T212 es que obliga al diseñador de la interfaz a usar Occam para generar sus programas, ya que las herramientas de C con las que se cuenta sólo pueden generar microcódigo para transputers de 32 bits.

Por otro lado, los programas de aplicación de la celda requieren de arreglos y funciones no soportadas por Occam, por lo que se percibe la necesidad de usar el lenguaje C paralelo para generación de este código. Los programas C y Occam pueden comunicarse entre sí como se indicó en el punto 3.3.9, sin embargo, su comunicación queda restringida a bytes cuando los programas se encuentran en transputers de distinta longitud de palabra, lo cual se presenta al querer comunicar los procesadores del controlador con el procesador de la interfaz. Este problema se debe a la diferencia en el manejo de tipos de datos en los transputer de 16 y 32 bits.

Lo anterior tiene como consecuencia una restricción en el protocolo que ha de comunicar el controlador con la interfaz RS-232: toda comunicación entre estas dos entidades debe hacerse a nivel byte; no se puede utilizar una transferencia de enteros, arreglos, etc.

Considerando estas restricciones en el ambiente de trabajo de la celda y deseando contar con los múltiples beneficios de un protocolo para manufactura como MiniMAP (ver sección 3.1.2.2), se propone que el subsistema de comunicación para la celda sea de tres capas; la comparación de este modelo con el sistema MiniMAP se puede observar en la Figura 28.

Las diferencia en la capa física y de enlace del subsistema propuesto con respecto a MiniMAP se debe a que, en la capa física, se trata de explotar al máximo las ventajas que ofrece la comunicación nativa de los transputers utilizando sus enlaces y, en la capa de enlace, debido a las restricciones arriba mencionadas se requiere de un protocolo orientado a bytes.

En la capa de aplicación, se adopta MMS para estar dentro de la estandarización mundial, en cuanto a intercambio de mensajes de manufactura se refiere (ver punto 3.1.2.2).

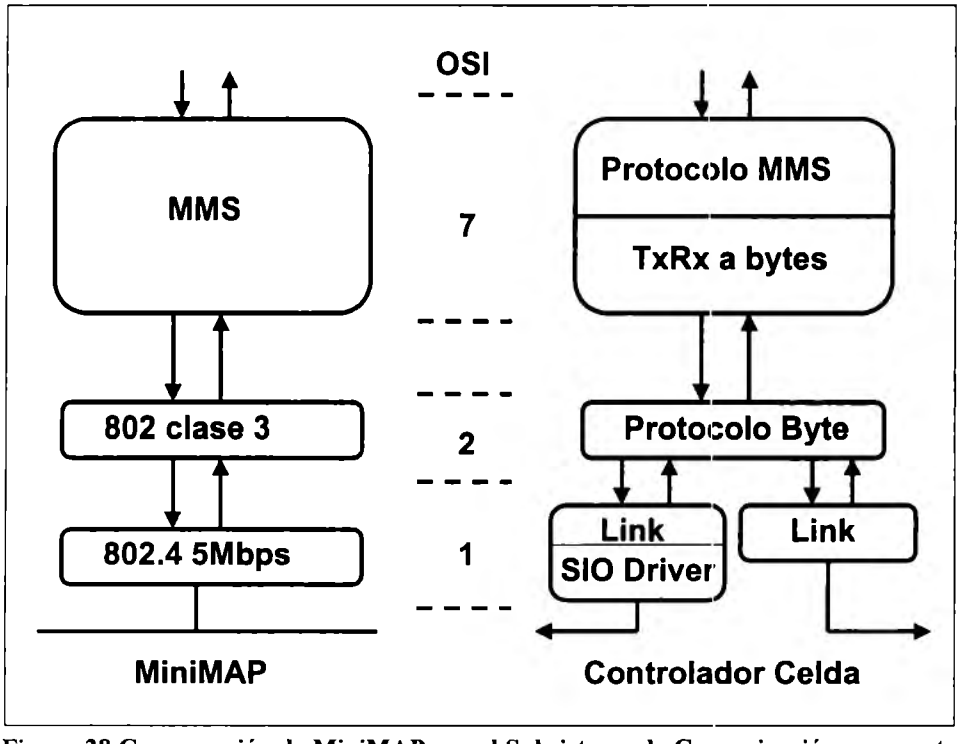


Figura 28 Comparación de MiniMAP con el Subsistema de Comunicación propuesto para la Celda de Manufactura ITESM - CEM

Para la implementación de este subsistema se requieren los siguientes tres protocolos de comunicación:

A nivel capa física se adopta la comunicación de enlaces de transputer (refiérase al punto 3.3.2) si la comunicación es hacia otro procesador del controlador. Si la comunicación está dirigida hacia el robot, entonces se hace necesaria la interfaz SIO-Driver, la cual se describe en el punto 4.1.1.

A nivel de enlace se adopta el protocolo orientado a bytes propuesto por la M.C. Mota González, mismo que se describe en el punto 0.

A nivel capa de aplicación se implementará un protocolo que esté basado en MMS, el cual es el objetivo central de la presente tesis, y es tratado desde la sección 0 y hasta el final del presente trabajo.

4.1.1 INTERFAZ RS-232 – TRANSPUTERS

El análisis de esta interfaz está siendo desarrollado dentro del mismo proyecto que esta investigación por el Ing. Edgar Morales [14]. Esta interfaz consiste de un módulo TRAM SIO-232 (refiérase a la sección 3.4) y de un programa que corre en paralelo con los programas de aplicación de usuario.

El programa que está en diseño es un protocolo para comunicar procesos en los transputers y los robots; corre en el transputer integrado en el módulo SIO-232 que es de 16 bits, lo que implica que los datos que se transfieran hacia este TRAM y se reciban de él deben ser de esta longitud de palabra. Los transputers restantes del controlador son de 32 bits, por lo que para evitar que los programas del controlador tengan que hacer rutinas adicionales, se propone que las comunicaciones de y hacia el SIO-232 se hagan en bytes.

Hay dos protocolos básicos de comunicación para el controlador SIO-232, que son:

```
tag; char                    BYTE;BYTE
tag;word.count::stringBYTE;INT16::BYTE
```

Estos protocolos permiten enviar caracteres, cadenas y comandos al proceso SIO-Driver que corre en el TRAM SIO. Los datos son siempre regresados del controlador como información de un solo caracter, es decir, en la forma:

```
tag; char                    BYTE;BYTE
```

4.1.1.1 Comandos del controlador

Existen comandos del protocolo para habilitar o deshabilitar el *handshaking*, seleccionar una o varias velocidades de transmisión preestablecidas y también para configurar velocidades de transmisión especiales. No todas las velocidades posibles del UART están preestablecidas, pero la capacidad de programar cualquier velocidad está abierta, inicializando directamente los registros del UART: ACR (auxilliary control register), MR0A (registro modo 0) y CSR (clock select registers).

4.1.1.2 Comandos Preestablecidos del Protocolo

Hay dos etiquetas usadas para indicar al controlador que el siguiente caracter es un comando. Una de estas etiquetas se usa para indicar las características del puerto A y otra para el puerto B. Cada etiqueta afecta al puerto correspondiente de manera similar.

Comando del Protocolo:

```
Out ! sio.cmd.A; char          o
      Out ! sio.cmd.B; char
```

sio.cmd.A	causa que el siguiente comando afecte las características del puerto A.
sio.cmd.B	causa que el siguiente comando afecte las características del puerto B.

El caracter "char" que sigue a los comandos debe ser uno de los siguientes:

Sio.enable.hs	comandos para el *handshaking*
Sio.disable.hs	
Sio.baud.1200	Velocidades normales
Sio.baud.2400	
Sio.baud.4800	
Sio.baud.9600	
Sio.baud.19.2K	
Sio.baud.28.8K	Velocidades extendidas
Sio.baud.57.6K	
Sio.baud.115.2K	
Sio.timer.mode	Velocidad contador/temporizador

Estas constantes normalmente se utilizan en la inicialización del SIO, aunque éste puede ser configurado dinámicamente.

4.1.1.3 Comunicación con el controlador

La información es enviada en bytes por controlador de la celda hacia el controlador del SIO a través de un enlace de transputer, para que este último la retransmita por un canal específico UART usando RS-232; esto se puede realizar de dos formas:

1) Para enviar un caracter de un nodo del transputer hacia el módulo SIO para transferir por el canal UART A, se ejecuta el siguiente código Occam:

```
Out ! sio.char.A; char          ó
```

```
SEQ i = 0 FOR (SIZE cadena)
  Out ! sio.char.A; cadena[i]
```

2) Para enviar una cadena se necesita la siguiente instrucción:

```
Out ! sio.string.A; (INT16 longitud)::cadena
```

Con estas bases de comunicación hacia el controlador del SIO, se implementa un protocolo que se describe a continuación.

4.1.1.4 Protocolo de SIO-232 y SIO-422

El controlador del SIO-232 se puede ver de manera abstracta como el conjunto de cuatro procesos concurrentes: interfaz de recepción, rutina de interrupciones, servidor de transmisión y servidor de recepción, sincronizados a través de canales y comunicados hacia otros procesos por medio un canal de entrada *IN* y uno de salida *OUT*.

4.1.1.4.1 Interfaz de Recepción

Este proceso (ver Figura 29 etiqueta "IN_RX") es el encargado de recibir los datos que se desean transmitir por RS-232; utiliza el canal *IN* y puede atender mensajes con 3 tipos de formatos: tag::char, tag::wcount::string y tag::cmd, para procesar un caracter "char", una cadena "string" de longitud "wcount" o un comando de configuración "cmd", respectivamente. En la etiqueta "tag" además de indicarse el tipo de mensaje de que se trata, se señala a que canal se afectará, al "A" o al "B".

La interfaz al recibir información de cualquiera de los dos primeros tipos, la retransmite a la rutina de interrupciones para que ésta la monte en los espacios del DUART para la transmisión RS-232, después de lo cual regresa a escuchar el canal *IN*.

Si la interfaz recibe un mensaje del tercer tipo, utiliza el comando recibido para configurar el DUART y regresa al punto "IN_RX" de la Figura 29.

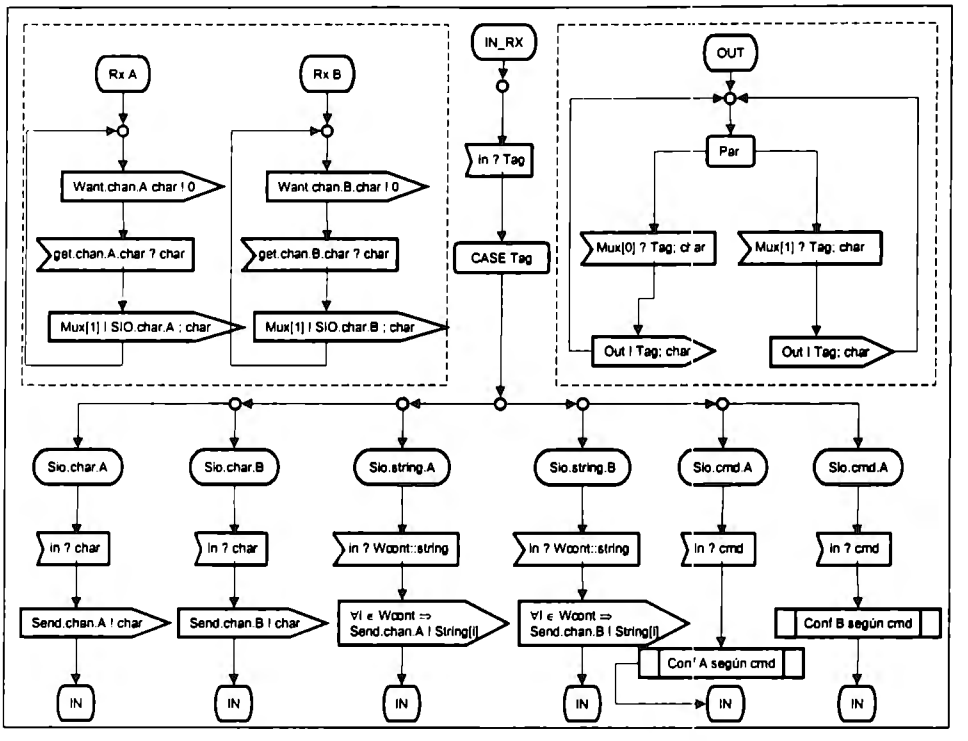


Figura 29 Esquema del Proceso Controlador de la Interfaz SIO-232.

4.1.1.4.2 Rutina de Interrupciones

Esta proceso es el encargado de sincronizarse con el DUART y los datos de transmisión y recepción a través de sus interrupciones; puede observarse en la Figura 30. Al iniciar el programa se habilitan las interrupciones de recepción para que el DUART grabe lo que llegue por RS-232 en los buffers de recepción después de lo cuál llega al punto marcado con la etiqueta "a". Estando en este punto el proceso puede realizar cinco acciones dependiendo de contenido de las variables y canales. Los dos primeros casos (etiquetas "b" y "c") se presentan cuando hay espacio en el buffer de transmisión y se desea transmitir un caracter a través del canal "A" ó "B" del DUART; el proceso coloca el caracter en el buffer correspondiente e incrementa un contador. El tercer y cuarto caso (etiquetados con "d" y "e") se presentan cuando en buffer de recepción no está vacío y el servidor de recepción esta listo para procesar información, el buffer de recepción contiene datos si el DUART ha recibido información; la rutina de interrupciones toma el dato indicado del buffer y lo transmite al proceso servidor de recepción, decrementando el número de datos válidos del buffer y regresando al punto "a".

La última de las acciones de este proceso sucede cuando hay una recepción de información por el canal de eventos; ésta opción queda libre para el usuario para interactuar con el controlador de una manera directa.

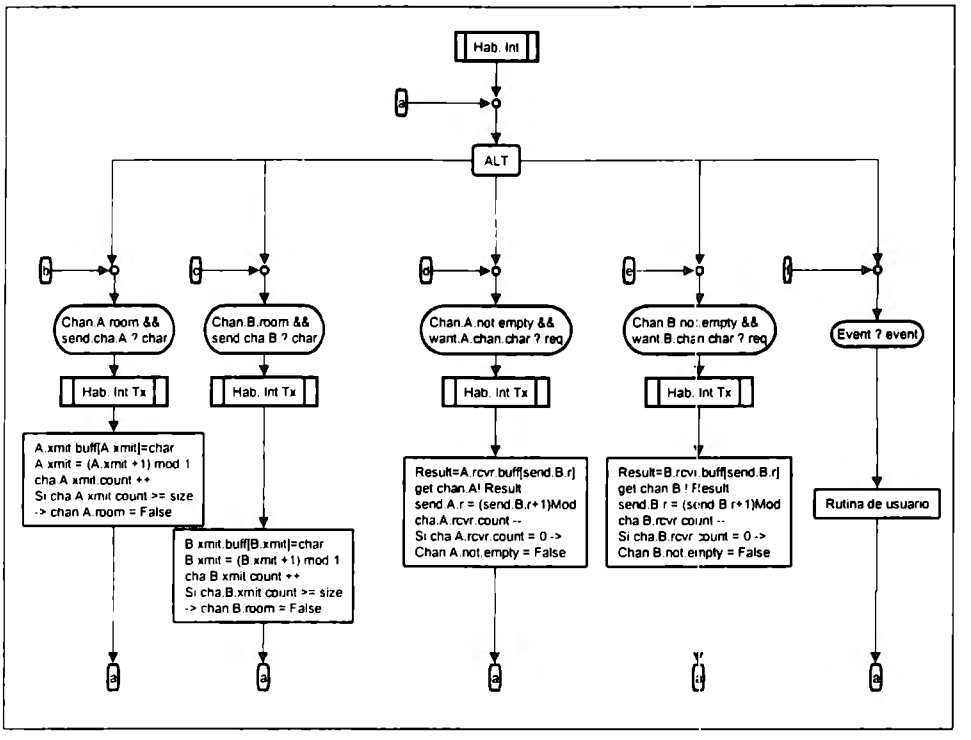


Figura 30 Esquema de la Rutina de Interrupciones del Controlador de la Interfaz SIO-232.

4.1.1.4.3 Servidor de Recepción

La tarea fundamental de este proceso es la de sincronizar los datos que lee la rutina de interrupciones con los datos que lee el proceso que esta siendo uso de la interfaz SIO-232, su esquema operativo puede verse en la Figura 29 en las secciones marcadas con las etiquetas "Rx A" y "Rx B". Cada vez que está listo, envía un mensaje a la rutina de interrupciones y espera por la recepción de un dato que ésta última le manda cuando el DUART ha hecho una recepción RS-232; al recibir el dato lo retransmite al servidor de transmisión a través de los canales *mux*, después de lo cual vuelve a indicar que está listo.

4.1.1.4.4 Servidor de Transmisión

Este proceso es el que sirve de enlace hacia los procesos que hacen uno de la interfaz SIO-232, transmitiéndoles la información que ésta recibe a través del puerto RS-232. Su tarea consiste en recibir la información del servidor de recepción y retransmitírsela a los procesos externos; el diagrama operativo de este proceso se puede observar en la figura antes mencionada en la etiqueta "OUT".

4.1.2 PROTOCOLO EN LA CAPA DE ENLACE

La M. en C. Mota González en [18] propone un protocolo orientado a byte para la comunicación entre transputers. Este protocolo consta de tres capas llamadas Aplicación, Enlace y Capa Física (ver Figura 31).

La capa de Aplicación es la interfaz del usuario al robot en ambos sentidos de la comunicación:

- Recibe los comandos del usuario y los transmite hacia el robot a través de las capas de enlace y física.
- Recibe los datos que envía el Robot a través de las dos capas inferiores para presentarlos al usuario.

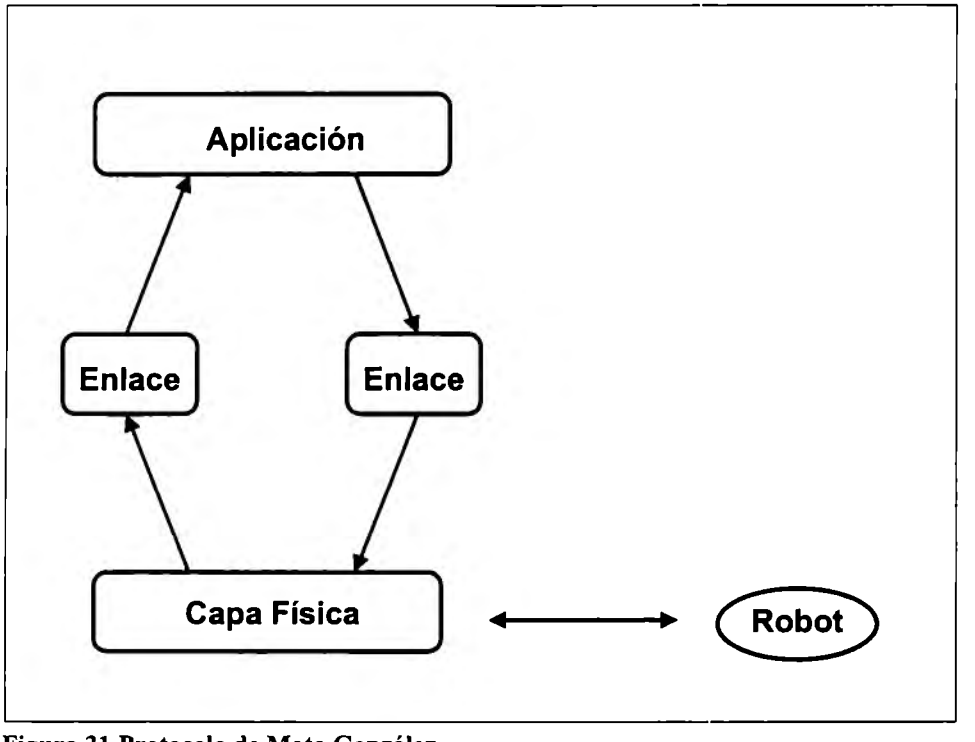


Figura 31 Protocolo de Mota González

El comando insertado por el usuario es procesado byte a byte detectando su terminación por medio de un caracter de escape ("\"). La capa de aplicación verifica la entrega del byte enviado esperando siempre un caracter de reconocimiento positivo (*ACK.caracter*) o negativo (*error*). La espera por este reconocimiento no puede ser indefinida en tiempo por lo que hace uso de un temporizador. El término de la comunicación por parte de esta capa, se lleva a cabo transmitiendo un caracter de fin de la transmisión.

En la parte receptora de esta capa, desde el punto de vista del usuario, además de los caracteres de reconocimiento, se reciben los caracteres de reconocimiento de terminación.

La capa de enlace de este protocolo tiene como objetivo asegurar que los mensajes provenientes de la capa física lleguen eventualmente a la capa superior; para ello, implementa un mecanismo de reenvío de mensajes. El reenvío se activa cuando no se recibe un reconocimiento de recepción del mensaje y cuenta con un ciclo finito, delimitado por un número fijo de intentos de transmisión del mensaje. Así mismo, se considera la implementación de un buffer de mensajes que tiene como objetivo coordinar las velocidades de consumo de mensajes por parte de la aplicación y el arribo de mensajes enviados por parte de la capa inferior.

En el otro sentido de la comunicación, esta capa tiene funciones análogas a las mencionadas en el párrafo anterior, sólo que aquí el consumidor es la capa inferior y el generador de mensajes es la capa de aplicación. Sin embargo, en esta etapa de enlace no se espera por un reconocimiento de transmisión, ya que supone una comunicación asíncrona con el robot y que este tipo de falla será detectado por un módulo externo de tolerancia a fallas. En las figuras 24 y 25 se tiene un esquema del protocolo de la capa de enlace, en la Figura 32 la comunicación de la Capa Aplicación a la Física y en la Figura 33 de la Capa Física a la de Aplicación.

La capa física de esta implementación es la comunicación nativa de los sistemas transputers y una interfaz propietaria Transputer-RS-232 [19] desarrollada en el ITESM-CEM y Toluca.

La capa de enlace del protocolo de Mota cumple con los requisitos necesarios para la capa de enlace del subsistema propuesto en este trabajo por lo que se adopta como una fase completamente validada [18].

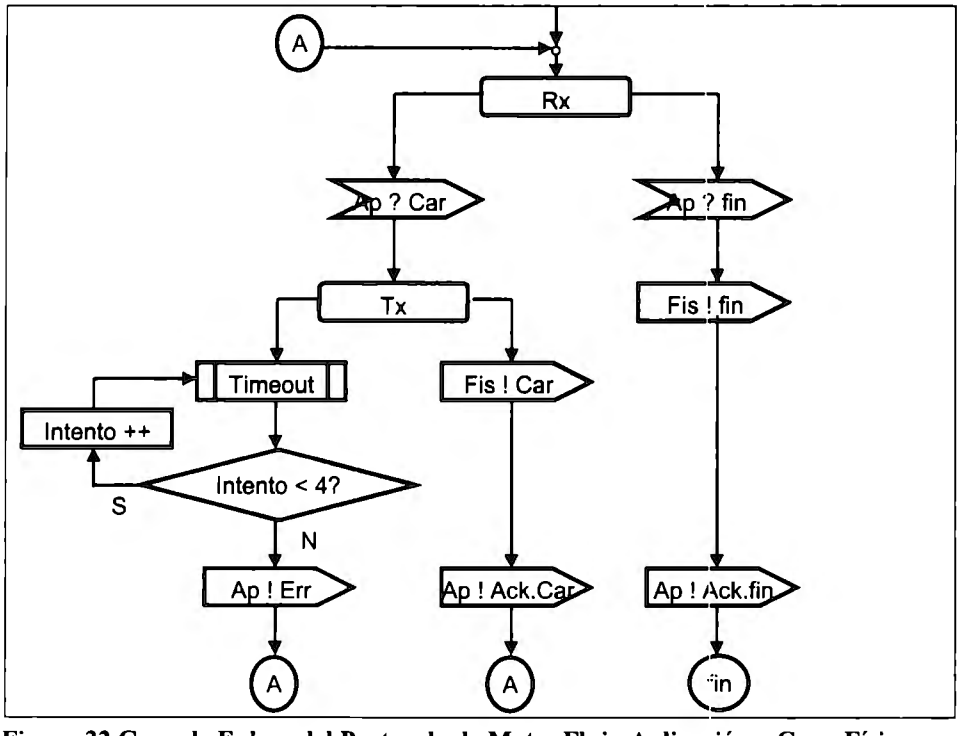


Figura 32 Capa de Enlace del Protocolo de Mota: Flujo Aplicación a Capa Física.

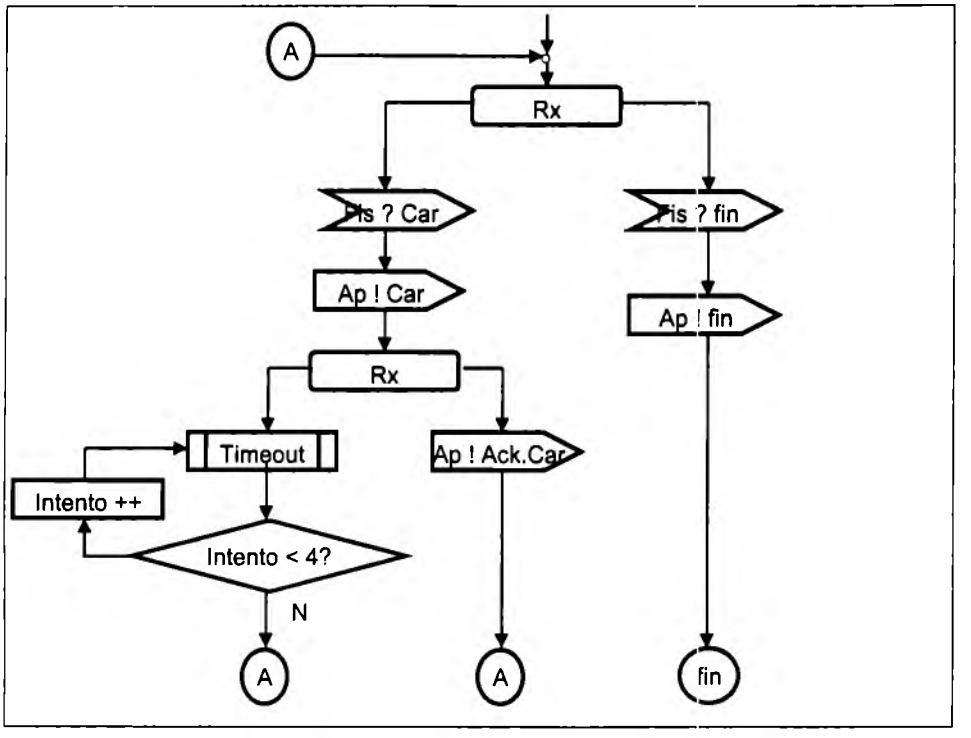


Figura 33 Capa de Enlace del Protocolo de Mota: Flujo Capa Física a Aplicación.

4.2 PROTOCOLO PROPUESTO BASADO EN MMS

Ya se cuenta con infraestructura que brinda toda la potencialidad de un sistema centralizado – distribuido basado en procesadores de alta velocidad [13]; se tiene además un protocolo para garantizar las funcionalidades que la capa de enlace de datos OSI requiere [18], y un protocolo que confiablemente transmite desde un enlace de transputers a un robot comunicado por el protocolo RS-232 y viceversa [14]. Hasta este nivel se podrían empezar a generar programas de aplicación para el control de la celda, sin embargo, estas aplicaciones tendrían que generar su propio código de interfaz para envío y recepción de comando específicos para cada robot, de tal manera que por cada aplicación existente en la celda habría un protocolo de intercambio de mensajes de manufactura. Esto implica en el peor de los casos que haya tantas interfaces como programas de aplicación existan. Lo anterior genera la necesidad de crear un protocolo único para el intercambio de mensaje de manufactura. Pero debe ser un protocolo que lejos de ser propietario de la celda del ITESM, cumpla con la estandarización hacia donde se está moviendo el mundo de la manufactura, la especificación MMS. Este es el objetivo central del presente trabajo de tesis.

MMS se mencionó en forma general en el punto 3.1.3, es tiempo de aterrizar esta filosofía en nuestra celda. Implementar MMS para la celda es un trabajo complejo porque hay que cumplir con las especificaciones sin olvidarse de que el protocolo esté correctamente validado.

La implementación de MMS en la celda para el presente trabajo consiste en generar un código que haga las funciones de la máquina de estados MMS, respetando la sintaxis de los tipos de mensajes que puede enviar y recibir una entidad MMS. Se trata de que la celda esté preparada para que cuando sean adquiridos robots que manejen este estándar únicamente se tenga que conectarlos a la red. No se trata de una implementación del protocolo en cada uno de los robots existentes, ya que éstos no cuentan con un software de adaptación al estándar.

El desarrollo de cómo se llegó a la implementación de MMS en la celda está contenido en de aquí y hasta el final del presente trabajo, tal y como se fueron presentando las etapas de diseño, validación e implementación.

4.3 BASES DEL PROTOCOLO

En esta sección se presentan las bases que se consideran para el diseño del protocolo. Primeramente se aborda la forma general en la que se intercambia la información, la cuál esta basada en la filosofía Cliente-Servidor. Después se destacan de los servicios que el protocolo debe prestar según MMS y los requerimientos del medio ambiente donde se correrá el protocolo, es decir, los del nuevo controlador de la celda. En los últimos puntos de esta sección se definen los tipos de mensajes válidos para el protocolo y el formato de éstos.

4.3.1 EL INTERCAMBIO DE INFORMACIÓN

Como se mencionó anteriormente MMS está basado en el modelo Cliente-Servidor, y el intercambio de datos se hace por medio de Unidades de Datos del Protocolo, PDU (Protocol Data Unit). Estos PDUs son perfectamente conocidos por ambas entidades y si alguna de ellas no reconociera alguno, lo rechazaría inmediatamente; el esquema de la Figura 11 muestra esta interacción.

En un ambiente MMS generalmente el controlador de la celda es el cliente y los robots y máquinas automatizadas son los servidores; esto es debido a que los controladores solicitan información sobre las actividades de cada robot, quienes a su vez esperan por instrucciones de los controladores.

4.3.2 SERVICIOS DEL PROTOCOLO

Lo primero que se requiere para que se pueda intercambiar información es que se establezca una asociación entre las entidades comunicantes (refiérase al punto 3.1.3.2). En este proceso, además de reconocer a la entidad que se quiere asociar, se negocian parámetros para el intercambio de mensajes, tales como:

- Número Máximo soportado de Peticiones en espera.
- Número Máximo soportado de Indicaciones en espera.

- Límite superior de anidamiento soportado en la definición de variables.

El proceso de asociación es el primer paso en el protocolo y su complemento es el proceso de "conclusión de la asociación". En la Figura 34 se puede observar este intercambio de mensajes entre las entidades para establecer y liberar una asociación.

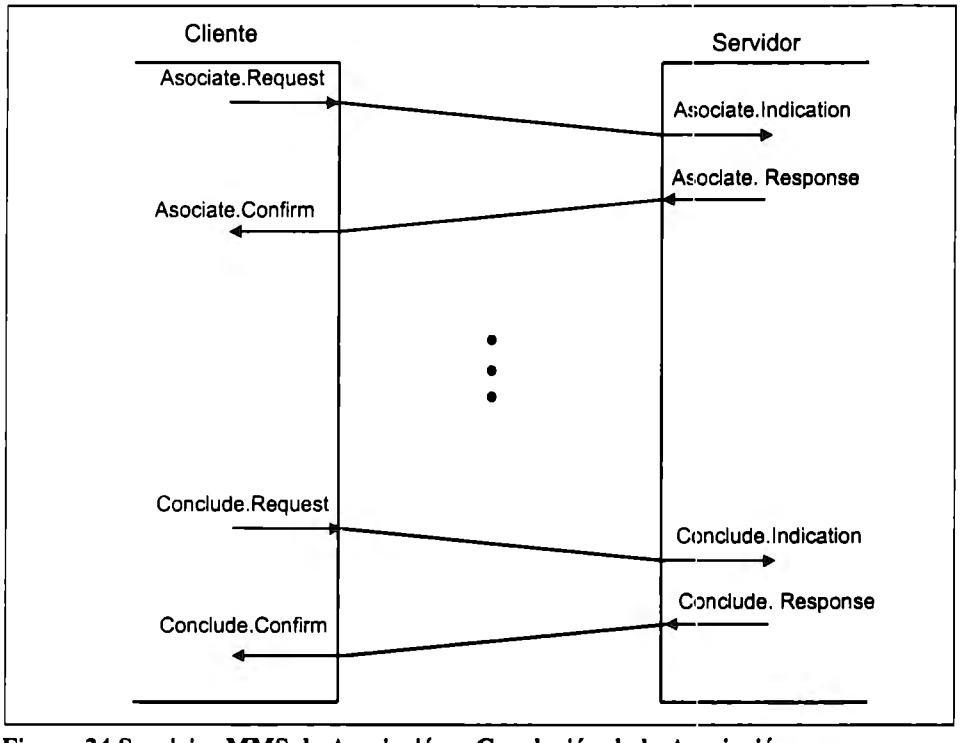


Figura 34 Servicios MMS de Asociación y Conclusión de la Asociación.

Otra forma de deshacer la asociación es abortarla; este servicio a diferencia de los dos antes descritos, es un servicio sin confirmación empleado por la máquina de protocolo cuando por alguna razón debe terminar expeditamente la asociación. Este Servicio se muestra en la Figura 35.

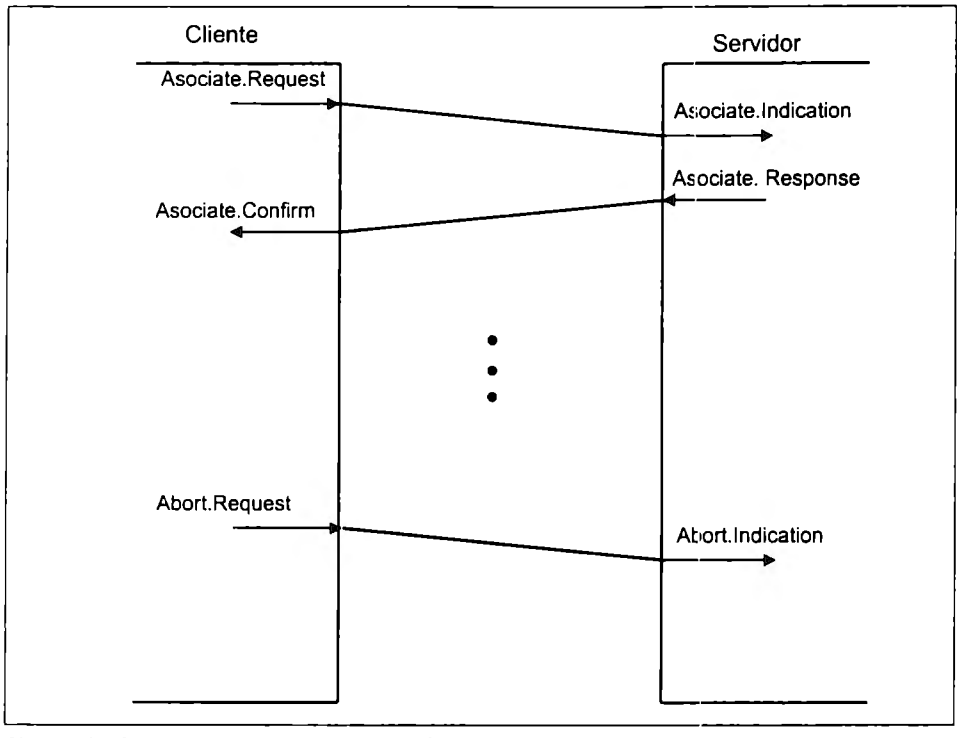


Figura 35 Servicios MMS de Asociación y Aborto de la Asociación.

Una vez hecha la asociación, hay dos tipos básicos de servicios que debe brindar la máquina de protocolo: el Servicio Confirmado y el Servicio sin Confirmación. La mayoría de los mensajes en una celda de manufactura tienden a ser confirmados debido a que el controlador generalmente siempre exige una respuesta sobre la información o comando enviado al robot y depende de será respuesta la decisión que tomará para la siguiente instrucción. El servicio sin confirmación es habitualmente usado en el sentido robot hacia el controlador, utilizado principalmente para transferir datos y estado periódicos de las acciones que el robot está realizando, para que el controlador pueda así monitorearlos de una manera completa. Las alarmas también son del tipo no confirmado, pues la entidad que las emite no puede esperar por una respuesta para apagarse o realizar otra acción ante cualquier contingencia.

El intercambio lógico de mensajes en los servicios confirmados y no confirmados se puede apreciar en la Figura 36.

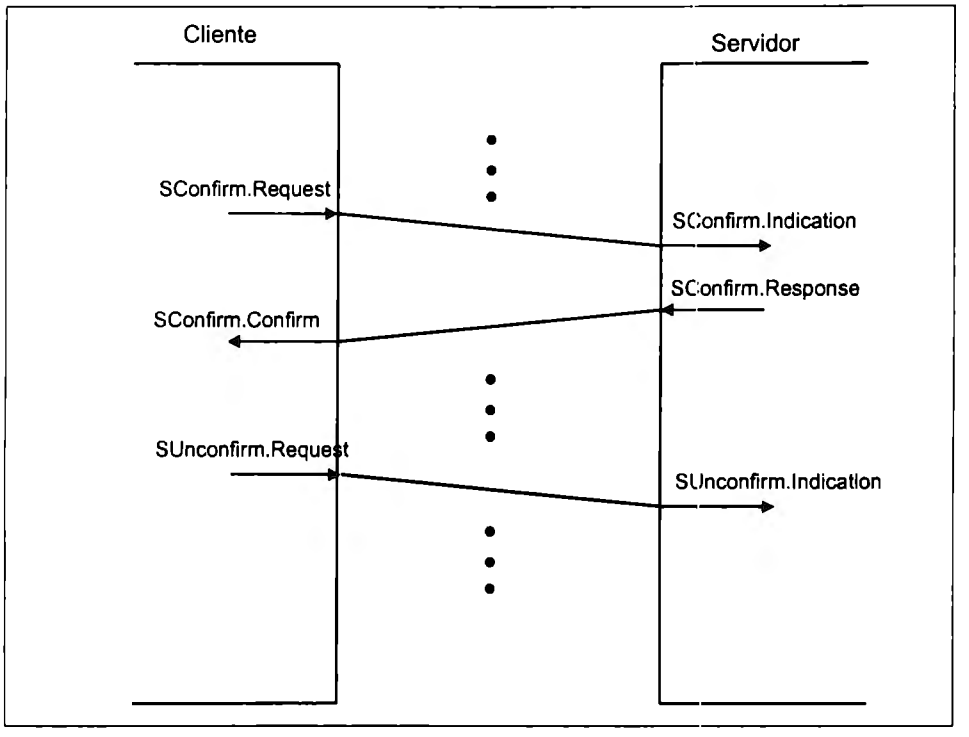


Figura 36 Servicios MMS Confirmado y Sin Confirmación.

Otro servicio propuesto por MMS es el servicio *Reject*. La máquina del protocolo debe ser capaz de rechazar cualquier servicio cuando el PDU no sea llenado correctamente, cuando el servicio solicitado sobrepase las especificaciones negociadas en el establecimiento de la asociación, o bien, cuando el servicio solicitado no se pueda procesar.

Adicionalmente se requiere un servicio muy particular a esta celda, el cual no está incluido en las especificaciones MMS. Se trata de un servicio en el cual el proceso (cliente) que intenta hacer la asociación, verifique que exista el servidor del servicio. Sin este servicio el protocolo supondría que debe terminar el servidor cuando no haya más asociaciones abiertas y para poder continuar con la operación de la celda se tendría que reinicializar tanto al cliente como al servidor. Además, en este proyecto se está proponiendo que el controlador de la celda cuente con un mecanismo de tolerancia a fallas [15], por lo que sí el cliente llegara a fallar el respaldo de este cliente entraría a sustituirlo para que el controlador dinámicamente se recupere de la falla. Para poder responder a este requerimiento de nuestro sistema usaremos el servicio *Login*.

El cliente solo podrá tratar de hacer asociaciones si se puede firmar con el servidor, es decir, su vida esta condicionada a la existencia del servidor. Por otro lado, el servidor puede vivir aún y si no existe eventualmente el o los clientes.

En un principio, se podría pensar en que el servicio de crear una asociación se encarga también del servicio *Login*, pero esto implicaría alterar el concepto de establecimiento de asociación que define MMS, lo cual no es conveniente por lo que se agrega como un servicio independiente. Este servicio es asimétrico y se puede observar en la Figura 37.

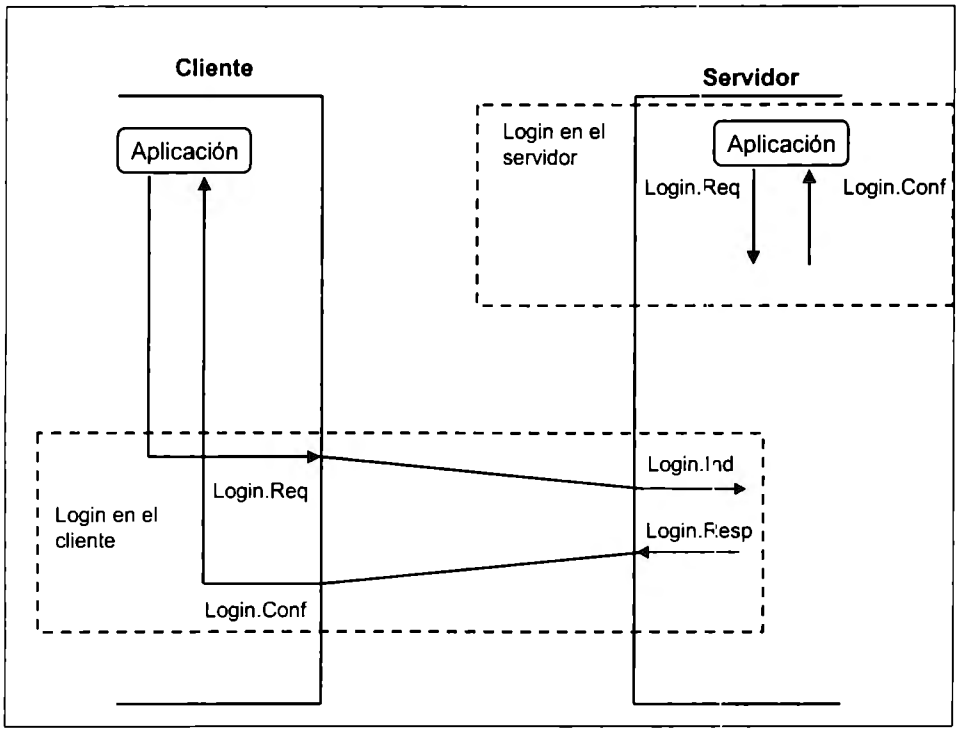


Figura 37 Servicio Login para el controlador de la celda.

Finalmente, *Login* tiene como contraparte el servicio *Logout* que tiene un comportamiento distinto dependiendo del estado en que se encuentre la máquina del Protocolo. Los dos estados de la máquina que hacen diferenciar este servicio son:

- *Estado Sin Enlace (Stand Alone).* El servidor se encuentra esperando a que se firme algún cliente. El cliente se ha desconectado del servidor porque este se dio de baja y está esperando a que la capa superior se desconecte.

- *Estado Enlazado.* Si se trata del servidor, tiene al menos un cliente conectado. Si es la entidad cliente, se ha conectado al servidor previamente.

Si la máquina se encuentra en el primer estado, el servicio no saldrá a la máquina de la entidad remota por lo que no se generará la indicación ni la respuesta del servicio. Si la máquina se encuentra en el segundo estado, entonces el servicio se convierte en un servicio completo. El esquema del servicio se observa en la Figura 38.

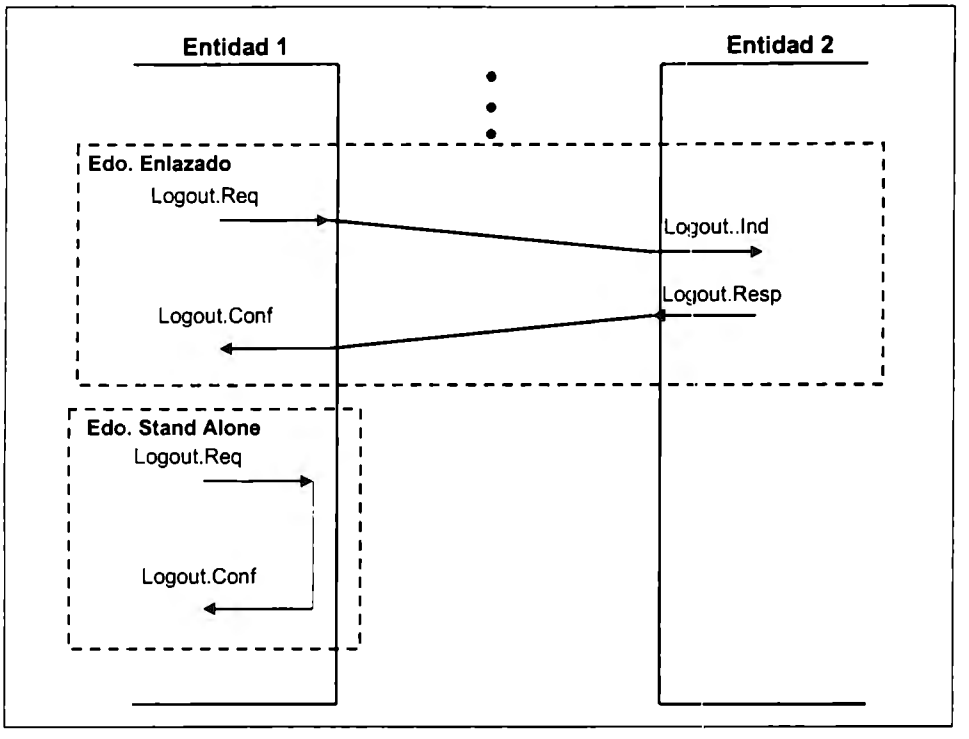


Figura 38 Servicio *Logout* para el controlador de la celda.

Servicio	Objetivo	MMS ó Propietario
Login	Cliente.- Notificar la existencia de la entidad en el servidor. Servidor.- Dar de alta el servicio.	Propietario
Logout	Dar de baja la entidad.	Propietario
Associate	Negociar los parámetros de la comunicación entre las entidades.	MMS
Conclude	Dar de baja la asociación.	MMS
Reject	Rechazar el servicio solicitado.	MMS
Confirm	Petición de un servicio que requiere de una respuesta por parte de la entidad remota. Esta respuesta puede ser afirmativa o negativa.	MMS
Uconfirm	Petición de un servicio que no necesita respuesta por parte de la entidad remota. Por lo general es un servicio que genera el servidor, reportando un dato periódico, un estado de alguna variable para supervisión del controlador, o la generación de una alarma.	MMS
Abort	Servicio utilizado por la máquina de estado cuando no puede continuar con el protocolo. Es una salida de emergencia de la conversación establecida.	MMS

Tabla 14. Servicios que provee la máquina de protocolo MMS del controlador.

La Tabla 14 muestra un resumen de los servicios que deberá incluir el protocolo MMS del controlador de la celda.

Teniendo identificados los servicios que debe proveer el protocolo, el siguiente paso es definir las suposiciones sobre el medio ambiente de trabajo del controlador.

4.3.3 REQUERIMIENTOS DEL PROTOCOLO DEBIDO AL MEDIO AMBIENTE DE TRABAJO DEL CONTROLADOR

Las bases en las cuales se diseña el presente protocolo obedecen al ambiente de trabajo del controlador y son:

- Los transputers están montados sobre una misma tarjeta madre, las distancias de comunicación son muy cortas y la circuitería permite tener un bajo nivel de ruido, por lo que puede asegurarse la integridad del mensaje transmitido a través de sus enlaces.

- Un enlace puede llegar a fallar e inclusive un mismo procesador tiene esta probabilidad. El mensaje transmitido puede llegar a perderse.

- Lo anterior se soluciona usando temporizadores para esperar la respuesta al mensaje transmitido, para los servicios de esta naturaleza. Si después de la expiración del temporizador no se ha recibido respuesta, entonces se procede a retransmitir el mensaje. El procedimiento anterior se repite un número específico de veces. Si después de estos intentos no se ha recibido respuesta para la petición pendiente, se procede a regresar un estado de error a la aplicación que solicitó el servicio. Este reenvío de mensajes puede provocar que el receptor obtenga mensajes duplicados, por lo que el protocolo debe ser capaz de manejar la duplicidad de mensajes.

- La aplicación que llama a la máquina de protocolo sabe llenar el PDU MMS correspondiente al servicio; sin embargo, puede ser que esa aplicación haya equivocado el PDU o que algún dato de éste no sea coherente con la negociación que se llevó a cabo en el establecimiento de la asociación, por lo que el protocolo debe ser capaz de manejar el rechazo de PDU's.

4.3.4 LOS TIPOS DE PDU EN MMS

Aunque los servicios presentados en el punto 4.3.2 pueden parecer muy reducidos, la complejidad de MMS radica en todos los tipos de PDU's que pueden viajar en tales servicios; la sintaxis MMS se puede consultar en [27]. Para mostrar la manipulación del PDU MMS que debe hacer la aplicación que usa MMS se supondrá que se quiere escribir un dato en una variable del servidor.

1) Elegimos el tipo de PDU. En este caso *Confirmed-Request-PDU.*
2) Un *Confirmed-Request-PDU* implica una secuencia de las siguientes declaraciones:
   a) El Invoke-Id que es el identificador de petición. Generalmente un número progresivo.
   b) Y la petición de servicio confirmado. *ConfirmedServiceRequest*

3) La petición *ConfirmedServiceRequest* es una alternativa de todos los servicios confirmados existentes para MMS, por mencionar algunos: status, *getNameList, identify, rename, read, write*. Para nuestro ejemplo escojamos *write*.
4) Al escoger el servicio write implica una secuencia del tipo *WriteRequest*.
5) La secuencia *WriteRequest* nos obliga a llenar los campos del PDU llamados: y *VariableAccessSpecificatn y ListOfData*.
   a) *VariableAccessSpecificatn* nos presenta una alternativa: *ListofVariable o VariableListName*. Por simplicidad para el ejemplo seleccionemos *VariableListName*.
      i) *VariableListName* implica llenar los campos relacionados a *ObjectName*.
      ii) *ObjectName* nos presenta la alternativa de llenar el nombre específico de la máquina virtual de manufactura, el dominio especifico o el identificador en la asociación. Por simplicidad escojamos la primera alternativa conocida como *vmd-specific*.
      iii) *Vmd-specific* nos implica llenar un campo de tipo *identifier*, que es una cadena de caracteres.
   b) *ListOfData* es una lista de datos con su valor. Los tipos de datos pueden ser: arreglos, estructuras, booleanos, etc. Y el valor depende del tipo.

Este pequeño ejercicio nos permite percatarnos de la complejidad que se presenta cuando no se tiene experiencia en el entendimiento del llenado del PDU. Sin embargo, para establecer un buen protocolo MMS no se requiere ser un experto en el manejo de estos PDU's; lo importante es conocer los tipos de servicio, el sentido en la comunicación y los diferentes tipos de PDU para poder establecer la colección de mensajes válidos en el protocolo.

4.3.5 TIPOS DE MENSAJES EN EL PROTOCOLO

La selección de los tipos de mensaje que tiene el modelo de validación del protocolo se hace teniendo en cuenta que deben ser los menos posibles y que se deben incluir todos aquellos que causen un cambio en su comportamiento. No basta con elegir solo los Tipos de Servicio que el protocolo ofrece con sus respectivas primitivas, pues en cada tipo de servicio pueden viajar diferentes tipos de PDU validos. Para obtener estos mensajes se analizó servicio por servicio considerando los PDU's que pueden viajar en cada servicio según el estándar MMS. La Tabla 15 muestra la matriz formada para obtener estos tipos de mensajes; en la primera y segunda columna se encuentran los servicios y sus primitivas correspondientes y, en la fila sombreada de cada servicio los PDU's válidos.

A manera de ejemplo a continuación se describe la forma de deducir los mensajes válidos para el servicio *Associate*:

1. La aplicación de usuario envía a la MMPM (máquina de Protocolo MMS) la primitiva *Associate.request*.- Con esta primitiva solo pueden viajar el *initiate_RequestPDU,* en donde viajan, entre otros parámetros, los correspondientes de la negociación. De aquí se genera el primer tipo de mensaje para este servicio: *MAsRq_initiate_RequestPDU.*

2. La MMPM recibe la petición *Associate.request*.- Sí el PDU recibido está bien formado, la MMPM genera un servicio *Associate.indication* que envía a la MMPM remota, lo cual nos conduce al tipo de mensaje: *MAsIn_initiate_RequestPDU*. Si el PDU no está bien formado entonces genera una confirmación *Associate.request* con un *PDU_reject*, lo que implica el tipo *MAsCf_reject_PDU*. Por otro lado, sí el PDU está bien formado pero por alguna razón la MMPM no puede procesar el servicio, por que llego al límite de asociaciones disponibles, por ejemplo, la MMPM regresará a la Aplicación que lo llamó una confirmación *Associate.request* con un *PDU_Error*.

3. La MMPM remota recibe la indicación del servicio *Associate*.- Si esta MMPM puede seguir procesando el servicio, entonces transmite a la capa de aplicación una indicación con el *initiate_RequestPDU*, que es el mismo tipo de mensaje que se generó en el punto 2. Sí la MMPM no puede darle seguimiento al servicio entonces genera una respuesta *Associate* con un *PDU_error*, lo que implica un nuevo tipo de mensaje: *MAsCf_initiate_ErrorPDU*.

4. La aplicación de la entidad remota recibe la petición de asociación.- La función ejecutiva de esa entidad podrá devolver una respuesta aceptación o de error, lo que implica los tipos: *MAsRs_initiate_ResponsePDU* y *MAsRs_initiate_ErrorPDU*

5. La MMPM recibe la respuesta de la aplicación.- La aplicación retransmite hacia la MMPM original si el PDU está bien formado, los tipos correspondientes (en este caso son los mismos que en el punto anterior). Si un PDU no está bien formado, la MMPM regresa un *reject_PDU* a su capa de aplicación correspondiente, lo que nos implica un tipo de mensaje adicional: *MAsIn_reject_PDU*

6. La MMPM recibe la respuesta de la MMPM remota.- Esta máquina sólo retransmite el tipo de mensaje.

7. Note que en el punto 3 y 6 nunca se genera un *PDU_reject* debido a que se supone que la MMPM de donde proviene el servicio no podrá enviar un PDU mal formado.

Análogamente se analizaron todos los servicios del protocolo y las combinaciones posibles con los tipos de PDU de la MMS, resultando los mensajes que se listan a continuación:

1. MAsRq_initiate_RequestPDU
2. MAsIn_initiate_RequestPDU
3. MAsIn_reject_PDU
4. MAsRs_initiate_ResponsePDU
5. MAsRs_initiate_ErrorPDU
6. MAsCf_initiate_ResponsePDU
7. MAsCf_initiate_ErrorPDU
8. MAsCf_reject_PDU
9. MRlRq_conclude_RequestPDU
10. MRlIn_conclude_RequestPDU
11. MRlIn_reject_PDU

12. MRlRs_conclude_ResponsePDU
13. MRlRs_conclude_ErrorPDU
14. MRlCf_conclude_ResponsePDU
15. MRlCf_conclude_ErrorPDU
16. MRlCf_reject_PDU
17. MCfRq_confirmed_RequestPDU
18. MCfIn_confirmed_RequestPDU
19. MCfIn_reject_PDU
20. MCfRs_confirmed_ResponsePDU
21. MCfRs_confirmed_ErrorPDU
22. MCfCf_confirmed_ResponsePDU
23. MCfCf_confirmed_ErrorPDU
24. MCfCf_reject_PDU
25. MUfRq_unconfirmed_PDU
26. MUfIn_unconfirmed_PDU
27. MUfIn_reject_PDU
28. MAbIn_abort_PDU
29. MLiRq_XXXPDU
30. MLiIn_XXXPDU
31. MLiCf_XXXPDU
32. MLiCf_ErrPDU
33. MLoRq_XXXPDU
34. MLoIn_XXXPDU
35. MLoCf_XXXPDU

En la Tabla 15 se muestra la matriz de donde provienen los tipos de mensajes del protocolo.

Servicio	**Primitiva**	**Initiate_Request PDU**	**Initiate_Response PDU**	**Initiate_Error PDU**	**reject_PDU**
Associate	Request	MAsRq_ initiate_RequestPDU	X	X	X
	Indication	MAsRq_ initiate_RequestPDU	X	X	MasIn_ reject_PDU
	Response	X	MAsRs_ initiate_ResponsePDU	MasRs_ Initiate_ErrorPDU	X
	Confirm	X	MAsCf_ initiate_ResponsePDU	MasCf_ Initiate_ErrorPDU	MasCf_ Reject_PDU
Servicio	**Primitiva**	**conclude_Request PDU**	**conclude_Response PDU**	**Conclude_Error PDU**	**reject_PDU**
Release	Request	MRlRq_ Conclude_RequestPDU	X	X	X
	Indication	MRlIn_ Conclude_RequestPDU	X	X	MRlIn_ reject_PDU

	Response	X	MRIRs_ conclude_ResponsePDU	MRIRs_ Conclude_ErrorPDU	X
	Confirm	X	MAsCf_ conclude_ResponsePDU	MasCf Initiate_ErrorPDU	MAsCf_ reject_PDU
Servicio	**Primitiva**	**Confirmed_Request PDU**	**confirmed_Response PDU**	**Confirmed_Error PDU**	**reject_PDU**
Confirmed Service	Request	MCfRq_ Confirmed_RequestPDU	X	X	X
	Indication	Mcfln_ Confirmed_RequestPDU	X	X	Mcfln_ reject_PDU
	Response	X	MCfRs_ confirmed_ResponsePDU	MCfRs_ Confirmed_ErrorPDU	X
	Confirm	X	MCfCf_ confirmed_ResponsePDU	MCfCf_ Confirmed_ErrorPDU	MCfCf_ reject_PDU
Servicio	**Primitiva**	**Unconfirmed_ PDU**	**confirmed_Response PDU**		**reject_PDU**
Unconfirmed Service	Request	MufRq_ Unconfirmed_PDU	X	X	X
	Indication	X	Mufln_ unconfirmed_PDU	X	MUfln_ reject_PDU
Servicio	**Primitiva**	**Unconfirmed_ PDU**			
Abort	Request, Indication	Mabln_ Abort_PDU	X	X	X
Servicio	**Primitiva**	**Unconfirmed_ PDU**		**ErrorPDU**	
Login	Request	MliRq_ XXXPDU	X	X	X
	Indication	Mliln_ XXXPDU	X	X	X
	Confirm	MliCf_ XXXPDU	X	MliCf_ ErrPDU	X
Servicio	**Primitiva**	**Unconfirmed_PDU**			
Logout	Request	MloRq_ XXXPDU	X	X	X
	Indication	Mloln_ XXXPDU	X	X	X
	Confirm	MloCf_ XXXPDU	X	X	X

Tabla 15 Matriz para obtener los tipos de mensajes del protocolo.

4.3.6 FORMATO DE MENSAJES

El formato de cada mensaje está bien determinado por la especificación de manufactura que se puede estudiar en [27]. Sin embargo, para el modelado del protocolo se debe incluir solamente los parámetros que pueden modificar el comportamiento del modelo, todos los parámetros adicionales son omitidos y únicamente se usarán al tiempo de implementación. El análisis del formato se plantea por tipo de servicio del protocolo y es detallado a continuación.

4.3.6.1 *Login y logout*

En este servicio no hay parámetros adicionales que cambien el comportamiento del protocolo, debido a que estos servicios traen como consecuencia una acción general para la aplicación que los ejecuta contra la MMSPM.

4.3.6.2 Associate, Conclude y Abort

Considerando que la aplicación puede solicitar hasta "n" asociaciones a la MMSPM, se debe adicionar al formato del mensaje un entero que identifique a la asociación a la que se hace referencia en la MMSPM local. Además, para relacionar la asociación con el número con la que es conocida en el sistema remoto, se adiciona el identificador remoto de la asociación. El formato requerido es:

TipoMensaje, IdAssLoc, IdAssRmt

donde

TipoMensaje := Cualquier tipo de mensaje relacionado con el *Associate, Conclude* o *Abort.*
IdAssLoc := número de la asociación en la entidad local.
IdAssRmt := número de la asociación en la entidad remota.

4.3.6.3 Servicio No Confirmado

Este servicio sólo se puede llevar a cabo una vez asociadas las entidades y está siempre relacionado a una de ellas, por lo que se requiere de los dos identificadores mencionados en el punto anterior.

4.3.6.4 Servicio Confirmado

Al igual que para el servicio no confirmado, para éste se requiere indicar con qué asociaciones está ligado; pero además, este servicio puede dejar pendientes hasta "k" peticiones y hasta "j" indicaciones, lo que conlleva a agregar al formato un identificador de la petición pendiente a la

que corresponde el PDU en la MMSPM local y también el identificador correspondiente a la indicación en el MMSPM remoto; el formato para este tipo de mensaje es:

TipoMensaje, IdAssLoc, IdAssRmt, IdReqLoc, IdIndRmt

donde

TipoMensaje := Cualquier tipo de mensaje relacionado con el Servicio Confirmado
IdAssLoc := número de la asociación en la entidad local.
IdAssRmt := número de la asociación en la entidad remota.
IdReqLoc := número de la petición en la entidad local.
IdIndRmt := número de la indicación en la entidad remota.

4.3.6.5 *Reject*

Los parámetros para rechazar un PDU dependen de qué tipo sea el PDU a rechazar, pero en el peor de los casos se requieren los parámetros mencionados en el punto 0.

4.3.6.6 Formato General

Aunque en el modelo de validación se puede manejar un formato para cada tipo de mensaje, se decidió generalizar este formato con el que incluye el mayor número de parámetros para una comprensión más fácil del lector del modelo del protocolo. Los parámetros no utilizados se llenan con un valor cualquiera.

El formato general utilizado en el modelo es:

TipoMensaje, IdAssLoc, IdAssRmt, IdReqLoc, IdIndRmt

donde

TipoMensaje := Cualquiera de los 35 numerados en el punto 4.3.5.
IdAssLoc := número de la asociación en la entidad local.
IdAssRmt := número de la asociación en la entidad remota.
IdReqLoc := número de la petición en la entidad local.
IdIndRmt := número de la indicación en la entidad remota.

5 VALIDACIÓN DEL PROTOCOLO

Una vez determinadas las funciones que tiene que desarrollar el protocolo y plantear los requerimientos que debe satisfacer debido al medio ambiente en el que se va a implementar, el siguiente paso es validarlo. Para ello es necesario modelar cada una de las funciones del protocolo y plantearlo a través de un modelo de validación, el cual se describe a continuación y cuyo código puede ser consultado en el Apéndice B.

5.1 MODELO DE VALIDACIÓN

El modelo presentado para la validación del protocolo propuesto consiste en cuatro procesos, además del proceso iniciador *init*, que define los tipos de mensaje y lanza en paralelo los procesos principales (refiérase al punto 3.2.6.2).

Los dos procesos más importantes son **Smmspm** y **Cmmspm** que modelan las máquinas del protocolo MMS en el servidor y en el cliente respectivamente. Estos son los procesos donde se centra la validación del protocolo. Los procesos Cupper y Supper modelan el comportamiento general que tendrán las aplicaciones del cliente y el servidor que usen el protocolo MMS. El esquema general de las interacciones de estos procesos se muestra en la Figura 39 y se detallan en los próximos puntos.

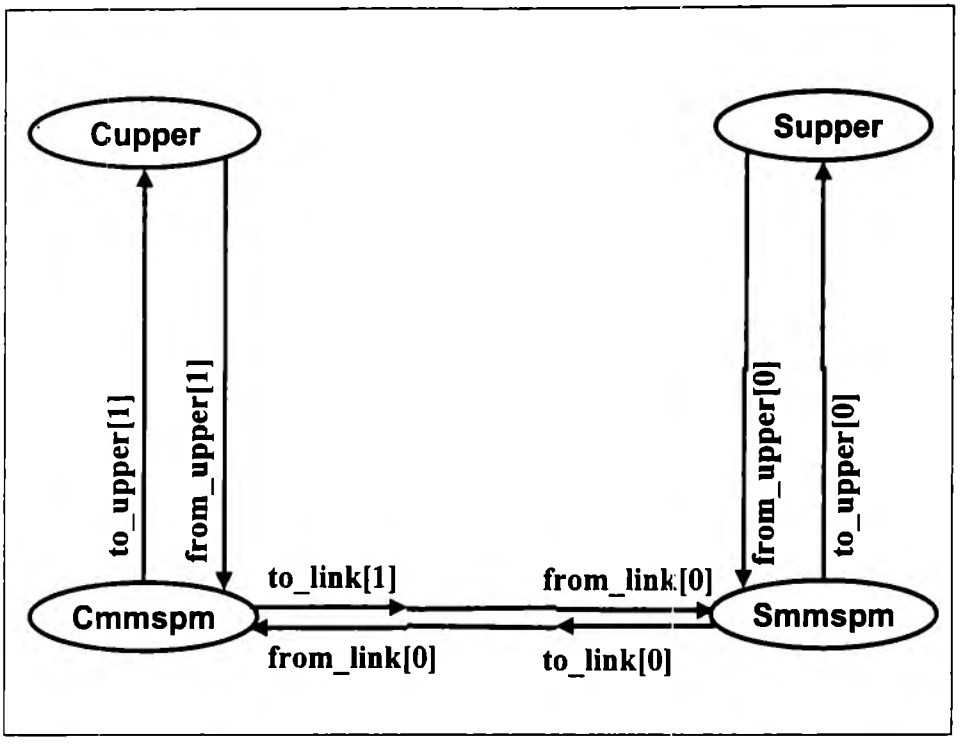


Figura 39 Esquema General del Modelo de Validación para el protocolo MMS.

5.1.1 MÁQUINA DEL PROTOCOLO MMS CLIENTE

El proceso **Cmmspm** representa y modela las funciones que tiene a cargo la máquina del protocolo MMS para la entidad cliente. Este proceso recibe información del cliente y, de acuerdo a las reglas preestablecidas por el protocolo, regresa o no algún tipo de mensaje. En la Figura 40 se aprecia el diagrama general de este proceso y se describe su funcionamiento a continuación.

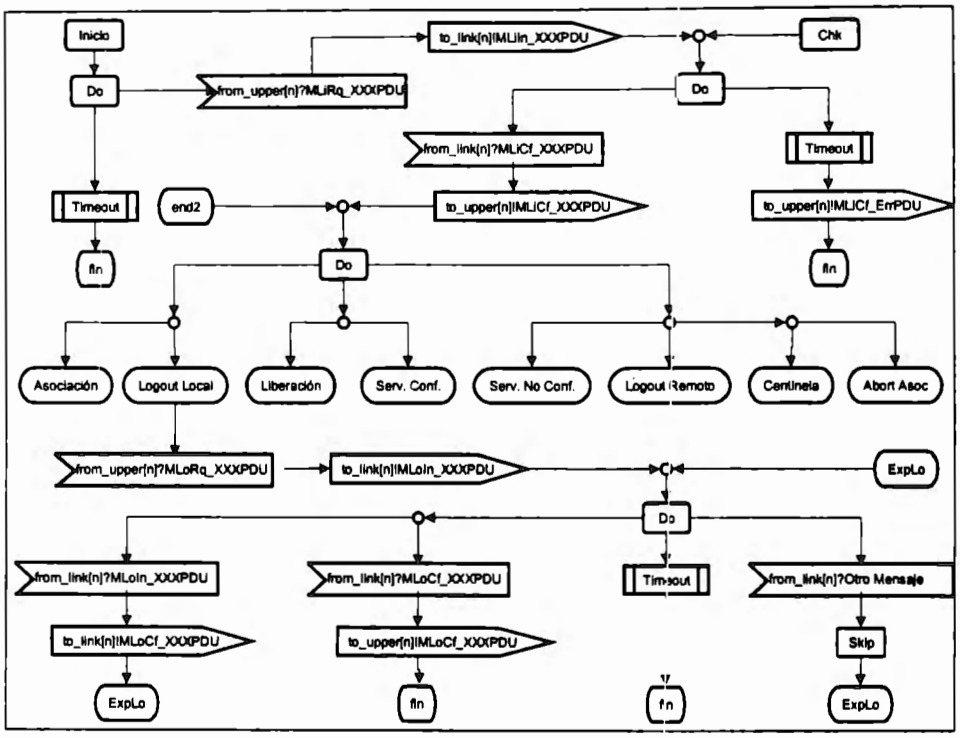


Figura 40 Modelo de Validación General de la MMSPM para la endtidad cliente

5.1.1.1 Estado Inicial

Una vez hecha la inicialización de las variables, el proceso espera por la petición de *Login* que el cliente debe enviarle (ver Figura 40); si ésta no llegase por alguna razón y el temporizador se agotara, la máquina del protocolo terminaría su ejecución.

Recibida la petición de *Login*, **Cmmspm** trata de comunicarse con su similar en la entidad del servidor enviándole una indicación de *Login*; con esta acción la máquina pasa a un estado etiquetado con "chk" en la figura antes mencionada, en el cual checa si la máquina remota está levantada esperando recibir una confirmación de *Login*.

5.1.1.2 Estado Check

Si la confirmación de *Login* no llega por alguna circunstancia y el temporizador se agota, el proceso **Cmmspm** enviaría a su capa superior, un error de *Login* y terminaría su ciclo de vida.

Al recibir la confirmación de *Login*, **Cmmspm** transfiere esta confirmación a su capa superior indicándole con esta acción que sí existe el servidorm y entra al estado Conectado, marcado en la Figura 40 como "end2". En este estado, la mmspm es capaz de dar los servicios de: Asociación, Liberación de Asociación, Abortar Asociación, Servicio Confirmado, Servicio No confirmado, Logout Remoto, Logout Local y el servicio inherente del Centinela, marcados con las respectivas etiquetas en la figura mencionada y descritos a continuación.

5.1.1.3 Estado Conectado

Si cmmspm recibiera en este estado una petición de *Logout* por parte de la entidad local, enviaría una indicación de este servicio a la entidad remota y esperaría por la respuesta de ésta. Este estado está etiquetado como "ExpLo" en la Figura 41. Si se recibe una indicación de *Logout* remoto, cmmspm eliminaría todas las asociaciones que tuviera pendientes y enviaría una confirmación de *Logout* a la máquina remota entrando así a un estado *StandAlone*. Para eliminar todas las asociaciones, la mmspm sigue el flujo detallado a continuación y mostrado en la figura antes mencionada.

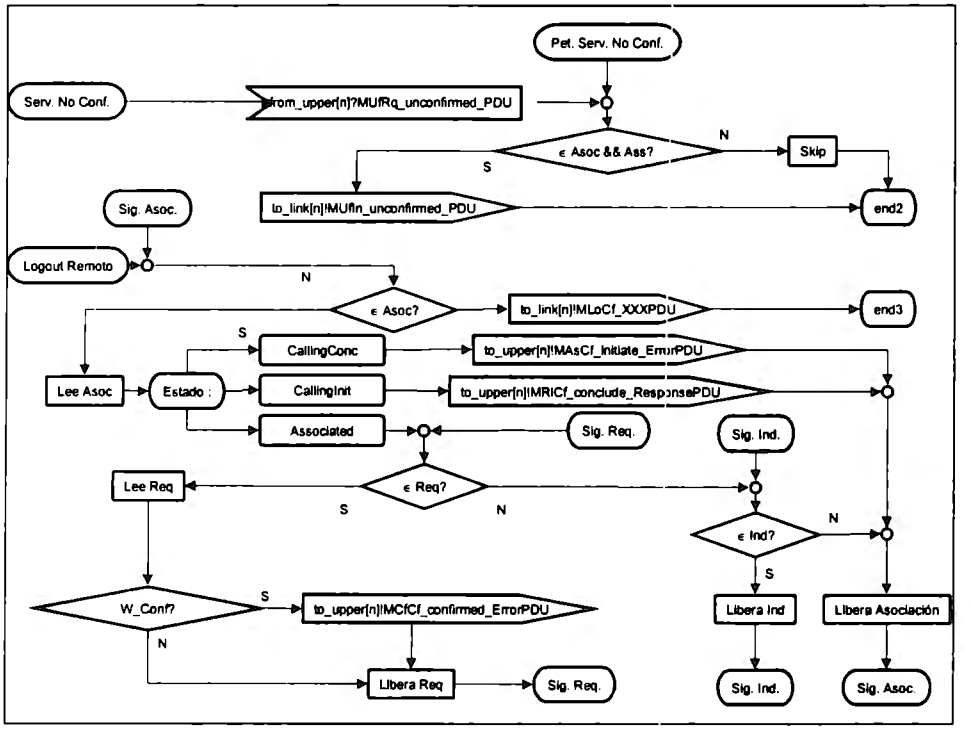


Figura 41 Modelo de Validación Cmmspm: Servicios *Logout* Remoto y Servicio No Confirmado.

Si el estado de la asociación es *Calling Conc*, significa que cmmspm está esperando la confirmación para concluir esta asociación, por lo que antes de eliminarla envía un mensaje de error de conclusión a la capa superior. Hace algo similar si el estado de la asociación es *Calling Init*, con la diferencia de que el mensaje enviado es un error de iniciación de asociación. Si el estado es *Associate*, elimina todas las peticiones e indicaciones pendientes, y después libera la asociación.

Para eliminar una petición pendiente, la máquina verifica que el estado de ésta sea *wait_conf;* si es así, envía un mensaje de error de servicio confirmado a la capa superior. Posteriormente libera la petición.

A diferencia de la eliminación de una petición, al liberar una indicación la máquina no genera mensaje alguno.

5.1.1.4 Estado ExpLo

En este estado (Ver Figura 40), la máquina del cliente espera por una confirmación de *Logout* y al llegar ésta, la retransmite a su capa superior y da por terminada su ejecución. Si el mensaje esperado no llega antes de que se agote el temporizador, cmmspm da como finalizado el servicio y transmite a la capa superior un error para que ésta decida si se aborta completamente o no el *Login*.

Estando en este estado, cmmspm probablemente recibe otros mensajes provenientes de la entidad remota, debido a que éstos estaban en tránsito; tales mensajes deben ser omitidos y sólo se responderá a una indicación de *Logout* remota, si llega.

5.1.1.5 StandAlone

En este estado, la máquina sólo espera que la capa superior genere la petición de *Logout*, por lo que todas las peticiones diferentes a ese servicio serán contestadas inmediatamente con un error del servicio solicitado. Una vez llegada este tipo de petición, la máquina envía una confirmación de *Logout* y termina su ejecución.

5.1.2 APLICACIÓN CLIENTE

El proceso Cupper modela el comportamiento que la aplicación debe guardar para poder comunicarse adecuadamente con la Máquina del Protocolo MMS Cliente descrita en el punto 5.1.1. El esquema de este proceso se muestra en la Figura 42 y en la Figura 43, descritas en los siguientes párrafos.

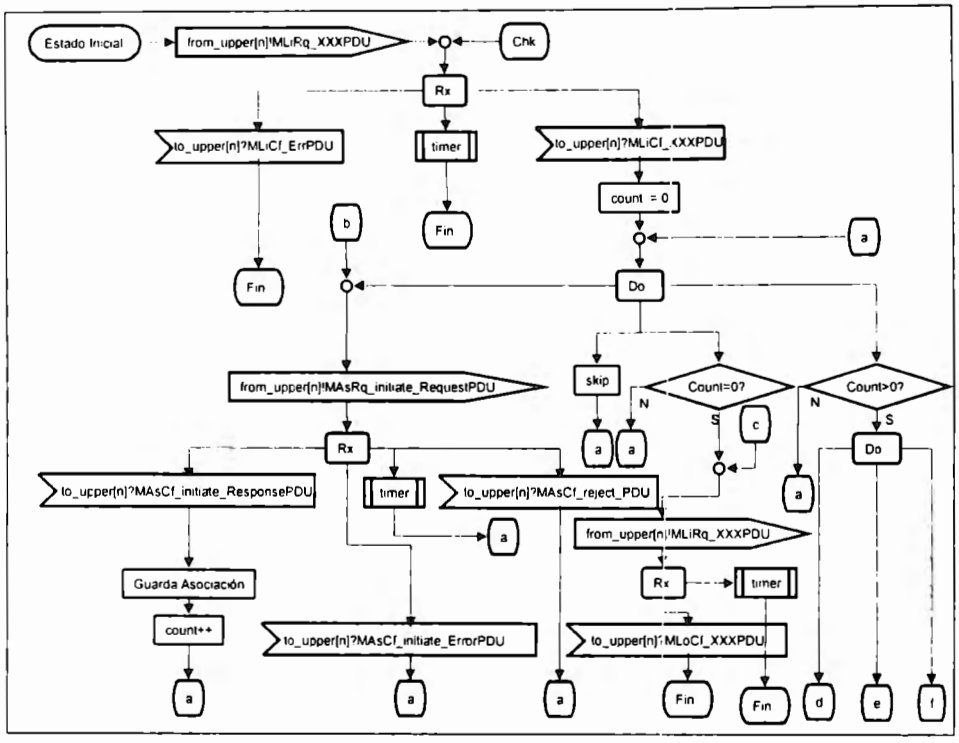


Figura 42 Diagrama de Flujo del Modelo Aplicación Cliente I.

5.1.2.1 Estado Inicial

Una vez inicializadas las variables locales, Cupper envía un mensaje a la MMSPM local para indicar que requiere conectarse a la entidad remota, esto es, envía una primitiva *LoginRequest* y pasa a un estado "Check".

5.1.2.2 Estado Check

En este estado Cupper espera por un mensaje tipo *LoginConfirm*, lo cual indica que la máquina remota existe y está lista para establecer la comunicación, lo que lleva a este proceso a un estado *Ready*. Es importante recordar que esta parte del protocolo es adicional al MMS y tiene como objetivo verificar únicamente la existencia de la máquina remota.

Si en lugar de un *LoginConfirm* se recibe un *LoginConfirmError*, la máquina daría por hecho que es imposible comunicarse con su similar y abortaría cualquier intento por comunicarse. El mismo comportamiento se observa si el *LoginCofirm* no llega antes de que el tiempo predeterminado de espera concluya.

5.1.2.3 Estado Ready

Al entrar a este estado (etiquetado como "a" en la Figura 42), Cupper inicializa un contador de asociaciones en 0 y en este momento, la aplicación es capaz de hacer cualquiera de las siguientes

acciones: solicitar una asociación, dar de baja una asociación, solicitar un servicio confirmado, solicitar un servicio no confirmado o notificar su salida de la red. Además de estas acciones, que se describen en los párrafos siguientes, se agrega una en la que sólo figura la instrucción *SKIP*, esto para simular que la aplicación puede ejecutar cualquier otra instrucción que no tenga relación con el protocolo pero que implica que está ocupada. Lo anterior es importante para que el validador pueda encontrar todos los estados posibles más apegados a la realidad.

5.1.2.4 Solicitando Asociación

Este estado está marcado con la etiqueta "b" en la figura mencionada e inicia enviando un mensaje de *PeticionAsociacion*. Aunque en la práctica, en este mensaje viajan los valores negociables de MMS (refiérase a la sección 3.1.3.2) que son determinantes para que las entidades acuerden si se pueden o no asociar, en el modelo de validación no se requieren ya que las dos alternativas son simuladas por la entidad remota contestando aleatoriamente con las primitivas *RespuestaAsociacion* y *RespuestaAsociacionError*.

Cupper espera en este estado hasta que recibe un mensaje de su máquina local al que responde con una acción determinada y va al estado "Ready".

Si recibe una *RespuestaAsociacion*, significa que la máquina local y la remota han registrado la asociación exitosamente y puede proceder a solicitar peticiones de servicio confirmado y de servicio no confirmado. La aplicación guarda el identificador de la asociación e incrementa el número de asociaciones conseguidas *count* en una unidad. *Count* le sirve a la máquina para saber si tiene al menos una asociación establecida para que pueda solicitar los servicios que requieren que así sea.

La recepción de la primitiva *RespuestaAsociacionError* significa que alguna de las dos máquinas no tuvo éxito al registrar la asociación, por lo que la aplicación no podrá solicitar servicios utilizando esta asociación. En la implementación del protocolo, este mensaje lleva consigo el tipo de error y en qué máquina se produjo, tal y como lo requiere MMS.

Si se recibe un *RechazoAsociación*, significa que el PDU de petición de asociación no fue llenado correctamente, por lo que la aplicación debe corregirlo si quiere que procesarla. En este PDU viaja más información para indicar cuál es el defecto que tiene el PDU, esto se puede observar solo en la implementación, ya que esta información es irrelevante para el comportamiento del modelo.

5.1.2.5 Firmando la Salida

Para registrar la salida, el número de asociaciones conseguidas debe ser cero (si se trata de una salida normal), aunque puede ser que la aplicación requiera firmar la salida sin haber dado de baja sus asociaciones apropiadamente, esto está contemplado por la MMSPM local (refiérase a la punto 5.1.1).

Para iniciar la salida Cupper envía una *PeticionLogout* y espera por una *ConfirmacionLogout*, después de la cual termina su ciclo de vida.

Si *ConfirmacionLogout* no llega antes de que expire el temporizador preestablecido, la aplicación puede asumir que no existe la máquina remota y por lo tanto terminar su ejecución.

5.1.2.6 Servicio Confirmado

Teniendo al menos una asociación, la aplicación puede solicitar un servicio confirmado (Etiqueta "d", Figura 43) transmitiendo el mensaje *PeticionServConf* indicando a través de cuál asociación será efectuado el servicio. En la implementación este mensaje lleva, además, los datos del servicio requerido y parámetros de entrada/salida correspondientes.

Hecha la petición, Cupper espera por alguno de los mensajes siguientes:

- *RespuestaServConf.*- Si se recibe esta primitiva, entonces el servicio se efectuó exitosamente y el mensaje contiene los valores de las variables de regreso y demás información útil.
- *ErrorServConf.*- Este mensaje indica que el servicio no fue completado y en su cuerpo se describen las razones y la entidad en la que se generó el error. Para la validación esta información no es relevante pero sí para la implementación.
- *RechazoPDU.*- Este mensaje indica que el PDU de servicio confirmado no se envió adecuadamente y el cuerpo del mensaje indica el porqué.

Si ninguno de los tres mensajes no llega antes de que el temporizador se agote, entonces Cupper asimila el servicio como un *ErrorServConf.* Cualquier que sea el caso, Cupper regresa al estado *ready* después de tomar las acciones mencionadas.

5.1.2.7 Servicio No Confirmado

Como en el caso anterior, la aplicación requiere tener una asociación establecida antes de solicitar el servicio, si no es así la máquina simplemente omite la solicitud.

Esta sección del protocolo es muy simple, sólo se envía la *PeticionServNoConf* a la máquina local que contiene el identificador de la asociación por donde se desea efectuar el servicio y, en la implementación, los datos del servicio y parámetros correspondientes. Una vez enviado este mensaje, Cupper puede regresar inmediatamente al estado *Ready* ya que este servicio, como su nombre lo indica, no requiere de confirmación.

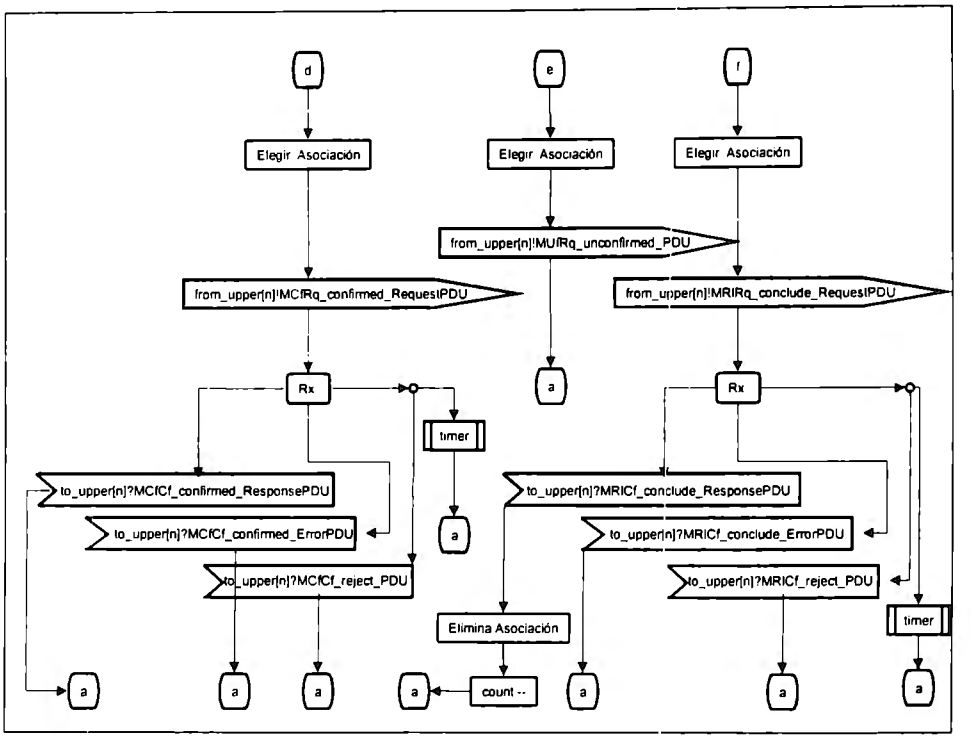


Figura 43 Diagrama de Flujo del Modelo Aplicación Cliente II.

5.1.2.8 Dando de Baja una Asociación

Esta acción sólo tiene sentido si la asociación existe, es decir, si el número de asociaciones conseguidas por la aplicación es mayor a cero (*count* > 0), sin embargo, si la aplicación trata de dar de baja una asociación no existente, la máquina local genera una respuesta positiva que indica que la asociación ha sido dada de baja, ya que realmente no existe (refiérase a la sección anterior).

Para dar de baja una asociación (etiqueta "f" en la Figura 43), Cupper envía la primitiva *PeticionConclude* y espera por la recepción del mensaje *RespuestaConclude*, el cual indica una baja exitosa de la asociación, procediendo a eliminarla de sus registros locales y a decrementar el contador de asociaciones conseguidas.

Si en lugar de recibir *RespuestaConclude* recibe *ConcludeError*, simplemente regresa al estado *ready*, considerando que esta asociación aún existe.

En este estado también se puede dar el caso de que el PDU de *PeticionConclude* sea rechazo, lo que es indicado a Cupper mediante la primitiva *RechazoConclusion*.

5.1.3 MÁQUINA DEL PROTOCOLO MMS SERVIDOR

El proceso **Smmspm** es el encargado de modelar el comportamiento de la máquina del protocolo MMS en la entidad servidor.

Smmspm interactúa con su proceso superior, el servidor, y con la máquina MMS de la entidad remota; atiende las peticiones y respuestas del servidor y las transmite a la máquina remota, o bien, rechaza estas primitivas regresando una respuesta al servidor. Además, esta máquina atiende las indicaciones que provienen de la entidad cliente y es encarga de transferir las confirmaciones de los servicios, cuando esto se requiere.

Aunque el comportamiento de esta máquina es análogo a la máquina en el cliente, difiere en algunos aspectos ya que no hay que olvidar que se trata de un protocolo asimétrico. La Figura 44 muestra el diagrama general de este proceso y las figuras subsecuentes en esta sección corresponden al detalle de este modelo de validación que a continuación se explica.

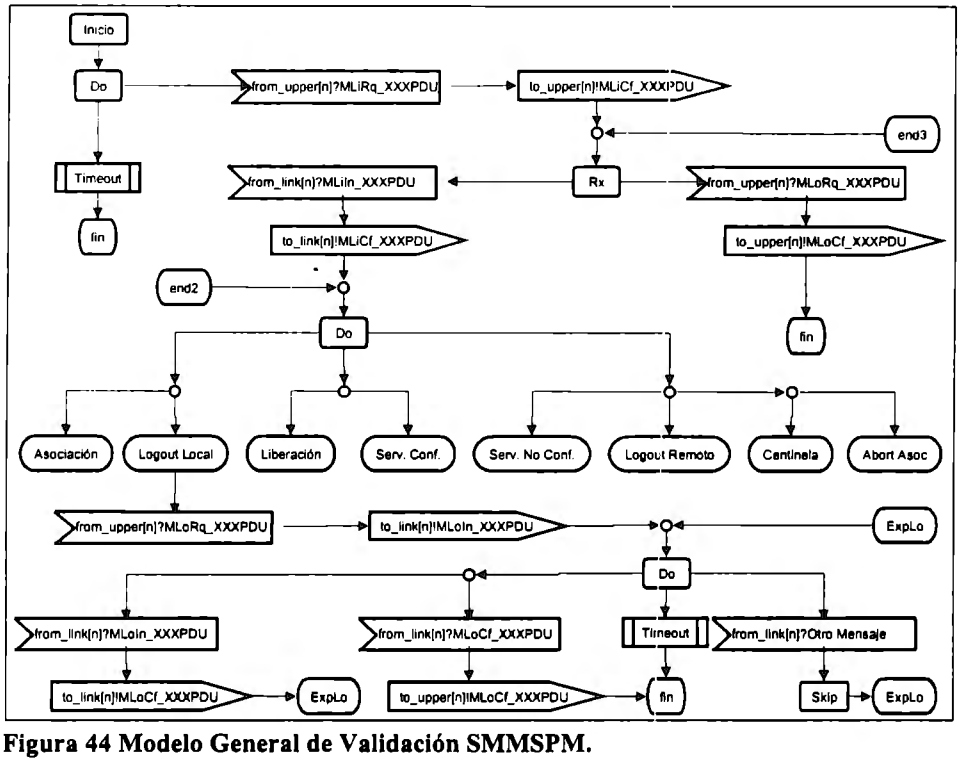


Figura 44 Modelo General de Validación SMMSPM.

5.1.3.1 Estado Inicial

Smmspm inicializa las variables locales y espera por una *PeticionLogin* por parte del proceso servidor. Si ésta no llega antes de que se agote el temporizador preestablecido correspondiente, la máquina daría por hecho que no existe el proceso y terminaría su ejecución.

En el caso contrario, si la petición llega, entonces **Smmspm** pasa a un estado llamado *StandAlone*.

5.1.3.2 Estado StandAlone

La principal característica de este estado es que **Smmspm** ha registrado al servidor local, pero aún no tiene registrado al cliente. Este es un estado final válido (ver 3.2.6.6.3), es decir, el modelo puede permanecer aquí sin límite de tiempo. Para el validador Xspin esto se debe marcar con la etiqueta "end" y por ello este estado se puede localizar en la Figura 44 con la etiqueta "end3".

En este estado, **Smmspm** está escuchando mensajes que provienen del servidor local y de la máquina MMS remota. Si llega una *Peticiónlogout* por parte del servidor, la máquina le responde inmediatamente con una primitiva *ConfirmacionLogout* y termina su ejecución; esto lo puede hacer sin más acciones ya que no tiene registrado al cliente.

Si recibe una *IndicacionLogin* por parte de la máquina remota, entonces le envía a ésta una primitiva *ConfirmacionLogin* y pasa a un estado *conectado*.

5.1.3.3 Estado Conectado

En este estado **Smmspm** tiene registrado tanto al servidor como al cliente remoto y está listo para atender las peticiones de este último; Este estado se encuentra marcado con la etiqueta "end2" y es un estado final aceptable ya que la máquina puede permanecer indefinidamente aquí. En esta situación **Smmspm** puede atender los servicios de Logout por parte del servidor, servicios de Asociación, servicios de Liberación de Asociación, Servicios No Confirmados y Servicios Confirmado y además, brinda el servicio de Centinela, todos estos descritos en los párrafos restantes de esta sección.

5.1.3.4 Servicio de Logout Local

Este servicio inicia cuando **Smmspm** recibe una primitiva del tipo *PeticionLogout* que genera el proceso Servidor cuando pretende dar por terminado su ciclo de vida. El primer evento que genera el **Smmspm** es el envío de la primitiva *IndicacionLogout* a la máquina del protocolo del cliente y después espera en el estado marcado como *ExpLo* (ver Figura 44). En este estado la máquina espera por la primitiva *ConfirmacionLogout* y al llegar ésta, la retransmite al Servidor para indicarle que puede terminar su ejecución y ella hace lo mismo.

Smmspm puede recibir otras primitivas que la máquina remota pudo generar antes de emitir la *ConfirmacionLogout*; si la primitiva es del tipo *IndicacionLogout* significa que la máquina remota también pretende terminar su ejecución, por lo que **Smmspm** envía una *ConfirmacionLogout* hacia el Cliente y regresa a su estado *ExpLo*. Si la primitiva es de cualquier otro tipo, **Smmspm** simplemente la omite y permanece en *ExpLo*.

Si **Smmspm** no recibe la primitiva *ConfirmacionLogout* antes de que su temporizador expire, entonces da por hecho que la máquina remota no existe y genera una *ConfirmacionLogout* que envía al Servidor, terminando su ejecución.

5.1.3.5 Servicio Indicación de Asociación

Este es el primer servicio MMS que solicita la entidad cliente cuando quiere inicializar propiamente la conversación con el Servidor. En este mensaje viajan los parámetros negociables de la asociación por la parte del cliente; el servidor compara estos valores con sus parámetros correspondientes y determina los valores que serán usados para la asociación. Estos parámetros de negociación son mencionados en la sección 3.1.3.

La recepción de este tipo de mensaje obliga a **Smmspm** a entrar en un estado marcado en la Figura 45 como "IndAsociacion". La primera tarea que realiza el proceso es verificar si esta asociación existe, lo cual puede suceder si esta indicación es producto de una retransmisión; si no es así, lo cual es lo más común, el proceso checa que exista espacio disponible en su tabla de asociaciones. Si encontró espacio, registra esta asociación con estado *CalledInit*, retransmite al proceso Servidor la indicación de asociación y regresa al estado *end2*. Si el proceso no encuentra espacio para la nueva asociación, responde a la entidad remota con un mensaje *ConfirmacionAsociacionError* regresando al estado *end2*.

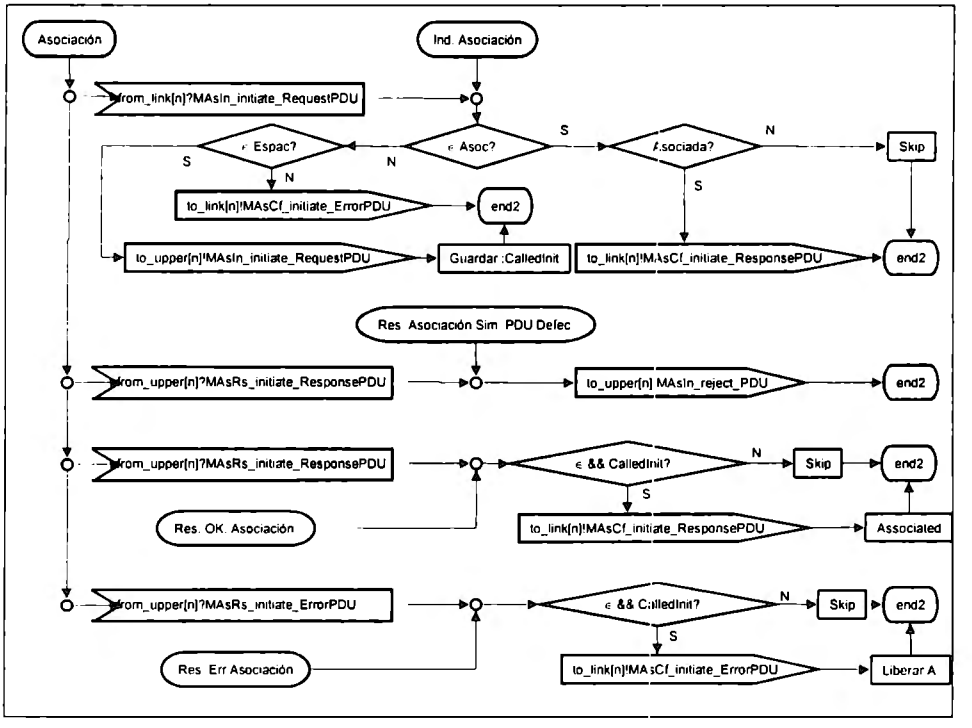


Figura 45 Modelo de Validación SMMSPM: Servicio de Asociación.

Por otro lado, si la asociación ya esta registrada y su estado es *Associated*, entonces **Smmspm** responde al cliente que la asociación esta completa con la primitiva *ConfirmacionAsociacion*. Si la asociación existe pero aún no está asociada, **Smmspm** simplemente omite este mensaje. En cualquiera de los dos casos anteriores, **Smmspm** regresa al estado estable *end2*.

5.1.3.6 Servicio Respuesta de Asociación Afirmativa

Este servicio es solicitado por la capa superior de **Smmspm**, el proceso Servidor, y lo hace a través de la primitiva *RespuestaAsociacion*, lo cual obliga al proceso **Smmspm** a ir al estado marcado como "ResOKAsociacion" en la figura antes mencionada. Después de haber recibido este mensaje, el proceso verifica que la asociación involucrada exista y que su estado sea *CalledInit*. Si es así, la marca como *Associated*, envía a la entidad solicitante una respuesta positiva de confirmación usando la primitiva *RespuestaConfirmacion* y regresa al estado *end2*. Si la condición mencionada no se cumple, la máquina omite este mensaje y va al estado *end2*.

5.1.3.7 Servicio Respuesta de Asociación Negativa

Este servicio ocurre cuando el servidor no puede atender la asociación y lo indica enviando la primitiva *RespuestaAsociacionError* a **Smmspm**. A la recepción de este mensaje, **Smmspm** entra al estado etiquetado como "Resp. Err. Asociación" de la Figura 45 y verifica que la asociación exista y esté marcada como *CalledInit*. Si la condición se cumple, **Smmspm** elimina la asociación de la tabla local y envía el mensaje *ConfirmacionAsociacionError*. Si no se cumple simplemente omite el mensaje recibido. En ambos casos el proceso regresa al estado final aceptable *end2*.

5.1.3.8 Servicio Respuesta de Asociación con PDU defectuoso

Este servicio modela la situación que no es usual pero puede suceder, cuando el Servidor envía la respuesta de la asociación, negativa o afirmativa, y **Smmspm** encuentra datos erróneos al procesarla. **Smmspm** responde a este servicio con la primitiva *IndicacionAsociacionRechaza* y regresa al estado *end2*. Este servicio está etiquetado como "Resp. Asoc. Sim. PDU Defectuoso" en la figura antes citada.

5.1.3.9 Servicio Indicación Liberación de Asociación

Este servicio es generado por la entidad cliente para notificar al servidor que requiere dar de baja una asociación y se lo indica mediante la primitiva *IndicacionConclusion*. Este servicio está etiquetado en la Figura 46 como "Ind Liberación", donde se aprecia que el paso siguiente a la recepción del mensaje citado, es verificar que exista la asociación. Si existe y está asociada, **Smmspm** la marca como *CalledConc*; si existe pero no está asociada, omite el mensaje; por último, si no existe regresa una confirmación exitosa de la conclusión a la entidad cliente. Cualquiera que sea el caso, el protocolo regresa al estado de donde partió, el *end2*.

5.1.3.10 Servicio Respuesta de Liberación Afirmativa

El servidor genera la primitiva *RespuestaConclusion* cuando la asociación se puede liberar sin problema alguno y la envía a **Smmspm**; al recibir este mensaje, se verifica que la asociación exista y esté en estado *CalledConc*, si es así se libera la asociación y si no, simplemente se omite el mensaje. Cualquiera que sea la situación de la asociación, el paso final de **Smmspm** es regresar al estado *end2*.

5.1.3.11 Servicio Respuesta de Liberación Negativa

Smmspm detecta que el servidor tuvo problemas en el proceso de liberación de asociación, si recibe la primitiva *RespuestaConclusionError* (ver Figura 46). Al recibir este mensaje, **Smmspm** verifica la existencia de la liberación y que esté marcada como *CalledConc*; si es así la marca como *Associated* y regresa al estado end2. Si la asociación no existe o si no está marcada como *CalledConc*, entonces la máquina omite el mensaje y se coloca en el estado final end2.

5.1.3.12 Servicio Respuesta de Liberación con PDU defectuoso

Este servicio, al igual que el de *RespuestaAsociacion* con PDU defectuoso, es un modelado de la recepción de una respuesta, afirmativa o negativa, a la liberación de asociación por parte del servidor, pero con un PDU que no puede ser procesado por la máquina del protocolo. La reacción a este servicio es el envío de un mensaje de rechazo hacia el Servidor: *IndicacionLiberacionRechazo*. Esta parte del protocolo se observa en la Figura 46 en la parte etiquetada como "Resp. Liberación PDU defectuoso".

5.1.3.13 Servicio Indicación Servicio Confirmado

Cuando la Mms del cliente requiere un servicio confirmado (refiérase al punto 5.1.1.3), genera la primitiva *IndicaciónServConf* que es procesada por **Smmspm** como a continuación se describe y como se puede observar en la Figura 47 en el punto marcado como "Ind. Serv. Conf.". Primeramente, **Smmspm**, verifica que exista la asociación y que esté marcada como *Associated*, si no es así regresa al cliente un el mensaje *ConfirmaciónServConfError*; si se cumple la condición anterior, verifica que no esté registrada la indicación en la tabla local de indicaciones, si ya está registrada simplemente omite el mensaje. Después de la anterior validación busca espacio para la indicación, si no hay regresa un mensaje del tipo *ConfirmaciónServConfError*, pero si hay espacio registra la indicación con un estado *W_Resp*. Cualquiera que sea el caso, **Smmspm** regresa a *end2* después de haber atendido este servicio.

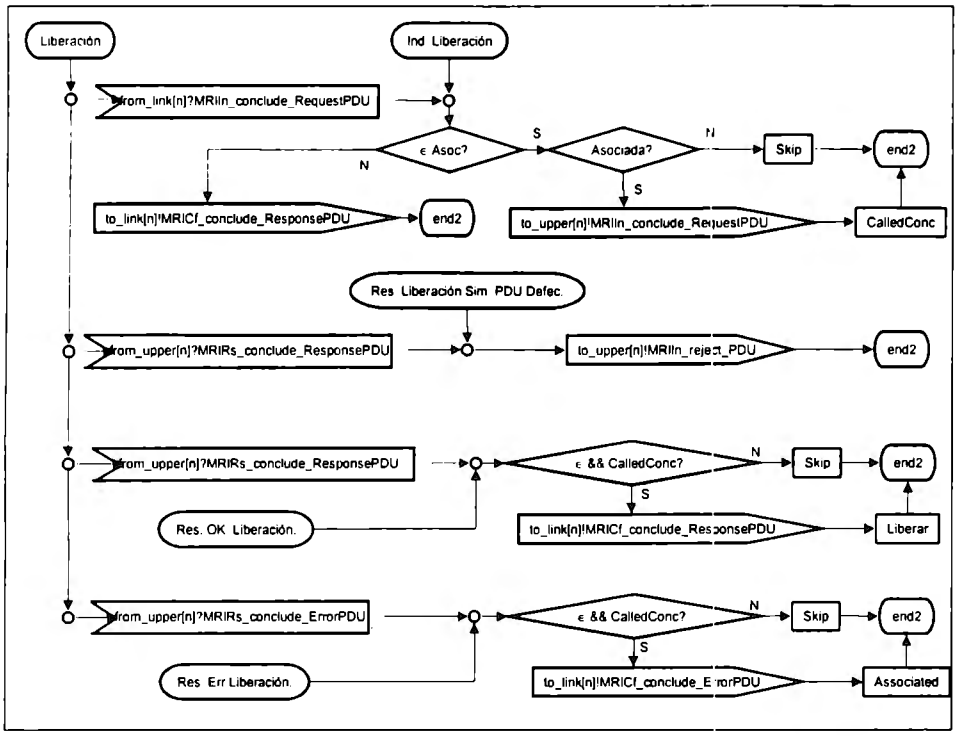


Figura 46 Modelo de Validación SMMSPM: Servicio de Liberación de Asociación.

5.1.3.14 Servicio Respuesta Servicio Confirmado Afirmativa o Negativa

Este tipo de mensaje es enviado por el servidor y contiene el resultado de la operación solicitada por el cliente, por ejemplo, el estado de un semáforo, valores de variables, posición de la herramienta del robot, etc., si es afirmativa y la razón de la falla si es negativa. Al recibir **Smmspm** esta primitiva, verifica que la indicación exista en la tabla correspondiente y esté marcada como *W_Resp* en la correspondiente tabla (ver Figura 47), si es así, la da de baja y transmite hacia el cliente el PDU recibido cambiando la primitiva a *ConfirmaciónServConf.* Si alguna de las condiciones no se cumple, entonces **Smmspm** omite el mensaje. En todo caso, **Smmspm** regresa a *end2* al terminar de atender este servicio.

5.1.3.15 Servicio Respuesta Servicio Confirmado con PDU defectuoso

Este servicio emula la situación que se presenta cuando el servidor ha generado un mensaje *RespuestaServConf* pero por alguna razón **Smmspm** no puede procesar el PDU y entonces tiene que rechazarlo; para hacer esto, genera un mensaje *IndicacionServConfRechazo.* Refiérase a la figura mencionada en el párrafo anterior en el punto marcado como "Resp. Serv. Conf. PDU defectuoso".

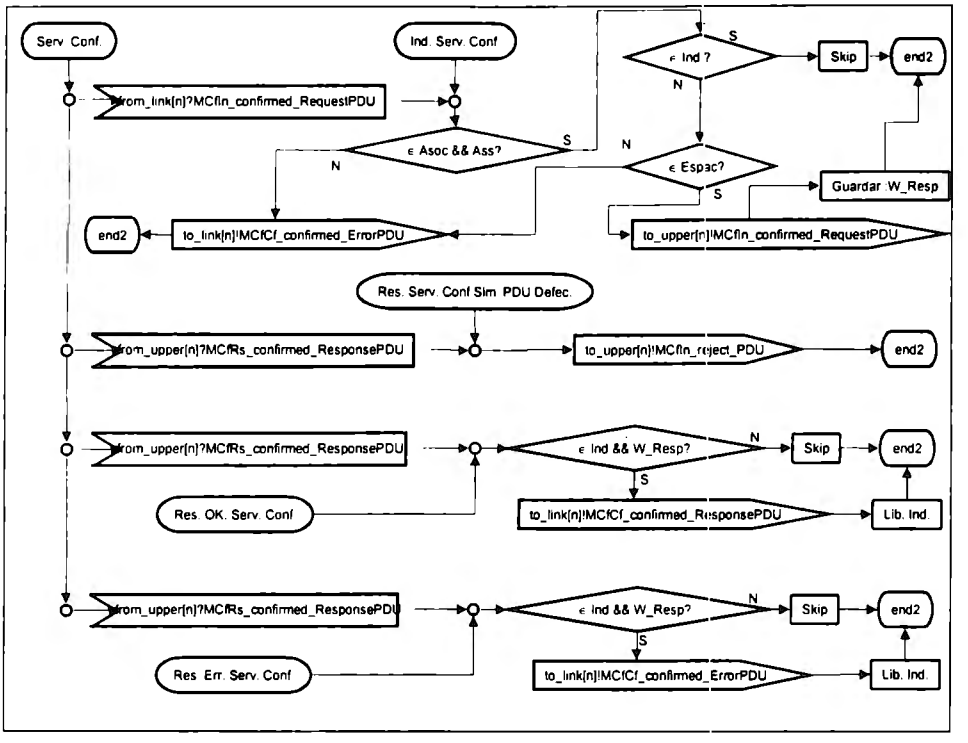


Figura 47 Modelo de Validación SMMSPM: Servicio Confirmado.

5.1.3.16 Servicio Indicación Servicio No Confirmado

Para los servicios no confirmados, a diferencia de los servicios confirmados, no se requiere de registrar el estado dentro del protocolo, lo que los hace sumamente más sencillos de procesar. La *IndicaciónServNoConf* la recibe **Smmspm** de la entidad remota y si existe la asociación indicada en el PDU y está marcada como *Associated*, la retransmite hacia el servidor. Si las condiciones anteriores no se cumplen, la máquina omite el mensaje. En la Figura 48 se observa el esquema de este servicio donde se aprecia que su ultima acción es regresar al estado *end2*.

5.1.3.17 Servicio de Logout Remoto

Este servicio sucede cuando el Servidor desea abandonar su operación y lo notifica a **Smmspm** a través de la primitiva *LogoutRemoto*. Para que este servicio sea completado exitosamente, **Smmspm** debe liberar exitosamente todas las asociaciones existentes; después de cumplir con esta tarea envía un mensaje *ConfirmacionLogout* hacia el servidor. Para deshacerse de las asociaciones existentes, **Smmspm** toma las acciones descritas a continuación e ilustradas en la Figura 48.

Para cada asociación existente, **Smmspm** verificar que su estado sea *Associated*, si no lo es simplemente la libera; si ese es su estado, **Smmspm** libera todas las peticiones registradas en la tabla de peticiones de la forma en que se indica en el siguiente párrafo y libera todas las indicaciones eliminándolas simplemente de su tabla de indicaciones.

Cada petición es eliminada de la tabla de peticiones, pero si esta marcada como *W_Conf.* (lo que significa que se espera la confirmación del servicio que representa tal petición), entonces, envía al servidor un PDU con la primitiva *ConfirmaciónServConfError* antes de la eliminación.

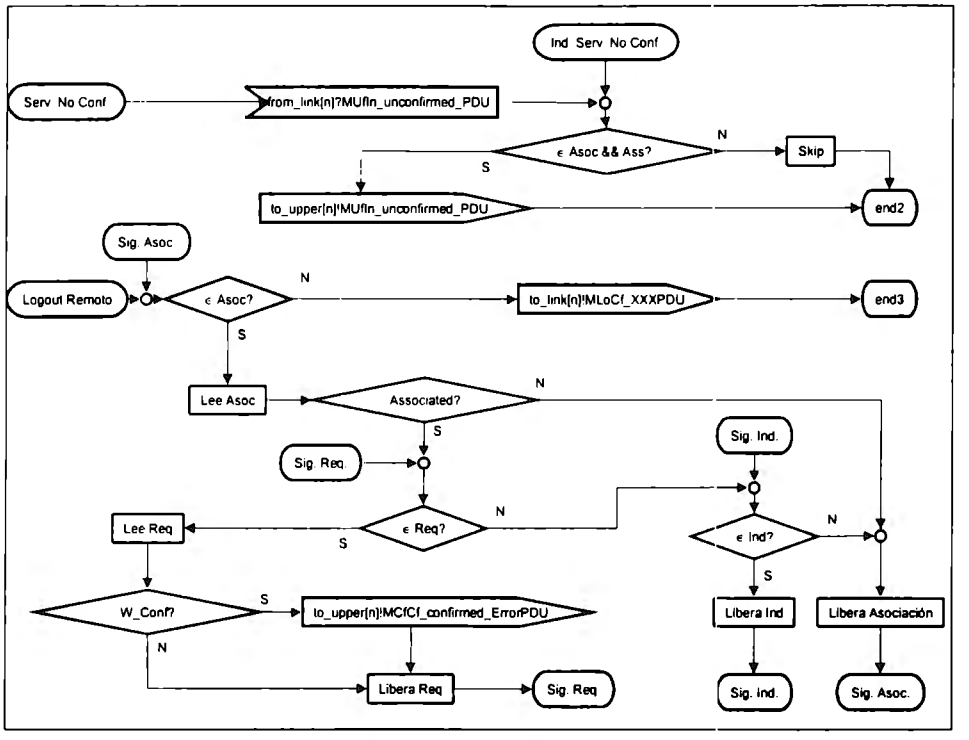


Figura 48 Modelo de Validación SMMSPM: Servicios No Confirmado y de Logout Remoto.

5.1.4 SERVICIO CENTINELA

El servicio en el que se requiere que el objeto, asociación, petición o indicación, sea confirmado por un proceso distinto al que lo está atendiendo antes de regresar una respuesta al solicitante, se presta marcándolo con un estado, según su máquina de estados. Por ejemplo, en una petición de asociación, la mmspm local la registra y la marca como *CallingInit* y envía una indicación a la máquina remota, regresando para atender otra petición. Lo anterior es con el objetivo de que la máquina trabaje de forma asíncrona respecto a su similar remota, lo cuál genera una mayor optimización en el tiempo de respuesta de las máquinas, sin embargo, la principal desventaja es que implica la existencia de algún mecanismo de mantenimiento de los estados "pendientes" de los objetos para evitar que estos se queden en tales tipos de estados. El propósito principal del proceso Centinela es, precisamente, dicho el encargado de dicho mantenimiento; Centinela es un proceso paralelo concurrente con la máquina de estados mms y, la mayoría del tiempo está inhabilitado pero cuando despierta, revisa todos los objetos, los que están esperando una respuesta son marcados como candidatos a eliminarse y envía una retransmisión del mensaje que dejo al objeto en tal estado de espera. Si Centinela encuentra que el objeto está ya marcado como candidato a eliminarse cuando hace su revisión, entonces genera el mensaje de error del servicio

y los transmite al proceso que esta esperando repuesta, eliminando el objeto de la tabla a la cual pertenece. Este servicio aplica a ambas máquinas, por ello se explica en una sección independiente a ellas.

5.1.4.1 Centinela dando mantenimiento a las Asociaciones

Centinela hace un barrido en la tabla de asociaciones, buscando aquellas que estén en estado distinto a *Associated,* es decir, que se encuentren en estado *"CallingInit", "CalledInit", "CallingConc"* o *"CalledConc"* y hace el mantenimiento de la siguiente manera.

Sí Centinela encuentra una asociación marcada como *CallingInit*, significa que ésta espera una *ConfirmacionAsociacion* de la mmspm remota, y por ello retransmite el mensaje de *IndicacionAsociacion* marcando esta asociación como candidata a eliminación. Si ya esta marcada, entonces la desaloja de la tabla de asociaciones y envía una *ConfirmacionAsociacionError* al proceso de capa superior.

Si el estado de la Asociación es *CalledInit,* significa que mmspm envió una primitiva *IndicacionAsociacion* a la capa superior, por lo que si no esta marcada como candidata a eliminarse, retransmite el mensaje y la marca. Por otro lado, si ya esta marcada, la elimina de la tabla y transmite un mensaje *ConfirmacionAsociacionError* a la mmspm remota.

Si la asociación se encuentra *CallingConc* y no ha sido marcada como candidata a eliminarse, retransmite un mensaje *IndicacionLiberacion* a la máquina remota y la marca como candidata. En caso contrario, regresa a la asociación a su estado "Associated" y envía a la capa superior una primitiva *ConfirmacionLiberacionError* para indicarle que la asociación no pudo ser purgada.

El último de los estados de espera de una asociación es cuando está CalledConc; este implica una retransmisión del mensaje *IndicacionLiberacion* a la capa superior si la asociación no ha sido previamente marcada por Centinela, ya que sí es así, Centinela regresa la asociación a estado Associated y responde a la máquina remota con *ConfirmacionLiberacionError.*

5.1.4.2 Centinela dando mantenimiento a las Peticiones e Indicaciones

Para que la existencia de una petición o indicación tenga sentido alguno, la asociación a la cual esta ligada debe estar asociada (*Associated*), por lo que al barrer la tabla de asociaciones, Centinela se detiene en las marcadas con tal estado y revisa su tabla de peticiones e indicaciones correspondientes verificando si están en estado pendiente, el cuál corresponde a *W_Conf* para las peticiones y a *W_Resp* para las indicaciones.

Si la petición esta en estado *W_Conf*, significa que la mmspm correspondiente envió previamente a su análoga remota una *IndicacionServConf.* Si la petición no ha sido aún marcada por Centinela, éste retransmite esta primitiva y la marca. De otro modo, la elimina y notifica a la capa superior el error mediante la primitiva *ConfirmacionServConfError.*

La indicación se convierte en *W_Resp* cuando la mmspm envía una *IndicacionServConf* a su capa superior, por ello, sí no es candidata a eliminarse, Centinela retransmite este mensaje y la marca. Sin embargo, si ya es candidata, entonces la desecha y notifica a la mmspm remota que el servicio confirmado fracasó valiéndose de la primitiva *ConfirmacionServConfError.*

5.1.5 PROCESO SERVIDOR

El proceso Servidor es incluido en el modelo de validación del protocolo para simular el comportamiento que tiene un Servidor MMS. Este proceso, en Promela, solo incluye las tareas concernientes al protocolo, las cuales se detallan en los siguientes párrafos.

5.1.5.1 Requerimiento Login

La tarea inicial del proceso servidor es firmarse dentro del entorno; el objetivo de esta tarea es la sincronización de este proceso con el proceso **Smmspm** y esto se lleva a cabo usando la primitiva *PeticionLogin.* Una vez emitida esta primitiva, el proceso Servidor espera en un estado etiquetado como "Rx" en la Figura 49, donde espera la primitiva *ConfirmacionLogin* para después pasar al estado *Conectado* etiquetado en la figura como "end1".

Si **Server** no recibe *ConfirmacionLogin,* antes de que se agote el tiempo predeterminado de espera o si recibe el mensaje *ConfirmacionLoginError* por parte **Smmspm**, termina su ejecución.

5.1.5.2 Estado Conectado

Este estado está etiquetado como "end" en la Figura 49 y es el estado fundamental del Servidor, ya que en este momento está disponible para atender las peticiones generadas por el cliente; estos pueden ser: Asociación, Liberación de Asociación, Servicios Confirmados y Servicios No Confirmado; Eventualmente puede pasar al estado Logout. Todos estos servicios se describen en los siguientes párrafos.

5.1.5.3 Servicio Indicación de Asociación

Este servicio es detectado por el servidor al recibir la primitiva *IndicacionAsociación* la cual llega con el PDU que contiene los parámetros negociables para una asociación del tipo MMS (refiérase al punto 3.1.3.2). La respuesta a esta indicación puede ser afirmativa, en cuyo caso se agregan los parámetros para la asociación, o negativa. Después de responder a la Máquina **Smmspm** con cualquiera de las primitivas *RespuestaAsociacion* ó *RespuestaAsociacionError*, el Servidor regresa al estado "end".

5.1.5.4 Servicio Indicación de Liberación de Asociación

La llegada de la primitiva *IndicacionLiberacion* le permite al servidor realizar tareas administrativas de desalojo de la asociación, como por ejemplo, liberación del semáforo de control de recursos. Después de realizar tales tareas el servidor debe responder al **Smmspm** ya sea afirmativa o negativamente valiéndose de los mensajes *RespuestaLiberacion* y *RespuestaLiberacionError*. Una vez enviado el mensaje correspondiente **Server** regresa a "end".

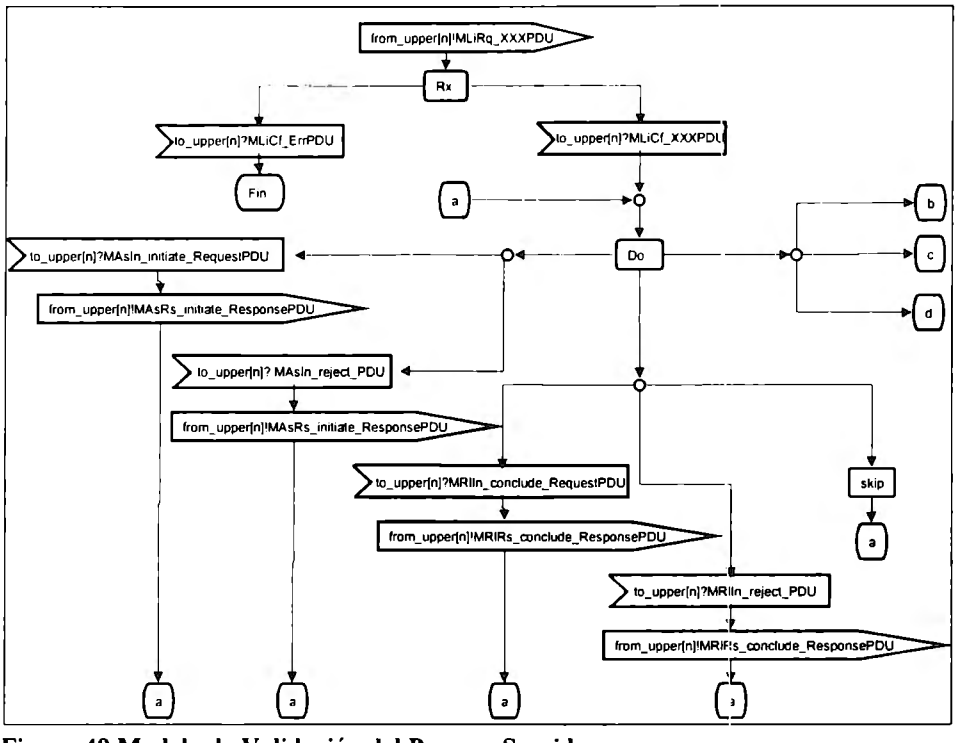


Figura 49 Modelo de Validación del Proceso Servidor.

5.1.5.5 Servicio Indicación de Servicio Confirmado

La primitiva *IndicacionServConf* indica al **Server** que el cliente solicita un servicio que requiere una respuesta; el servicio solicitado y sus parámetros correspondientes están contenidos en el PDU recibido y son accesados por **Server** para ejecutar las tareas correspondientes. Una vez hechas estas tareas y dependiendo del estado final resultante, **Server** responde a **Smmspm** afirmativa o negativamente usando las primitivas *RespuestaServConf* o *RespuestaServConfError*, respectivamente.

5.1.5.6 Servicio Indicación de Servicio No Confirmado

Los Servicios no confirmados no requieren de respuesta alguna por parte de **Server**, por lo que al recibir la primitiva *IndicacionServNoConf*, la cual indica esta clase de servicio, no tiene más que volver al estado "end", desde el punto de vista del protocolo, ya que en realidad, **Server** tiene que

ejecutar los procedimientos pertinentes para satisfacer la indicación del servicio. Esta parte del protocolo se puede observar en la Figura 50.

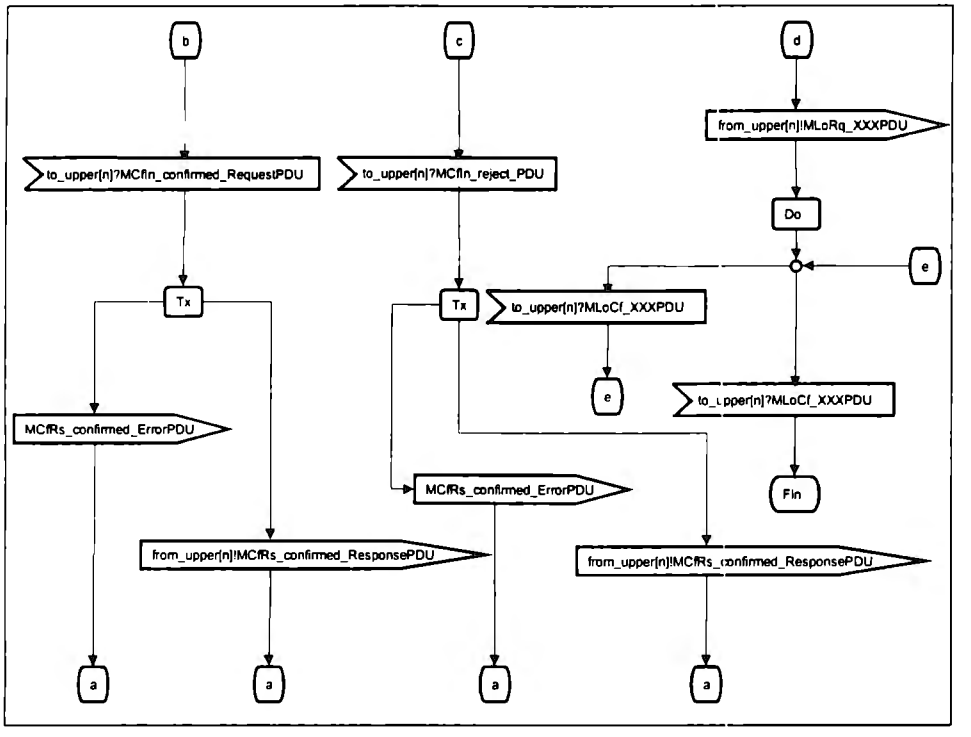


Figura 50 Modelo de Validación del Servidor

5.1.5.7 Servicio a PDUs Rechazados

Una parte importante dentro del modelado del protocolo es la que satisface a la premisa de que las máquinas del protocolo deben interactuar con cualquier entidad que pueda entender el mismo; tales entidades, el servidor y los clientes, pueden empacar los PDUs con información errónea, por lo que las máquinas deben de ser capaces de rechazar este tipo de mensaje. **Smmspm** modela este rechazo de PDUs y envía las primitivas adecuadas según el tipo de servicio del que se trate. **Server** solo tiene que simular la retransmisión del mensaje en forma correcta. Lo anterior es aplicable a los cuatro servicios previos a éste y se puede observar en los esquemas de las figuras antes mencionadas.

5.1.5.8 Servicio de LogOut

El estado "end" del proceso Servidor es un estado final aceptable, ya que teóricamente puede permanecer brindando servicios a los clientes conectados, pero también existe la posibilidad, y una posibilidad con probabilidad alta, de que el servidor eventualmente quiera terminar la conversación con los clientes registrados. Para indicar esto, el servidor emite un mensaje de tipo *PeticionLogout*, después de lo cual espera un tiempo determinado por la respuesta por parte de **Smmspm**. Si la respuesta recibida es del tipo *ConfirmacionLogout*, o si el tiempo se agota antes

de recibir respuesta alguna, **Smmspm** termina su ejecución; pero por otro lado, si la respuesta recibida es una *ConfirmacionLogoutError* **Smmspm** regresa al estado final "end", para poder decir si trata de abandonar su registro de una manera impropia.

5.2 MODELADO DE LOS REQUERIMIENTOS DEL PROTOCOLO

Esta sección tiene como propósito el remarcar la forma en como se modelaron los requerimientos del protocolo para una validación completa.

5.2.1 RECHAZO DE PDUS

Se han explicado los cuatro procesos que modelan el comportamiento completo del protocolo propuesto: **Client**, **Cmmspm**, **Smmspm** y **Server**, así como también de los diferentes tipos de mensajes posibles emitidos por cada un estos programas según la especificación MMS. Sin embargo, el software desarrollado en el presente trabajo correrá en el nuevo controlador y eventualmente interactuará con programas de control cargados en los robots y en el mismo controlador que se comunique usando MMS. Existe la posibilidad de que algunos tengan versiones MMS distintas y/o no soporten algunos servicios, por lo que se hace necesario una validación de los servicios que viajan usando el protocolo y de los formatos de los PDUs transmitidos. El anterior problema está muy relacionado con la implementación del protocolo, pero en la cuestión del tipo de mensajes lo está con el modelo mismo, por lo que se debe validar

El rechazo de PDUs solo puede existir del **Client** al **Cmmspm** o del **Server** al **Smmspm**, ya que **Cmmspm** y **Smmspm** son producto de un mismo desarrollo, el de esta tesis. Así que para modelar esta situación, los procesos **Cmmspm** y **Smmspm** están preparados para recibir el mensaje del tipo *XXXRechazo,* donde *XXX* representa la primitiva normal especificada en MMS pero que al pasar una función "ValidatePDU" se convirtió en un rechazo del mismo. La función "ValidatePDU" no es relevante en el modelo de la validación, por lo que no se incluye, pero sí lo es el resultado que genera, ya que esto modifica el comportamiento de las entidades. Por otro lado, **Client** y **Server** deben estar preparadas para manejar los mensajes del tipo *XXXRechazo,* y a la recepción de éstas deben retransmitir la primitiva modificando el PDU o enviar una primitiva de error. El detalle de estos servicios se encuentra detallado en la sección 5.1 para cada uno de los procesos.

5.2.2 PÉRDIDA DE MENSAJES

Los enlaces de los transputers son un medio de comunicación muy eficaz [13], sin embargo existe la posibilidad de falla y por lo tanto de pérdidas del mensaje. Además de una falla de los enlaces, se debe tener presente que los mensajes viajan fuera del controlador a través de conectores físicos RS232 donde la probabilidad de mensaje omitidos es mayor. Para modelar esta situación se sustituyo en el código de los procesos una secuencia *if-ji,* con dos alternativas (ver Figura 51); la primera de ella es la secuencia original precedente a la recepción del mensaje y la

segunda una instrucción *skip*. Si el proceso elige esta última, se comportara como si no hubiese recibido tal mensaje, es decir que hubo un error de comunicación. El ejemplo de la Figura 51(a) fue extraído del código del proceso **Cmmspm** usado para la validación del protocolo y corresponde a la recepción de la primitiva *PeticionLogout* por parte del proceso **Client**, a lo que la máquina responde con una transmisión de una *IndicacionLogout* hacia la entidad remota y se va a un estado de expiración local. La parte (b) muestra como se modifica el código para emular la pérdida del mensaje, **Cmmspm** recibe la primitiva *PeticionLogout* y elige entre seguir el camino descrito o ignorar el mensaje manteniéndose en el estado Asociado.

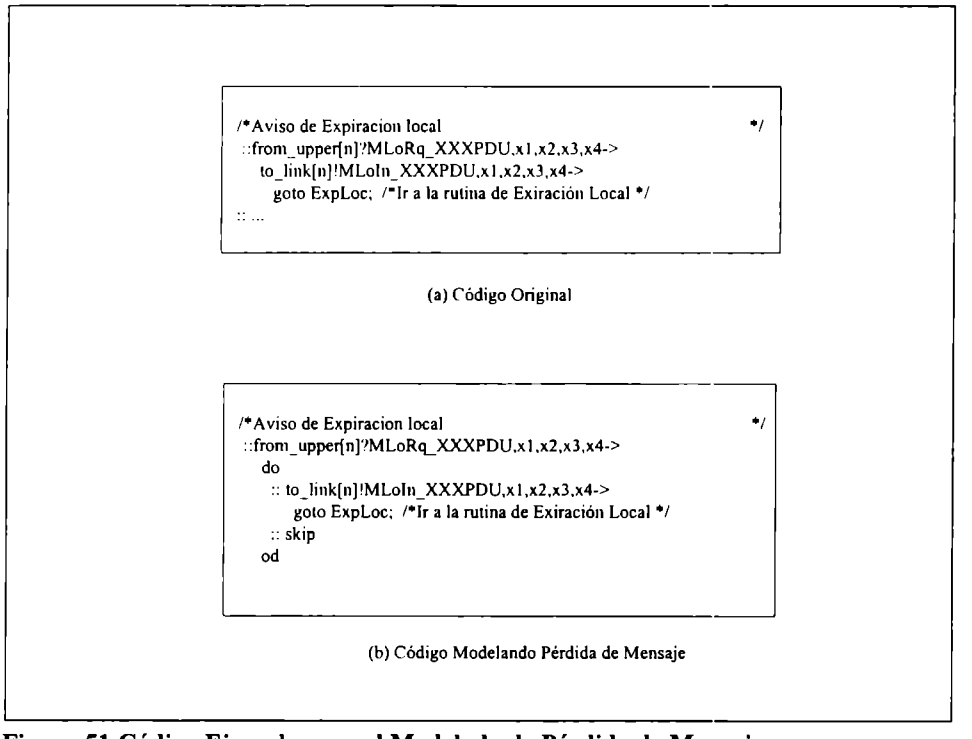


Figura 51 Código Ejemplo para el Modelado de Pérdida de Mensajes.

5.2.3 DUPLICIDAD DE MENSAJES

Para evitar esperas indefinidas se implementan temporizadores, que garantizan que la ejecución de los procesos sigue adelante aún y si no se recibe el mensaje esperado; después de la expiración del temporizador generalmente se retransmite el mensaje, porque se asumen que se perdió el primero. Si esta pérdida no fuera cierta y el mensaje "respuesta" estuviera en camino al tiempo del agotamiento del tiempo de espera, el proceso receptor recibirá el mismo mensaje dos veces y debería omitir el segundo. Para ello se implementa un etiquetado en los mensajes, marcando el número de asociación según la máquina local, el número de asociación según la máquina remota, el número de petición según la máquina remota y el número de indicación de la máquina local. Estas etiquetas permiten verificar el estado de la asociación, petición o indicación antes de responder apropiadamente al servicio aún y cuando se trate de un mensaje duplicado.

En la Figura 52 se muestra como ejemplo para este modelado una sección del código del proceso **Smmspm** que corresponde a la recepción de una *IndicacionAsociacion* con la que el cliente solicita el establecimiento de una asociación. La máquina local del cliente alojó tal asociación y le otorgó el número "l", posteriormente envía la indicación (probablemente más de una vez) y espera por una respuesta por parte **Smmspm**. En la primera parte del procedimiento, ciclo *do-od*, **Smmspm** checa en su tabla de asociaciones locales para determinar si la asociación "l" ha sido o no registrada previamente, si es así lo indica en la bandera *bExiste*. En la segunda parte, estructura *fi-if*,el procedimiento muestra cuatro alternativas, las dos primeras son válidas cuando la asociación no ha sido registrada previamente y el comportamiento está determinado por el cupo o no de otra asociación en la tabla de asociaciones. Las dos alternativas restantes existen para el caso de que la asociación ya haya sido registrada previamente y seguramente se trata de la recepción de un mensaje duplicado. Si el estado de la asociación es *Associated*, **Smmspm** responde a la máquina del cliente con una primitiva *ConfirmacionAsociacion*, para indicar que tal asociación es totalmente válida. Si su estado es distinto, simplemente hace caso omiso del mensaje.

```
::from_link[n]?MAsIn_initiate_RequestPDU,x1,l,x3,x4 ->
  atomic{
    bExiste=FALSE;
    j=0;
    do
    ::(j<MC && InUse[j]==TRUE && AssIdRmt[j]==l)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j<MC && (InUse[j]!=TRUE || AssIdRmt[j]!=l))->
      j=j+1;
    ::(j>=MC)->
      break;
    od;
  };
  if
  ::(bExiste==FALSE && NCD>0)-> busca lugar en tabla
    NCD=NCD-1;
    InUse[i]=TRUE;
    AssId[i]=s;
    AssIdRmt[i]=l;
    State[i]=CalledInit;
    to_upper[n]!MAsIn_initiate_RequestPDU,s,l,x3,x4;
    s=(s+1)%MA
  ::(bExiste==FALSE && NCD<=0)->
    to_link[n]!MAsCf_initiate_ErrorPDU,l,x2,x3,x4;
  ::(bExiste!=FALSE && State[i]==Associated)->
    k==AssId[i];
    to_link[n]!MAsCf_initiate_ResponsePDU,l,k,x3,x4;
  ::(bExiste!=FALSE && State[i]!=Associated)-> skip;
  fi
```

Figura 52 Código Ejemplo para el Modelado de Duplicidad de Mensajes.

Análogamente a este ejemplo, existen secciones en el código de los procesos para manejar la recepción de mensajes duplicados.

5.2.4 ESTADOS FINALES ACEPTABLES

Uno de los requerimientos de corrección principales aplicables a un modelo de validación es que esté libre de deadlocks. Un deadlock se detecta rápidamente cuando el proceso se queda

esperando indefinidamente a que suceda un evento, sin embargo, existen ciertos estados en los que el proceso puede estar en espera eternamente sin ser estos necesariamente un deadlock; un ejemplo de este tipo de procesos es un servidor, ya que su finalidad es estar disponible para atender peticiones de sus clientes, sin importar que en lapsos prolongados no haya un sólo cliente que atender. A un estado de espera de este tipo se le conoce como estado final aceptable (refiérase al punto 3.2.6.6.3). Todos los estados en que el proceso tiene el riesgo de quedarse esperando un mensaje son detectados por el validador SPIN y reportados por este como estados finales erróneos; para evitarlo se le debe indicar explícitamente que para el protocolo son aceptables, marcándolos con etiquetas con prefijo "end" (refiérase al punto 3.2.6.6.3).

En el modelo de validación propuesto en la presente tesis se pueden detectar cuatro estados finales aceptables en lo que se refiere a las máquinas del protocolo MMS. Para la máquina del cliente son:

- Estado *StandAlone*. (Refiérase a la etiqueta "End3" en el código del proceso **Cmmspm** incluido en el Apéndice B). El proceso **Cmmspm** llega a este estado después de haber recibido la indicación de terminación del servidor (*LOGOUT_Indicaction*), para esperar que el proceso **Client** termine la ejecución local. **Cmmspm** no indica a **Client** de la terminación remota inmediatamente, pero todo servicio solicitado por este último es rechazado por el primero indicando que la razón del error es la correspondiente a la baja del proceso servidor. Toda petición del cliente a su máquina tiene que ser procesada hasta que el proceso cliente solicite su terminación mediante la primitiva *LOGOUT_Request*.

- Estado *Group*. (Refiérase a la etiqueta "End2" en el código del proceso **Cmmspm** incluido en el Apéndice B). La estancia de este estado por parte **Cmmspm** significa que tanto el proceso servidor como el cliente están activos; todas las peticiones generadas por **Client** y todas las respuestas generadas por **Server** son procesadas según el estado de la máquina del protocolo. Es posible que el proceso se mantenga en este estado permanentemente.

En el proceso de la máquina del servidor también hay estados finales aceptables:

- Estado *StandAlone*. Es el estado normal de **Smmspm** cuando no hay clientes conectados; significa además que el proceso **Server** está disponible para atender las peticiones remotas. La estancia en este estado puede ser perpetua aunque realmente no hay sentido alguno en tener un servidor disponible si no hay clientes que deseen accesarlo. Este estado esta marcado con la etiqueta "End3" en el código relacionado con este proceso, incluido en el Apéndice B

- Estado *Group*. **Cmmspm** está en este estado cuando tiene al menos algún cliente registrado y puede continuar allí indefinidamente ya que es su objetivo primario. Este estado esta marcado con la etiqueta "End2" en el código de este proceso incluido en el Apéndice B

Para verificar esta propiedad por medido de XSPIN, se marca la opción "In" obteniéndose los resultados mostrados en la Figura 53.

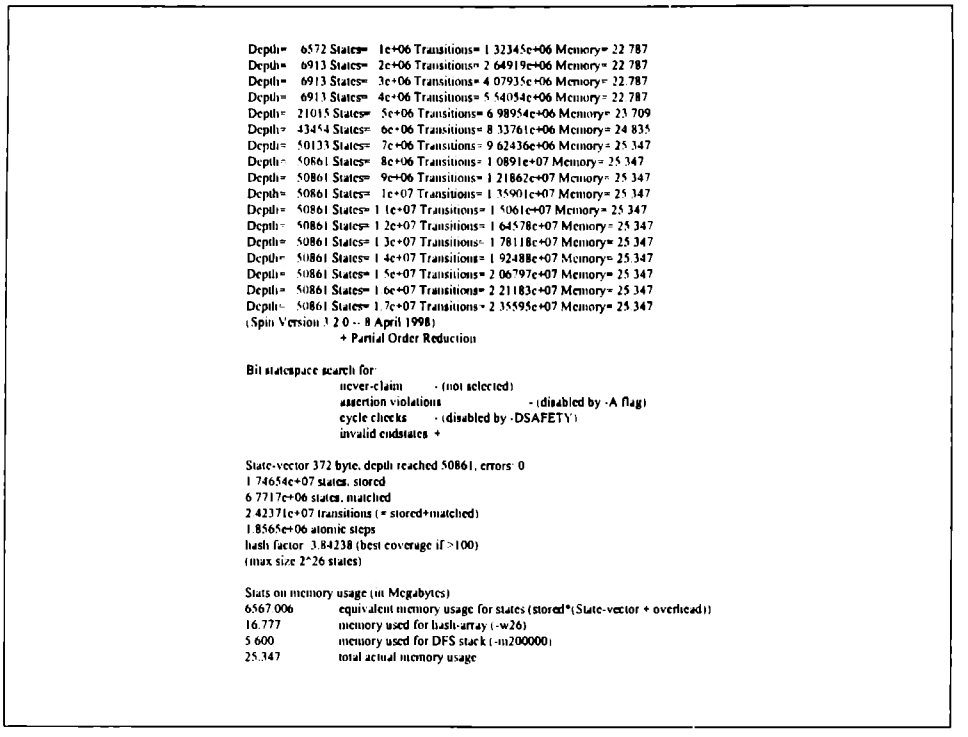


Figura 53 Resultados de la Validación del Modelo para los Estados Finales.

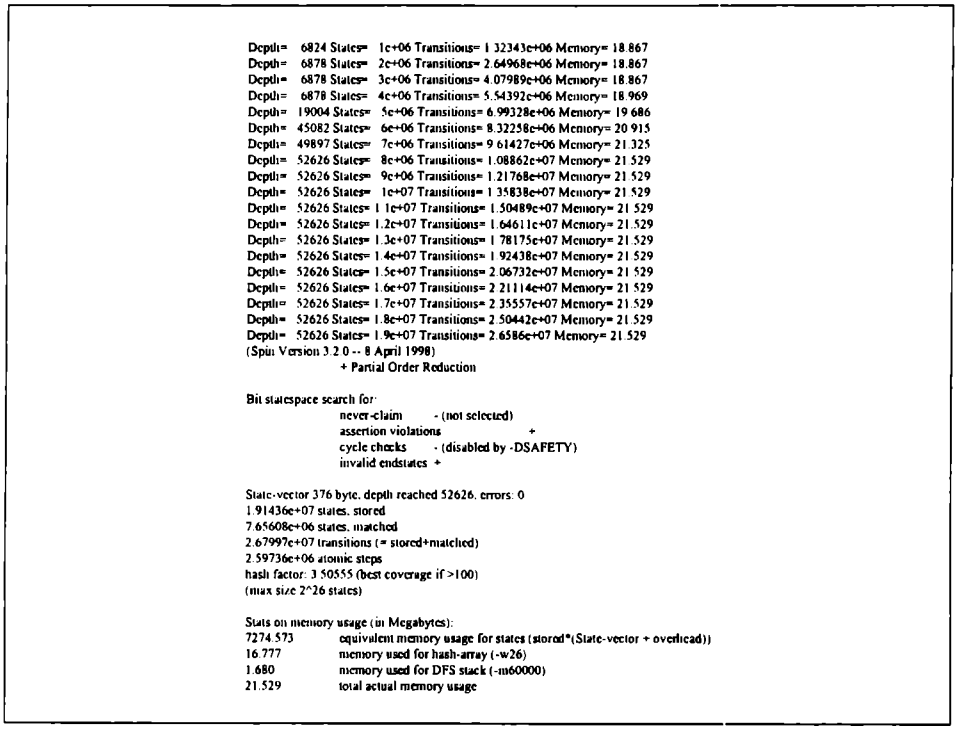
5.2.5 INVARIANTES DEL SISTEMA

La invariante para este sistema (consulte el punto 3.2.6.6.2), es que el número de asociaciones en todo momento y en cada máquina siempre debe ser menor o igual al tamaño de los arreglos hechos con para estos fines. El número de asociaciones en operación es contabilizado en la variable NCD_SM para la máquina del servidor y en NCD_CM para la máquina del cliente; la variable global MC representa el número de elementos de los arreglos que soportan la información de las asociaciones en ambas máquinas y por ello el máximo de asociaciones permitidas para evitar el desborde de los arreglos.

La invariante es verificada agregando un proceso adicional (refiérase al punto antes citado) llamado **monitor** el cual consta del siguiente código:

```
proctype monitor(){
 if
 ::assert(NCD_SM<=MC);
 ::assert(NCD_CM<=MC);
 fi
}
```

Como se puede notar, el proceso básicamente ejecuta una instrucción en toda su existencia. XSPIN se encarga de correr cada posibilidad del código original del modelo seguida por la ejecución de este pequeño proceso, buscando alguna que pueda violar cualquiera de las dos aserciones. El resultado de esta prueba se muestra en la Figura 54, el cual comprueba que esta invariante es respetada por el protocolo.



```
Depth=   6824 States=   1e+06 Transitions= 1.32343e+06 Memory= 18.867
Depth=   6878 States=   2e+06 Transitions= 2.64968e+06 Memory= 18.867
Depth=   6878 States=   3e+06 Transitions= 4.07989e+06 Memory= 18.867
Depth=   6878 States=   4e+06 Transitions= 5.54392e+06 Memory= 18.969
Depth=  19004 States=   5e+06 Transitions= 6.99328e+06 Memory= 19.686
Depth=  45082 States=   6e+06 Transitions= 8.32258e+06 Memory= 20.915
Depth=  49897 States=   7e+06 Transitions= 9.61427e+06 Memory= 21.325
Depth=  52626 States=   8e+06 Transitions= 1.08862e+07 Memory= 21.529
Depth=  52626 States=   9e+06 Transitions= 1.21768e+07 Memory= 21.529
Depth=  52626 States=   1e+07 Transitions= 1.35838e+07 Memory= 21.529
Depth=  52626 States= 1.1e+07 Transitions= 1.50489e+07 Memory= 21.529
Depth=  52626 States= 1.2e+07 Transitions= 1.64611e+07 Memory= 21.529
Depth=  52626 States= 1.3e+07 Transitions= 1.78175e+07 Memory= 21.529
Depth=  52626 States= 1.4e+07 Transitions= 1.92438e+07 Memory= 21.529
Depth=  52626 States= 1.5e+07 Transitions= 2.06732e+07 Memory= 21.529
Depth=  52626 States= 1.6e+07 Transitions= 2.21114e+07 Memory= 21.529
Depth=  52626 States= 1.7e+07 Transitions= 2.35557e+07 Memory= 21.529
Depth=  52626 States= 1.8e+07 Transitions= 2.50442e+07 Memory= 21.529
Depth=  52626 States= 1.9e+07 Transitions= 2.6586e+07 Memory= 21.529
(Spin Version 3.2.0 -- 8 April 1998)
        + Partial Order Reduction

Bit statespace search for:
        never-claim         - (not selected)
        assertion violations        +
        cycle checks        - (disabled by -DSAFETY)
        invalid endstates   +

State-vector 376 byte, depth reached 52626, errors: 0
1.91436e+07 states, stored
7.65608e+06 states, matched
2.67997e+07 transitions (= stored+matched)
2.59736e+06 atomic steps
hash factor: 3.50555 (best coverage if >100)
(max size 2^26 states)

Stats on memory usage (in Megabytes):
7274.573    equivalent memory usage for states (stored*(State-vector + overhead))
16.777      memory used for hash-array (-w26)
1.680       memory used for DFS stack (-m60000)
21.529      total actual memory usage
```

Figura 54 Resultados de la Validación de Invariantes del Sistema

Análogamente se puede probar que el número de peticiones e indicaciones en todo momento es menor o igual al permitido por los arreglos de almacenamiento de las mismas.

5.2.6 PROGRESO DEL PROTOCOLO

Finalmente dentro de las pruebas a las que fue sometido el modelo del protocolo, esta la de corroborar su avance o progresos (refiérase al punto 3.2.6.6.4), para ello se partió de la hipótesis:

"Si se solicita un servicio a alguna máquina del protocolo, eventualmente esta regresa una respuesta positiva o negativa."

Esta aseveración se verificó con todos los servicios otorgados en el protocolo; para efectos de ejemplificar se toma el servicio de *Login*. Al igual que en la validación del punto anterior, se agrega un proceso adicional al modelo, el proceso never (refiérase al punto 3.2.6.6.5); en el cual se plantea la hipótesis mencionada de la siguiente manera:

```
never{
 do
 ::!from_upper[0]?[MLiRq_XXXPDU] -> skip;
 ::from_upper[0]?[MLiRq_XXXPDU]  -> goto accept0;
 od;
accept0:
 do
 ::!to_upper[0]?[MLiCf_XXXPDU]
  && !to_upper[0]?[MLiCf_ErrPDU];
 od;
}
```

Como se mencionó en el punto antes citado, el proceso never nunca debe llagar a su última línea para que sea válido la propiedad contenida en su interior para XSPIN. Como resultado de esta validación se obtuvo la salida mostrada en la Figura 55.

```
Depth=   6824 States=   1e+06 Transitions= 1.32343e+06 Memory= 18.867
Depth=   6878 States=   2e+06 Transitions= 2.64968e+06 Memory= 18.867
Depth=   6878 States=   3e+06 Transitions= 4.07989e+06 Memory= 18.867
Depth=   6878 States=   4e+06 Transitions= 5.54392e+06 Memory= 18.969
Depth=  19004 States=   5e+06 Transitions= 6.99328e+06 Memory= 19.686
Depth=  45082 States=   6e+06 Transitions= 8.32258e+06 Memory= 20.915
Depth=  49897 States=   7e+06 Transitions= 9.61427e+06 Memory= 21.325
Depth=  52626 States=   8e+06 Transitions= 1.08862e+07 Memory= 21.529
Depth=  52626 States=   9e+06 Transitions= 1.21768e+07 Memory= 21.529
Depth=  52626 States=   1e+07 Transitions= 1.35838e+07 Memory= 21.529
Depth=  52626 States= 1.1e+07 Transitions= 1.50489e+07 Memory= 21.529
Depth=  52626 States= 1.2e+07 Transitions= 1.64611e+07 Memory= 21.529
Depth=  52626 States= 1.3e+07 Transitions= 1.78175e+07 Memory= 21.529
Depth=  52626 States= 1.4e+07 Transitions= 1.92438e+07 Memory= 21.529
Depth=  52626 States= 1.5e+07 Transitions= 2.06732e+07 Memory= 21.529
Depth=  52626 States= 1.6e+07 Transitions= 2.21114e+07 Memory= 21.529
Depth=  52626 States= 1.7e+07 Transitions= 2.35557e+07 Memory= 21.529
Depth=  52626 States= 1.8e+07 Transitions= 2.50442e+07 Memory= 21.529
Depth=  52626 States= 1.9e+07 Transitions= 2.6586e+07 Memory= 21.529
Depth=  52626 States= 2.0e+07 Transitions= 2.78956e+07 Memory= 21.529
Depth=  52626 States= 2.1e+07 Transitions= 2.90152e+07 Memory= 21.529
Depth=  52626 States= 2.2e+07 Transitions= 3.01236e+07 Memory= 21.529

(Spin Version 3.2.0 -- 8 April 1998)
        + Partial Order Reduction

Bit statespace search for
        never-claim     - (not selected)
        assertion violations            - (not selected)
        cycle checks    +
        invalid endstates -
State-vector 398 byte, depth reached 52626, errors: 0
2.0802e+07 states, stored
9.32162e+06 states, matched
3.01236e+07 transitions (= stored+matched)
2.65932e+06 atomic steps
hash factor: 4.956326 (best coverage if >100)
(max size 2^26 states)

Stats on memory usage (in Megabytes)
7456.573        equivalent memory usage for states (stored*(State-vector + overhead))
16.777          memory used for hash-array (-w26)
```

Figura 55 Resultados de la Validación del progreso del protocolo.

6 IMPLEMENTACIÓN DEL PROTOCOLO

Este capitulo tiene como objetivo el de presentar la implementación del protocolo MMS propuesto en este trabajo. Primeramente se da un enfoque global de la aplicación y después se presentan los detalles de ésta; para evitar repeticiones con lo expuesto en el capítulo "Validación del Protocolo", los algoritmos y procesos sólo se detallan en los puntos que cambian debido a las herramientas de programación usadas. El código fuente de esta implementación se incluye en el Apéndice C.

6.1 GENERALIDADES

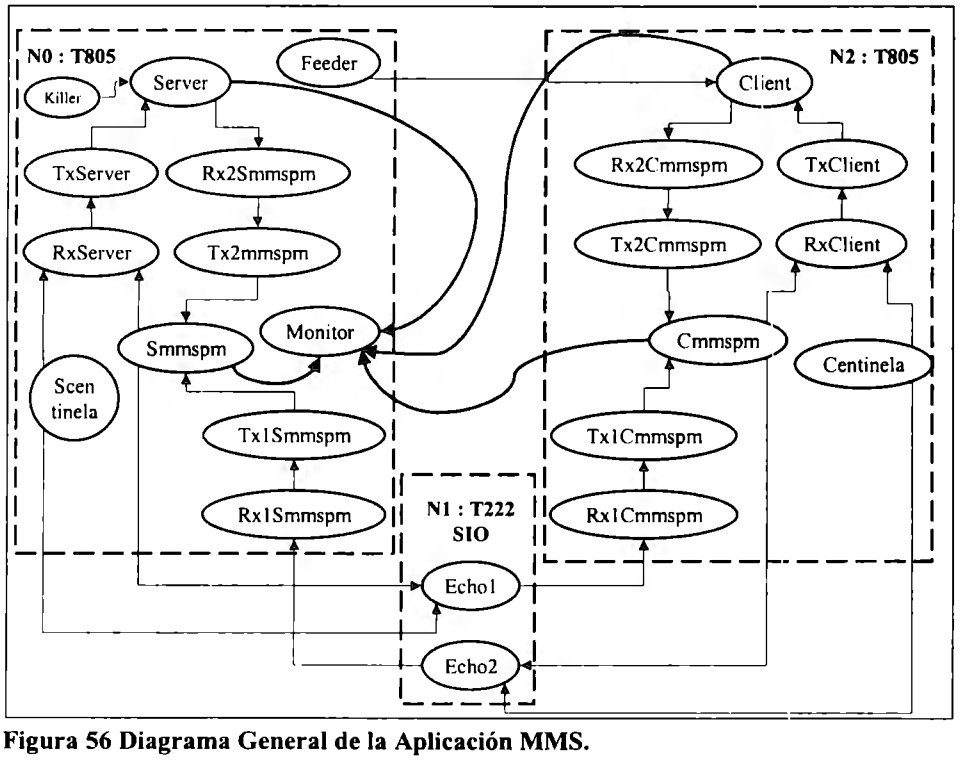


Figura 56 Diagrama General de la Aplicación MMS.

La aplicación que se describe en este capítulo tiene como fin la implementación del protocolo diseñado en la presente tesis, ajustándose al modelo validado que se abordó en el capítulo anterior. Aunque el alcance de está tesis es implementar solamente el protocolo MMS, se hace necesario adicionar algunos procesos de aplicación con el objeto de poder apreciar el comportamiento que tiene el protocolo; esta aplicación se referirá en lo sucesivo como Aplicación MMS.

La aplicación MMS consiste de un ambiente Cliente MMS, un ambiente Servidor MMS, un proceso alimentador de comandos llamado **Feeder**, un proceso de paro para el servidor llamado **Killer** y un proceso monitor del sistema nombrado **Monitor** (ver Figura 56). El ambiente Cliente MMS consiste, a su vez, de un proceso llamado **Client**, el cual recibe y procesa un grupo de instrucciones MMS enviadas por el proceso **Feeder** y de otros procesos que se encargan de que se lleve a cabo la comunicación tipo MMS. Análogamente, el ambiente Servidor MMS esta formado por un proceso llamado **Server**, encargado de responder a las peticiones que el cliente haga y un conjunto de procesos concurrentes para la comunicación MMS; dentro de estos dos ambientes, los procesos que cambiarán en toda aplicación que utilice la implementación MMS propuesta en este trabajo serán solamente **Client** y **Server**.

Adicionalmente a estos dos ambientes, descritos más a detalle en los puntos 6.1.1.1 y 6.1.1.2, existe un proceso llamado **Feeder** que se encarga de leer un archivo de comandos y transmitírselos al proceso **Client**; un proceso llamado **Killer** que se encarga de enviar al proceso **Server** un mensaje de paro cuando el operador del sistema así se lo indica; y por último un **Monitor** que tiene como finalidad presentar en la pantalla de la computadora host la recepción los mensajes recibidos por los cuatro nodos importantes de la aplicación MMS: **Client**, **Server**, **Cmmspm** y **Smmspm**, con el fin de ilustrar el comportamiento interno del protocolo.

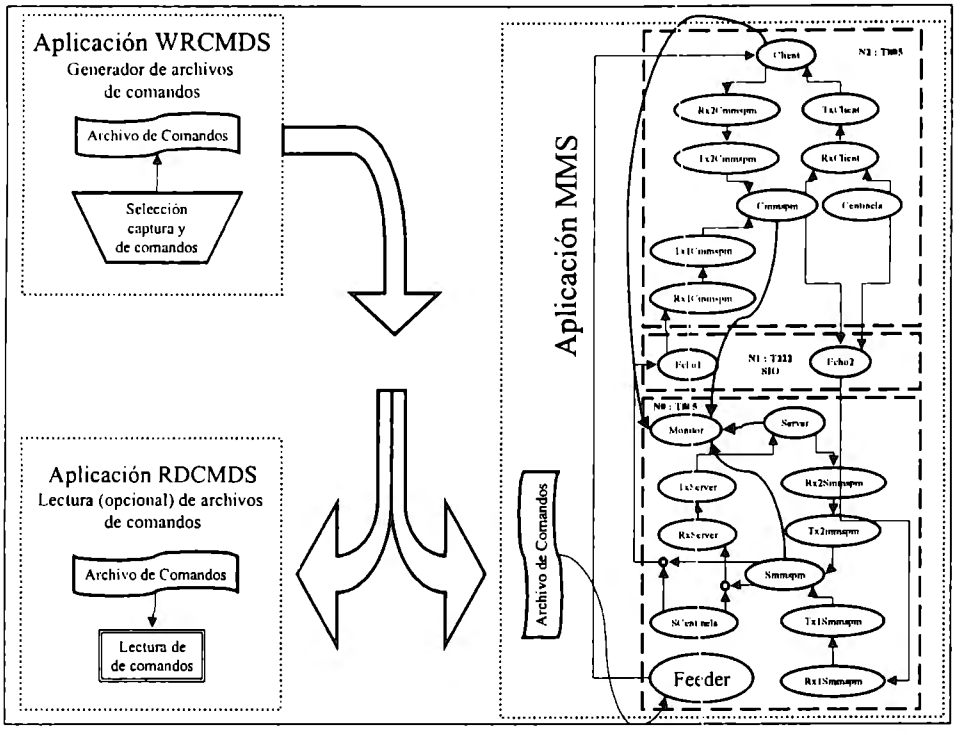


Figura 57 Relación entre la Aplicación MMS y las Aplicaciones Wrcmds y RdCmds.

Como se mencionó en los párrafos anteriores la Aplicación MMS requiere de un archivo de comandos MMS, los cuales están empacados como cadenas de caracteres. Dicha cadena es inteligible para cualquier usuario o proceso que no contenga los procedimientos necesarios para interpretarla como un PDU de MMS; por esta razón se le proporciona al usuario de este software, dos sistemas adicionales: "wrcmds" y "rdcomds"; el primero para escribir en un archivo los comandos a transmitir a través de la red y el segundo para que, opcionalmente, el usuario pueda leer los comandos MMS empacados en el archivo; estas dos aplicaciones se detallan en el punto 6.1.2 en este mismo capítulo. Para una mejor compresión de la relación que existe entre las aplicaciones proporcionadas por este trabajo la Figura 57 muestra un esquema de estas tres aplicaciones.

6.1.1 PROCESOS

A continuación se detallan los procesos involucrados en la Aplicación MMS.

6.1.1.1 Ambiente Servidor MMS.

El ambiente MMS del servidor está conformado por los procesos **Server**, TxServer, RxServer, **Smmspm**, SCentinela, Rx2Smmspm, Tx2mmspm, Rx1Smmspm, Tx1Smmspm, los cuales se describen en los siguientes párrafos y se muestran en la Figura 58.

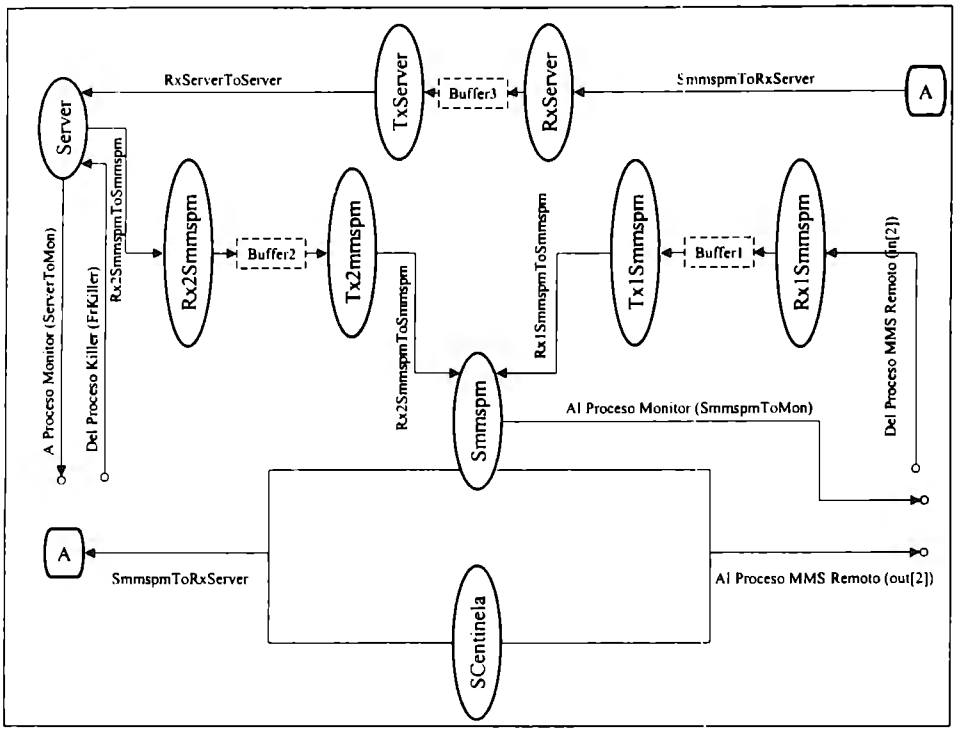


Figura 58 Esquema del Ambiente Servidor MMS.

6.1.1.1.1 Server

Este proceso tiene como finalidad ejecutar todas las actividades relacionadas con los servicios posibles de la aplicación. Según MMS, la entidad servidor generalmente está representada por los robots y las máquinas automatizadas aunque también existe la posibilidad que uno ó más procesadores del controlador haga funciones de cliente y servidor a un mismo tiempo. En el caso de que el servidor sea un robot, en este proceso se monta la función ejecutiva del robot (refiérase al punto 3.1.3); para esta aplicación, en este proceso se montaron las rutinas de servicio a las peticiones que hace el cliente remoto; el código se incluye en el punto 11.1 del Apéndice C. Este proceso consta de un canal de datos de entrada y uno de salida para intercambiar datos con su máquina de protocolo MMS. Exclusivamente para detener la ejecución de este proceso, que realmente puede ser temporalmente indefinida, se adicionó un canal de entrada por donde recibe la instrucción de paro que le da el proceso **Killer**. Existe un cuarto canal, que es de salida y sirve para retransmitir los mensajes recibidos por este proceso al **Monitor** para que el usuario pueda enterarse de la actividad realizada por él. **Killer**, **Monitor** y sus canales respectivos no forman parte integral del protocolo MMS implementado en esta tesis, son incluidos como parte de la aplicación demostrativa de éste.

6.1.1.1.2 RxServer y TxServer

Estos dos procesos sirven para emular el canal de recepción del servidor con una longitud tres, la cual puede ser configurada fácilmente modificando el archivo de configuración MMSCONTS.H. Para más detalles de las funciones de estos procesos refiérase al punto 6.2.3.

6.1.1.1.3 Smmspm

Los procesos **Smmspm** y **Cmmspm** son los más importantes dentro de esta implementación, ya que en ellos está contenido toda la esencia del protocolo y su funcionamiento. Este proceso se auxilia de las funciones, estructuras, variables y constantes incluidas en los archivos MMSPRMCF.H y MMSPRMCH.H, los cuales se encuentran junto al código del **Smmspm** en el Apéndice C. **Smmspm** consta de cinco canales dos de entrada y tres de salida, dos para la comunicación con el proceso Servidor, dos para la comunicación con la entidad remota y uno para indicarle al proceso Monitor del progreso del protocolo; éste último canal no forma parte básica para la implementación del protocolo solo fue montado para esta aplicación.

6.1.1.1.4 SCentinela

Scentinela es un proceso que está estrechamente relacionado con **Smmspm**, ya que es el encargado de asegurarse que ningún servicio confirmado registrados en la máquina del protocolo del servidor se quede pendiente; para ello hace uso de rutinas de retransmisión de mensajes pendientes de respuestas y de eliminación de peticiones e indicaciones consideradas sin respuestas. **Scentinela** consta de dos canales de salida que están compartidos con el **Smmspm**, uno dirigido hacia el servidor y otro hacia el cliente. El código de este proceso esta indicado en el punto 11.3 del Apéndice C.

6.1.1.1.5 Rx1Smmspm, Tx1Smmspm, Rx2Smmspm y Tx2mmspm

Estos procesos están encargados de mantener los canales de entrada al proceso **Smmspm** con una capacidad de almacenamiento de hasta 3 mensajes en espera (refiérase al punto 6.2.3). Los primeros dos procesos reciben la información que proviene de la entidad remota, y los dos restantes se encargan de recibir los mensajes del proceso Servidor.

6.1.1.2 Ambiente Cliente MMS.

El ambiente Cliente MMS lo componen los procesos **Client**, **TxClient**, **RxClient**, **Cmmspm**, **CCentinela**, **Tx1Cmmspm**, **Rx1Cmmspm**, **Rx2Cmmspm** y **Tx2Cmmspm**, los cuales se detallan a continuación y la relación entre ellos se puede observar en el esquema de la Figura 59.

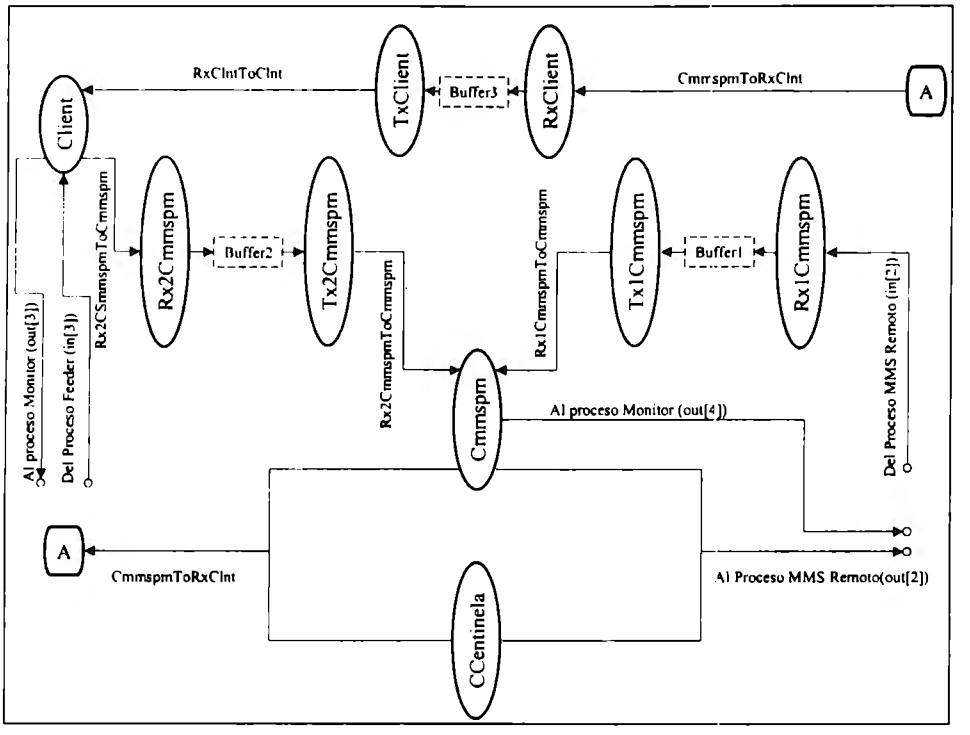


Figura 59 Esquema del Ambiente Cliente MMS.

6.1.1.2.1 Proceso Client

En este proceso es el que modifica el usuario para ajustarlo a las necesidades de la aplicación; es el punto de acceso al Ambiente Cliente MMS. Como demostración, en esta aplicación, este proceso se dedica a procesar todas las instrucciones que el operador le indica a través de un archivo. El archivo de comandos realmente es leído por un proceso independiente que corre en el nodo 0 de la tarjeta el cual se describe más adelante en esta misma sección. El proceso **Client** graba todos los comandos que tiene que procesar en un arreglo de comandos local, después procesa uno a uno dependiendo del tipo de comando de que se trate. Este proceso consta de cuatro canales dos de entrada y dos de salida; uno de los canales de entrada lo usa para recibir mensajes del proceso que lee el archivo de comando, **Feeder**, el canal de entrada restante y uno de los canales de salida los utiliza para comunicarse con su máquina de protocolo MMS, el canal de salida restante es usado para enviarle datos al proceso **Monitor**. El código fuente para este programa se encuentra en el punto 11.2 del Apéndice C.

6.1.1.2.2 TxClient y RxClient

Estos dos procesos desempeñan la función de una canal de longitud > 0 (ver punto 6.2.3), para el canal de recepción del proceso **Client**. Los datos que son manejados por este par de procesos provienen de la máquina local del protocolo del cliente.

6.1.1.2.3 Cmmspm

Cmmspm es el proceso que hace las funciones de la máquina del protocolo MMS para la entidad cliente, es auxiliado por procedimientos, estructuras, variables y constantes incluidas en los archivos MMSPRMCH.H y MMSPRMCF.H. Esta máquina consta de cinco canales, dos dirigidos hacia el Cliente y dos hacia el Servidor; aunque sus mensajes están dirigidos a éstos dos procesos, realmente los envía a los procesos receptores de éstos. El quinto canal es de salida y está direccionado al **Monitor** para informarle de los mensajes que ha recibido, este canal no es parte esencial de la implementación de MMS sino que está incluido para tener una herramienta visual del avance del protocolo. El código de este programa está en el archivo CMMSPM.H agregado a este documento en el Apéndice C.

6.1.1.2.4 CCentinela

Es el proceso encargado de dar seguimiento a los servicios pendientes registrados en las tablas de conexiones, de peticiones e indicaciones (ver punto 6.2.1). El proceso **CCentinela** hace un barrido de las tablas mencionadas para detectar servicios que estén pendientes; si detecta alguno lo marca como candidato a eliminación y retransmite el mensaje del cual se espera respuesta para despachar este servicio. Si el registro ya está marcado como candidato entonces lo elimina de la tabla, realizando la parte complementaria del servicio para indicar a la entidad correspondiente el error en el procesamiento del servicio solicitado. El comportamiento de este proceso está explicado más detalladamente en el capítulo de validación en el punto 5.1.4 mientras que el código C involucrado se encuentra en el mismo archivo que el proceso **Cmmspm**.

6.1.1.2.5 Rx1Cmmspm, Tx1Cmmspm, Rx2Cmmspm y Tx2Cmmspm

Los dos primero procesos emulan un canal de longitud mayor a 0 para el canal de entrada de los mensajes que van de la entidad remota a **Cmmspm** y los dos últimos tienen el mismo objetivo pero para el canal que comunica al proceso **Client** con **Cmmspm**. Para un mayor detalle del algoritmo seguido por estos proceso refiérase al punto 6.2.3

6.1.1.3 Otros Procesos.

Además de los dos ambientes existen otros procesos sin ser parte esencial del protocolo MMS complementan la aplicación; tales procesos son **Feeder, Killer, Echo1** y **Echo2** (ver Figura 60); son descritos a continuación.

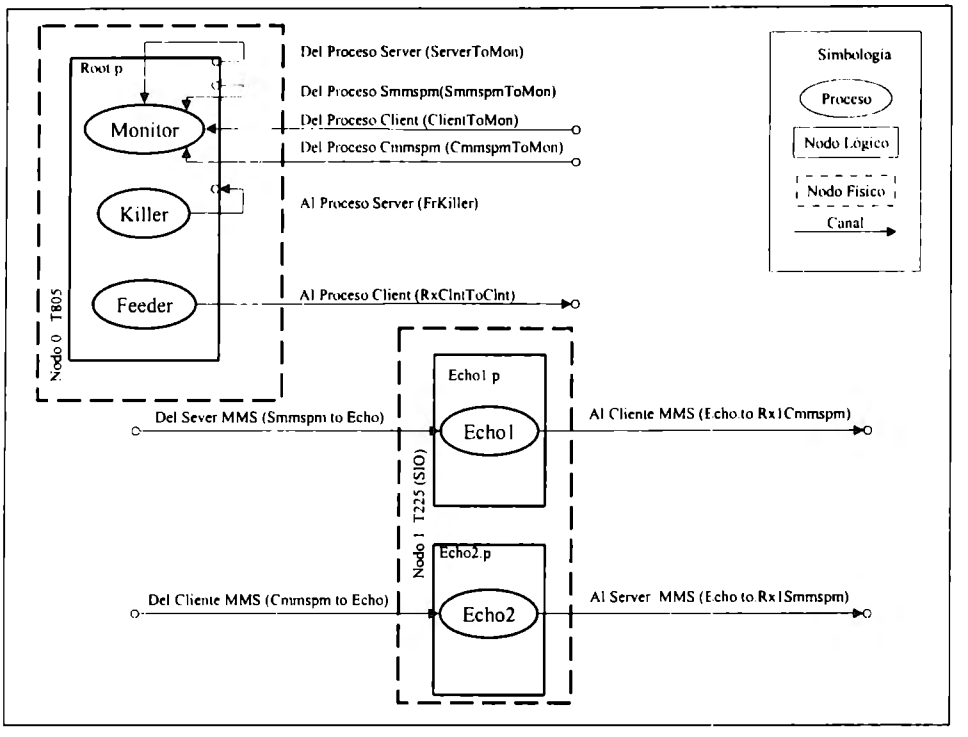


Figura 60 Otros Procesos de la Aplicación MMS.

6.1.1.3.1 Feeder

La existencia de este proceso es relevante sólo para esta aplicación en particular y no es imprescindible para alguna otra aplicación que requiera de la implementación del protocolo MMS aquí propuesta. El proceso Feeder tiene como objetivo transmitir los comandos que están contenidos en un archivo que previamente se grabó por medio de la aplicación "Wrcmds" (refiérase al punto 6.1.2); esta labor de lectura no fue incluida directamente en el proceso **Client** para no obligar a que el ambiente cliente MMS corriera en nodo 0, ya que las funciones de entrada/salida requeridas en la lectura están restringidas a procesos que corran en tal nodo. El código fuente de este proceso esta contenido en el archivo FEEDER.H que se puede consultar en el Apéndice C.

6.1.1.3.2 Killer

Con el objeto de detener la ejecución del proceso **Server** de una manera controlada, se incluyó este proceso que reside en el nodo 0 debido a que requiere de funciones de entrada/salida. El funcionamiento de este proceso es muy simple, solamente está monitoreando el dispositivo de entrada del host y si detecta que el caracter "&" fue introducido por el operador, envía un mensaje al proceso **Server** para que este detengan su ejecución. El mensaje enviado por **Killer** no forma parte integral del protocolo sino que es requerido por esta aplicación; es un mecanismo muy fino de detener el funcionamiento del proceso servidor. El código de este programa esta incluido en el archivo ROOT.C.

6.1.1.3.3 Echo1 y Echo2

Como se mencionó en el punto 4.3.3, uno de los requerimientos de este protocolo debido al medio de trabajo, es que los mensajes se deben poder transmitir hacia el controlador de la interfaz SIO 232/422 que corre en un transputer T225. Para asegurar que esta implementación es capaz de comunicarse con un proceso Occam en un transputer de 16 bits se crearon **echo1** y **echo2** que, teniendo estas características, no hacen más que retransmitir los mensajes que reciban. De esta forma, cuando el controlador del SIO esté liberado, esta implementación puede ser fácilmente adaptada para comunicarse a través del protocolo RS232/RS422.

6.1.2 APLICACIONES "ESCRIBE ARCHIVO DE COMANDOS" Y "LEE ARCHIVO DE COMANDOS"

Como se mencionó al inicio del punto 6.1, la forma de interactuar con la Aplicación MMS es a través de archivos de comandos que contienen PDUs de tipo MMS empacada como cadenas de caracteres. La razón de que los PDUs estén empacados de tal forma se detalla en el punto 6.2.2.

La aplicación WRCMDS graba un archivo con la secuencia de PDUs que le indique el usuario; el software registra un PDU a la vez interactuando con el usuario para completar todos los datos requeridos por el PDU en cuestión. Esta aplicación es muy sencilla y consta del programa principal WRCMDS.C y de los archivos includes MMSPRIMI.H, MMSTYPES.H y MMSCTES.H, compartidos con la Aplicación MMS.

La aplicación RDCMDS puede utilizarse opcionalmente para leer el tipo y los datos de los PDUs que contiene un archivo de comandos MMS grabados por la aplicación WRCMDS; para ejecutar esta aplicación solo basta dar el nombre del archivo que se quiere consultar. El programa principal es llamado RDCMDS.C y también comparte con las otras aplicaciones archivos tales como MMSPRIMI.H, MMSTYPES.H y MMSCTES.H.

El código fuente de ambas aplicaciones puede consultarse en el Apéndice D y la forma de generar y ejecutar los archivos btl se encuentra en el Apéndice F.

6.1.3 PROGRAMA DE CONFIGURACION

Como se planteó en el punto 3.3.8, una de las características más importantes en una aplicación diseñada para transputers es el archivo de configuración. Para la Aplicación MMS se decidió utilizar el lenguaje de configuración Occam debido a que se combinan procesos de origen Occam y C dentro de ésta, y el lenguaje Occam permite hacer paso de canales a ambos tipos de programas mediante el uso de interfaces incluidas para este propósito, las cuales se detallan en el punto 6.2.5 de este mismo capítulo.

El archivo de configuración para la Aplicación MMS se llama MMS.PGM y se incluye en el Apéndice E; la Figura 61 muestra la distribución de los objetos según este archivo. En el archivo MMS.PGM se puede observar que se está utilizando una red de procesadores que tiene al menos tres transputers, un T225 y dos T805 con 16KBytes para el primero y 2Mbytes para los dos

últimos definidos en la sección de NETWORK. Después de esta sección, en la sección de MAPPING, se definen 4 nodos lógicos root.p, client.p, echo1.p y echo2.p los cuales son mapeados al primer T805, al segundo T805 y los dos últimos al T225.

Finalmente en la sección correspondiente a la configuración, CONFIG, se definen los canales relacionados con el Servidor MMS, con el Cliente MMS, con el proceso alimentador de comandos MMS y con el proceso **Monitor;** La última sección del código corresponde a el lanzamiento en paralelo de los cuatro procesos, los cuales son:

- root.occ.- Interfaz Occam-C para el ambiente Servidor MMS y los procesos **Feeder**, **Killer** y **Monitor**.
- client.occ.- Interfaz Occam-C para el ambiente Cliente MMS.
- Echo.- Proceso **Echo**.

root.occ y client.occ son lanzados en los nodos lógicos root.p y client.p, respectivamente; una instancia de **Echo** se corre en el nodo echo1.p y otra en echo2.p.

Un buen diseño en una aplicación para transputers es aquél que permite que los procesos involucrados puedan ser montados en determinados transputers según sea la necesidad de la aplicación, el número de transputers con que se cuente en la red y la topología de esta última, cambiando simplemente el mapeo de los nodos lógicos en los transputers adecuados; todo lo anterior considerando que los procesos que hacen uso de las funciones de comunicación con el host deben estar forzosamente en el nodo 0 de la red; lo cual aplica para el objeto root.occ de la aplicación MMS ya que usa el host como salida del monitoreo de los mensajes que circulan por las máquinas MMS, por el **Server** y por el **Client,** y como entrada del mensaje de terminación del Servidor MMS ("&"). En realidad, la implementación del protocolo, ni en la parte del cliente ni en la del servidor, requiere de intercambio de datos con el host, es la aplicación en particular la que lo necesita.

Los nodos lógicos restantes pueden ser mapeados a cualquiera de los transputers de que estén definidos en la red sin más acciones que las de crear los objetos según sea el tipo de transputer a ocupar y modificar el archivo de configuración.

Por otro lado, los archivos de configuración para las aplicaciones WRCMDS y RDCMDS se llaman WRCMDS.CFS y RDCMDS.CFS y están escritos en C; estos archivos son muy simples ya que solo requieren de un procesador en el cual corre un único proceso generado con código C. Los archivos de configuración también se incluyen en el Apéndice E.

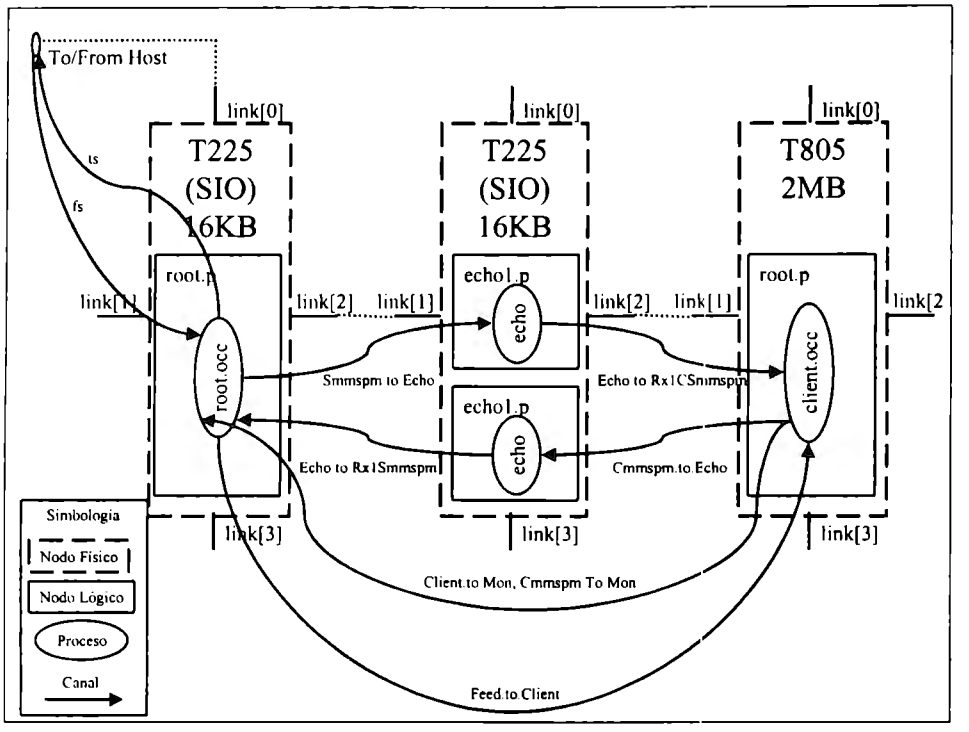


Figura 61 Distribución de la Aplicación MMS según MMS.PGM

6.1.4 LISTA DE ARCHIVOS DE LAS APLICACIONES

Los archivos involucrados en cada una de las aplicaciones se presentan en las tablas a manera de resumen.

Archivo	Descripción
Client.c	Código fuente en C del ambiente del Cliente MMS, recibe como parámetros de entrada un canal de salida y dos de entrada. Define las variables, canales y procesos involucrados en este ambiente y los lanza en paralelo. Hace uso de los archivos includes siguientes: Mmsprimi.H, Mmsctes.H, Mmstypes.H, Mmspdu.H, Mmstype2.H, Mmspdutr.H, Trxbuff.H Y Cmmspm.H
Client_o.occ	Código fuente en Occam de la interfaz Occam-C usada para el proceso **Client**.
Cmmspm.h	Archivo que incluye el código fuente en C para la máquina del protocolo MMS de la entidad Cliente.
Echo.occ	Código fuente en Occam del proceso que emula un eco en los PDUs transmitidos entre el Servidor MMS y el Cliente MMS.
Feeder.h	Archivo que incluye el código fuente en C del proceso alimentador de comando (Feeder).
Mms.pgm	Archivo de configuración de procesos escrito en Occam.

Mmsctes.h	Archivo donde se concentran las constantes usadas relacionadas con esta implementación el protocolo MMS.
Mmsdecod.h	Código fuente en C del proceso que manda los mensajes a pantalla.
Mmsparam.h	Archivo donde se concentran los parámetros de configuración de esta implementación el protocolo MMS.
Mmspdu.h	En este archivo se encuentra la traducción en código C de la especificación para el PDU de MMS original en ASN.1
Mmspdutr.h	Este archivo contiene funciones para convertir el PDU MMS a una cadena de caracteres y viceversa.
Mmsprimi.h	Aquí se incluyen las macro-definiciones relacionadas con las primitivas del protocolo MMS de esta implementación.
Mmsprmc0.h	Funciones de la Máquina del Protocolo MMS.
Mmsprmc1.h	Funciones de la Máquina del Protocolo MMS.
Mmsprmc3.h	Funciones de la Máquina del Protocolo MMS.
Mmsprmc4.h	Funciones de la Máquina del Protocolo MMS.
Mmsprmcf.h	Funciones de la Máquina del Protocolo MMS.
Mmstype2.h	En este archivo, junto con Mmstypes.h, se define los tipos internos a esta implementación.
Mmstypes.h	En este archivo, junto con Mmstypes.h, se define los tipos internos a esta implementación.
Root.c	Código fuente en C del ambiente del Servidor MMS, recibe como parámetros de entrada 2 canales de salida y uno de entrada. Define los procesos y canales necesarios para la implementación del ambiente Servidor MMS y, adicionalmente, el proceso Killer y el Proceso Feeder con sus respectivos canales. El código del Killer también es incluido en este archivo y hace referencia a los archivos Mmsprimi.H, Mmsctes.H, Mmstypes.H, Mmspdu.H, Mmstype2.H, Mmspdutr.H, Trxbuff.H, Smmspm.H, Feeder.H
Root_o.occ	Código fuente en Occam de la interfaz Occam-C usada para el proceso Root.
Smmspm.h	Código fuente en C de la máquina del protocolo MMS para la entidad Servidor.
Trxbuff.h	En este archivo se encuentra el código fuente de los procesos que emulan un canal de longitud >0 y que forman parte integral de esta implementación de MMS.

Tabla 16 Descripción de los archivos de la aplicación MMS.

Archivo	**Descripción**
Rdcmds.C	Código fuente de aplicación Rdcmds usada para leer los archivos de trabajo escritos por la aplicación Wrcmds.btl; incluye los archivos mmsprimi.h, mmsctes.h, mmstypes.h, mmspdu.h, mmstype2.h y mmspdutr.h que se mencionan en la Tabla 16.
Rdcmds.Cfs	Código de configuración en C para la aplicación Rdcmds.btl

Wrcmds.C	Código fuente en C del proceso que escribe comandos MMS a un archivi de trabajo. Hace referencia a los archivos mmsprimi.h, mmsctes.h, mmstypes.h, mmspdu.h, mmstype2.h, mmspdutr.h mencionados en la Tabla 16.
Wrcmds.Cfs	Código de configuración en C para la aplicación Wrcmds.btl

Tabla 17 Descripción de los archivos de las aplicaciones Wrcmds y Rdcomds.

6.2 DETALLES DE LA IMPLEMENTACIÓN

En esta sección de muestran algunos de los detalles más importantes en la implementación del modelo de validación del protocolo MMS.

6.2.1 REGISTRO DE LAS ASOCIACIONES, PETICIONES E INDICACIONES

Como se mencionó en la sección 3.1.3 , el protocolo MMS requiere de que se guardan ciertos parámetros relacionados con las asociaciones, peticiones e indicaciones para el funcionamiento adecuado del mismo, estos parámetros son descritos en las siguientes tablas.

Parámetro	Tipo	Uso
in_use	Booleano	Indica sí el registro de la Indicación o Petición esta siendo usado, es decir, si la información contenida en los demás campos tiene algún sentido para la máquina del protocolo.
invoke_id	Entero	Identificador de la petición o indicación. La MMSPM hace referencia a este número para la búsqueda de la información relacionada con la petición o indicación que se esté procesando
Mmpm_machine_state	Entero	Valor que indica en que estado se encuentra la petición/indicación. Los valores posibles se encuentran definidos en el archivo Mmsprmc0.
TimerState	Entero	Valor utilizado por el proceso centinela para dar mantenimiento a las peticiones e indicaciones. Los valores posibles están definidos en la enumeración llamada Timer_State_type que se puede consultar en el archivo Mmsprmc0.

Tabla 18 Descripción de los parámetros relacionados con las indicaciones y peticiones MMS.

Parámetro	Tipo	Uso

in_use	Booleano	Indica sí el registro de la tabla de asociaciones está en uso, es decir, si la información contenida en los demás campos tiene algún sentido para la máquina del protocolo.
Id	Entero	Identificador de asociación. La MMSPM hace referencia a este número para la búsqueda de la información relacionada con la asociación a la que se está atendiendo.
State	Entero	Valor que indica en que estado se encuentra la asociación.. Los valores posibles se encuentran definidos en el archivo Mmsprmc0.
TimerState	Entero	Valor utilizado por el proceso centinela para dar mantenimiento a las asociaciones que se encuentran en un estado diferente a ASSOCIATED. Los valores posibles están definidos en la enumeración llamada Timer_State_type que se puede consultar en el archivo Mmsprmc0.
Role	Entero	Indica si la asociación fue establecida a petición del proceso local o a petición del proceso remoto.
MaxSegmentSize	Entero	Parámetro negociado que indica cual el máximo de segmentos soportados para esta asociación.
NegociatedMaxServ OutstandingCalling	Entero	Parámetro negociado que indica cuantas peticiones máximas puede soportar esta asociación.
NegociatedMaxServ OutstandingCalled	Entero	Parámetro negociado que indica cuantas indicaciones máximas puede soportar esta asociación.
NegociatedDataStructure NestingLevel	Entero	Parámetro negociado que indica cual el nivel máximo soportado para la definición de variables por ambos nodos que participan en esta asociación.
PendRqtTbl	Apuntador a estructura	Apuntador a la tabla de Peticiones pendientes de esta asociación.
PendIndTbl	Apuntador a estructura	Apuntador a la tabla de Indicaciones pendientes de esta asociación.

Tabla 19 Descripción de los parámetros relacionados con las Asociaciones MMS.

La base de datos requerida para guardar la información indicada en las tablas anteriores está implementada como estructuras que se definen en el archivo Mmsprmc0.h (ver Apéndice C) y que son referidas por los procesos y funciones relacionadas con las máquinas de estados del protocolo MMS.

6.2.2 TRANMISIÓN DE PDUS MMS

El poder del protocolo MMS radica en la estructura estandarizada que conforma todo PDU intercambiado en el ambiente MMS; esta estructura es muy compleja ya que tratar de incluir todos tipos posibles de mensajes (refiérase al punto 4.3.4). Esta estructura debe ser convertida a una cadena de caracteres para poderla transmitirla a través de los canales utilizando las funciones proporcionadas por la herramienta de programación C.

Para ilustrar la forma en que se transmiten y reciben los PDUs MMS se extrajo una sección del código del proceso **Server** que corresponde a la etapa de *Logout*, el cual se puede observar en la Figura 62. El proceso recibe la trama haciendo una llamada a la instrucción *ChanIn* (refiérase al Apéndice A) en el cuál especifica que el canal por donde desea recibir que en este caso es el *FrClnt*, el apuntador de la cadena en donde debe dejar los datos recibidos, *sCmd* y la longitud de los datos esperados *TRAMA_LEN*. Esta instrucción es bloqueante y regresa el control a **Server** hasta que se ha recibido una trama de la longitud especificada; sí se requiere que la recepción sea no bloqueante se debe usar un mecanismo como el que se explica en el punto 6.2.4. La cadena de caracteres recibida es un PDU MMS empacado por lo que hay que desempacarlo para poder trabajarlo adecuadamente; esto lo lleva a cabo la función *UnpackTrama*, la cual deja el PDU obtenido en la estructura de tipo MMS llamada *SMAleaf*. Esta es una recepción típica de un PDU MMS.

```
/*****************************************************************************/
/*** ESTADO DEL PROCESO: LOFF                                              ***/
void ExpLoc(Channel *FrClnt,Channel *ToClnt,Channel *ToServer){
tTrama sCmd;
aleaf SMAleaf;
Primitive_Type Primitiva;

  while(TRUE){
    ChanIn(FrClnt,sCmd,TRAMA_LEN);
    UnpackTrama(&SMAleaf,sCmd);
    MMSDecod(sCmd,1);
    Primitiva=SMAleaf.Primitive;
    switch(Primitiva){
      case LOGOUT_Indication:
        SMAleaf.Primitive=LOGOUT_Confirm;
        RdId(&(SMAleaf.Id),1);              /*leo registro id*/
        PackTrama(&SMAleaf,sCmd);
        ChanOut(ToClnt,sCmd,TRAMA_LEN);
        break;
    .
    .
    .
    }
  }
}
/***                                                                  LOFF  ***/
/*****************************************************************************/
```

Figura 62 Ejemplo de la transmisión y recepción de PDU MMS a través de los canales de los procesos.

En la transmisión se ejecuta el procedimiento contrario, el proceso maneja la estructura MMS haciendo las evaluaciones y asignaciones pertinentes, empaca la estructura usando la instrucción *PackTrama* y la cadena resultante la transmite usando la instrucción *ChanOut*. Un ejemplo de la transmisión se observa en la misma figura en el caso en que la primitiva recibida sea *LOGOUT_Indicaction.*

La función para empacar un PDU MMS a una cadena plana, y su función complementaria, se pueden consultar en el archivo Mmspdutr.h para los detalles. Es conveniente mencionar que para el manejo de las cadenas que representan PDUs MMS no es recomendable usar las funciones para manejo de cadenas ordinarias incluidas en la librería string.h, ya que al empacar un PDU puede ser que los caracteres obtenidos sean de control, e inclusive el que indica terminador de cadena. El archivo Mmspdutr.h se incluye en el Apéndice C al final de este documento

6.2.3 CANALES DE LONGITUD MAYOR A CERO

Los canales implementados en los transputers son de tipo síncrono (refiérase a la sección 3.3), es decir, el proceso que desea transmitir a través de un canal, tiene que esperar en tal instrucción hasta que exista un proceso que intente recibir datos de este mismo canal; esto es debido a que no cuentan con una estructura para almacenar mensajes y por ello se les conoce como canales de longitud cero. Existe un segundo tipo de canales, los del tipo asíncrono, los cuales si cuentan con una estructura de almacenamiento temporal conocida como buffer; el número de mensajes que se pueden almacenar (longitud del canal) determina el número de transmisiones que puede quedar pendientes antes de bloquear al proceso transmisor.

Uno de los requerimientos que se detectaron en la validación fue la necesidad de contar con canales con longitud mayor o igual a tres para evitar deadlocks por falta de recursos; este problema se resolvió sustituyendo dos procesos cooperativos del tipo Productor-Consumidor, un arreglo compartido y la filosofía de la zona crítica en lugar de un canal de recepción simple [3]. Estos procesos se muestran la Figura 63(B).

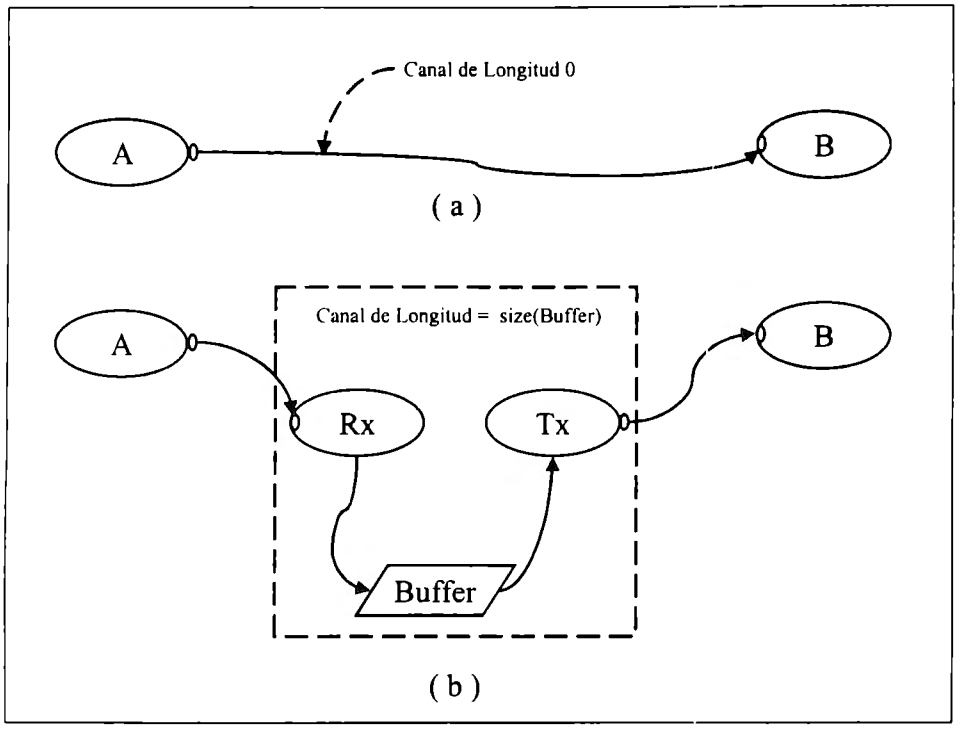


Figura 63 Implementación de Canal con longitud mayor a cero

Si el proceso A desea transmitir el mensaje "M" al proceso B de manera asíncrona, es decir, sin tener que esperar a que B esté disponible para recibir, entonces debe hacerlo a través de la implementación del canal de longitud mayor a 0; por lo que A envía "M" al proceso receptor de B "Rx" y no a B directamente. El proceso receptor de B, lo guarda en un buffer que comparte con el proceso transmisor "Tx", cuya tarea es vaciar tal buffer y enviar el contenido a B. El diagrama detallado de la operación de estos procesos se puede apreciar en el esquema de la Figura 64.

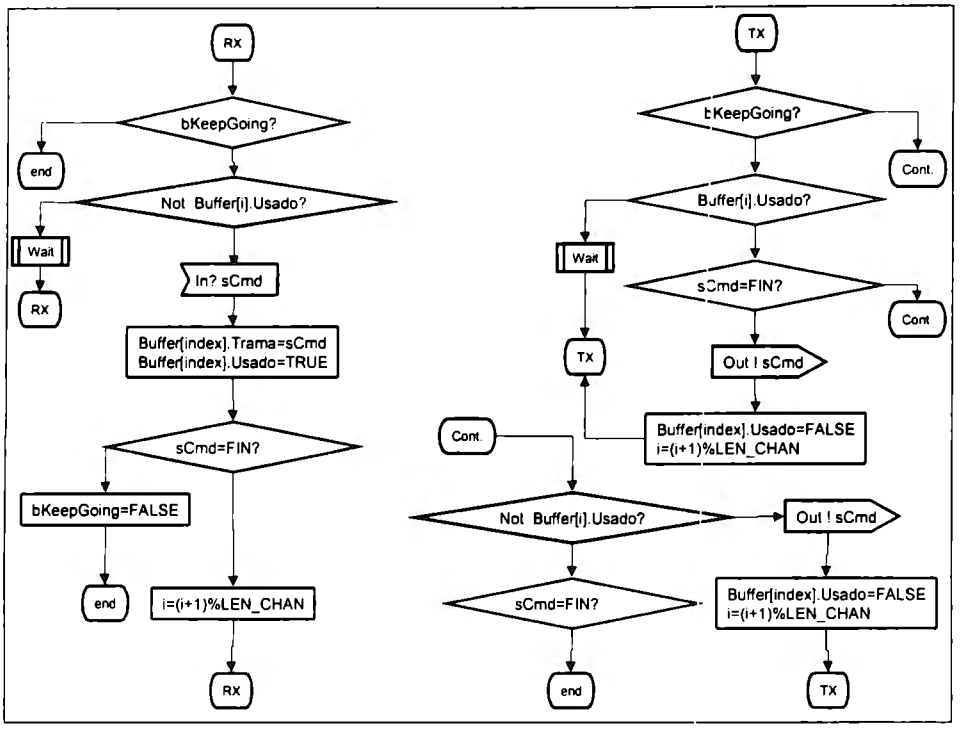


Figura 64 Procesos Rx y Tx para la implementación del canal de longitud > 0.

El proceso **Tx** y **Rx** comparten los recursos *bKeepGoing* y *Buffer*, el primero es una variable booleana inicializada como verdadera usada para que **Rx** indique a **Tx** que continúe vaciando *Buffer*, y el segundo es un arreglo de una estructura de control que contiene un comando y una bandera del tipo booleana llamada *Usado* que indica si el comando es válido y está listo para ser transmitido.

Al inicio, **Rx** verifica si el elemento al que apunta la variable local *index* está disponible; si no es así significa que el comando almacenado en ese elemento no ha sido transmitido por **Tx** por lo que se duerme por un intervalo predefinido después del cuál verifica nuevamente. Por el contrario, si el elemento está disponible entonces **Rx** espera hasta que **A** le envía un mensaje y lo guarda en *Buffer* marcando *Usado* como verdadero; después **Rx** verifica si el tipo de comando que acaba de escribir es del tipo "FIN", si es el caso avisa a **Tx** que es el último de los mensajes cambiando *bKeepGoing* a falso; si el comando no es "FIN", **Rx** incrementa su apuntador a *Buffer* y regresa al punto inicial.

El proceso **Tx**, por su parte, inicia analizando si el comando al que apunta su apuntador a *Buffer* está usado; si no lo está espera a que **Rx** escriba un mensaje válido en tal posición durmiéndose por un tiempo limitado. Si el registro está marcado como usado, significa que el comando referido debe ser transmitido y **Tx** lo hace siempre y cuando el mensaje no sea del tipo "FIN", por que si es así, tiene que abandonar su ejecución. En la transmisión del mensaje, **Tx** TIENE que esperar a que **B** esté disponible para recibir el mensaje y se bloquea hasta que así sea; mientras tanto **Rx** puede seguir recibiendo mensajes de **A** sin que éste último se bloquee como consecuencia de la indisponibilidad de **B** siempre y cuando haya espacio en *Buffer*. Una vez

transmitido el mensaje "M" a **B**, **Tx** libera el registro ocupado por "M", marcando la bandera de *Usado* como falso. **Tx** incrementa su apuntador y vuelve a su estado inicial. Si detecta que debe detener su ejecución (bKeepGoing=FALSE), entonces se dirige al siguiente ciclo el cual esta marcado en la Figura 64 con la etiqueta "cont". Este pequeño bucle funciona de manera similar al primero y está incluido para evitar que se quede un mensaje en *Buffer* sin que sea transmitido al proceso B.

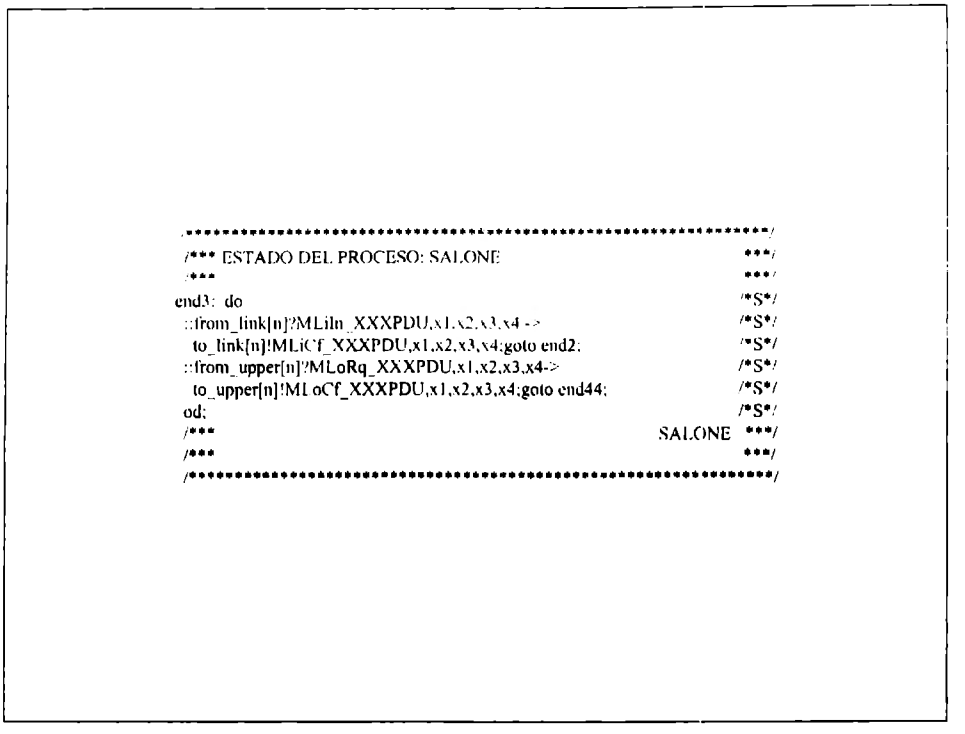
Esta emulación de canal de longitud mayor a cero es muy útil y, como se aprecia en la Figura 56, es utilizada seis veces en la implementación del protocolo MMS, una para cada canal de recepción de los procesos principales. El código de estos dos procesos esta incluido en el punto 11.7.

6.2.4 DISPOSICIÓN PARA ESCUCHAR SIMULTÁNEAMENTE DOS CANALES

En distintas partes del código del modelo de validación para las máquinas del protocolo MMS del Cliente y del Servidor se requiere que el proceso involucrado sea capaz de escuchar simultáneamente del canal conectado al Servidor y del conectado al Cliente y, una vez detectado un mensaje en alguno de ellos, se tiene que llevar a cabo una rutina de servicio de acuerdo al canal de donde proviene el mensaje y al tipo de éste; para implementar esto en código C paralelo se requirió del uso de las instrucciones *ProcTimer* y *ProcWait* (Ver Apéndice A).

A manera de ejemplo, se toma un fragmento del código del proceso **Smmspm** del modelo de validación Promela en que la máquina del servidor se encuentra *StandAlone*, es decir, el servidor se ha registrado y está lista para que un cliente lo haga; en esta situación pueden suceder dos acontecimientos, por un lado se puede recibir un mensaje del servidor en el que requiera terminar su ciclo de vida *RequerimientoLogout* y por otro, el cliente puede enviar una primitiva *IndicaciónLogin;* cada mensaje es temporalmente independiente del otro, por lo que **Smmspm** debe estar monitoreando ambas posibilidades sin ninguna prioridad. En Promela, esto es entendido por el validador al incluir las dos alternativas como enunciados dentro de una instrucción do-od marcado con la etiqueta "end3" de la Figura 65. El proceso se detendrá en esta línea hasta que alguna de los dos enunciados se vuelva ejecutable, es decir, cuando se reciba alguno de los dos mensajes posibles.

La implementación de este comportamiento en C se puede observar en la Figura 66. En este código se observa el uso de la instrucción *ProcAlt* a la cual se le están pasando como parámetros los canales *FrServer* y *FrClnt* que son los que conecta al **Smmspm** con el servidor y con el cliente, respectivamente. La ejecución de este proceso se detiene indefinidamente hasta que se detecte la recepción por alguno de estos dos canales y la función indica el canal que contiene mensaje regresando la posición que éste guarda en la lista de parámetros, es decir, 0 si se detectó un mensaje en *FrServer* y 1 si se detectó en *FrClnt.*



```
/*****************************************************************/
/*** ESTADO DEL PROCESO: SALONE                                ***/
/***                                                           ***/
end3:  do                                                      /*S*/
  ::from_link[n]?MLiIn_XXXPDU,x1,x2,x3,x4 ->                   /*S*/
    to_link[n]!MLiCf_XXXPDU,x1,x2,x3,x4;goto end2;             /*S*/
  ::from_upper[n]?MLoRq_XXXPDU,x1,x2,x3,x4->                   /*S*/
    to_upper[n]!MLoCf_XXXPDU,x1,x2,x3,x4;goto end44;           /*S*/
  od;                                                          /*S*/
  /***                                                 SALONE ***/
  /***                                                       ***/
  /*****************************************************************/
```

Figura 65 Proceso Smmspm en Promela. Oyente simultáneo de dos canales

```
/*****************************************************************************/
/*** ESTADO DEL PROCESO: SALONE      (end3)                    ***/
int end3(Channel *FrClnt,Channel *ToClnt,Channel *FrServer,Channel *ToServer){
tTrama sCmd;aleaf SMAleaf;int ch;
 while(TRUE){
  ch=ProcAlt(FrServer,FrClnt,NULL);
  switch (ch){
   case 0:
    ChanIn(FrServer,sCmd,TRAMA_LEN);
    UnpackTrama(&SMAleaf,sCmd);
    MMSDecod(sCmd,ch);
    if(SMAleaf.Primitive==LOGOUT_Request){
     SMAleaf.Primitive=LOGOUT_Confirm;
     PackTrama(&SMAleaf,sCmd);
     ChanOut(ToServer,sCmd,TRAMA_LEN);
     end44(ToServer,ToClnt);                    /*goto end44*/
     return(FALSE);      }
    break;
   case 1:
    ChanIn(FrClnt,sCmd,TRAMA_LEN);
    UnpackTrama(&SMAleaf,sCmd);
    MMSDecod(sCmd,ch);
    if(SMAleaf.Primitive==LOGIN_Indication){
     SMAleaf.Primitive=LOGIN_Confirm;
     RdId(&(SMAleaf.Id),1);          /*leo registro id*/
     PackTrama(&SMAleaf,sCmd);
     ChanOut(ToClnt,sCmd,TRAMA_LEN);
     return(TRUE);                  /*goto end2*/      }
    break;  } }
***                                    SALONE  ***/
/*****************************************************************************/
```

Figura 66 Proceso Smmspm en C paralelo. Oyente simultáneo de dos canales

6.2.5 INTERFAZ OCCAM - C

Como se mencionó en el punto 6.1.3, para esta implementación se requirió de las interfaces Occam-C tipo 2 y 3 (refiérase al punto 3.3.9) para lanzar los procesos principales de los ambientes Servidor MMS y Cliente MMS respectivamente.

La interfaz root_o.cc es del tipo 2, ya que se requiere que algunos de los procesos que se crean dentro del ambiente Servidor MMS tengan comunicación con el host. Es el caso del proceso **Feeder**, el cual abre un archivo del host, lo lee y lo cierra; **Killer** también requiere de comunicación con el host ya que se función estaba basada en el poleo del teclado; por último, **Monitor** requiere escribir en pantalla los mensajes enviados por los procesos supervisados. Además de estar comunicados hacia el host estos procesos interactuan con otros que se pueden encontrar en procesadores distintos al suyo.

La interfaz client_o.occ es del tipo 3, ya que los procesos del ambiente cliente se comunican con procesos que residen en otros procesadores pero no se requiere una comunicación hacia el host.

El código de ambas interfaces es muy sencillo, ya que su tarea se reduce a instanciar a los procesos client.c y root.c, los cuales son los que realmente realizan la administración de los recursos; este código se incluye en el Apéndice C al final de este documento.

7 CONCLUSIONES.

Una Celda de Manufactura es un conjunto de máquinas, robots, bandas y otros servomecanismos que son utilizados para realizar una tarea específica dentro de un proceso de manufactura; por ejemplo, dentro de una planta de ensamble automotriz, la sección encargada del pintado final de la unidad es una celda. La introducción de computadoras, programas y demás componentes computacionales para el control de estos elementos dio como origen a la llamada Manufactura Integrada por Computadora, que tiene entre otros objetivos el optimizar la producción.

La Ciencia de la Computación también puede ayudar a que la operación de la celda pueda ser adaptada rápidamente a nuevos requerimientos de la producción y además, puede reasignar las tareas de cada elemento de la celda en caso de que alguno de estos falle; estos dos aspectos hacen que la celda sea flexible y tolerante a fallas; esto se logra ejecutando procesos concurrentes y comunicantes para el control de la operación de la celda.

Una plataforma que permite que una celda tenga las características mencionadas es el transputer. Este es un microprocesador de alto rendimiento que soporta el procesamiento paralelo en el mismo circuito integrado ya que cuentan con un mecanismo de asignación de CPU altamente eficiente para los procesos que corren en un mismo transputer; el tiempo de cambio de contexto es menor a un microsegundo. Si los transputers son interconectados a través de sus enlaces, el procesamiento paralelo se puede llevar a cabo en distintos procesadores; la sincronización de tareas y el intercambio de información se implementa a través de los enlaces. Existen varias tarjetas madres de transputers que permiten la creación de redes de estos; una de ellas es la tarjeta IMS B008 que cuenta con el CO04, un múltiple interruptor digital controlado por software.

Un requerimiento implícito en la implementación de una celda integrada por computadora es que debe contar una comunicación en tiempo real y confiable; los transputers soportan este tipo de comunicación ya que sus principales características en este sentido son:

- Mecanismo de asignación de tiempo de CPU altamente eficiente.
- Enlaces y canales de comunicación altamente eficientes y confiables.
- Manejo de Prioridades para los procesos paralelos.
- Implementación simple del manejo de interrupciones.

- Programación sencilla de temporizadores, que permite el control de tiempos y un poleo no bloqueante.
- Colocación de variables en direcciones específicas de memoria.

Además la comunicación de la celda debe estar orientada a los estándares internacionales para que la integración de nuevos elementos a la celda sea simple y de bajo costo; se debe evitar en la medida de lo posible las soluciones propietarias. MAP es el estándar en materia de la comunicación en las aplicaciones de manufactura y MiniMAP específicamente en las celdas; este último es un modelo de tres capas (aplicación, enlace y física) cuyo objetivo es trabajar con los mensajes más simples para hacer la comunicación más eficiente. MiniMAP establece reglas para las tres capas pero su importancia radica en la especificación de los mensajes de manufactura MMS el cual determinar las reglas y formatos validos para los mensajes intercambiados en una celda de manufactura para la generalidad de los elementos de una celda; aunado a este estándar existen otras reglas llamadas estándares asociados MMS que son aplicables según la naturaleza del dispositivo de manufactura: para robots, para máquinas de control numérico y para controladores programables.

Un protocolo óptimo para la celda debe asegurar que en el intercambio de la información se respete el estándar MMS explotando al máximo las características de los transputers antes mencionadas. Un protocolo define un formato conciso para los mensajes válidos que intercambiaran los nodos participantes en la comunicación (sintaxis), las reglas procedurales para el intercambio de información (gramática) y el vocabulario de mensajes válidos con su significado (semántica). En el diseño del protocolo se debe cuidar que éste sea simple, modular y que este completamente especificado; lo último significa que todas las condiciones posibles en la comunicación deben ser tomadas en cuenta para que las entidades comunicantes puedan comportarse adecuadamente cuando estas se presenten.

Una fase muy importante en la implantación del protocolo es la verificación de que éste cumpla con los objetivos para los que fue diseñado; a esto se le conoce como validación. La validación y el diseño del protocolo no son dos etapas independientes, sino que se deben llevar a cabo a la par, diseñar, validar, volver a diseñar y a validar y así sucesivamente hasta que el protocolo satisfaga los requerimientos. Existen dos tipos de validación, la exhaustiva y la no-exhaustiva; con la primera todas las secuencias del protocolo posibles son sometidas a verificación, la segunda sólo prueba algunas secuencias, escogidas según una técnica y es muy útil cuando los estados del protocolo son muchos y la verificación exhaustiva requeriría demasiado poder computacional.

La validación permite garantizar que el protocolo este completamente especificado, sin embargo una verificación manual sería muy lenta y sujeta a errores humanos, por lo que se hace uso de las herramientas de validación. Spin es un software muy usado para la validación de protocolos y es de dominio público. Esta herramienta puede realizar validaciones exhaustivas y no exhaustivas, probando propiedades diversas como: estados finales inaceptables, ciclos erróneos, aserciones, invariantes del sistema y requerimientos temporales. La validación del protocolo se hace sobre un modelo formal, se debe asegurar que la implementación sea una equivalencia total a este modelo.

El protocolo diseñado en este trabajo respeta las normas establecidas por MMS y su implementación fue realizada haciendo uso de las características técnicas de los transputers y sus redes. Ha sido verificado usando la validación no exhaustiva para verificar la ausencia de estados finales inaceptables, ausencia de ciclos erróneos y el mantenimiento de las invariantes del sistema, usando la herramienta Xspin.

7.1 TRABAJO A FUTURO.

- Este software está completo para que los procesos participantes lo usen como un mecanismo de comunicación; Sin embargo, es conveniente remarcar que esta implementación debe ser generalizada si se requiere que un proceso servidor tenga más de un cliente en forma simultánea.

- La integración del controlador de las interfaces RS-232 y RS-422 no debe representar desgaste alguno en la implementación de la comunicación real al robot debido a que el software realizado en este trabajo contempla que existe un proceso en el transputer del SIO (T222), sin embargo es un trabajo adicional a esta tesis.

8 BIBLIOGRAFIA.

[1] J. Corona, J.Sánchez, C. Rodriguez, I. Rodumín, F.Ramos y R. Valdivia "Tecnologías de Comunicación Avanzada en Robótica y Celdas Flexibles de Manufactura". Documento del Proyecto NSF-CONACyT. México. 1996.

[2] R. Bottle. "Modulo de Interfase Programable". Tesis de Licenciatura. ITESM – CEM. México 1992.

[3] G.Coulouris. "Distibuited Systems: Concepts and Design". Ed. Addison Wesley. 1994.

[4] Andrew S. Tanenbaum, "Computer Networks". Prentice Hall, 1988. 2nd. edition

[5] L. J. McGuffin, L. O. Reid, S. R. Sparks. "MAP/TOP in CIM, Distribuiting Computing". Paper. IEEE Network. 1989.

[6] Dieter Hogrefe, "Validation of SDL systems". Paper. Computer Network and ISDN. Volumen 28, 1996.

[7] Ana R. Carvalli, Byonung. Moon Chin. "Testing methods for SDL systems". Paper. Computer Network and ISDN. Volumen 28, 1996.

[8] A. Valenzano, C. Demartini, L.Ciminiera, "MAP and Top Communications: standards and applications". Ed. Addison-Wesley. 1991.

[9] Henrik A. Schutz. "The Role of MAP in Factory Integration". Paper. IEEE Network. 1988

[10] Simon S. Chung. "The Control of a Model Factory: a MAP Work Place". Paper. IEEE Network. 1989.

[11] C.J. Cassidy. "Mastering Communications Using the Manufacturing Message Specification". Welding Institute. Reino Unido.

[12] Sánchez J., "A Flexible Manufacturing Cell Distribuited Controller". 4th International Symposium on Applied Corporate Computing (ISACC). Monterrey, México, 1996.

[13] Salmerón G. Mirna. "Diseño de la Arquitectura de Hardware utilizando transputers para el Conttrol de una Celda Flexible de Manufactura". Tesis. de Maestría del ITESM- CEM. México, 1997.

[14] Morales S. Edgar "Diseño de Interfaz y Conmutador de Comunicación entre Controlador y Elementos de una Celda de Manufactura Flexible". Propuesta de Tesis. ITESM- CEM. México, 1996.

[15] Zavala Vega Laura Elena, "Implementación de distintas Técnicas de Tolerancia a Fallas en una Celda Flexible de Manufactira". Propuesta de Tesis. ITESM- CEM. México, 1996.

[16] ANSI C Toolset User Guide, SGS-Thompson Microelectronics, September 1995

[17] J.N. Daigle, A.Seidmann, J.R. Pimentel. "Communications for Manufacturing : An Overview." IEEE May 1988.

[18] Mota González Sara. "Protocolos de Comunicación en una Arquitectura de Transputers para Controlar una celda flexible de manufactura". Tesis. de Maestría del ITESM- CEM. Touca, México, 1997.

[19] Alarcón Félix Jaimes, "Diseño de una Arquitectura Reconfigurable basada en Transputers", Tesis de Maestría ITESM CEM, 1997.

[20] Gerarld J. Holzmann. "Design and Validation of Computer Protocols". Prentice Halls.

[21] "Transtech SIO232". Transtech Parallel Systems. E.U. 1991.

[22] Webber do Prado Maria F., "Projeto EPA/MAP". Documento en Internet: [http:// www.penta.ufrgs.br/ rc952/ trab2/ mms3.html]. 1998.

[23] Mackiewicz Ralph, "An Overview to the Manufacturing Message Specification". Documento en Internet: [ftp:// litsun.epfl.ch/MMS/ MMS.txt]. Sisco Inc, 1998.

[24] "World-wide Enviroment for Learning LOTOS". Documento en Internet: [http:// www.cs.stir.ac.uk/ ~kjt/ research/well/intro.html]

[25] "Formal Description Techniques". Documento en Internet: [http:// www.fsz.bme.hu/ ~mohacsi/ formal.html]

[26] "Specification and Description Language". Documento en Internet: [http:// www.tdr.dk/ public/ SDL/]

[27] Pierre Castori, "On-line Access to the MMS Abstract Syntax". Documento en Internet: [http:// litwww.epfl.ch/MMS/ mms_abstract_syntax.html]. Ecole Polytecnique Fédérale de Laussane, 1998.

[28] Bell Labs, "SPIN Readme". Documento en Internet: [http:// cm.bell-labs.com/ cm/ cs/ what/ spin/ Man/ README.html]. Bell Labs, 1998.

9 APÉNDICE A

ALGUNAS FUNCIONES DE C PARALELO

Los canales son un medio de comunicación que sirven para transferir datos de un proceso a otro o para sincronizar las acciones entre procesos paralelos. Si un proceso necesita esperar a que otro alcance un estado en especial un canal es de suma utilidad; Si un proceso tiene un acceso exclusivo a un recurso particular y actúa como servidor de otros procesos, los canales pueden servirle como un mecanismo de atención a peticiones.

Cualquier tipo de dato se puede pasar a través de un canal pero el usuario debe asegurar que los procesos comunicantes entiendan el mismo protocolo para que los datos sean interpretados correctamente. Los canales sólo operan en una dirección; los procesos que requieren enviarse información en ambos sentidos, requieren de dos canales.

Los canales entre procesadores son implementados sobre los enlaces; los canales entre procesos que corren en un mismo transputer son conocidos como canales *soft;* los enlaces de los transputers también son conocidos como canales *hard*. Los canales pueden ser declarados como variables de cualquier tipo pero para usarse deben inicializarse correctamente; esto se hace automáticamente para los canales que son creados por el configurador y para los alojados por un programa C se debe hacer explícitamente usando una de las siguientes funciones:

```
Channel * ChanAlloc (void);
void ChanInit (Channel *)
```

La función *ChanAlloc* aloja el espacio para un canal e inicializa el canal antes de regresar. Si el canal ya ha sido alojado en memoria, puede ser inicializado con la instrucción *ChanInit*.

El valor inicial de un canal es una constante llamada *NotProcess_p*, que se define en channel.h.; este valor indica que el no existe proceso alguno esperando en el canal para escribir ni para leer. Si un canal se debe pasar como parámetro a dos procesos entonces se debe inicializar antes de pasarlo y no dejar que uno de los procesos sea el que lo haga.

Para escribir a los canales se proveen las siguientes instrucciones:

```
void ChanOut  (Channel *c, void *cp, int count);
void ChanOutChar (Channel *c, unsigned char);
void ChanOutInt (Channel *c, int n);
void DirectChanOut  (Channel *c, void *cp, int count);
void DirectChanOutChar (Channel *c, unsigned char);
void DirectChanOutInt (Channel *c, int n);
```

Las funciones *ChanOutChar* y *ChanOutInt* transfieren un caracter y un entero respectivamente; *ChanOut* es una función general de transferencia que envía *count* bytes de datos de un arreglo de cualquier tipo de dato apuntado por *cp*. Las tres últimas rutinas tienen funciones equivalentes a las primeras pero ejecutan la operación directamente usando la instrucción de los transputers *output*; por ello estas rutinas sólo pueden usarse en las siguientes clases de canales:

En los canales que comparte procesos en un mismo procesador (canales *soft*).
En los canales directos; los cuales son los canales *hard* que comparte procesadores adyacentes y que sólo existen dos de ellos, uno en cada dirección.

Cada una de las funciones de salida representa una sola comunicación; el proceso no continuará hasta que la transferencia esté completa. Estas funciones tienen sus complementarias para la lectura de valores de un canal, las cuales son:

```
void ChanIn  (Channel *c, void *cp, int count);
unsigned char ChanOutChar (Channel *c );
int n ChanOutInt (Channel *c );
void DirectChanIn  (Channel *c, void *cp, int count);
unsigned char DirectChanOutChar (Channel *c );
int n DirectChanOutInt (Channel *c );
```

Pero debido a que existen muchos casos en los que un proceso debe escuchar de varios canales y desea saber cual de ellos tiene datos listos, se agregan las siguientes funciones que ayudan a determinar que canal alterno está listo para ser leido:

```
int ProcAlt (Channel * c1, ... );
```

```
int ProcAltList (Channel * *clist);
int ProcSkipAlt (Channel * cl, ... );
int ProcSkipAltList (Channel * *clist);
```

Todas estas funciones utilizan como parámetros una lista de apuntadores a canal terminadas por un apuntador a NULL; *ProcAlt* y *ProcAltSkip* regresan el índice (que empieza en cero) de la lista para indicar que canal está listo para la transferencia de datos.

El siguiente es un ejemplo simple de un servidor que convierte a mayúsculas y regresa los datos al cliente que envió la solicitud.

```
void upserver (Process * p, Channel * ins[], Channel * outs[])
{
        size_t i;
        char data;

        p = p;
        /*bucle infinito, este proceso nunca termina*/
        for (;;)
        {
                /*Espera por datos de entrada*/
                i = ProcAltList (ins);

                /*Lee y convierte los datos*/
                data = ChanInChar (ins[i]);
                data = toupper (data);

                /*Envia la conversión por el canal correspondiente*/
                ChanOutChar (outs[i], data);
        }
}
```

ProcAlt y *ProcAltList* son funciones bloqueantes y regresan hasta que uno de los canales este listo. Las variantes de estas funciones *ProcSkipAlt* y *ProcSkipAltList*, pueden usarse para verificar si hay o no un canal listo; regresan el valor (-1) si ninguno de ellos contiene datos, de otra manera trabajan exactamente como *ProcAlt* y *ProcAltList*, respectivamente.

Por otro lado, los transputers cuentan con dos relojes integrados en el mismo chip, uno usado para los procesos de alta prioridad y uno para los de baja; el de baja velocidad corre a una razón de 15625 pulsaciones por segundo, el da alta a un millón de pulsaciones por segundo. Cada proceso lee el reloj correspondiente según sea su prioridad.

Un proceso puede esperara por cierto tiempo (medido en pulsaciones por segundo), usando las siguientes funciones:

```
void ProcAfter (int t);
void ProcWait (int t);
```

Ambas funciones esperan por un período después del cual regresan: *ProcAfter* espera hasta que se alcance la lectura absoluta del tiempo en el reloj. Si el tiempo indicado no es mayor que el actual, la función regresa inmediatamente. *ProcWait* suspende el proceso hasta que ha transcurrido el intervalo indicado, es decir, retrasa la ejecución por un número especifico de pulsaciones. Si se especifica un tiempo negativo, la función regresa inmediatamente.

Existe dos funciones que permiten que los procesos selecciones un canal de recepción que este listo dentro de un tiempo especifico. Si ningún canal se hace disponible durante este tiempo, entonces la función regresa y el proceso sigue con su ejecución. Estas funciones son:

```
int ProcTimeAlt (int time, Channel * C1, ... );
int ProcTimeAltList (int time, Channel * * clist);
```

Estas funciones regresan (-1) si el tiempo expira antes de que haya datos disponibles en alguno de los canales incluidos en la lista.

10 APÉNDICE B

CÓDIGO DEL MODELO DE VALIDACIÓN PARA EL PROTOCOLO MMS

10.1 MMS_MAL.PRO

```
/***                              ***/
/***          MMS_Val             ***/
/***                              ***/

#define TRUE  1
#define FALSE 0
#define QSZ 3   /*tamanio del canal entre app y maq del proto   */
#define QSZ2 3  /*tamanio del canal entre maquinas del protocolo*/
mtype={
  IdleConn,CallingInit,CalledInit,CallingConc,CalledConc,Associated,
  Idle,W_Conf,W_Resp,
  MAsRq_initiate_RequestPDU,MAsRq_initiate_XXXPDU,
  MAsIn_initiate_RequestPDU,MAsIn_initiate_XXXPDU,MAsIn_reject_PDU,
  MAsRs_initiate_ResponsePDU,MAsRs_initiate_XXXPDU,MAsRs_initiate_ErrorPDU,
  MAsCf_initiate_ResponsePDU,MAsCf_initiate_XXXPDU,MAsCf_initiate_ErrorPDU,MAsCf_reject_PDU,
  MRlRq_conclude_RequestPDU,MRlRq_conclude_XXXPDU,
  MRlIn_conclude_RequestPDU,MRlIn_conclude_XXXPDU,MRlIn_reject_PDU,
  MRlRs_conclude_ResponsePDU,MRlRs_conclude_XXXPDU,MRlRs_conclude_ErrorPDU,
  MRlCf_conclude_ResponsePDU,MRlCf_conclude_XXXPDU,MRlCf_conclude_ErrorPDU,MRlCf_reject_PDU,
  MCfRq_confirmed_RequestPDU,MCfRq_confirmed_XXXPDU,
  MCfIn_confirmed_RequestPDU,MCfIn_confirmed_XXXPDU,MCfIn_reject_PDU,
  MCfRs_confirmed_ResponsePDU,MCfRs_confirmed_XXXPDU,MCfRs_confirmed_ErrorPDU,
  MCfCf_confirmed_ResponsePDU,MCfCf_confirmed_XXXPDU,MCfCf_confirmed_ErrorPDU,MCfCf_reject_PDU,
  MUfRq_unconfirmed_PDU,MUfRq_unconfirmed_XXXPDU,
  MUfIn_unconfirmed_PDU,MUfIn_unconfirmed_XXXPDU,MUfIn_reject_PDU,
  MAbIn_abort_PDU,MAbIn_abort_XXXPDU,
  MLiRq_XXXPDU,MLiIn_XXXPDU,MLiCf_XXXPDU,MLiCf_ErrPDU,
  MLoRq_XXXPDU,MLoIn_XXXPDU,MLoCf_XXXPDU
}

chan to_link[2]   = [QSZ2] of {byte,byte,byte,byte,byte};
chan from_link[2] = [QSZ2] of {byte,byte,byte,byte,byte};
chan to_upper[2]  = [QSZ] of {byte,byte,byte,byte,byte};
chan from_upper[2] = [QSZ] of {byte,byte,byte,byte,byte};

#include "cmmsprm.pro"
#include "smmsprm.pro"
#include "cupper.pro"
#include "supper.pro"
/* Para chechar las invariantes del sistema:
#include "inv_sys1.pro"
*/
/* Para checar el progreso en el Login*/
#include "nev_li1.pro"

init
{
  atomic{
    to_link[0]=from_link[1];
    to_link[1]=from_link[0];
    run smmspm(0); run cmmspm(1);
    run supper(0); run cupper(1);
/*    run monitor(); */
  }
}
```

10.2 CUPPER.PRO

```
/***                          ***/
/***          CUPPER          ***/
/***                          ***/


proctype cupper(bit n)
{
  /*****************************************************************************/
  /***                            Variables                                  ***/
  /*****************************************************************************/
  byte dummy;          /*variable dummy*/
  byte x1;             /*otra variable dummy*/
  byte x2;             /*otra variable dummy*/
  byte x3;             /*otra variable dummy*/
  byte x4;             /*otra variable dummy*/
  byte j;
  byte k;              /*assid local                    */
  byte l;              /*assid remota                   */
  byte Associa[MC];
  byte q;              /*apuntador a una asociacion a guardar*/
  byte p;              /*apuntador a una asociacion a concluir*/
  byte PetConf;        /*apuntador a una asociacion disponible para enviar un ser conf*/
  byte p1;             /*apuntador a una asociacion a concluir*/
  byte count;          /*control para dar de baja asociaciones*/
  byte AsocCons;       /*control para dar de baja asociaciones*/
  byte Nevers;         /*valor invalido para s*/

  /*****************************************************************************/
  /***                            Inicializacion                             ***/
  /*****************************************************************************/
 q=0;
  Nevers=255;
  do
  ::(q<MC)->
    Associa[q]=Nevers;
    q=q+1
  ::(q>=MC)->
    break;
  od;
  q=0;
  count=0;
  /***                            Inicializacion                             ***/
  /*****************************************************************************/

  /*****************************************************************************/
  /***                            Login                                      ***/
  /*****************************************************************************/
  from_upper[n]!MLiRq_XXXPDU,dummy,dummy,dummy,dummy->
  do
  ::to_upper[n]?MLiCf_XXXPDU,x1,x2,x3,x4->goto cont;
  ::to_upper[n]?MLiCf_ErrPDU,x1,x2,x3,x4->goto fin;
  od;
  /***                            Login                                      ***/
  /*****************************************************************************/

  /*****************************************************************************/
  /***                            Ciclo principal                            ***/
  /*****************************************************************************/
cont:  do
  /*requiero asociacion*/
  ::from_upper[n]!MAsRq_initiate_RequestPDU,dummy,dummy,dummy,dummy->
    do
    ::to_upper[n]?MAsCf_initiate_ResponsePDU,x1,x2,x3,x4->
      count=count+1;
      AsocCons=AsocCons+1;
      Associa[q]=x1;
      q=(q+1)%MC;
      break;
    ::to_upper[n]?MAsCf_initiate_ErrorPDU,x1,x2,x3,x4->break;
    ::to_upper[n]?MAsCf_reject_PDU,x1,x2,x3,x4->break;
```

```
    od;
   /*enviar servicions confirmados*/
   ::(count>0)->
     j=PetConf;
     do
     ::(j<MC && Associa[j]!=Nevers)->
       PetConf=j;
       break;
     ::(j<MC && Associa[j]==Nevers)->
       j=(j+1)%MC;
     od;
     p1=Associa[PetConf];
     from_upper[n]!MCfRq_confirmed_RequestPDU,p1,x2,x3,x4->
     if
       ::to_upper[n]?MCfCf_confirmed_ResponsePDU,x1,x2,x3,x4;
       ::to_upper[n]?MCfCf_confirmed_ErrorPDU,x1,x2,x3,x4;
       ::to_upper[n]?MCfCf_reject_PDU,x1,x2,x3,x4;
     fi;
     do
     ::PetConf=(PetConf+1)%MC->break;
     ::PetConf=PetConf->break;
     od;
   /*enviar servicions no confirmados*/
   ::(count>0)->
     j=PetConf;
     do
     ::(j<MC && Associa[j]!=Nevers)->
       PetConf=j;
       break;
     ::(j<MC && Associa[j]==Nevers)->
       j=(j+1)%MC;
     od;
     p1=Associa[PetConf];
     from_upper[n]!MUfRq_unconfirmed_PDU,p1,x2,x3,x4->
     do
     ::PetConf=(PetConf+1)%MC->break;
     ::PetConf=PetConf->break;
     od;

   /*dar de baja asociaciones*/
   ::(count>0)->
     j=p;
     do
     ::(j<MC && Associa[j]!=Nevers)->
       p=j;
       break;
     ::(j<MC && Associa[j]==Nevers)->
       j=(j+1)%MC;
     od;
     p1=Associa[p];
     from_upper[n]!MRlRq_conclude_RequestPDU,p1,dummy,dummy,dummy;
     do
     ::to_upper[n]?MRlCf_conclude_ResponsePDU,x1,x2,x3,x4->
       count=count-1;
       Associa[p]=Nevers;
       p=(p+1)%MC;
       break;/*concluyo OK*/
     ::to_upper[n]?MRlCf_conclude_ErrorPDU,x1,x2,x3,x4->break;/*NO Concluyo*/
     ::to_upper[n]?MRlCf_reject_PDU,x1,x2,x3,x4->break;
     od;
   ::(count==0)->skip;
   ::(count==0)->goto termina;
   od;
   /*****************************************************************************/
   /***                          Terminacion                                  ***/
   /*****************************************************************************/
termina: from_upper[n]!MLoRq_XXXPDU,dummy,dummy,dummy,dummy->
   do
   ::to_upper[n]?MLoCf_XXXPDU,x1,x2,x3,x4-> goto fin;
   od;
   /***                          Terminacion                                  ***/
fin: skip;}
```

10.3 CMMSPRM.PRO

```
/***                          ***/
/***          CMMSPM          ***/
/***                          ***/

#define MC    4       /*No. maximo de conexiones          */
#define MR    1       /*No.max de pet.simultaneas x conex.*/
#define MA   40       /*No. maximo de asociaciones        */
#define MI   200      /*No. maximo del id de la peticion  */
byte NCD_CM;          /*numero de conexiones disponibles  */
proctype cmmspm(bit n)
{
  /*****************************************************************************/
  /***                          Variables                                    ***/
  /*****************************************************************************/
  bool bExiste;                       /*indica si assid esta registrado   */
  byte NRD[MC];                       /*# de petciones disp. por conex.   */
  byte NID[MC];                       /*# de indicaciones disp. por conex.*/
  byte q;                             /*apuntador a conexion del timer    */
  byte i;                             /*apuntador a conexion libre        */
  byte k;                             /*assid local                       */
  byte l;                             /*assid remota                      */
  byte k1;                            /*reqid local                       */
  byte l1;                            /*indid remota                      */
  byte s;                             /*ass disponible                    */
  byte t;                             /*id rqt proximo disponible         */
  byte u;                             /*id ind proximo disponible         */
  byte j;             /*indice                          */
  byte x1;            /*variable dummy                  */
  byte x2;            /*variable dummy                  */
  byte x3;            /*variable dummy                  */
  byte x4;            /*variable dummy                  */
  byte AssId[MC];     /*buffer para guardar #ass local  */
  byte AssIdRmt[MC];  /*buffer para guardar #ass remota */
  byte State[MC];     /*buffer para guardar el edo      */
  bool InUse[MC];     /*buffer para guardar #ass remota */
  byte RqtId[MC];     /*buffer para guardar #pet local  */
  byte RqtRmtId[MC];  /*buffer para guardar #pet remota */
  byte IndId[MC];     /*buffer para guardar #ind rmt    */
  byte RqtState[MC];  /*buffer para guardar #pet local  */
  byte IndState[MC];  /*buffer para guardar #pet local  */
  byte RqtInUse[MC];  /*buffer para guardar #ind rmt    */
  byte IndInUse[MC];  /*buffer para guardar #ind rmt    */
/*  byte NumSerUnC;      Numero de Pet de Serv. No COnf*/
  /*****************************************************************************/
  /***                  Inicializacion de Variables                          ***/
  /*****************************************************************************/
/*  j=0;
  do
  ::(j<MC)->atomic{
    InUse[j]=FALSE;
    NRD[j]=MR;
    NID[j]=MR;
    RqtInUse[j]=FALSE;
    IndInUse[j]=FALSE;
    j=j+1;
    }
  ::(j>=MC)->
    break;
  od;*/
  NCD_CM=MC;
  s=(l3+n*l1);
    NRD=MR;
    NID=MR;
    RqtInUse=FALSE;
    IndInUse=FALSE;
  /***                                                        INICIA   ***/
  /***                                                                 ***/
  /*****************************************************************************/
```

```
  /*****************************************************************************/

  /*****************************************************************************/
  /*****************************************************************************/
  /*** ESTADO DEL PROCESO: BEGIN                                             ***/
  /***                                                                       ***/
  from_upper[n]?MLiRq_XXXPDU,x1,x2,x3,x4->                                  /*C*/
    to_link[n]!MLiIn_XXXPDU,x1,x2,x3,x4;goto chk;                           /*C*/
  /***                                                          BEGIN        ***/
  /***                                                                       ***/
  /*****************************************************************************/
  /*****************************************************************************/

  /*****************************************************************************/
  /*****************************************************************************/
  /*** ESTADO DEL PROCESO: CHK                                               ***/
  /***                                                                       ***/
chk:  do
  ::from_link[n]?MLiCf_XXXPDU,x1,x2,x3,x4->                                 /*C*/
    to_upper[n]!MLiCf_XXXPDU,x1,x2,x3,x4;goto end2;                         /*C*/
  ::timeout-> to_upper[n]!MLiCf_ErrPDU,x1,x2,x3,x4;goto end44;              /*C*/
  od;
  /***                                                           CHK         ***/
  /***                                                                       ***/
  /*****************************************************************************/
  /*****************************************************************************/

  /*****************************************************************************/
  /*****************************************************************************/
  /*** ESTADO DEL PROCESO: GROUP                                             ***/
  /***                                                                       ***/
end2: do
  /*Aviso de Expiracion local*/
  ::from_upper[n]?MLoRq_XXXPDU,x1,x2,x3,x4->                                /*C*/
    to_link[n]!MLoIn_XXXPDU,x1,x2,x3,x4->goto ExpLoc;                       /*C*/
  /*Aviso de Expiracion remota*/                                            /*C*/
  ::from_link[n]?MLoIn_XXXPDU,x1,x2,x3,x4-> goto ExpRmt;                    /*C*/

  /*****************************************************************************/
  /***                    Peticion de Asociacion                           ***/
  /*****************************************************************************/
  /**Pet. de Asoc. loc. PDU correcto **/
  ::from_upper[n]?MAsRq_initiate_RequestPDU,x1,x2,x3,x4->
    if
    ::(NCD_CM>0)->
      atomic{
      j=0;
      do
      ::(j<MC && InUse[j]==TRUE)->
        j=j+1;
      ::(j<MC && InUse[j]!=TRUE)->
        i=j;
        break;
      ::(j>=MC)->
        break;
      od;
      };
      NCD_CM=NCD_CM-1;
      InUse[i]=TRUE;
      AssId[i]=s;
      AssIdRmt[i]=0;
      State[i]=CallingInit;
      to_link[n]!MAsIn_initiate_RequestPDU,x1,s,x3,x4;
      s=(s+1)%MA
    ::(NCD_CM<=0)->
      to_upper[n]!MAsCf_initiate_ErrorPDU,x1,x2,x3,x4;
    fi

  /**Pet. de Asoc. loc. Emular PDU incorrecto **/
  ::from_upper[n]?MAsRq_initiate_RequestPDU,x1,x2,x3,x4->
    to_upper[n]!MAsCf_reject_PDU,x1,x2,x3,x4;

  /*****************************************************************************/
  /***                    Indicacion de Asociacion                         ***/
```

```
/*****************************************************************************/
/*****************************************************************************/
/***                    Respuesta de Asociacion                            ***/
/*****************************************************************************/
/*****************************************************************************/
/***                    Confirmacion de Asociacion                         ***/
/*****************************************************************************/
/**Cnf. de Asoc. rmt. PDU correcto Afirmativa**/
::from_link[n]?MAsCf_initiate_ResponsePDU,k,l,x3,x4 ->
  bExiste=FALSE;
  j=0;
  atomic{
    do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k) )->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
    od;
  };
  if
  ::(bExiste==TRUE && State[i]==CallingInit)->
    State[i]=Associated;
    AssIdRmt[i]=l;
    to_upper[n]!MAsCf_initiate_ResponsePDU,k,l,x3,x4;
  ::(bExiste==TRUE && State[i]!=CallingInit)->skip;
  ::(bExiste!=TRUE)->
    to_link[n]!MAbIn_abort_PDU,l,k,x3,x4;
  fi

/**Cnf. de Asoc. rmt. PDU correcto Error**/
::from_link[n]?MAsCf_initiate_ErrorPDU,k,l,x3,x4  ->
  bExiste=FALSE;
  j=0;
  atomic{
    do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k) )->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
    od;
  };
  if
  ::(bExiste==TRUE && State[i]==CallingInit)->
    InUse[i]=FALSE;
    State[i]=IdleConn;
    NCD_CM=NCD_CM+1;
    to_upper[n]!MAsCf_initiate_ErrorPDU,k,l,x3,x4;
  ::(bExiste==TRUE && State[i]!=CallingInit)->skip;
  ::(bExiste!=TRUE)->skip;
  fi

/*****************************************************************************/
/***                    Peticion de Liberacion                             ***/
/*****************************************************************************/
/**Pet. de Libe. loc. PDU correcto **/
::from_upper[n]?MRlRq_conclude_RequestPDU,k,l,x3,x4  ->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k) )->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
```

```
  };
  if
  ::(bExiste==TRUE && State[i]==Associated)->
    State[i]=CallingConc;
    l=AssIdRmt[i];
    to_link[n]!MR1In_conclude_RequestPDU,l,k,x3,x4;
  ::(bExiste==TRUE && State[i]==CallingConc)->skip;
  ::(bExiste==TRUE && State[i]!=Associated && State[i]!=CallingConc)->
    to_upper[n]!MR1Cf_conclude_ErrorPDU,k,l,x3,x4;
  ::(bExiste!=TRUE)->
    to_upper[n]!MR1Cf_conclude_ResponsePDU,k,l,x3,x4;
  fi

/**Pet. de Libe. loc. Emular PDU incorrecto **/
::from_upper[n]?MR1Rq_conclude_RequestPDU,x1,x2,x3,x4  ->
  to_upper[n]!MR1Cf_reject_PDU,x1,x2,x3,x4;

/*****************************************************************************/
/***                    Indicacion de Liberacion                           ***/
/*****************************************************************************/
/*****************************************************************************/
/***                    Respuesta  de Liberacion                           ***/
/*****************************************************************************/
/*****************************************************************************/
/***                    Confirmacion de Liberacion                         ***/
/*****************************************************************************/
/**Cnfs. de Libe. rmt. PDU correcto Afirmativo**/
::from_link[n]?MR1Cf_conclude_ResponsePDU,k,l,x3,x4->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k) )->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE && State[i]==CallingConc)->
    l=AssIdRmt[i];
    InUse[i]=FALSE;
    State[i]=IdleConn;
    NCD_CM=NCD_CM+1;
    to_upper[n]!MR1Cf_conclude_ResponsePDU,k,l,x3,x4;
  ::(bExiste==TRUE && State[i]!=CallingConc)->
    InUse[i]=FALSE;
    State[i]=IdleConn;
    NCD_CM=NCD_CM+1;
  ::(bExiste!=TRUE)->skip;
  fi

/**Cnf. de Libe. rmt. PDU correcto Error**/
::from_link[n]?MR1Cf_conclude_ErrorPDU,k,l,x3,x4 ->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k) )->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE && State[i]==CallingConc)->
    l=AssIdRmt[i];
    State[i]=Associated;
```

```
    to_upper[n]!MRlCf_conclude_ErrorPDU,l,k,x3,x4;
  ::(bExiste==TRUE && State[i]==Associated)->skip
  ::(bExiste==TRUE && State[i]!=CallingConc && State[i]!=Associated)->skip;
  ::(bExiste!=TRUE)->
    to_link[n]!MAbIn_abort_PDU,l,k,x3,x4;
  fi

/*****************************************************************************/
/***                  Peticion de Servicio Confirmado                      ***/
/*****************************************************************************/
/**Pet. de S.Conf. loc. PDU correcto **/
::from_upper[n]?MCfRq_confirmed_RequestPDU,k,l,x3,x4  ->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k || State[j]!=Associated))->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k && State[j]==Associated)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE && NRD[i]>0)->
     RqtInUse[i]=TRUE;
     NRD[i]=NRD[i]-1;
     RqtId[i]=t;
     RqtState[i]=W_Conf;
     l=AssIdRmt[i];
     to_link[n]!MCfIn_confirmed_RequestPDU,l,k,x3,t;
     t=(t+1)%MI;


  ::(bExiste!=TRUE || NRD[i]<=0)->
    to_upper[n]!MCfCf_confirmed_ErrorPDU,k,l,x3,x4;
  fi

/*Pet de Serv. Conf. Emular PDU incorrecto*/
  ::from_upper[n]?MCfRq_confirmed_RequestPDU,x1,x2,x3,x4  ->
    to_upper[n]!MCfCf_reject_PDU,x1,x2,x3,x4;

/*****************************************************************************/
/***                   Confirmacion de Servicio Confirmado                 ***/
/*****************************************************************************/
/**Cnf. de S.Conf. rmt. PDU correcto Afirmativo**/
::from_link[n]?MCfCf_confirmed_ResponsePDU,k,l,k1,l1 ->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k || RqtInUse[i]!=TRUE || RqtId[i]!=k1))->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k && RqtInUse[i]==TRUE && RqtId[i]==k1)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE && RqtState[i]==W_Conf)->
    RqtInUse[i]=FALSE;
    NRD[i]=NRD[i]+1;
    RqtState[i]=Idle;
    to_upper[n]!MCfCf_confirmed_ResponsePDU,k,l,k1,l1;
  ::(bExiste!=TRUE || RqtState[i]!=W_Conf)->skip;
  fi

/**Cnf. de S.Conf. rmt. PDU correcto Error**/
::from_link[n]?MCfCf_confirmed_ErrorPDU,k,l,k1,l1 ->
  bExiste=FALSE;
```

```
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k || RqtInUse[i]!=TRUE || RqtId[i]!=k1))->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k && RqtInUse[i]==TRUE && RqtId[i]==k1) ->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE && RqtState[i]==W_Conf)->
    RqtInUse[i]=FALSE;
    NRD[i]=NRD[i]+1;
    RqtState[i]=Idle;
    to_upper[n]!MCfCf_confirmed_ErrorPDU,k,l,k1,l1;
  ::(bExiste!=TRUE || RqtState[i]!=W_Conf)->skip;
  fi

/*****************************************************************************/
/***                    Peticion de Servicio Sin Confirmacion                ***/
/*****************************************************************************/
/**Pet. de S.UnConf. loc. PDU correcto **/
::from_upper[n]?MUfRq_unconfirmed_PDU,k,l,x3,x4->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k || State[j]!=Associated))->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k && State[j]==Associated)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE)->
     l=AssIdRmt[i];
     to_link[n]!MUfIn_unconfirmed_PDU,x1,x2,x3,x4;
  ::(bExiste!=TRUE)->skip;
  fi

  od;
  /***                                                          GROUP     ***/
  /***                                                                    ***/
  /************************************************************************/
  /************************************************************************/

  /************************************************************************/
  /************************************************************************/
  /*** ESTADO DEL PROCESO: SALONE                                         ***/
  /***                                                                    ***/

end3:  do
  ::from_upper[n]?MLoRq_XXXPDU,x1,x2,x3,x4->                          /*C*/
    to_upper[n]!MLoCf_XXXPDU,x1,x2,x3,x4;goto end44;                  /*C*/
  ::from_upper[n]?MAsRq_initiate_RequestPDU,x1,x2,x3,x4->
    to_upper[n]!MAsCf_initiate_ErrorPDU,x1,x2,x3,x4;


  /*CASO PARA CUANdo termino remotamente*/

  /**Pet. de Libe. loc. PDU correcto **/
  ::from_upper[n]?MRlRq_conclude_RequestPDU,k,l,x3,x4  ->
    to_upper[n]!MRlCf_conclude_ResponsePDU,k,l,x3,x4;
  ::from_upper[n]?MCfRq_confirmed_RequestPDU,k,l,x3,x4  ->
    to_upper[n]!MCfCf_confirmed_ErrorPDU,k,l,x3,x4;
  ::from_upper[n]?MUfRq_unconfirmed_PDU,x1,x2,x3,x4  -> skip;

  od;
```

```
  /***                                                                    SALONE   ***/
  /***                                                                             ***/
  /*********************************************************************************/
  /*********************************************************************************/

  /*********************************************************************************/
  /*********************************************************************************/
  /*** ESTADO DEL PROCESO: LOFF                                                    ***/
  /***                                                                             ***/
ExpLoc: do
  ::from_link[n]?MLoIn_XXXPDU,x1,x2,x3,x4->                                      /*C*/
    to_link[n]!MLoCf_XXXPDU,x1,x2,x3,x4;                                         /*C*/
  ::from_link[n]?MLoCf_XXXPDU,x1,x2,x3,x4->                                      /*C*/
    to_upper[n]!MLoCf_XXXPDU,x1,x2,x3,x4;goto end44;                             /*C*/
  od;
  /***                                                                    LOFF     ***/
  /***                                                                             ***/
  /*********************************************************************************/
  /*********************************************************************************/

  /*********************************************************************************/
  /*********************************************************************************/
  /***              Rutina de Terminacion Remota                                 ***/
  /***                                                                             ***/
ExpRmt:  q=0;
  do   /*limpio mis tablas*/
  ::(q<MC && InUse[q]==TRUE && State[q]==CallingInit)->
    InUse[q]=FALSE;
    k=AssId[q];
    State[q]=IdleConn;
    NCD_CM=NCD_CM+1;
    to_upper[n]!MAsCf_initiate_ErrorPDU,k,x2,x3,x4;
    q=q+1;
  ::(q<MC && InUse[q]==TRUE && State[q]==CallingConc)->
    InUse[q]=FALSE;
    k=AssId[q];
    l=AssIdRmt[q];
    State[q]=IdleConn;
    NCD_CM=NCD_CM+1;
    to_upper[n]!MR1Cf_conclude_ResponsePDU,k,l,x3,x4;
    q=q+1;
  ::(q<MC && InUse[q]==TRUE && State[q]==Associated && RqtInUse[q]==TRUE && RqtState[q]==W_Conf)-
>
    RqtInUse[i]=FALSE;
    RqtState[i]=Idle;
    k=AssId[q];
    l=AssIdRmt[q];
    k1=RqtId[i];
    NRD[i]=NRD[i]+1;
    to_upper[n]!MCfCf_confirmed_ErrorPDU,k,l,k1,x4;
    q=q+1;
  ::(q<MC && InUse[q]==TRUE && State[q]==Associated && IndInUse[i]==TRUE && IndState[i]==W_Resp)-
>
    IndInUse[i]=FALSE;
    IndState[i]=Idle;
    k1=IndId[i];
    l1=RqtRmtId[i];
    k=AssId[q];
    l=AssIdRmt[q];
    NID[i]=NID[i]+1;
  ::(q<MC && InUse[q]==TRUE && State[q]==Associated && (RqtInUse[q]!=TRUE || RqtState[q]!=W_Conf)
&& (IndInUse[i]!=TRUE || IndState[i]!=W_Resp) )->
    q=q+1
  ::(q<MC && InUse[q]!=TRUE)->
    q=q+1
  ::(q>=MC)->
    q=0;
    break;
  od;
  /*Envio Mensaje de confirmacion de terminacion*/
  to_link[n]!MLoCf_XXXPDU,x1,l,x3,x4 -> goto end3;                                  /*C*/

  /***                                                                    ExpRmt   ***/
  /***                                                                             ***/
```

```
  /*****************************************************************************/
  /*****************************************************************************/


  /*****************************************************************************/
  /*****************************************************************************/
  /*** ESTADO DEL PROCESO: END                                                 ***/
  /***                                                                         ***/

end44: skip;                                                                /*C*/
  /***                                                              END      ***/
  /***                                                                       ***/
  /*****************************************************************************/
  /*****************************************************************************/


}
```

10.4 SUPPER.PRO

```
/***                          ***/
/***          Supper          ***/
/***                          ***/

#define MC    4

proctype supper(bit n)
{
  /*****************************************************************************/
  /***                            Variables                                  ***/
  /*****************************************************************************/
  byte dummy;          /*variable dummy*/
  byte x1;             /*otra variable dummy*/
  byte x2;             /*otra variable dummy*/
  byte x3;             /*otra variable dummy*/
  byte x4;             /*otra variable dummy*/
  byte j;
  byte k;              /*assid local                    */
  byte l;              /*assid remota                   */
  byte Associa[MC];
  byte q;              /*apuntador a una asociacion a guardar*/
  byte p;              /*apuntador a una asociacion a concluir*/
  byte p1;             /*apuntador a una asociacion a concluir*/
  byte count;          /*control para dar de baja asociaciones*/
  byte AsocCons;       /*control para dar de baja asociaciones*/
  byte Nevers;         /*valor invalido para s*/

  /*****************************************************************************/
  /***                           Inicializacion                              ***/
  /*****************************************************************************/
  q=0;
  Nevers=255;
  do
  ::(q<MC)->
    Associa[q]=Nevers;
    q=q+1
  ::(q>=MC)->
    break;
  od;
  q=0;
  count=0;
  /***                           Inicializacion                              ***/
  /*****************************************************************************/

  /*****************************************************************************/
  /***                           Login                                       ***/
  /*****************************************************************************/
  from_upper[n]!MLiRq_XXXPDU,dummy,dummy,dummy,dummy->
  do
  ::to_upper[n]?MLiCf_XXXPDU,x1,x2,x3,x4->goto cont;
  ::to_upper[n]?MLiCf_ErrPDU,x1,x2,x3,x4->goto fin;
```

```
  od;
  /***                              Login                                    ***/
  /*****************************************************************************/


  /*****************************************************************************/
  /***                          Ciclo principal                              ***/
  /*****************************************************************************/
cont:  do
  ::(count==0)->skip;
  ::(count==0)->goto termina;
    /*servicios:*/
  ::to_upper[n]?MAsIn_initiate_RequestPDU,k,l,x3,x4->
    from_upper[n]!MAsRs_initiate_ResponsePDU,k,l,dummy,dummy;
  ::to_upper[n]? MAsIn_reject_PDU,k,l,x3,x4;     /*Retransmite la resp. de ini*/
    from_upper[n]!MAsRs_initiate_ResponsePDU,k,l,dummy,dummy;
  ::to_upper[n]?MRlIn_conclude_RequestPDU,k,l,x3,x4->
    from_upper[n]!MRlRs_conclude_ResponsePDU,k,l,dummy,dummy;
  ::to_upper[n]?MRlIn_reject_PDU,k,l,x3,x4;      /*Retransmite la resp. de conc*/
    from_upper[n]!MRlRs_conclude_ResponsePDU,k,l,dummy,dummy;
  ::to_upper[n]?MCfIn_confirmed_RequestPDU,k,l,x3,x4->
    from_upper[n]!MCfRs_confirmed_ResponsePDU,k,l,x3,x4;
  ::to_upper[n]?MCfIn_reject_PDU,x1,x2,x3,x4->
    from_upper[n]!MCfRs_confirmed_ResponsePDU,x1,x2,x3,x4;
  ::to_upper[n]?MUfIn_unconfirmed_PDU,x1,x2,x3,x4->skip;
    od;

  /*****************************************************************************/
  /***                          Terminacion                                  ***/
  /*****************************************************************************/
termina: from_upper[n]!MLoRq_XXXPDU,dummy,dummy,dummy,dummy->
  do
  ::to_upper[n]?MLoCf_XXXPDU,x1,x2,x3,x4-> goto fin;
  ::to_upper[n]?MAsIn_initiate_RequestPDU,x1,x2,x3,x4->skip;
  ::to_upper[n]?MAsIn_reject_PDU,x1,x2,x3,x4->skip;
  ::to_upper[n]?MRlIn_conclude_RequestPDU,x1,x2,x3,x4->skip;
  ::to_upper[n]?MRlIn_reject_PDU,x1,x2,x3,x4->skip;
  ::to_upper[n]?MCfIn_confirmed_RequestPDU,x1,x2,x3,x4->skip;
  ::to_upper[n]?MCfIn_reject_PDU,x1,x2,x3,x4->skip;
  ::to_upper[n]?MUfIn_unconfirmed_PDU,x1,x2,x3,x4->skip;
  od;
  /***                          Terminacion                                  ***/
  /*****************************************************************************/
fin: skip;
}
```

10.5 SMMSPRM.PRO

```
/***                          ***/
/***          SMMSPM          ***/
/***                          ***/
#define MC    4       /*No. maximo de conexiones          */
#define MR    1       /*No.max de pet.simultaneas x conex.*/
#define MA   40       /*No. maximo de asociaciones        */
#define MI   200      /*No. maximo del id de la peticion  */
byte NCD_SM;
proctype smmspm(bit n)
{
  /*****************************************************************************/
  /***                          Variables                                    ***/
  /*****************************************************************************/
  bool bExiste;                          /*indica si assid esta registrado   */
  byte NRD[MC];                          /*# de petciones disp. por conex.   */
  byte NID[MC];                          /*# de indicaciones disp. por conex.*/
  byte q;                                /*apuntador a conexion del timer    */
  byte i;                                /*apuntador a conexion libre        */
  byte k;                                /*assid local                       */
  byte l;                                /*assid remota                      */
  byte k1;                               /*reqid local                       */
  byte l1;                               /*indid remota                      */
```

```
  byte s;                                  /*ass disponible                  */
  byte t;                                  /*id rqt proximo disponible       */
  byte u;                                  /*id ind proximo disponible       */
  byte j;              /*indice                          */
  byte x1;             /*variable dummy                  */
  byte x2;             /*variable dummy                  */
  byte x3;             /*variable dummy                  */
  byte x4;             /*variable dummy                  */
  byte AssId[MC];      /*buffer para guardar #ass local  */
  byte AssIdRmt[MC];   /*buffer para guardar #ass remota */
  byte State[MC];      /*buffer para guardar el edo      */
  bool InUse[MC];      /*buffer para guardar #ass remota */
  byte RqtId[MC];      /*buffer para guardar #pet local  */
  byte RqtRmtId[MC];   /*buffer para guardar #pet remota */
  byte IndId[MC];      /*buffer para guardar #ind rmt    */
  byte RqtState[MC];   /*buffer para guardar #pet local  */
  byte IndState[MC];   /*buffer para guardar #pet local  */
  byte RqtInUse[MC];   /*buffer para guardar #ind rmt    */
  byte IndInUse[MC];   /*buffer para guardar #ind rmt    */

  /*************************************************************************/
  /***                  Inicializacion de Variables                      ***/
  /*************************************************************************/
  j=0;
  do
  ::(j<MC)->atomic{
    InUse[j]=FALSE;
    NRD[j]=MR;
    NID[j]=MR;
    RqtInUse[j]=FALSE;
    IndInUse[j]=FALSE;
    j=j+1;
    }
  ::(j>=MC)->
    break;
  od;
  NCD_SM=MC;
  s=(13+n*11);
    NRD=MR;
    NID=MR;
    RqtInUse=FALSE;
    IndInUse=FALSE;


  /*************************************************************************/
  /*************************************************************************/
  /*** ESTADO DEL PROCESO: BEGIN                                         ***/
  /***                                                                   ***/
  from_upper[n]?MLiRq_XXXPDU,x1,x2,x3,x4->                             /*S*/
  to_upper[n]!MLiCf_XXXPDU,x1,x2,x3,x4;                                /*S*/
  /***                                                    BEGIN        ***/
  /***                                                                 ***/
  /*************************************************************************/
  /*************************************************************************/

  /*************************************************************************/
  /*************************************************************************/
  /*** ESTADO DEL PROCESO: SALONE                                        ***/
  /***                                                                   ***/
end3:  do                                                              /*S*/
  ::from_link[n]?MLiIn_XXXPDU,x1,x2,x3,x4 ->                           /*S*/
    to_link[n]!MLiCf_XXXPDU,x1,x2,x3,x4;goto end2;                     /*S*/
  ::from_upper[n]?MLoRq_XXXPDU,x1,x2,x3,x4->                           /*S*/
    to_upper[n]!MLoCf_XXXPDU,x1,x2,x3,x4;goto end44;                   /*S*/
  od;                                                                  /*S*/
  /***                                                    SALONE       ***/
  /***                                                                 ***/
  /*************************************************************************/
  /*************************************************************************/

  /*************************************************************************/
  /*************************************************************************/
  /*** ESTADO DEL PROCESO: GROUP                                         ***/
  /***                                                                   ***/
```

```
end2: do                                                                    /*S*/
  /*Aviso de Expiracion local*/                                             /*S*/
  ::from_upper[n]?MLoRq_XXXPDU,x1,x2,x3,x4->                                /*S*/
    /*Aqui va la rutina de descarga de informacion*/                        /*S*/
    to_link[n]!MLoIn_XXXPDU,x1,x2,x3,x4->goto ExpLoc;                       /*S*/
  /*Aviso de Expiracion remota*/                                            /*S*/
  ::from_link[n]?MLoIn_XXXPDU,x1,x2,x3,x4->goto ExpRmt;                     /*S*/
    /*Aqui va la rutina de descarga de informacion*/                        /*S*/


  /*************************************************************************/
  /***                    Peticion de Asociacion                         ***/
  /*************************************************************************/
  /*************************************************************************/
  /***                    Indicacion de Asociacion                       ***/
  /*************************************************************************/
  /**Ind. de Asoc. rmt. PDU correcto **/
  ::from_link[n]?MAsIn_initiate_RequestPDU,x1,1,x3,x4 ->
    atomic{
      bExiste=FALSE;
      j=0;
      do
      ::(j<MC && InUse[j]==TRUE && AssIdRmt[j]==1)->
         bExiste=TRUE;
         i=j;
         break;
      ::(j<MC && (InUse[j]!=TRUE  || AssIdRmt[j]!=1))->
         j=j+1;
      ::(j>=MC)->
         break;
      od;
    };
    if
    ::(bExiste==FALSE && NCD_SM>0)->
      atomic{
        j=0;
        do
        ::(j<MC && InUse[j]==TRUE)->
          j=j+1;
        ::(j<MC && InUse[j]!=TRUE)->
          i=j;
          break;
        ::(j>=MC)->
          break;
        od;
      };
      NCD_SM=NCD_SM-1;
      InUse[i]=TRUE;
      AssId[i]=s;
      AssIdRmt[i]=1;
      State[i]=CalledInit;
      to_upper[n]!MAsIn_initiate_RequestPDU,s,1,x3,x4;
      s=(s+1)%MA
    ::(bExiste==FALSE && NCD_SM<=0)->
      to_link[n]!MAsCf_initiate_ErrorPDU,1,x2,x3,x4;
    ::(bExiste!=FALSE && State[i]==Associated)->
      k==AssId[i];
      to_link[n]!MAsCf_initiate_ResponsePDU,1,k,x3,x4;
    ::(bExiste!=FALSE && State[i]!=Associated)-> skip;
    fi
  /*************************************************************************/
  /***                     Respuesta de Asociacion                       ***/
  /*************************************************************************/
  /**Res. de Asoc. rmt. PDU correcto Afirmativa**/
  ::from_upper[n]?MAsRs_initiate_ResponsePDU,k,1,x3,x4 ->
    bExiste=FALSE;
    j=0;
    atomic{
    do
      ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k))->
        j=j+1;
      ::(j<MC && InUse[j]==TRUE && AssId[j]==k)->
        bExiste=TRUE;
        i=j;
```

```
        break;
      ::(j>=MC)->break;
    od;
    };
    if
    ::(bExiste==TRUE && State[i]==CalledInit)->
      l=AssIdRmt[i];
      State[i]=Associated;
      to_link[n]!MAsCf_initiate_ResponsePDU,l,k,x3,x4;
    ::(bExiste!=TRUE || State[i]!=CalledInit)->skip;
    fi

  /**Res. de Asoc. rmt. PDU correcto Error**/
  ::from_upper[n]?MAsRs_initiate_ErrorPDU,k,l,x3,x4 ->
    bExiste=FALSE;
    j=0;
    atomic{
      do
      ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k))->
        j=j+1;
      ::(j<MC && InUse[j]==TRUE && AssId[j]==k)->
        bExiste=TRUE;
        i=j;
        break;
      ::(j>=MC)->break;
      od;
    };
    if
    ::(bExiste==TRUE && State[i]==CalledInit)->
      l=AssIdRmt[i];
      InUse[i]=FALSE;
      State[i]=IdleConn;
      NCD_SM=NCD_SM+1;
      to_link[n]!MAsCf_initiate_ErrorPDU,l,k,x3,x4;
    ::(bExiste!=TRUE || State[i]!=CalledInit)->skip;
    fi

  /**Res. de Asoc. rmt. Emular PDU incorrecto **/
  ::from_upper[n]?MAsRs_initiate_ResponsePDU,x1,x2,x3,x4 ->
    to_upper[n]!MAsIn_reject_PDU,x1,x2,x3,x4;

/*****************************************************************************/
/***                    Confirmacion de Asociacion                         ***/
/*****************************************************************************/
/*****************************************************************************/
/***                    Peticion de Liberacion                             ***/
/*****************************************************************************/
/*****************************************************************************/
/***                    Indicacion de Liberacion                           ***/
/*****************************************************************************/
/**Ind. de Libe. rmt. PDU correcto **/
::from_link[n]?MRlIn_conclude_RequestPDU,k,l,x3,x4 ->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k) )->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE && State[i]==Associated)->
    State[i]=CalledConc;
    l=AssIdRmt[i];
    to_upper[n]!MRlIn_conclude_RequestPDU,k,l,x3,x4;
  ::(bExiste==TRUE && State[i]==CalledConc)->skip;
  ::(bExiste==TRUE && State[i]!=Associated && State[i]!=CalledConc)->skip;
  ::(bExiste!=TRUE )->
    to_link[n]!MRlCf_conclude_ResponsePDU,l,k,x3,x4;
```

```
  fi

/*****************************************************************************/
/***                      Respuesta  de Liberacion                         ***/
/*****************************************************************************/
/**Res. de Libe. loc. PDU correcto afirmativo**/
::from_upper[n]?MRlRs_conclude_ResponsePDU,k,l,x3,x4 ->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k) )->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE && State[i]==CalledConc)->
    l=AssIdRmt[i];
    InUse[i]=FALSE;
    State[i]=IdleConn;
    NCD_SM=NCD_SM+1;
    to_link[n]!MRlCf_conclude_ResponsePDU,l,k,x3,x4;
  ::(bExiste!=TRUE || State[i]!=CalledConc)-> skip;
  fi

/**Res. de Libe. loc. PDU correcto Error**/
::from_upper[n]?MRlRs_conclude_ErrorPDU,k,l,x3,x4 ->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k) )->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE && State[i]==CalledConc)->
    l=AssIdRmt[i];
    State[i]=Associated;
    to_link[n]!MRlCf_conclude_ErrorPDU,l,k,x3,x4;
  ::(bExiste!=TRUE || State[i]!=CalledConc)-> skip;
  fi

/**Res. de Libe. loc. emulacion PDU incorrecto **/
::from_upper[n]?MRlRs_conclude_ResponsePDU,x1,x2,x3,x4 ->
  to_upper[n]!MRlIn_reject_PDU,x1,x2,x3,x4;

/*****************************************************************************/
/***                       Confirmacion de Liberacion                      ***/
/*****************************************************************************/
/*****************************************************************************/
/***                     Peticion de Servicio Confirmado                   ***/
/*****************************************************************************/
/*****************************************************************************/
/***                     Indicacion de Servicio Confirmado                 ***/
/*****************************************************************************/
/**Ind. de S.Conf. rmt. PDU correcto **/
::from_link[n]?MCfIn_confirmed_RequestPDU,k,l,x3,l1 ->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k || State[j]!=Associated))->
      j=j+1;
```

```
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k && State[j]==Associated)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE && NID[i]>0)->
     IndInUse[i]=TRUE;
     NID[i]=NID[i]-1;
     RqtRmtId[i]=l1;
     IndId[i]=u;
     IndState[i]=W_Resp;
     l=AssIdRmt[i];
     to_upper[n]!MCfIn_confirmed_RequestPDU,k,l,u,l1;
     u=(u+1)%MI;
  ::(bExiste==TRUE && NRD[i]<=0)->/*SUPONGO QUE ES MJE REPETIDO*/
     skip;
  ::(bExiste!=TRUE)->
    to_link[n]!MCfCf_confirmed_ErrorPDU,l,k,x3,x4;
  fi

/**Ind. de S.Conf. rmt. PDU incorrecto **/
::from_link[n]?MCfIn_confirmed_XXXPDU,x1,l,x3,x4 ->skip; caso(1)

/*****************************************************************************/
/***                    Respuesta de Servicio Confirmado                   ***/
/*****************************************************************************/
/**Res. de S.Conf. loct. PDU correcto Afirmativa**/
::from_upper[n]?MCfRs_confirmed_ResponsePDU,k,l,k1,l1 ->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k || IndInUse[i]!=TRUE || IndId[i]!=k1))->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k && IndInUse[i]==TRUE && IndId[i]==k1)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE && IndState[i]==W_Resp)->
    IndState[i]=Idle;
    l1=RqtRmtId[i];
    l=AssIdRmt[i];
    IndInUse[i]=FALSE;
    NID[i]=NID[i]+1;
    to_link[n]!MCfCf_confirmed_ResponsePDU,l,k,l1,k1;
  ::(bExiste!=TRUE || IndState[i]!=W_Resp)->skip;
  fi

/**Res. de S.Conf. loct. PDU correcto Error**/
::from_upper[n]?MCfRs_confirmed_ErrorPDU,k,l,k1,l1 ->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k || IndInUse[i]!=TRUE || IndId[i]!=k1))->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k && IndInUse[i]==TRUE && IndId[i]==k1)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE && IndState[i]==W_Resp)->
    IndState[i]=Idle;
    l1=RqtRmtId[i];
```

```
    l=AssIdRmt[i];
    IndInUse[i]=FALSE;
    NID[i]=NID[i]+1;
    to_link[n]!MCfCf_confirmed_ErrorPDU,l,k,l1,k1;
  ::(bExiste!=TRUE || IndState[i]!=W_Resp)->skip;
  fi

/**Res. de S.Conf. loc. Emular PDU incorrecto **/
::from_upper[n]?MCfRs_confirmed_ResponsePDU,x1,x2,x3,x4 ->
  to_upper[n]!MCfIn_reject_PDU,x1,x2,x3,x4;

/*****************************************************************************/
/***                    Confirmacion de Servicio Confirmado                ***/
/*****************************************************************************/
/*****************************************************************************/
/***                    Indicacion de Servicio Sin Confirmacion            ***/
/*****************************************************************************/
/**Ind. de S.UnConf. rmt. PDU correcto **/
::from_link[n]?MUfIn_unconfirmed_PDU,k,l,x3,x4 ->
  bExiste=FALSE;
  j=0;
  atomic{
  do
    ::(j<MC && (InUse[j]!=TRUE || AssId[j]!=k || State[j]!=Associated))->
      j=j+1;
    ::(j<MC && InUse[j]==TRUE && AssId[j]==k && State[j]==Associated)->
      bExiste=TRUE;
      i=j;
      break;
    ::(j>=MC)->break;
  od;
  };
  if
  ::(bExiste==TRUE)->
     l=AssIdRmt[i];
     to_upper[n]!MUfIn_unconfirmed_PDU,k,l,x3,x4
  ::(bExiste!=TRUE)->skip;
  fi

  od;
  /***                                                        GROUP     ***/
  /***                                                                  ***/
  /*****************************************************************************/
  /*****************************************************************************/

  /*****************************************************************************/
  /*****************************************************************************/
  /*** ESTADO DEL PROCESO: LOFF                                         ***/
  /***                                                                  ***/
ExpLoc: do                                                            /*S*/
  ::from_link[n]?MLoIn_XXXPDU,x1,x2,x3,x4->                           /*S*/
    to_link[n]!MLoCf_XXXPDU,x1,x2,x3,x4;                              /*S*/
  ::from_link[n]?MLoCf_XXXPDU,x1,x2,x3,x4->                           /*S*/
    to_upper[n]!MLoCf_XXXPDU,x1,x2,x3,x4;goto end44;                  /*S*/
  ::from_link[n]?MAsIn_initiate_RequestPDU,x1,l,x3,x4 ->
    skip;
  ::from_link[n]?MRlIn_conclude_RequestPDU,l,k,x3,x4      ->
    skip;
  ::from_link[n]?MCfIn_confirmed_RequestPDU,x1,x2,x3,x4 ->
    skip;
  ::from_link[n]?MUfIn_unconfirmed_PDU,x1,x2,x3,x4 ->
    skip;
  od;                                                                 /*S*/
  /***                                                          LOFF    ***/
  /***                                                                  ***/
  /*****************************************************************************/
  /*****************************************************************************/

  /*****************************************************************************/
  /*****************************************************************************/
  /***              Rutina de Terminacion Remota                       ***/
  /***                                                                 ***/
ExpRmt:  q=0;
  do   /*limpio mis tablas*/
```

```
  ::(q<MC && InUse[q]==TRUE && State[q]==CalledInit)->
    InUse[q]=FALSE;
    k=AssId[q];
    l=AssIdRmt[q];
    State[q]=IdleConn;
    NCD_SM=NCD_SM+1;
    q=q+1;
  ::(q<MC && InUse[q]==TRUE && State[q]==CalledConc)->
    InUse[q]=FALSE;
    k=AssId[q];
    l=AssIdRmt[q];
    State[q]=IdleConn;
    NCD_SM=NCD_SM+1;
    q=q+1;
  ::(q<MC && InUse[q]==TRUE && State[q]==Associated && RqtInUse[q]==TRUE && RqtState[q]==W_Conf)-
>
    RqtInUse[i]=FALSE;
    RqtState[i]=Idle;
    k=AssId[q];
    l=AssIdRmt[q];
    k1=RqtId[i];
    NRD[i]=NRD[i]+1;
    to_upper[n]!MCfCf_confirmed_ErrorPDU,k,l,k1,x4;
    q=q+1;
  ::(q<MC && InUse[q]==TRUE && State[q]==Associated && IndInUse[i]==TRUE && IndState[i]==W_Resp)-
>
    IndInUse[i]=FALSE;
    IndState[i]=Idle;
    k1=IndId[i];
    l1=RqtRmtId[i];
    k=AssId[q];
    l=AssIdRmt[q];
    NID[i]=NID[i]+1;
  ::(q<MC && InUse[q]==TRUE && State[q]==Associated && (RqtInUse[q]!=TRUE || RqtState[q]!=W_Conf
|| IndInUse[i]!=TRUE && IndState[i]!=W_Resp))->
    q=q+1
  ::(q<MC && InUse[q]!=TRUE)->
    q=q+1
  ::(q>=MC)->
    q=0;
    break;
  od;
  /*Envio Mensaje de confirmacion de terminacion*/
  to_link[n]!MLoCf_XXXPDU,x1,l,x3,x4 -> goto end3;                              /*C*/

  /***                                                          ExpRmt    ***/
  /***                                                                    ***/
  /*************************************************************************/
  /*************************************************************************/

  /*************************************************************************/
  /*************************************************************************/
  /*** ESTADO DEL PROCESO: END                                            ***/
  /***                                                                    ***/
end44: skip;
  /***                                                          END       ***/
  /***                                                                    ***/
  /*************************************************************************/
  /*************************************************************************/
}
```

11 APÉNDICE C

FUENTES DE LA IMPLEMENTACIÓN DEL PROTOCOLO MMS

11.1 ROOT.C

```
#include <channel.h>
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <iocntrl.h>
#include <process.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "mmsprimi.h"    /*      Primitivas MMS                          */
#include "mmsctes.h"     /*      Constante MMS                           */
#include "mmstypes.h"    /*      Tipos MMS                               */
#include "mmspdu.h"      /*      Definicion PDU MMS                      */
#include "mmstype2.h"    /*      Tipos MMS                               */
#include "mmspdutr.h"    /*      Fns de Conversion de MMSPDU a un string */
#include "TrxBuff.h"     /*      Procesos p emular canal de long. n      */
#include "SMMSPM.h"      /*      Proceso MAq Edos MMS del servidor       */
#include "feeder.h"      /*      Proceso que recibe los datos desde T0   */
/*****************************************************************************/
/**********                          KILLER                          **********/
void Killer_proc(Process * p,Channel *Killer){
Boolean bStop=TRUE;
int Stop;
char schar[1];
  p=p;
  while(bStop){
    Stop=pollkey();
    if (Stop==38){
      ChanOut(Killer,&schar,1);
      bStop=FALSE;
    }
  }
  return;
}
/**********                          KILLER                          **********/
/*****************************************************************************/

/*****************************************************************************/
/**********                          MONITOR                         **********/
void Monitor_proc(Process * p,Channel *FrClient,Channel *FrCmmspm,
Channel *FrSmmspm,Channel *FrServer){
Boolean bMonitoring=TRUE;
char sTmp[11];
char Node[11];
char sTx[3];
char sVal[2];
tTrama sCmd;
tTrama sPrntCmd,sPrntCmd2;
int ch;
  while(bMonitoring){
    ch=ProcAlt(FrClient,FrCmmspm,FrSmmspm,FrServer,NULL);
    switch(ch){
      case 0:
        strcpy(sTmp,"Client");
        ChanIn(FrClient,sCmd,TRAMA_LEN);
        break;
      case 1:
        strcpy(sTmp,"Cmmspm");
        ChanIn(FrCmmspm,sCmd,TRAMA_LEN);
        break;
```

```
      case 2:
        strcpy(sTmp,"Smmspm");
        ChanIn(FrSmmspm,sCmd,TRAMA_LEN);
        break;
      case 3:
        strcpy(sTmp,"Server");
        ChanIn(FrServer,sCmd,TRAMA_LEN);
        if (strncmp(sFIN,sCmd,3)==0){
          bMonitoring=FALSE;
        }
        break;
    }
    strcat(sTmp,"[");
    GetTx(sCmd,sTx,sVal);
    strcat(sTmp,sTx);
    strcat(sTmp,"]");
    strcpy(Node,sVal);
    strcat(Node,sTmp);
    sPrntCmd[0]='\0';
    sPrntCmd2[0]='\0';
    if (bMonitoring){
    MMSDecod2(sCmd,sPrntCmd,sPrntCmd2);
/*      printf("\n%s:%s\n\t%s",Node,sPrntCmd,sPrntCmd2);*/
      printf("\n%s:%s",Node,sPrntCmd);
    }
    else{
      bMonitoring=TRUE;
      while(bMonitoring){
          ch=ProcSkipAlt(FrClient,FrCmmspm,FrSmmspm,FrServer,NULL);
          switch(ch){
           case -1:
             bMonitoring=FALSE;
             break;
           case 0:
                   strcpy(sTmp,"ClienX");
                ChanIn(FrClient,sCmd,TRAMA_LEN);
             break;
           case 1:
             strcpy(sTmp,"CmmspX");
             ChanIn(FrCmmspm,sCmd,TRAMA_LEN);
             break;
           case 2:
             strcpy(sTmp,"SmmspX");
             ChanIn(FrSmmspm,sCmd,TRAMA_LEN);
             break;
           case 3:
             strcpy(sTmp,"ServeX");
             ChanIn(FrServer,sCmd,TRAMA_LEN);
             break;
          }
          sPrntCmd[0]='\0';
          sPrntCmd2[0]='\0';
          if (bMonitoring){
            strcat(sTmp,"[");
            GetTx(sCmd,sTx,sVal);
            strcat(sTmp,sTx);
            strcat(sTmp,"]");
            strcpy(Node,sVal);
            strcat(Node,sTmp);
            MMSDecod2(sCmd,sPrntCmd,sPrntCmd2);
            printf("\n%s:%s",Node,sPrntCmd);
          }
          else{
            printf("\n%s:%s",sTmp,sFIN);
          }
      }
    }
  }
  return;
}
/**********                          MONITOR                                **********/
/*****************************************************************************/

/*****************************************************************************/
```

```
/**********                        Proceso Server                        **********/
/**********                                                              **********/
  /*****************************************************************************/
  /*** ESTADO DEL PROCESO: END                                                ***/
void fin(Channel *ToClnt,Channel *ServerToMon){
  ChanOut(ToClnt,sFIN,TRAMA_LEN);
  ChanOut(ServerToMon,sFIN,TRAMA_LEN);
  return;
}
  /***                                                          END      ***/
  /*****************************************************************************/

void server_proc(Process * p,Channel *c, Channel *d,Channel *FromKiler,
Channel *ServerToMon)
{
tTrama sCmd;
int bKeepGoing = TRUE;
int iPid=1546;
aleaf SAleaf;
Boolean bStop;
int ch;
char sChar[1];

  /*****************************************************************************/
  /***                          Inicializacion                               ***/
  p = p;
  /***                                                          INICIA ***/
  /*****************************************************************************/
  /*****************************************************************************/
  /***                          Login                                        ***/
  SAleaf.Primitive=LOGIN_Request;
  SAleaf.Id.AssId=0;
  SAleaf.Id.AssIdRmt=0;
  SAleaf.Id.PId=iPid;
  SAleaf.Id.PIdRmt=0;
  PackTrama(&SAleaf,sCmd);
  ChanOut(d,sCmd,TRAMA_LEN);
  bStop=TRUE;
  while(bStop){
    ChanIn(c,sCmd,TRAMA_LEN);
    SendMonitor(ServerToMon,sCmd,FROMSMMSPM,VALIDO);
    UnpackTrama(&SAleaf,sCmd);
    switch(SAleaf.Primitive){
      case LOGIN_Confirm:
        bStop=FALSE;
        break;                  /*goto cont;*/
      case LOGIN_ConfirmErr:
        fin(d,ServerToMon);               /*goto fin; */
        return;
        break;
    }
  }
  /***                          Login                                        ***/
  /*****************************************************************************/
  /*****************************************************************************/
  /***   cont                   Ciclo principal                              ***/
  while (bKeepGoing){
    ch=ProcAlt(c,FromKiler,NULL);
    if (ch==0){
      ChanIn(c,sCmd,TRAMA_LEN);
               SendMonitor(ServerToMon,sCmd,FROMSMMSPM,VALIDO);
      UnpackTrama(&SAleaf,sCmd);
/*        printf("\nRECIBI MSJ  :%d-%d",SAleaf.Primitive,SAleaf.Id.PId);*/
      switch(SAleaf.Primitive){
        case A_ASSOCIATE_indication:
          SAleaf.Primitive=A_ASSOCIATE_response;
          SAleaf.MmsPdu.designator=MMSpdu_initiate_ResponsePDU;
          PackTrama(&SAleaf,sCmd);
          ChanOut(d,sCmd,TRAMA_LEN);
          break;
        case A_RELEASE_indication:
          SAleaf.Primitive=A_RELEASE_response;          /*Retransmite la resp. de conc*/
          SAleaf.MmsPdu.designator=MMSpdu_conclude_ResponsePDU;
          PackTrama(&SAleaf,sCmd);
```

```
            ChanOut(d,sCmd,TRAMA_LEN);
            break;
          case A_MMS_Indication:
            if(SAleaf.MmsPdu.designator==MMSpdu_confirmed_RequestPDU){
              SAleaf.Primitive=A_MMS_Response;          /*Retransmite la resp. del serv conf*/
              SAleaf.MmsPdu.designator=MMSpdu_confirmed_ResponsePDU;
              PackTrama(&SAleaf,sCmd);
              ChanOut(d,sCmd,TRAMA_LEN);
            }
            if(SAleaf.MmsPdu.designator==MMSpdu_Unconfirmed_RequestPDU){
              /*atencion al servicio no confirmado*/
            }
            break;
          default:
            break;
        }
      }
      else{
        ChanIn(FromKiler,&sChar,1);
        bKeepGoing=FALSE;/*goto termina;*/
      }
    }
    /*****************************************************************************/
    /*** (termina)                    Terminacion                              ***/
    SAleaf.Primitive=LOGOUT_Request;
    SAleaf.Id.AssId=0;
    SAleaf.Id.AssIdRmt=0;
    PackTrama(&SAleaf,sCmd);
    ChanOut(d,sCmd,TRAMA_LEN);
    bKeepGoing=TRUE;
    while (bKeepGoing){
      ChanIn(c,sCmd,TRAMA_LEN);
      SendMonitor(ServerToMon,sCmd,FROMSMMSPM,INVALIDO);
      UnpackTrama(&SAleaf,sCmd);
      switch(SAleaf.Primitive){
        case LOGOUT_Confirm: /*goto fin;*/
          fin(d,ServerToMon);
          return;
          break;
        default:            /*skip*/
          break;
      }
    }
    /***                          Terminacion                                  ***/
    /*****************************************************************************/
}

/*****************************************************************************/
/**********                        Entry Point                        **********/
/**********                                                           **********/
int main (int argc, char *argv[], char *envp[],
          Channel *in[],  int inlen,
          Channel *out[], int outlen)
{

Process  *RxSmmspmFrClnt,*TxSmmspmFrClnt,
         *Smmspm,*Server,
         *RxServer,*TxServer,
         *RxSmmspmFrServer,*TxSmmspmFrServer,
         *Feeder,*Killer,
         *Monitor;
Channel  *Rx1SmmspmToSmmspm,
         *SmmspmToRxServer,*RxServerToServer,
         *ServerToRx2Smmspm,*Rx2SmmspmToSmmspm,
         *FrKiller,
         *SmmspmToMon,*ServerToMon;
int      i;
int      bKeepGoingSmms=TRUE,bKeepGoing1=TRUE,bKeepGoing2=TRUE,bKeepGoing3=TRUE;

  assert(inlen  > 2);
  assert(outlen > 2);
  for (i=0;i<(LEN_CHAN*3);i++){
    Buffer[i].Usado=FALSE;
  }
```

```
  Rx1SmmspmToSmmspm= ChanAlloc();
  SmmspmToRxServer = ChanAlloc();
  RxServerToServer = ChanAlloc();
  ServerToRx2Smmspm= ChanAlloc();
  Rx2SmmspmToSmmspm= ChanAlloc();
  FrKiller         = ChanAlloc();
  SmmspmToMon      = ChanAlloc();
  ServerToMon      = ChanAlloc();
  RxSmmspmFrClnt   = ProcAlloc(RxProc,     0, 4, in[2],1,&bKeepGoing1,0);
  TxSmmspmFrClnt   = ProcAlloc(TxProc,     0, 4, Rx1SmmspmToSmmspm,1,&bKeepGoing1,0);
  Smmspm           = ProcAlloc(SmmspMProc, 0, 6,
Rx1SmmspmToSmmspm,out[2],Rx2SmmspmToSmmspm,SmmspmToRxServer,SmmspmToMon,&bKeepGoingSmms);
  RxServer         = ProcAlloc(RxProc,     0, 4, SmmspmToRxServer,3,&bKeepGoing3,0);
  TxServer         = ProcAlloc(TxProc,     0, 4, RxServerToServer,3,&bKeepGoing3,0);
  Server           = ProcAlloc(server_proc,0, 4,
RxServerToServer,ServerToRx2Smmspm,FrKiller,ServerToMon);
  RxSmmspmFrServer = ProcAlloc(RxProc,     0, 4, ServerToRx2Smmspm,2,&bKeepGoing2,0);
  TxSmmspmFrServer = ProcAlloc(TxProc,     0, 4, Rx2SmmspmToSmmspm,2,&bKeepGoing2,0);
  Feeder           = ProcAlloc(feeder_proc,0, 1, out[3]);
  Killer           = ProcAlloc(Killer_proc,0, 1, FrKiller);
  Monitor          = ProcAlloc(Monitor_proc,0, 4,in[3],in[4],SmmspmToMon,ServerToMon);
if ((RxSmmspmFrClnt == NULL)    || (TxSmmspmFrClnt == NULL)    ||
    (Smmspm == NULL)            || (Server == NULL)            ||
    (RxServer == NULL)          || (TxServer == NULL)          ||
    (RxSmmspmFrServer == NULL) || (TxSmmspmFrServer == NULL) ||
    (Feeder == NULL)            || (Killer == NULL)            ||
    (Monitor== NULL)                                          ){
    printf ("Could not allocate processes.\n");
    exit (EXIT_FAILURE);
  }
  ProcRun (Feeder);
  ProcJoin(Feeder,NULL);
  ProcAllocClean (Feeder);
  ProcPar
(Smmspm,Server,RxSmmspmFrClnt,TxSmmspmFrClnt,RxServer,TxServer,RxSmmspmFrServer,TxSmmspmFrServer,
Killer,Monitor,NULL);
  ProcAllocClean (Server);
  ProcAllocClean (Smmspm);
  ProcAllocClean (RxServer);
  ProcAllocClean (TxServer);
  ProcAllocClean (RxSmmspmFrServer);
  ProcAllocClean (TxSmmspmFrServer);
  ProcAllocClean (RxSmmspmFrClnt);
  ProcAllocClean (TxSmmspmFrClnt);
  ProcAllocClean (Killer);
  ProcAllocClean (Monitor);
  printf("\n Fin");
}
```

11.2 CLIENT.C

```
#include <channel.h>
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <process.h>
#include <string.h>
#include <math.h>
#include "mmsprimi.h"    /*      Primitivas MMS                              */
#include "mmsctes.h"     /*      Constante MMS                               */
#include "mmstypes.h"    /*      Tipos MMS                                   */
#include "mmspdu.h"      /*      Definicion PDU MMS                          */
#include "mmstype2.h"    /*      Tipos MMS                                   */
#include "mmspdutr.h"    /*      Fns de Conversion de MMSPDU a un string     */
#include "TrxBuff.h"     /*      Procesos p emular canal de long. n          */
#include "cmmspm.h"      /*      Procesos de la maquina de protocolo MMS para el cliente*/

int ListenFeeder(Channel *FrFeeder,tStackTrama *sCmdBuff){
/*** Fn. ListenFeeder Recibe los comandos que le envia el proc Feeder       **/
```

```
int iComando=0;
int bKeepGoing = TRUE;
tTrama sCmd;
  while (bKeepGoing){
    ChanIn(FrFeeder,sCmd,TRAMA_LEN);
    if (strncmp(sFIN,sCmd,3)==0)
      break;
    Mstrcpy((*sCmdBuff)[iComando],sCmd);
    iComando++;
  }
  return(iComando);
}

int GetLoginPrim(tStackTrama *sCmdBuff,int icount){
/*** Fn. GetLoginPrim Busca el indice donde se está la primitiva Login      **/
int i=0;
tTrama sCmd;
aleaf AleafTmp;
  while(i<icount){
    Mstrcpy(sCmd,(*sCmdBuff)[i]);
    UnpackTrama(&AleafTmp,sCmd);
    if(AleafTmp.Primitive==LOGIN_Request)
      break;
    i++;
  }
  if(i>=icount)
    i=-1;
  return(i);
}


/*****************************************************************************/
/**********                          Proceso Cliente                  **********/
/**********                                                           **********/
void fin(Channel *ToMmspm){
  ChanOut(ToMmspm,sFIN,TRAMA_LEN);
  return;
}

  /*****************************************************************************/
  /***                          Terminacion                                  ***/
void termina(Channel *FrMmspm, Channel *ToMmspm,tTrama *sCmd){
/*** Fn. SendLogout Envia la Prim. Logout y espera respuesta                  **/
Boolean bStop=TRUE;
aleaf AleafTmp;
  ChanOut(ToMmspm,(*sCmd),TRAMA_LEN);
  while(bStop){
    ChanIn(FrMmspm,sCmd,TRAMA_LEN);
    UnpackTrama(&AleafTmp,(*sCmd));
    if(AleafTmp.Primitive==LOGOUT_Confirm)
        break;
  }
  fin(ToMmspm);                              /*goto fin*/
  return;
}
  /***                          Terminacion                                  ***/
  /*****************************************************************************/


void ClientProc(Process * p,Channel *FrMmspm, Channel *ToMmspm,Channel *FrFeeder,
Channel *ClientToMon)
{
tTrama sCmd;
tStackTrama sCmdBuff;
int i,j,iComando=0;
aleaf   CAleaf;
Boolean bStop;
Primitive_Type Primitiva;

  /*****************************************************************************/
  /***                          Inicializacion                               ***/
  p = p;
  iComando=ListenFeeder (FrFeeder,&sCmdBuff);
  /***                          Inicializacion                               ***/
```

```
   /*****************************************************************************/
   /*****************************************************************************/
   /***                          Login                                        ***/
   j = GetLoginPrim (&sCmdBuff,iComando);
   Mstrcpy(sCmd,sCmdBuff[j]);
   ChanOut(ClientToMon,sCmd,TRAMA_LEN);
   ChanOut(ToMmspm,sCmd,TRAMA_LEN);
   bStop=TRUE;
   while(bStop){
     ChanIn(FrMmspm,sCmd,TRAMA_LEN);
     UnpackTrama(&CAleaf,sCmd);
     switch(CAleaf.Primitive){
       case LOGIN_Confirm:
         bStop=FALSE;
         break;                                  /*goto cont*/
       case LOGIN_ConfirmErr:
         fin(ToMmspm);
         return;                                 /*goto fin*/
         break;
     }
   }
   /***                          Login                                        ***/
   /*****************************************************************************/
   /*****************************************************************************/
   /***  (cont)                   Ciclo principal                              ***/
   for (i=j+1;i<iComando;i++){
     Mstrcpy(sCmd,sCmdBuff[i]);
     ChanOut(ClientToMon,sCmd,TRAMA_LEN);
     UnpackTrama(&CAleaf,sCmd);
     Primitiva=CAleaf.Primitive;
     switch(Primitiva){
       case LOGOUT_Request:
         termina(FrMmspm, ToMmspm,&sCmd);        /*goto termina*/
         return;
         break;
       case A_ASSOCIATE_request:                 /*requiero asociacion*/
         ChanOut(ToMmspm,sCmd,TRAMA_LEN);
         bStop=TRUE;
         while(bStop){
           ChanIn(FrMmspm,sCmd,TRAMA_LEN);
           UnpackTrama(&CAleaf,sCmd);
           if(CAleaf.Primitive==A_ASSOCIATE_confirm){/*goto cont*/
               /*AQUI LA APLICACION REAL DEBE EVALUAR EL PDU: ok, ERR, REJECT*/
               bStop=FALSE;                      /*AP:no interesa si se pudo o no establecer la
asociacion*/
           }
         }
         break;
       case A_RELEASE_request:                   /*dar de baja asociaciones*/
         ChanOut(ToMmspm,sCmd,TRAMA_LEN);
         bStop=TRUE;
         while(bStop){
           ChanIn(FrMmspm,sCmd,TRAMA_LEN);
           UnpackTrama(&CAleaf,sCmd);
           if(CAleaf.Primitive==A_RELEASE_confirm){
               /*AQUI LA APLICACION REAL DEBE EVALUAR EL PDU: ck, ERR, REJECT*/
               bStop=FALSE;
           }
         }
         break;
       case A_MMS_Request:
         if (CAleaf.MmsPdu.designator==MMSpdu_confirmed_RequestPDU){/*enviar servicions
confirmados*/
           ChanOut(ToMmspm,sCmd,TRAMA_LEN);
           bStop=TRUE;
           while(bStop){
             ChanIn(FrMmspm,sCmd,TRAMA_LEN);
             UnpackTrama(&CAleaf,sCmd);
             if(CAleaf.Primitive==A_MMS_Confirm){
               /*AQUI LA APLICACION REAL DEBE EVALUAR EL PDU: ok, ERR, REJECT*/
               bStop=FALSE;
             }
           }
         }
```

```
        else{                                   /*enviar servicios NO confirmados*/
          ChanOut(ToMmspm,sCmd,TRAMA_LEN);
        }
        break;
      default:
        ChanOut(ToMmspm,sCmd,TRAMA_LEN);
        break;
    }
  }
}

/*****************************************************************************/
/**********                        Entry Point                      **********/
/**********                                                         **********/
int main (int argc, char *argv[], char *envp[],
          Channel *in[],  int inlen,
          Channel *out[], int outlen)
{
Process *Client,*CmmspM,
        *RxCmmspMFrServer,*TxCmmspMFrServer,
        *RxClnt,*TxClnt,
        *RxCmmspMFrClnt,*TxCmmspMFrClnt;
Channel *ClntToRx2Cmmspm,
        *Rx1CmmspmToCmmspm,
        *CmmspmToRxClnt,*RxClntToClnt,
        *Rx2CmmspmToCmmspm;
int i,bKeepGoingCmms=TRUE,bKeepGoing1=TRUE,bKeepGoing2=TRUE,bKeepGoing3=TRUE;

  assert(inlen  > 2);
  assert(outlen > 2);
  for (i=0;i<(LEN_CHAN*3);i++){
    Buffer[i].Usado=FALSE;
  }
  RxClntToClnt      = ChanAlloc ();
  ClntToRx2Cmmspm   = ChanAlloc ();
  Rx1CmmspmToCmmspm = ChanAlloc ();
  CmmspmToRxClnt    = ChanAlloc ();
  RxClntToClnt      = ChanAlloc ();
  ClntToRx2Cmmspm   = ChanAlloc ();
  Rx2CmmspmToCmmspm = ChanAlloc ();
  RxCmmspMFrServer = ProcAlloc (RxProc,     0, 4, in[2],1,&bKeepGoing1,0);
  TxCmmspMFrServer = ProcAlloc (TxProc,     0, 4, Rx1CmmspmToCmmspm,1,&bKeepGoing1,0);
  CmmspM           = ProcAlloc (CmmspMProc,0, 6,
Rx1CmmspmToCmmspm,out[2],Rx2CmmspmToCmmspm,CmmspmToRxClnt,out[4],&bKeepGoingCmms);
  RxClnt           = ProcAlloc (RxProc,     0, 4, CmmspmToRxClnt,3,&bKeepGoing3,0);
  TxClnt           = ProcAlloc (TxProc,     0, 4, RxClntToClnt,3,&bKeepGoing3,0);
  Client           = ProcAlloc (ClientProc,0, 4, RxClntToClnt,ClntToRx2Cmmspm,in[3],out[3]);
  RxCmmspMFrClnt   = ProcAlloc (RxProc,     0, 4, ClntToRx2Cmmspm,2,&bKeepGoing2,0);
  TxCmmspMFrClnt   = ProcAlloc (TxProc,     0, 4, Rx2CmmspmToCmmspm,2,&bKeepGoing2,0);
if ((Client           == NULL)  || (CmmspM           == NULL) ||
    (RxCmmspMFrServer == NULL)  || (TxCmmspMFrServer == NULL) ||
    (RxClnt           == NULL)  || (TxClnt           == NULL) ||
    (RxCmmspMFrClnt   == NULL)  || (TxCmmspMFrClnt   == NULL)
    ){
    exit (EXIT_FAILURE);
  }

ProcPar(Client,CmmspM,RxCmmspMFrServer,TxCmmspMFrServer,RxClnt,TxClnt,RxCmmspMFrClnt,TxCmmspMFrCl
nt,NULL);
  ProcAllocClean (Client);
  ProcAllocClean (CmmspM);
  ProcAllocClean (RxCmmspMFrServer);
  ProcAllocClean (TxCmmspMFrServer);
  ProcAllocClean (RxClnt);
  ProcAllocClean (TxClnt);
  ProcAllocClean (RxCmmspMFrClnt);
  ProcAllocClean (TxCmmspMFrClnt);
}
/*AP:CASO EXCLUSIVO PARA ESTA APLICACION*/
```

11.3 SMMSPM.H

```
# include "MMSDecod.h"    /*      Procesos p emular canal de long. n      */
# include "mmsprmcf.h"
# include "mmsprmc0.h"
# include "mmsprmc1.h"
# include "mmsprmc3.h"
# include "mmsparam.h"
# include "mmsprmc4.h"

ConnTblType ConnTbl;

int ListenChans(Channel *FrServer, Channel *FrClnt,aleaf *AleafTmp,tTrama *sCmd){
int ch;
    ch=ProcAlt(FrServer,FrClnt,NULL);
    switch(ch){
      case 0:
        ChanIn(FrServer,(*sCmd),TRAMA_LEN);
        break;
      case 1:
        ChanIn(FrClnt,(*sCmd),TRAMA_LEN);
        break;
    }
    UnpackTrama(AleafTmp,(*sCmd));
    return(ch);
}

/*****************************************************************************/
/**********                          Proceso SmmspM                  **********/
/**********                                                          **********/

  /*****************************************************************************/
  /*** ESTADO DEL PROCESO: END                                                 ***/
void end44(Channel *ToServer,Channel *ToClnt){
  ChanOut(ToClnt,sFIN,TRAMA_LEN);
  ChanOut(ToServer,sFIN,TRAMA_LEN);
  return;
}
  /***                                                              END      ***/
  /*****************************************************************************/

  /*****************************************************************************/
  /*** ESTADO DEL PROCESO: SALONE          (end3)                              ***/
int end3(Channel *FrClnt,Channel *ToClnt,Channel *FrServer,Channel *ToServer){
tTrama sCmd;
aleaf SMAleaf;
int ch;
  while(TRUE){
    ch=ProcAlt(FrServer,FrClnt,NULL);
    switch (ch){
      case 0:
        ChanIn(FrServer,sCmd,TRAMA_LEN);
        UnpackTrama(&SMAleaf,sCmd);
        MMSDecod(sCmd,ch);
        if(SMAleaf.Primitive==LOGOUT_Request){
          SMAleaf.Primitive=LOGOUT_Confirm;
          PackTrama(&SMAleaf,sCmd);
          ChanOut(ToServer,sCmd,TRAMA_LEN);
          end44(ToServer,ToClnt);                          /*goto end44*/
          return(FALSE);
        }
        break;
      case 1:
        ChanIn(FrClnt,sCmd,TRAMA_LEN);
        UnpackTrama(&SMAleaf,sCmd);
        MMSDecod(sCmd,ch);
        if(SMAleaf.Primitive==LOGIN_Indication){
          SMAleaf.Primitive=LOGIN_Confirm;
          RdId(&(SMAleaf.Id),1);              /*leo registro id*/
          PackTrama(&SMAleaf,sCmd);
          ChanOut(ToClnt,sCmd,TRAMA_LEN);
          return(TRUE);                                  /*goto end2*/
```

```
        }
        break;
    }
  }
}
  /***                                                                    SALONE   ***/
  /*****************************************************************************/
  /*****************************************************************************/
  /*** ESTADO DEL PROCESO: LOFF                                                ***/
void ExpLoc(Channel *FrClnt,Channel *ToClnt,Channel *ToServer){
tTrama sCmd;
aleaf SMAleaf;
Primitive_Type Primitiva;

  while(TRUE){
    ChanIn(FrClnt,sCmd,TRAMA_LEN);
    UnpackTrama(&SMAleaf,sCmd);
    MMSDecod(sCmd,1);
    Primitiva=SMAleaf.Primitive;
    switch(Primitiva){
      case LOGOUT_Indication:
        SMAleaf.Primitive=LOGOUT_Confirm;
        RdId(&(SMAleaf.Id),1);              /*leo registro id*/
        PackTrama(&SMAleaf,sCmd);
        ChanOut(ToClnt,sCmd,TRAMA_LEN);
        break;
      case LOGOUT_Confirm:
        ChanOut(ToServer,sCmd,TRAMA_LEN);
        end44(ToServer,ToClnt);
        return;
        break;
      default:
        break;
    }
  }
}
  /***                                                                      LOFF   ***/
  /*****************************************************************************/


void SmmspMProc(Process * p,Channel *FrClnt, Channel *ToClnt,Channel *FrServer, Channel
*ToServer,
Channel *SmmspmToMon,int *bKeepGoing)
{
tTrama sCmd;
aleaf SMAleaf;
Boolean bStop;
int ch,FRSERVER=0,FRCLNT=1;
Primitive_Type Primitiva;
Boolean bOk;
int     CId;

  /*****************************************************************************/
  /***                  Inicializacion de Variables                          ***/
  p = p;
  InitiateConnTbl(&ConnTbl);
  /***                  Inicializacion de Variables                          ***/
  /*****************************************************************************/
  /*****************************************************************************/
  /*** ESTADO DEL PROCESO: BEGIN                                               ***/
  bStop=TRUE;
  while(bStop){
    ChanIn(FrServer,sCmd,TRAMA_LEN);
    SendMonitor(SmmspmToMon,sCmd,FROMSERVER,VALIDO);
    UnpackTrama(&SMAleaf,sCmd);
    MMSDecod(sCmd,0);
    if(SMAleaf.Primitive==LOGIN_Request){
      WrId(SMAleaf.Id,0);                /*registro el Local id*/
      SMAleaf.Primitive=LOGIN_Confirm;
      PackTrama(&SMAleaf,sCmd);
      ChanOut(ToServer,sCmd,TRAMA_LEN);
      break;
    }
  }
```

```
  /***                                                                  BEGIN    ***/
  /*************************************************************************/
  /*** ESTADO DEL PROCESO: SALONE                                              ***/
  if (end3(FrClnt,ToClnt,FrServer,ToServer)==FALSE)
    return;
  /*************************************************************************/
  /*** ESTADO DEL PROCESO: GROUP                                               ***/
  while ((*bKeepGoing)){
    ch=ListenChans(FrServer,FrClnt,&SMAleaf,&sCmd);
    MMSDecod(sCmd,ch);
    Primitiva=SMAleaf.Primitive;
    if(ch==FRCLNT){
      SendMonitor(SmmspmToMon,sCmd,FROMCMMSPM,VALIDO);
      switch(Primitiva){
  /*************************************************************************/
  /***                    Aviso de Expiracion remota                         ***/
        case LOGOUT_Indication:
          SMAleaf.Primitive=LOGOUT_Confirm;                    /*goto ExpRmt*/
          RdId(&(SMAleaf.Id),1);                              /*leo registro id*/
          PackTrama(&SMAleaf,sCmd);
          ChanOut(TcClnt,sCmd,TRAMA_LEN);
          if (end3(FrClnt,ToClnt,FrServer,ToServer)==FALSE)
            return;
          break;
  /*************************************************************************/
  /***                   Indicacion de Asociacion                            ***/
        case A_ASSOCIATE_indication:
          GetConnection(&bOk,&ConnTbl,&CId);
          if(bOk==TRUE){                          /*Cargo estado,rol y Pid*/
            ConnTbl[CId].State=CalledInit;
            ConnTbl[CId].Role=Responder;
            /*Identificacion local*/
            SMAleaf.Id.PId=LocalPid;
            SMAleaf.Id.AssId=ConnTbl[CId].Id.AssId;
            /*Descargo datos de identificacion en la tabla de conexion*/
            WrIdentity(SMAleaf.Id,&ConnTbl,CId,0);
            /*Completa el pdu para obtener enviar el response*/
            BuildIniRspPDU(&SMAleaf);
            PackTrama(&SMAleaf,sCmd);
            ChanOut(ToServer,sCmd,TRAMA_LEN);
          }
          else{

FillErrorPDU(&(SMAleaf.MmsPdu),ServiceError_errorClass_t_initiate,ServiceError_errorClass_t_initi
ate_t__other);
          }
          break;
  /*************************************************************************/
  /***                     Indicacion de Liberacion                          ***/
        case A_RELEASE_indication:
          FindConnection(&bOk,&ConnTbl,&CId,SMAleaf.Id.AssId);
          if(bOk==TRUE){
            if(ConnTbl[CId].State==Associated){
              ConnTbl[CId].State=CalledConc;
              /*Rtx el pdu de indication*/
              ChanOut(ToServer,sCmd,TRAMA_LEN);
            }                                                    /*else skip*/
          }
          else{
            /*no Existe -> PDU Conf. OK*/
            SMAleaf.Primitive=A_RELEASE_confirm;
            SMAleaf.MmsPdu.designator=MMSpdu_conclude_ResponsePDU;
            /*Cargo el id en el PDU con el id en la tabla*/
            RdIdentity(&(SMAleaf.Id),ConnTbl,CId,1);
            /*envio*/
            PackTrama(&SMAleaf,sCmd);
            ChanOut(ToClnt,sCmd,TRAMA_LEN);
          }
          break;
        case A_MMS_Indication:
/*************************************************************************/
/***                  Indicacion de Servicio Confirmado                    ***/
          if(SMAleaf.MmsPdu.designator==MMSpdu_confirmed_RequestPDU){
            FindConnection(&bOk,&ConnTbl,&CId,SMAleaf.Id.AssId);
```

```
            if(bOk==TRUE && ConnTbl[CId].State==Associated){
              bOk=ProcessIndication(&SMAleaf,&(ConnTbl[CId]));
              if(bOk==TRUE){
                /*Rtx el pdu de indication*/
                SMAleaf.Primitive=A_MMS_Indication;
                /*Cargo el id en el PDU con el id en la tabla*/
                RdIdentity(&(SMAleaf.Id),ConnTbl,CId,0);
                /*envio*/
                PackTrama(&SMAleaf,sCmd);
                ChanOut(ToServer,sCmd,TRAMA_LEN);
              }
              else{   /*Error: no se alojar la indicacion*/
                SMAleaf.Primitive=A_MMS_Response;
                /*Cargo el id en el PDU con el id en la tabla*/
                RdIdentity(&(SMAleaf.Id),ConnTbl,CId,1);

FillErrorPDU(&(SMAleaf.MmsPdu),ServiceError_errorClass_t_initiate,ServiceError_errorClass_t_initi
ate_t__other);
                /*envio*/
                PackTrama(&SMAleaf,sCmd);
                ChanOut(ToClnt,sCmd,TRAMA_LEN);
              }
            }
            else{/*error no existe la asociacion, aviso del error al cliente*/
              SMAleaf.Primitive=A_MMS_Response;
              /*Cargo el id en el PDU con el id en la tabla*/
              RdIdentity(&(SMAleaf.Id),ConnTbl,CId,1);

FillErrorPDU(&(SMAleaf.MmsPdu),ServiceError_errorClass_t_initiate,ServiceError_errorClass_t_initi
ate_t__other);
              /*envio*/
              PackTrama(&SMAleaf,sCmd);
              ChanOut(ToClnt,sCmd,TRAMA_LEN);
            }
          }
/*****************************************************************************/
/***                  Indicacion de Servicio Sin Confirmacion              ***/
          else if(SMAleaf.MmsPdu.designator==MMSpdu_Unconfirmed_RequestPDU){
            FindConnection(&bOk,&ConnTbl,&CId,SMAleaf.Id.AssId);
            if(bOk==TRUE && ConnTbl[CId].State==Associated){
              /*Rtx el pdu de indication*/
              SMAleaf.Primitive=A_MMS_Indication;
              /*Cargo el id en el PDU con el id en la tabla*/
              RdIdentity(&(SMAleaf.Id),ConnTbl,CId,0);
              /*envio*/
              PackTrama(&SMAleaf,sCmd);
              ChanOut(ToServer,sCmd,TRAMA_LEN);
            }/*No existe la asociacion -> skip*/
          }
          break;

        default:
          PackTrama(&SMAleaf,sCmd);
          ChanOut(ToServer,sCmd,TRAMA_LEN);
          break;
      }
    }
    if(ch==FRSERVER){
      SendMonitor(SmmspmToMon,sCmd,FROMSERVER,VALIDO);
      switch(Primitiva){
  /*****************************************************************************/
  /***                        Aviso de Expiracion Local                      ***/
        case LOGOUT_Request:
          SMAleaf.Primitive=LOGOUT_Indication;                /*goto ExpLoc*/
          RdId(&(SMAleaf.Id),1);                            /*leo registro id*/
          PackTrama(&SMAleaf,sCmd);
          ChanOut(ToClnt,sCmd,TRAMA_LEN);
          ExpLoc(FrClnt,ToClnt,ToServer);
          return;
          break;
  /*****************************************************************************/
  /***                    Respuesta de Asociacion                            ***/
        case A_ASSOCIATE_response:
          if (SMAleaf.MmsPdu.designator!=MMSpdu_initiate_ResponsePDU){
```

```
          /*if not pdu valido reject*/
         }
         else{
           FindConnection(&bOk,&ConnTbl,&CId,SMAleaf.Id.AssId);
           if(bOk==TRUE && ConnTbl[CId].State==CalledInit){
             /*Cargo estado y parametros de la conexion*/
             ConnTbl[CId].State=Associated;

ConnTbl[CId].MaxSegmentSize=SMAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedMaxSegmentSize


ConnTbl[CId].NegociatedMaxServOutstandingCalling=SMAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.neg
ociatedMaxServOutstandingCalling;

ConnTbl[CId].NegociatedMaxServOutstandingCalled=SMAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.nego
ciatedMaxServOutstandingCalled;

ConnTbl[CId].NegociatedDataStructureNestingLevel=SMAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.neg
ociatedDataStructureNestingLevel;
             /*Marco el pdu como un confirmacion de asociacion, transmito*/
             SMAleaf.Primitive=A_ASSOCIATE_confirm;
             RdIdentity(&(SMAleaf.Id),ConnTbl,CId,1);
             PackTrama(&SMAleaf,sCmd);
             ChanOut(ToClnt,sCmd,TRAMA_LEN);
           }                                                    /*else skip*/
         }
       break;
  /*****************************************************************************/
  /***                    Respuesta  de Liberacion                           ***/
       case A_RELEASE_response:
         if (SMAleaf.MmsPdu.designator!=MMSpdu_conclude_ResponsePDU){
          /*if not pdu valido reject*/
         }
         else{
           FindConnection(&bOk,&ConnTbl,&CId,SMAleaf.Id.AssId);
           if(bOk==TRUE && ConnTbl[CId].State==CalledConc){
             /*Marco el pdu como un confirmacion de asociacion, transmito*/
             SMAleaf.Primitive=A_RELEASE_confirm;
             RdIdentity(&(SMAleaf.Id),ConnTbl,CId,1);
             /*Elimino la asociacion*/
             FreeConnection(&bOk,&ConnTbl,CId,SMAleaf.Id.AssId);
             PackTrama(&SMAleaf,sCmd);
             ChanOut(ToClnt,sCmd,TRAMA_LEN);
           }                                                    /*else skip*/
         }
       break;
/*****************************************************************************/
/***                   Respuesta de Servicio Confirmado                    ***/
       case A_MMS_Response:
         if (SMAleaf.MmsPdu.designator!=MMSpdu_confirmed_ResponsePDU &&
             SMAleaf.MmsPdu.designator!=MMSpdu_confirmed_ErrorPDU){
            /*if not pdu valido reject, si es respuessta afirmativa o negativa,es lo mismo*/
         }
         else{
           FindConnection(&bOk,&ConnTbl,&CId,SMAleaf.Id.AssId);
           if(bOk==TRUE && ConnTbl[CId].State==Associated){
             bOk=ProcessIndication(&SMAleaf,&(ConnTbl[CId]));
             if(bOk==TRUE){
               SMAleaf.Primitive=A_MMS_Confirm;
               /*Cargo el id en el PDU con el id en la tabla*/
               RdIdentity(&(SMAleaf.Id),ConnTbl,CId,1);
               /*envio*/
               PackTrama(&SMAleaf,sCmd);
               ChanOut(ToClnt,sCmd,TRAMA_LEN);
             }/*else No existe la ind en staus W_Response skip*/
           }/*else No existe la asociacion ->skip*/
         }
       break;


     }
   }
  }
  return;
```

```
  /***                                                                    GROUP     ***/
  /***                                                                              ***/
  /*********************************************************************************/
}
```

11.4 CMMSPM.H

```
# include "mmsprmcf.h"
# include "mmsprmc0.h"
# include "mmsprmc1.h"
# include "mmsprmc3.h"
# include "mmsparam.h"
# include "mmsprmc4.h"

ConnTblType ConnTbl;



int ListenChans(Channel *FrServer, Channel *FrClnt,
Channel *CmmspmToMon,aleaf *AleafTmp,tTrama *sCmd){
int ch;
    ch=ProcAlt(FrServer,FrClnt,NULL);
    switch(ch){
      case 0:
        ChanIn(FrServer,(*sCmd),TRAMA_LEN);
        ChanOut(CmmspmToMon,(*sCmd),TRAMA_LEN);
        break;
      case 1:
        ChanIn(FrClnt,(*sCmd),TRAMA_LEN);
        ChanOut(CmmspmToMon,(*sCmd),TRAMA_LEN);
        break;
    }
    UnpackTrama(AleafTmp,(*sCmd));
    return(ch);
}

/*********************************************************************************/
/**********                          Proceso CmmspM                      **********/
/**********                                                              **********/
  /*********************************************************************************/
  /***             Rutina de Terminacion Remota                                  ***/
void ExpRmt(){

}
  /***                                                                    ExpRmt    ***/
  /*********************************************************************************/
  /*********************************************************************************/
  /*** ESTADO DEL PROCESO: END                                                     ***/
void end44(Channel *ToServer,Channel *ToClnt){
  ChanOut(ToClnt,sFIN,TRAMA_LEN);
  ChanOut(ToServer,sFIN,TRAMA_LEN);
  return;
}
  /***                                                                    END       ***/
  /*********************************************************************************/
  /*********************************************************************************/
  /*** ESTADO DEL PROCESO: SALONE                                                  ***/
void end3(Channel *ToServer,Channel *FrClnt, Channel *ToClnt){
tTrama sCmd;
aleaf AleafTmp;
Boolean bStop=TRUE;
Primitive_Type Primitiva;

  while(bStop){
    ChanIn(FrClnt,sCmd,TRAMA_LEN);
    UnpackTrama(&AleafTmp,sCmd);
    Primitiva=AleafTmp.Primitive;
    switch(Primitiva){
      case LOGOUT_Request:
```

```
        AleafTmp.Primitive=LOGOUT_Confirm;
        PackTrama(&AleafTmp,sCmd);
        ChanOut(ToClnt,sCmd,TRAMA_LEN);
        end44(ToServer,ToClnt);                    /*goto end44;*/
        return;
        break;
      default:
        break;
    }
  }
}
  /***                                                                    SALONE   ***/
  /*************************************************************************/


  /*************************************************************************/
  /*** ESTADO DEL PROCESO: LOFF                                           ***/
void ExpLoc(Channel *FrServer, Channel *ToServer,Channel *ToClnt){
tTrama sCmd;
aleaf AleafTmp;
Boolean bStop=TRUE;

  while(bStop){
    ChanIn(FrServer,sCmd,TRAMA_LEN);
    UnpackTrama(&AleafTmp,sCmd);
    switch(AleafTmp.Primitive){
      case LOGOUT_Indication:
        AleafTmp.Primitive=LOGOUT_Confirm;
        RdId(&(AleafTmp.Id),1);              /*leo registro id*/
        PackTrama(&AleafTmp,sCmd);
        ChanOut(ToServer,sCmd,TRAMA_LEN);
        break;
      case LOGOUT_Confirm:
        ChanOut(ToClnt,sCmd,TRAMA_LEN);
        end44(ToServer,ToClnt);                    /*goto end44;*/
        return;
        break;
    }
  }
}
  /***                                                                  LOFF   ***/
  /*************************************************************************/

void CmmspMProc(Process * p,Channel *FrServer, Channel *ToServer,Channel *FrClnt,
Channel *ToClnt,Channel *CmmspmToMon, int *bKeepGoing)
{
tTrama sCmd;
aleaf   CMAleaf;
int bStop;
int ch,FRSERVER=0,FRCLNT=1;
Primitive_Type Primitiva;
Boolean bOk;
int     CId;

  /*************************************************************************/
  /***                  Inicializacion de Variables                      ***/
  p = p;
  InitiateConnTbl(&ConnTbl);
  /***                                                                INICIA   ***/
  /*************************************************************************/
  /*************************************************************************/
  /*** ESTADO DEL PROCESO: BEGIN                                          ***/
  bStop=TRUE;
  while(bStop){
    ChanIn(FrClnt,sCmd,TRAMA_LEN);
    ChanOut(CmmspmToMon,sCmd,TRAMA_LEN);
    UnpackTrama(&CMAleaf,sCmd);
    if(CMAleaf.Primitive==LOGIN_Request){
      CMAleaf.Id.PIdRmt=0;
      WrId(CMAleaf.Id,0);                      /*registro el Local id*/
      CMAleaf.Primitive=LOGIN_Indication;
      RdId(&(CMAleaf.Id),1);                   /*leo registro id*/
      PackTrama(&CMAleaf,sCmd);
      ChanOut(ToServer,sCmd,TRAMA_LEN);
```

```
      break;                                        /*goto chk*/
    }
  }
  /***                                                                    BEGIN    ***/
  /*************************************************************************************/
  /*************************************************************************************/
  /*** ESTADO DEL PROCESO: CHK                                                       ***/
  while(bStop){
    ChanIn(FrServer,sCmd,TRAMA_LEN);
    ChanOut(CmmspmToMon,sCmd,TRAMA_LEN);
    UnpackTrama(&CMAleaf,sCmd);
    if(CMAleaf.Primitive==LOGIN_Confirm){
      RmtPid=CMAleaf.Id.PIdRmt;
      ChanOut(ToClnt,sCmd,TRAMA_LEN);
      break;                                        /*goto end2;*/
    }
  }
  /***                                                                      CHK    ***/
  /*************************************************************************************/
  /*************************************************************************************/
  /*** (end2) ESTADO DEL PROCESO: GROUP                                              ***/
  while ((*bKeepGoing)){
    ch=ListenChans(FrServer,FrClnt,CmmspmToMon,&CMAleaf,&sCmd);
    Primitiva=CMAleaf.Primitive;
    if(ch==FRCLNT){
      switch (Primitiva){
        case LOGOUT_Request:                        /*Aviso de Expiracion local*/
          CMAleaf.Primitive=LOGOUT_Indication;
          RdId(&(CMAleaf.Id),1);                    /*leo registro id*/
          PackTrama(&CMAleaf,sCmd);
          ChanOut(ToServer,sCmd,TRAMA_LEN);
          ExpLoc(FrServer,ToServer,ToClnt);         /*goto ExpLoc;*/
          return;
          break;
  /*************************************************************************************/
  /***                    Peticion de Asociacion                                   ***/
        case A_ASSOCIATE_request:
          GetConnection(&bOk,&ConnTbl,&CId);
          if(bOk==TRUE){/*Cargo estado,rol y el Pid en la conexion*/
            ConnTbl[CId].State=CallingInit;
            ConnTbl[CId].Role=Initiator;
            ConnTbl[CId].Id.PId=LocalPid;
            ConnTbl[CId].Id.PIdRmt=RmtPid;
            /*Cargo la info de id para dar a conocerlo al proc. llamante*/
            RdIdentity(&(CMAleaf.Id),ConnTbl,CId,0);
            /*Construye el pdu de indication*/
            CMAleaf.Primitive=A_ASSOCIATE_indication;
            /*Cargo el id en el PDU con el id en la tabla*/
            RdIdentity(&(CMAleaf.Id),ConnTbl,CId,1);
            /*Completa el pdu para obtener enviar el indication y envio*/
            FillIniRqtPDU(&CMAleaf);
            PackTrama(&CMAleaf,sCmd);
            ChanOut(ToServer,sCmd,TRAMA_LEN);
          }
          else{/*error local*/

FillErrorPDU(&(CMAleaf.MmsPdu),ServiceError_errorClass_t_initiate,ServiceError_errorClass_t_initi
ate_t__other);
          }
          break;
/*************************************************************************************/
/***                    Peticion de Liberacion                                   ***/
        case A_RELEASE_request:
          if (CMAleaf.MmsPdu.designator!=MMSpdu_conclude_RequestPDU){
           /*if not pdu valido reject*/
          }
          else{
            FindConnection(&bOk,&ConnTbl,&CId,CMAleaf.Id.AssId);
            if(bOk==TRUE ){
              if(ConnTbl[CId].State==Associated){
                ConnTbl[CId].State=CallingConc;
                /*Construye el pdu de indication*/
                CMAleaf.Primitive=A_RELEASE_indication;
                /*Cargo el id en el PDU con el id en la tabla*/
```

```
                RdIdentity(&(CMAleaf.Id),ConnTbl,CId,1);
                /*envio*/
                PackTrama(&CMAleaf,sCmd);
                ChanOut(ToServer,sCmd,TRAMA_LEN);
              }
              else if(ConnTbl[CId].State!=Associated &&
ConnTbl[CId].State!=CallingConc){/*ConnTbl[CId].State==Associated*/
                /*Error->Construye el pdu de confirmacion c/error*/
                CMAleaf.Primitive=A_RELEASE_confirm;

FillErrorPDU(&(CMAleaf.MmsPdu),ServiceError_errorClass_t_initiate,ServiceError_errorClass_t_initi
ate_t__other);
                /*Cargo el id en el PDU con el id en la tabla*/
                RdIdentity(&(CMAleaf.Id),ConnTbl,CId,0);
                /*envio*/
                PackTrama(&CMAleaf,sCmd);
                ChanOut(ToClnt,sCmd,TRAMA_LEN);
              }
            }
            else{                                   /*bOk==TRUE*/
              /*No Existe ->Construye el pdu de confirmacion*/
              CMAleaf.Primitive=A_RELEASE_confirm;
              CMAleaf.MmsPdu.designator=MMSpdu_conclude_ResponsePDU;
              /*Cargo el id en el PDU con el id en la tabla*/
              RdIdentity(&(CMAleaf.Id),ConnTbl,CId,0);
              /*envio*/
              PackTrama(&CMAleaf,sCmd);
              ChanOut(ToClnt,sCmd,TRAMA_LEN);
            }
          }
          break;
        case A_MMS_Request:
/*****************************************************************************/
/***                    Peticion de Servicio Confirmado                    ***/
          if(CMAleaf.MmsPdu.designator==MMSpdu_confirmed_RequestPDU){
            if (CMAleaf.MmsPdu.designator!=MMSpdu_confirmed_RequestPDU){
              /*if not pdu valido reject.-sustituir por funcion validadora de pdu*/
            }
            else{
              FindConnection(&bOk,&ConnTbl,&CId,CMAleaf.Id.AssId);
              if(bOk==TRUE && ConnTbl[CId].State==Associated){
                bOk=ProcessRequest(&CMAleaf,&(ConnTbl[CId]));
                if(bOk==TRUE){
                  /*Construye el pdu de indication*/
                  CMAleaf.Primitive=A_MMS_Indication;
                  /*Cargo el id en el PDU con el id en la tabla*/
                  RdIdentity(&(CMAleaf.Id),ConnTbl,CId,1);
                  /*envio*/
                  PackTrama(&CMAleaf,sCmd);
                  ChanOut(ToServer,sCmd,TRAMA_LEN);
                }
                else{
                  /* Error: no se pudo alojar la peticion*/
                  CMAleaf.Primitive=A_MMS_Confirm;

FillErrorPDU(&(CMAleaf.MmsPdu),ServiceError_errorClass_t_initiate,ServiceError_errorClass_t_initi
ate_t__other);
                  /*Cargo el id en el PDU con el id en la tabla*/
                  RdIdentity(&(CMAleaf.Id),ConnTbl,CId,0);
                  /*envio*/
                  PackTrama(&CMAleaf,sCmd);
                  ChanOut(ToClnt,sCmd,TRAMA_LEN);
                }
              }
              else{
                /*error no existe la asociacion correspondiente,pdu de conf. de serv.conf
c/error*/
                CMAleaf.Primitive=A_MMS_Confirm;

FillErrorPDU(&(CMAleaf.MmsPdu),ServiceError_errorClass_t_initiate,ServiceError_errorClass_t_initi
ate_t__other);
                /*Cargo el id en el PDU con el id en la tabla*/
                RdIdentity(&(CMAleaf.Id),ConnTbl,CId,0);
                /*envio*/
```

```
                PackTrama(&CMAleaf,sCmd);
                ChanOut(ToClnt,sCmd,TRAMA_LEN);
              }
            }
          }
/*****************************************************************************/
/***                  Peticion de Servicio Sin Confirmacion                ***/
          else if(CMAleaf.MmsPdu.designator==MMSpdu_Unconfirmed_RequestPDU){
            if (CMAleaf.MmsPdu.designator!=MMSpdu_Unconfirmed_RequestPDU){
              /*if not pdu valido reject.-sustituir por funcion validadora de pdu*/
            }
            else{
              FindConnection(&bOk,&ConnTbl,&CId,CMAleaf.Id.AssId);
              if(bOk==TRUE && ConnTbl[CId].State==Associated){
                /*Construye el pdu de indication*/
                CMAleaf.Primitive=A_MMS_Indication;
                /*Cargo el id en el PDU con el id en la tabla*/
                RdIdentity(&(CMAleaf.Id),ConnTbl,CId,1);
                /*envio*/
                PackTrama(&CMAleaf,sCmd);
                ChanOut(ToServer,sCmd,TRAMA_LEN);
              }/*else no existe la asociacion -> skip*/
            }
          }
          else{
              /*pdu no valido -> reject*/
          }
          break;
/*****************************************************************************/
/***                  Peticion de Servicio Sin Confirmacion                ***/

        default:
          RdId(&(CMAleaf.Id),1);              /*leo registro id*/
          PackTrama(&CMAleaf,sCmd);
          ChanOut(ToServer,sCmd,TRAMA_LEN);
          break;
      }
    }
    if(ch==FRSERVER){
      switch (Primitiva){
  /*****************************************************************************/
  /***                     Aviso de Expiracion remota                        ***/
        case LOGOUT_Indication:                 /*Aviso de Expiracion remota*/

          CMAleaf.Primitive=LOGOUT_Confirm;     /*goto ExpRmt*/
          RdId(&(CMAleaf.Id),1);              /*leo registro id*/
          PackTrama(&CMAleaf,sCmd);
          ChanOut(ToServer,sCmd,TRAMA_LEN);
          end3(ToServer,FrClnt,ToClnt);          /*goto end3*/
          return;
          break;
  /*****************************************************************************/
  /***                     Confirmacion de Asociacion                        ***/
        case A_ASSOCIATE_confirm:
          if (CMAleaf.MmsPdu.designator!=MMSpdu_initiate_ResponsePDU){
           /*if not pdu valido reject*/
          }
          else{
            FindConnection(&bOk,&ConnTbl,&CId,CMAleaf.Id.AssId);
            if(bOk==TRUE ){
              if(ConnTbl[CId].State==CallingInit){
                if(CMAleaf.MmsPdu.designator==MMSpdu_initiate_ResponsePDU){
                  /*Cargo estado y parametros de la conexion*/
                  ConnTbl[CId].State=Associated;

ConnTbl[CId].MaxSegmentSize=CMAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedMaxSegmentSize
;

ConnTbl[CId].NegociatedMaxServOutstandingCalling=CMAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.neg
ociatedMaxServOutstandingCalling;

ConnTbl[CId].NegociatedMaxServOutstandingCalled=CMAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.nego
ciatedMaxServOutstandingCalled;
```

```
ConnTbl[CId].NegociatedDataStructureNestingLevel=CMAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.neg
ociatedDataStructureNestingLevel;
                WrIdentity(CMAleaf.Id,&ConnTbl,CId,0);
                /*Aviso que ya esta la conexion*/
                RdIdentity(&(CMAleaf.Id),ConnTbl,CId,0);
                PackTrama(&CMAleaf,sCmd);
                ChanOut(ToClnt,sCmd,TRAMA_LEN);
              }
              else if(CMAleaf.MmsPdu.designator!=MMSpdu_initiate_ResponsePDU){
                /*Error*/
              }
            }
            /*else skip;*/
          }
          else{
          /*      to_link[n]!MAbIn_abort_PDU,1,k,x3,x4;*/
          }
        }
        break;
  /*****************************************************************************/
  /***                    Confirmacion de Liberacion                         ***/
      case A_RELEASE_confirm:
        if (CMAleaf.MmsPdu.designator!=MMSpdu_conclude_ResponsePDU){
         /*if not pdu valido reject*/
        }
        else{
          FindConnection(&bOk,&ConnTbl,&CId,CMAleaf.Id.AssId);
          if(bOk==TRUE ){
            if(ConnTbl[CId].State==CallingConc){
            /*Aviso que ya esta desconectado*/
            CMAleaf.Primitive=A_RELEASE_confirm;
            CMAleaf.MmsPdu.designator=MMSpdu_conclude_ResponsePDU;
            /*Cargo el id en el PDU con el id en la tabla*/
            RdIdentity(&(CMAleaf.Id),ConnTbl,CId,0);
            /*Libero la conexion*/
            FreeConnection(&bOk,&ConnTbl,CId,CMAleaf.Id.AssId);
            /*envio*/
            PackTrama(&CMAleaf,sCmd);
            ChanOut(ToClnt,sCmd,TRAMA_LEN);
            }/*Si no es CallingConc skip Nunca debe pasar*/
          }/* Si no existe skip*/
        }
        break;
/*****************************************************************************/
/***                 Confirmacion de Servicio Confirmado                   ***/
      case A_MMS_Confirm:
        /*No importa si la respuesta es afirmativa o negativa*/
        if (CMAleaf.MmsPdu.designator!=MMSpdu_confirmed_ResponsePDU &&
            CMAleaf.MmsPdu.designator!=MMSpdu_confirmed_ErrorPDU){
         /*if not pdu valido reject*/
        }
        else{
          if(bOk==TRUE && ConnTbl[CId].State==Associated){
            bOk=ProcessRequest(&CMAleaf,&(ConnTbl[CId]));
            if(bOk==TRUE){
              /*Aviso que el servicio fue exitoso localmente*/
              CMAleaf.Primitive=A_MMS_Confirm;
              CMAleaf.MmsPdu.designator=MMSpdu_confirmed_ResponsePDU;
              /*Cargo el id en el PDU con el id en la tabla*/
              RdIdentity(&(CMAleaf.Id),ConnTbl,CId,0);
              /*envio*/
              PackTrama(&CMAleaf,sCmd);
              ChanOut(ToClnt,sCmd,TRAMA_LEN);
            }
            /*else Error al chechar la tabla de petciones: skip*/
          }
          /*else No existe la asoiciacion correspondienteskip*/
        }
        break;

      default:
        RdId(&(CMAleaf.Id),1);              /*leo registro id*/
        PackTrama(&CMAleaf,sCmd);
```

```
            ChanOut(ToServer,sCmd,TRAMA_LEN);
            break;
        }
      }
    }
    /***                                                                    GROUP    ***/
    /*************************************************************************************/
}
```

11.5 MAQUINA DEL PROTOCOLO MMS

```
/***                                                  ***/
/***       MMSPRMCF.H          ***/
/***                                                  ***/

/*Configuracion para la maquina de estados*/

/* Parametros de las tablas pet y req de cada asociacion*/
#define MAX_PENDING_REQUEST_NUMBER  4
#define MAX_PENDING_INDICATION_NUMBER 4

/* Parametros de la tabla de conexiones*/
#define MAX_CONNECTIONS 4
#define MAX_ASSOCIATIONS 500

/* Parametros de la tabla comunicacion interprocesos*/
#define MAX_PIPE 2

/* Unidad de Tiempo*/
int timeunit=15625; /*15625 es un segundo en procesos de prioridad normal*/

/***                                                  ***/
/***       MMSPRMC0.H          ***/
/***                                                  ***/

/*Definicion y manejo tabla de req e ind de cada asociacion*/

typedef enum { Idle,Wating_Conf,Wating_Resp  } mmpm_fsm_type;
typedef enum { IdleTimer, DeathMark} Timer_State_type;

typedef struct {
    Boolean             in_use;
    unsignedlong        invoke_id;
    mmpm_fsm_type       mmpm_machine_state;
    Timer_State_type    TimerState;
}pending_element_type;
typedef pending_element_type request_pending_table_type[MAX_PENDING_REQUEST_NUMBER];
typedef pending_element_type indication_pending_table_type[MAX_PENDING_INDICATION_NUMBER];

Boolean TableRqtIsFull(request_pending_table_type *the_table)
{
  int i;

  for (i=0;i<MAX_PENDING_REQUEST_NUMBER;i++)
  {
    if ( (*the_table[i]).in_use==FALSE )
      return (FALSE);
  }
  return (TRUE);
}

Boolean TableIndIsFull(indication_pending_table_type *the_table)
{
  int i;

  for (i=0;i<MAX_PENDING_INDICATION_NUMBER;i++)
  {
    if ( (*the_table[i]).in_use==FALSE )
      return (FALSE);
  }
```

```
  return (TRUE);
}

int PendingRqtNumb(request_pending_table_type *the_table)
{
  int i,count=0;

  for (i=0;i<MAX_PENDING_REQUEST_NUMBER;i++)
  {
    if ( (*the_table[i]).in_use==TRUE )
       count = count + 1;
  }
  return (count);
}

int PendingIndNumb(indication_pending_table_type *the_table)
{
  int i,count=0;

  for (i=0;i<MAX_PENDING_INDICATION_NUMBER;i++)
  {
    if ( (*the_table[i]).in_use==TRUE )
       count = count + 1;
  }
  return (count);
}

Boolean FindRqt(request_pending_table_type *the_table,Unsigned32 invoke_id)
{
  int i;

  for (i=0;i<MAX_PENDING_REQUEST_NUMBER;i++)
  {
    if ( (*the_table[i]).invoke_id == invoke_id )
      return (TRUE);
  }
  return (FALSE);
}

Boolean FindInd(indication_pending_table_type *the_table,Unsigned32 invoke_id)
{
  int i;

  for (i=0;i<MAX_PENDING_INDICATION_NUMBER;i++)
  {
    if ( (*the_table[i]).invoke_id == invoke_id )
      return (TRUE);
  }
  return (FALSE);
}

void AddRqt(request_pending_table_type *the_table,pending_element_type *the_item)
{
  int i;

  for (i=0;i<MAX_PENDING_REQUEST_NUMBER;i++)
  {
    if ( (*the_table[i]).in_use==FALSE )
    {
      (*the_table[i]).in_use = TRUE;
      (*the_table[i]).invoke_id = (*the_item).invoke_id;
      (*the_table[i]).mmpm_machine_state = (*the_item).mmpm_machine_state;
      (*the_table[i]).TimerState=IdleTimer;
    }
  }
}

void AddInd(indication_pending_table_type *the_table,pending_element_type *the_item)
{
  int i;

  for (i=0;i<MAX_PENDING_INDICATION_NUMBER;i++)
  {
    if ( (*the_table[i]).in_use==FALSE )
```

```
    {
      (*the_table[i]).in_use = TRUE;
      (*the_table[i]).invoke_id = (*the_item).invoke_id;
      (*the_table[i]).mmpm_machine_state = (*the_item).mmpm_machine_state;
      (*the_table[i]).TimerState=IdleTimer;
    }
  }
}

void RemoveRqt(request_pending_table_type *the_table,Unsigned32 invoke_id)
{
  int i;

  for (i=0;i<MAX_PENDING_REQUEST_NUMBER;i++)
  {
    if ( (*the_table[i]).invoke_id == invoke_id )
      (*the_table[i]).in_use = FALSE;
  }
}

void RemoveInd(indication_pending_table_type *the_table,Unsigned32 invoke_id)
{
  int i;

  for (i=0;i<MAX_PENDING_INDICATION_NUMBER;i++)
  {
    if ( (*the_table[i]).invoke_id == invoke_id )
      (*the_table[i]).in_use = FALSE;
  }
}

void InitiateRqt(request_pending_table_type *the_table)
{
  int i;

  for (i=0;i<MAX_PENDING_REQUEST_NUMBER;i++)
  {
    (*the_table[i]).in_use = FALSE;
  }
}

void InitiateInd(indication_pending_table_type *the_table)
{
  int i;

  for (i=0;i<MAX_PENDING_INDICATION_NUMBER;i++)
  {
    (*the_table[i]).in_use = FALSE;
  }
}


/***                                              ***/
/***       MMSPRMC1.H          ***/
/***                                              ***/

/*Definicion y manejo tabla de conexiones*/
typedef enum {IdleConn,
CallingInit,CalledInit,CallingConc,CalledConc,Associated}MMPM_Connect_State;
typedef enum {Initiator, Responder}Role_Type;
int LocalPid=0;
int RmtPid=0;
typedef struct {
    Boolean                         InUse;
    IdType                          Id;
    MMPM_Connect_State              State;
    Timer_State_type                TimerState;
    Role_Type                       Role;
    Integer32                       MaxSegmentSize;
    Integer16                       NegociatedMaxServOutstandingCalling;
    Integer16                       NegociatedMaxServOutstandingCalled;
    Integer8                        NegociatedDataStructureNestingLevel;
    request_pending_table_type      PendRqtTbl;
    indication_pending_table_type   PendIndTbl;
```

```
}ConnType;

typedef ConnType ConnTblType[MAX_CONNECTIONS];

ConnType ConnItem;

/*****************************************************************************/
/* InitiateConnTbl                                                           */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
void InitiateConnTbl(ConnTblType *ConnTbl)
{
  int i;

  for (i=0;i<MAX_CONNECTIONS;i++)
  {
    (*ConnTbl)[i].InUse=FALSE;
  }
}                                                    /* InitiateConnTbl */

/*****************************************************************************/
/* ConnTblIsFull:                                                            */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
Boolean ConnTblIsFull(ConnTblType *ConnTbl)
{
  int i;

  for (i=0;i<MAX_CONNECTIONS;i++)
  {
    if ( (*ConnTbl)[i].InUse==FALSE )
      return (FALSE);
  }
  return (TRUE);
}                                                    /* ConnTblIsFull */

/*****************************************************************************/
/* NextAssId:                                                                */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
Handle NextAssId(ConnTblType *ConnTbl)
{
int i;
Handle TopId=0;

  for (i=0;i<MAX_CONNECTIONS;i++)
  {
    if ( (*ConnTbl)[i].InUse==TRUE )
    {
      if( (*ConnTbl)[i].Id.AssId>TopId )
        TopId=(*ConnTbl)[i].Id.AssId;
    }
  }
  if (TopId ==MAX_ASSOCIATIONS) TopId=0;
  TopId=TopId + 1;
  return(TopId);
}                                                    /* NextAssId */

/*****************************************************************************/
/* AddAss:                                                                   */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
int AddAss(ConnTblType *ConnTbl)
{
int i;
```

```
  for (i=0;i<MAX_CONNECTIONS;i++)
  {
    if ( (*ConnTbl)[i].InUse==FALSE )
    {
      (*ConnTbl)[i].InUse=ConnItem.InUse;
      (*ConnTbl)[i].Id.AssId=ConnItem.Id.AssId;
      (*ConnTbl)[i].TimerState=IdleTimer;
      break;
    }
  }
  return(i);
}                                                                    /* AddAss */


/*****************************************************************************/
/* FindAss:                                                                  */
/* Encuentra el id de la conexion de la asociacion
*/
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
int FindAss(ConnTblType *ConnTbl,Handle AssId)
{
int i;
Boolean HereIs=FALSE;

  for (i=0;i<MAX_CONNECTIONS;i++)
  {
    if ((*ConnTbl)[i].InUse==TRUE && (*ConnTbl)[i].Id.AssId==AssId)
    {
      HereIs=TRUE;
      break;
    }
  }
  if(HereIs==TRUE)
    return(i);
  else
    return(-1);
}                                                                  /* FindAss */

/*****************************************************************************/
/* GetConnection: obtiene la referencia a una associacion                    */
/* bRtn.      out: resultado de la funcion.                                  */
/* ConnId: in: out: conexion id                                              */
/*****************************************************************************/
void GetConnection(Boolean *bRtn,ConnTblType *ConnTbl,int *ConnId)
{

Boolean Ok=FALSE;
int CId=-1;

  *bRtn=FALSE;
  Ok=ConnTblIsFull(ConnTbl);
  if (Ok==FALSE)
  {
    ConnItem.InUse=TRUE;
    ConnItem.Id.AssId=NextAssId(ConnTbl);
    CId=AddAss(ConnTbl);
    *bRtn= TRUE;
    InitiateRqt(&((*ConnTbl)[CId].PendRqtTbl));
    InitiateInd(&((*ConnTbl)[CId].PendIndTbl));
  }
  *ConnId=CId;
}                                                          /* GetConnection */

/*****************************************************************************/
/* FindConnection: obtiene la referencia a una associacion                   */
/* bRtn.      out: resultado de la funcion.                                  */
/*****************************************************************************/
void FindConnection(Boolean *bRtn,ConnTblType *ConnTbl,int *ConnId,Handle AssId)
{
int CId=-1;
  *bRtn=FALSE;
  CId=FindAss(ConnTbl,AssId);
```

```
  if(CId>=0)
    *bRtn= TRUE;
  *ConnId=CId;
}                                                          /* FindConnection*/

/*****************************************************************************/
/* FreeConnection: libera una conexion                                       */
/* bRtn.      out: si existia la asociacion TRUE                             */
/*****************************************************************************/
void FreeConnection(Boolean *bRtn,ConnTblType *ConnTbl,int ConnId,Handle AssId)
{
  *bRtn=FALSE;
  if((*ConnTbl)[ConnId].Id.AssId==AssId)
  {
   (*ConnTbl)[ConnId].InUse=FALSE;
    *bRtn= TRUE;
  }
}                                                          /* FreeConnection*/

/*****************************************************************************/
/* WrIdentity : Fn utilizada para bajar la identificacion que viene en el    */
/*              a la tabla de conexion                                       */
/*****************************************************************************/
void WrIdentity(IdType NewId,ConnTblType *ConnTbl,int ConnId,int Param)
{
  /*El id Asociacion Local esta en el AssIdRmt del Arreglo NewId*/
  if(Param>0){
    (*ConnTbl)[ConnId].Id.PId     =NewId.PIdRmt;
    (*ConnTbl)[ConnId].Id.AssIdRmt=NewId.AssId;
    (*ConnTbl)[ConnId].Id.PIdRmt  =NewId.PId;
  }
  else{
    /*El id Asociacion Local esta en el AssId del Arreglo NewId*/
    (*ConnTbl)[ConnId].Id.PId     =NewId.PId;
    (*ConnTbl)[ConnId].Id.AssIdRmt=NewId.AssIdRmt;
    (*ConnTbl)[ConnId].Id.PIdRmt  =NewId.PIdRmt;
  }
}                                                          /* WrIdentity */

/*****************************************************************************/
/* RdIdentity : Fn utilizada para subir la identificacion de la tabla de con-*/
/*              xion a aleaf                                                 */
/*****************************************************************************/
void RdIdentity(IdType *NewId,ConnTblType ConnTbl,int ConnId,int Param)
{
  /*El id Asociacion va en el AssIdRmt del Arreglo NewId*/
  if(Param>0){
    (*NewId).AssId=ConnTbl[ConnId].Id.AssIdRmt;
    (*NewId).PId=ConnTbl[ConnId].Id.PIdRmt;
    (*NewId).AssIdRmt=ConnTbl[ConnId].Id.AssId;
    (*NewId).PIdRmt=ConnTbl[ConnId].Id.PId;
  }
  else{
    /*El id Asociacion va en el AssId del Arreglo NewId*/
    (*NewId).AssId=ConnTbl[ConnId].Id.AssId;
    (*NewId).PId=ConnTbl[ConnId].Id.PId;
    (*NewId).AssIdRmt=ConnTbl[ConnId].Id.AssIdRmt;
    (*NewId).PIdRmt=ConnTbl[ConnId].Id.PIdRmt;
  }
}                                                          /* RdIdentity */

/*****************************************************************************/
/* WrId : Fn utilizada para bajar la identificacion de los procesos          */
/*****************************************************************************/
void WrId(IdType NewId,int Param)
{
  /*Intercambio Local-remoto*/
  if(Param>0){
    LocalPid=NewId.PIdRmt;
    RmtPid=NewId.PId;
  }
  else{
    /*Local-Local*/
    LocalPid=NewId.PId;
```

```
    RmtPid=NewId.PIdRmt;
  }
}                                                    /* WrId */

/*****************************************************************************/
/* RdId : Fn utilizada para subir la identificacion de los procesos          */
/*****************************************************************************/
void RdId(IdType *NewId,int Param)
{
  /*Intercambio Local-remoto*/
  if(Param>0){
    (*NewId).PId=RmtPid;
    (*NewId).PIdRmt=LocalPid;
  }
  else{
    /*Local-Local*/
    (*NewId).PId=LocalPid;
    (*NewId).PIdRmt=RmtPid;
  }
}                                                    /* RdId*/

/***                                          ***/
/***       MMSPRMC4.H          ***/
/***                                          ***/
/*Llena el MmsPdu con parametros locales*/
void FillIniRqtPDU(aleaf *aleaf)
{
  (*aleaf).MmsPdu.designator=MMSpdu_initiate_RequestPDU;
  (*aleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedMaxSegmentSize
=LocalMaxSegmentSize;
  (*aleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedMaxSegmentSize_present
=LocalMaxSegmentSize_present;
  (*aleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedMaxServOutstandingCalling
=LocalMaxServOutstandingCalling;
  (*aleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedMaxServOutstandingCalled
=LocalMaxServOutstandingCalled;
  (*aleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedDataStructureNestingLevel
=LocalDataStructureNestingLevel;

(*aleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedDataStructureNestingLevel_present=LocalDataS
tructureNestingLevel_present;
}

/*Llena el MmsPdu de error*/
void FillErrorPDU(MMSpdu *ErrMMSpdu,ServiceError_errorClass_t_Choice Error,int NumErr)
{
  switch(Error){
  case ServiceError_errorClass_t_initiate:
    (*ErrMMSpdu).designator=MMSpdu_initiate_ErrorPDU;
    (*ErrMMSpdu).u.MMSpdu_initiate_ErrorPDU.errorClass.designator=Error;

(*ErrMMSpdu).u.MMSpdu_initiate_ErrorPDU.errorClass.u.ServiceError_errorClass_t_initiate=NumErr;
    break;
  }
}

/* BuildIniRspPDU: Construye aleaf de respuesta temporal con los parametros MMS local y remoto*/
/* Para procosar despues la respuesta*/
void BuildIniRspPDU(aleaf *RmAleaf)
{
aleaf RspAleaf;

  RspAleaf.Primitive=A_ASSOCIATE_indication;
  /* Datos de identifiacion Locales y remotos*/
  RspAleaf.Id.AssId    =(*RmAleaf).Id.AssId;
  RspAleaf.Id.PId      =(*RmAleaf).Id.PId;
  RspAleaf.Id.AssIdRmt=(*RmAleaf).Id.AssIdRmt;
  RspAleaf.Id.PIdRmt   =(*RmAleaf).Id.PIdRmt;
  /*MmsPdu de Respuesta de iniciacion*/
  RspAleaf.MmsPdu.designator=MMSpdu_initiate_ResponsePDU;
  /*Obtengo la negociacion de parametros*/
  if( (*RmAleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedMaxSegmentSize >
LocalMaxSegmentSize)
      RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedMaxSegmentSize=LocalMaxSegmentSize;
```

```
  else

RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedMaxSegmentSize=(*RmAleaf).MmsPdu.u.MMSpdu
_initiate_RequestPDU.proposedMaxSegmentSize;
  if( (*RmAleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedMaxSegmentSize_present >
LocalMaxSegmentSize_present )

RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedMaxSegmentSize_present=LocalMaxSegmentSiz
e_present;
  else

RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedMaxSegmentSize_present=(*RmAleaf).MmsPdu.
u.MMSpdu_initiate_RequestPDU.proposedMaxSegmentSize_present;
  if( (*RmAleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedMaxServOutstandingCalling >
LocalMaxServOutstandingCalling )

RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedMaxServOutstandingCalling=LocalMaxServOut
standingCalling;
  else

RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedMaxServOutstandingCalling=(*RmAleaf).MmsP
du.u.MMSpdu_initiate_RequestPDU.proposedMaxServOutstandingCalling;
  if((*RmAleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedMaxServOutstandingCalled >
LocalMaxServOutstandingCalled)

RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedMaxServOutstandingCalled=LocalMaxServOuts
tandingCalled;
  else

RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedMaxServOutstandingCalled=(*RmAleaf).MmsPd
u.u.MMSpdu_initiate_RequestPDU.proposedMaxServOutstandingCalled;

if((*RmAleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedDataStructureNestingLevel>LocalDataStru
ctureNestingLevel)

RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedDataStructureNestingLevel=LocalDataStruct
ureNestingLevel;
  else

RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedDataStructureNestingLevel=(*RmAleaf).MmsP
du.u.MMSpdu_initiate_RequestPDU.proposedDataStructureNestingLevel;

if((*RmAleaf).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedDataStructureNestingLevel_present>Local
DataStructureNestingLevel_present)

RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedDataStructureNestingLevel_present=LocalDa
taStructureNestingLevel_present;
  else

RspAleaf.MmsPdu.u.MMSpdu_initiate_ResponsePDU.negociatedDataStructureNestingLevel_present=(*RmAle
af).MmsPdu.u.MMSpdu_initiate_RequestPDU.proposedDataStructureNestingLevel_present;
  (*RmAleaf)=RspAleaf;
}

/***                                              ***/
/***       MMSPARAM.H          ***/
/***                                              ***/
#define LocalMaxSegmentSize 310
#define LocalMaxSegmentSize_present 1
#define LocalMaxServOutstandingCalling 311
#define LocalMaxServOutstandingCalled 312
#define LocalDataStructureNestingLevel 313
#define LocalDataStructureNestingLevel_present 1
```

11.6 FEEDER.H

```
/*****************************************************************************/
/**********                        Proceso Feeder                   **********/
/**********                                                         **********/
void feeder_proc(Process * p,Channel *c)
```

```
{
FILE *FComandos;
char sComandos[9];
tTrama sCmd,sTempo;
tStackTrama sCmdBuff;
int i,j,iComando=0;
int nTop,nCountAc,nCount;
   p = p;
   printf("\nArchivo de Comandos del Cliente: ");
   gets(sComandos);
   if ( (FComandos=fopen(sComandos,"r")) == NULL )
   {
      printf("\nImposible Abrir Archivo: %s",sComandos);
      ChanOut(c,sFIN,TRAMA_LEN);
      exit(EXIT_FAILURE);
   }
   fseek(FComandos,0,SEEK_SET);
   while(!feof(FComandos))
   {
     nCountAc=0;
     while(nCountAc<TRAMA_LEN){
       if(fscanf(FComandos,"%s%n",sTempo,&nCount)<=0){
         nCountAc=-1;
         nCount=0;
         break;
       }
       else{
         if(nCountAc+nCount<TRAMA_LEN)
           nTop=nCountAc+nCount;
         else
           nTop=TRAMA_LEN;
         for(j=nCountAc;j<nTop;j++){
           sCmd[j]=sTempo[j-nCountAc];
         }
         nCountAc=nCountAc+nCount;
       }
     }
     if (nCountAc<0)
       break;
     sCmd[TRAMA_LEN-1]='\0';
     if (iComando<MAX_COMMAND){
       Mstrcpy(sCmdBuff[iComando],sCmd);
       printf(" \nComando  es :");
       for(j=0;j<TRAMA_LEN;j++){
         printf("%c",sCmdBuff[iComando][j]);
       }
     }
     if (iComando==MAX_COMMAND){
        printf("\nMax.(%d).Num.Cmds.Alcanzados!",iComando);
        break;
     }
      iComando++;
   }
   fclose(FComandos);
   printf("Archivo Cerrado\n");
   for (i=0;i<iComando;i++){
     sCmdBuff[iComando][TRAMA_LEN-1]='\0';
     Mstrcpy(sCmd,sCmdBuff[i]);
     ChanOut(c,sCmd,TRAMA_LEN);
   }
   ChanOut(c,sFIN,TRAMA_LEN);
}
```

11.7 TRXBUFF.H

```
TxTrama Buffer[LEN_CHAN*3];
/*****************************************************************************/
/**********                          Proceso Rx                      **********/
/**********     Enacargado de emular un canal de longitud n          **********/
void RxProc (Process * p,Channel *FrCanal1,int TrxNumb,int *bKeepGoing,int imas){
int index,offset;
```

```
tTrama sCmd;
aleaf SMAleaf;
int count=0;

  p = p;
  offset=(TrxNumb-1)*LEN_CHAN;
  index=offset;
  while ((*bKeepGoing)){
    if(Buffer[index].Usado==TRUE){
      ProcWait(TRX_WTIME);
    }
    else{
      ChanIn(FrCanal1,sCmd,TRAMA_LEN);
      count++;
      if (strncmp(sFIN,sCmd,3)==0){
          Mstrcpy(Buffer[index].Trama,sCmd);
          Buffer[index].Usado=TRUE;
          (*bKeepGoing)=FALSE;
      }
      else{
        UnpackTrama(&SMAleaf,sCmd);
        SMAleaf.Id.PId=SMAleaf.Id.PId+imas;
        PackTrama(&SMAleaf,sCmd);
        Mstrcpy(Buffer[index].Trama,sCmd);
        Buffer[index].Usado=TRUE;
        index=(index-offset+1)%LEN_CHAN;
        index=index+offset;
      }
    }
  }
  return;
}

/*****************************************************************************/
/**********                          Proceso Tx                     **********/
/**********     Enacargado de emular un canal de longitud n         **********/
void TxProc (Process * p,Channel *ToCanal2,int TrxNumb,int *bKeepGoing,int imas){
int index,offset;
tTrama sCmd;
aleaf SMAleaf;

  p = p;
  offset=(TrxNumb-1)*LEN_CHAN;
  index=offset;
  while (*bKeepGoing){
    if(Buffer[index].Usado==TRUE){
      Mstrcpy(sCmd,Buffer[index].Trama);
      UnpackTrama(&SMAleaf,sCmd);
      if (strncmp(sFIN,sCmd,3)==0){
        break;
      }
      SMAleaf.Id.PId=SMAleaf.Id.PId+imas;
      PackTrama(&SMAleaf,sCmd);
/*      printf("TX1:%d",imas);*/
      ChanOut(ToCanal2,sCmd,TRAMA_LEN);
      Buffer[index].Usado=FALSE;
      index=(index-offset+1)%LEN_CHAN;
      index=index+offset;
    }
    else{
      ProcWait(TRX_WTIME*4);
    }
  }
  while (Buffer[index].Usado==TRUE){
      Mstrcpy(sCmd,Buffer[index].Trama);
      if (strncmp(sFIN,sCmd,3)==0){
        break;
      }
      else{
        ChanOut(ToCanal2,sCmd,TRAMA_LEN);
        Buffer[index].Usado=FALSE;
        index=(index-offset+1)%LEN_CHAN;
        index=index+offset;
      }
```

```
/*      printf("TX2:%d",imas);*/
  }
  return;
}
```

11.8 MMSDECOD.H

```
void GetInfo(aleaf *pdu,char *Info){
   switch((*pdu).MmsPdu.designator){
     case MMSpdu_initiate_RequestPDU:
       strcpy(Info,"IniReqPDU");
       break;
     case MMSpdu_initiate_ResponsePDU:
       strcpy(Info,"IniResPDU");
       break;
     case MMSpdu_initiate_ErrorPDU:
       strcpy(Info,"IniErrPDU");
       break;
     case MMSpdu_conclude_RequestPDU:
       strcpy(Info,"ConReqPDU");
       break;
     case MMSpdu_conclude_ResponsePDU:
       strcpy(Info,"ConResPDU");
       break;
     case MMSpdu_conclude_ErrorPDU:
       strcpy(Info,"ConErrPDU");
       break;
     case MMSpdu_confirmed_RequestPDU:
       strcpy(Info,"CnfReqPDU");
       break;
     case MMSpdu_confirmed_ResponsePDU:
       strcpy(Info,"CnfResPDU");
       break;
     case MMSpdu_confirmed_ErrorPDU:
       strcpy(Info,"CnfErrPDU");
       break;
     case MMSpdu_Unconfirmed_RequestPDU:
       strcpy(Info,"UcfReqPDU");
       break;
     case MMSpdu_reject_PDU:
       strcpy(Info,"Reject_PDU");
       break;
     default:
       strcpy(Info,"Not Info");
       break;
   }
}

void MMSDecod(char *trama,int nEntidad){
int i;
char Info[255];
char Node[7];
union {
aleaf PduSw;
char TramSw[TRAMA_LEN];
}P;
  for(i=0;i<TRAMA_LEN;i++){
    P.TramSw[i]=trama[i];
  }
  return;
  switch(nEntidad){
    case 0:
      strcpy(Node,"Server");
      break;
    case 1:
      strcpy(Node,"Client");
      break;
  }
  GetInfo(&P.PduSw,Info);
  switch(P.PduSw.Primitive){
```

```
    case DUMMY_Primitive:
      printf("\n%s: DUMMY_Primitive\tId= %d\tRId=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,Info);
      break;
    case LOGIN_Request:
      printf("\n%s: LOGIN_Request\tId= %d\tRId=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,Info);
      break;
    case LOGIN_Indication:
      printf("\n%s: LOGIN_Indication\tId= %d\tRId=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,Info);
      break;
    case LOGIN_Confirm:
      printf("\n%s: LOGIN_Confirm\tId= %d\tRId=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,Info);
      break;
    case LOGIN_ConfirmErr:
      printf("\n%s: LOGIN_ConfirmErr\tId= %d\tRId=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,Info);
      break;
    case LOGOUT_Request:
      printf("\n%s: LOGOUT_Request\tId= %d\tRId=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,Info);
      break;
    case LOGOUT_Indication:
      printf("\n%s: LOGOUT_Indication\tId= %d\tRId=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,Info);
      break;
    case LOGOUT_Confirm:
      printf("\n%s: LOGOUT_Confirm\tId= %d\tRId=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,Info);
      break;
    case A_ASSOCIATE_request:
      printf("\n%s: A_ASSOCIATE_request\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    case A_ASSOCIATE_indication:
      printf("\n%s: A_ASSOCIATE_indication\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    case A_ASSOCIATE_response:
      printf("\n%s: A_ASSOCIATE_response\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    case A_ASSOCIATE_confirm:
      printf("\n%s: A_ASSOCIATE_confirm\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    case A_RELEASE_request:
      printf("\n%s: A_RELEASE_request\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    case A_RELEASE_indication:
      printf("\n%s: A_RELEASE_indication\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    case A_RELEASE_response:
      printf("\n%s: A_RELEASE_response\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    case A_RELEASE_confirm:
      printf("\n%s: A_RELEASE_confirm\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    case A_MMS_Request:
```

```
      printf("\n%s: A_MMS_Request\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    case A_MMS_Indication:
      printf("\n%s: A_MMS_Indication\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    case A_MMS_Response:
      printf("\n%s: A_MMS_Response\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    case A_MMS_Confirm:
      printf("\n%s: A_MMS_Confirm\tId= %d\tRId= %d\tAs= %d\tRAs=
%d\n\t%s",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt,Info);
      break;
    default:
      printf("\n%s: Unknown\tId= %d\tRId= %d",Node,P.PduSw.Id.PId,P.PduSw.Id.PIdRmt);
      break;
  }
}

void MMSDecod2(char *trama,char *TramaOut,char *TramaOut2){
int i;
char Info[255];
char Node[7];
char TramaTmp[255];
union {
aleaf PduSw;
char TramSw[TRAMA_LEN];
}P;
  TramaOut[0]='\0';
  TramaOut2[0]='\0';
  TramaTmp[0]='\0';
  for(i=1;i<TRAMA_LEN;i++){
    TramaTmp[i]=' ';
  }
  for(i=0;i<TRAMA_LEN;i++){
    P.TramSw[i]=trama[i];
  }
  GetInfo(&P.PduSw,Info);
  switch(P.PduSw.Primitive){
    case DUMMY_Primitive:
      sprintf(TramaTmp," DUMMY_Primitive\tId= %d\tRId= %d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt);
      break;
    case LOGIN_Request:
      sprintf(TramaTmp," LOGIN_Request\tId= %d\tRId= %d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt);
      break;
    case LOGIN_Indication:
      sprintf(TramaTmp," LOGIN_Indication\tId= %d\tRId= %d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt);
      break;
    case LOGIN_Confirm:
      sprintf(TramaTmp," LOGIN_Confirm\tId= %d\tRId= %d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt);
      break;
    case LOGIN_ConfirmErr:
      sprintf(TramaTmp," LOGIN_ConfirmErr\tId= %d\tRId= %d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt);
      break;
    case LOGOUT_Request:
      sprintf(TramaTmp," LOGOUT_Request\tId= %d\tRId= %d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt);
      break;
    case LOGOUT_Indication:
      sprintf(TramaTmp," LOGOUT_Indication\tId= %d\tRId= %d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt);
      break;
    case LOGOUT_Confirm:
      sprintf(TramaTmp," LOGOUT_Confirm\tId= %d\tRId= %d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt);
      break;
    case A_ASSOCIATE_request:
      sprintf(TramaTmp," A_ASSOCIATE_request\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
```

```
    case A_ASSOCIATE_indication:
      sprintf(TramaTmp," A_ASSOCIATE_indication\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
    case A_ASSOCIATE_response:
      sprintf(TramaTmp," A_ASSOCIATE_response\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
    case A_ASSOCIATE_confirm:
      sprintf(TramaTmp," A_ASSOCIATE_confirm\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
    case A_RELEASE_request:
      sprintf(TramaTmp," A_RELEASE_request\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
    case A_RELEASE_indication:
      sprintf(TramaTmp," A_RELEASE_indication\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
    case A_RELEASE_response:
      sprintf(TramaTmp," A_RELEASE_response\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
    case A_RELEASE_confirm:
      sprintf(TramaTmp," A_RELEASE_confirm\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
    case A_MMS_Request:
      sprintf(TramaTmp," A_MMS_Request\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
    case A_MMS_Indication:
      sprintf(TramaTmp," A_MMS_Indication\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
    case A_MMS_Response:
      sprintf(TramaTmp," A_MMS_Response\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
    case A_MMS_Confirm:
      sprintf(TramaTmp," A_MMS_Confirm\tId= %d\tRId= %d\tAs= %d\tRAs=
%d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt,
            P.PduSw.Id.AssId,P.PduSw.Id.AssIdRmt);
      break;
    default:
      sprintf(TramaTmp," Unknown\tId= %d\tRId= %d",P.PduSw.Id.PId,P.PduSw.Id.PIdRmt);
      break;
  }
  TramaTmp[60]='\0';
  strcpy(TramaOut,TramaTmp);
  strcpy(TramaOut2,Info);
}
```

11.9 MMSPDUTR.H

```
void SendMonitor(Channel *ToMon,char *trama,char *cFrProc,char *cValido){
  trama[TRAMA_LEN-3]=cFrProc[0];
  trama[TRAMA_LEN-2]=cValido[0];
  ChanOut(ToMon,trama,TRAMA_LEN);
```

```
void GetTx(char *trama,char *sRx,char *sVal){
  sRx[0]='\0';
  sVal[0]='\0';
  if(trama[TRAMA_LEN-3]==FROMSERVER[0]){
    strcpy(sRx,"SR");
  }
  else if(trama[TRAMA_LEN-3]==FROMSMMSPM[0]){
    strcpy(sRx,"SM");
  }
  else if(trama[TRAMA_LEN-3]==FROMCLIENT[0]){
    strcpy(sRx,"CL");
  }
  else if(trama[TRAMA_LEN-3]==FROMCMMSPM[0]){
    strcpy(sRx,"CM");
  }
  else {
    strcpy(sRx,"??");
  }
  if(trama[TRAMA_LEN-2]==VALIDO[0]){
    strcpy(sVal,"+");
  }
  else if(trama[TRAMA_LEN-2]==INVALIDO[0]){
    strcpy(sVal,"-");
  }
  else {
    strcpy(sVal," ");
  }
}

void Mstrcpy(char *tramaIn,char *tramaFrom){
int j;
  for(j=0;j<TRAMA_LEN;j++){
    tramaIn[j]=tramaFrom[j];
  }
}

void ClrPdu(aleaf *pdu){
int i;
union {
aleaf PduSw;
char TramSw[TRAMA_LEN];
}PduSwapTra;
  for(i=0;i<TRAMA_LEN-1;i++){
    PduSwapTra.TramSw[i]=' ';
  }
  PduSwapTra.TramSw[TRAMA_LEN-1]='\0';
  *pdu=PduSwapTra.PduSw;

}

void PackTrama(aleaf *pdu,char *trama){
int i;
union {
aleaf PduSw;
char TramSw[TRAMA_LEN];
}PduSwapTra;
  PduSwapTra.PduSw=*pdu;
  for(i=0;i<TRAMA_LEN-1;i++){
    trama[i]=PduSwapTra.TramSw[i];
  }
  trama[TRAMA_LEN-1]='\0';
}
void UnpackTrama(aleaf *pdu,char *trama){
int i;
union {
aleaf PduSw;
char TramSw[TRAMA_LEN];
}PduSwapTra;
  for(i=0;i<TRAMA_LEN;i++){
    PduSwapTra.TramSw[i]=trama[i];
  }
  *pdu=PduSwapTra.PduSw;
```

11.10 MMSPDU.H

```
typedef enum { ObjectName_vmd_specific, ObjectName_domain_specific  } ObjectName_Choice;

typedef enum { VariableSpecification_name  } VariableSpecification_Choice;

typedef enum { VariableAccessSpecification_listOfVariable,
VariableAccessSpecification_variableListName  } VariableAccessSpecification_Choice;


typedef enum { Values_Write_Response_failure, Values_Write_Response_success  }
Values_Write_Response_Choice;

typedef enum { Data_booleant, Data_integert, Data_unsigned, Data_octet_string,
Data_visible_string  } Data_Choice;

typedef enum { AccessResult_failure, AccessResult_success  } AccessResult_Choice;

typedef enum { RejectPDU_rejectReason_confirmed_requestPDU,
RejectPDU_rejectReason_confirmed_responsePDU, RejectPDU_rejectReason_confirmed_errorPDU,
RejectPDU_rejectReason_pdu_error, RejectPDU_rejectReason_conclude_requestPDU,
RejectPDU_rejectReason_conclude_responsePDU, RejectPDU_rejectReason_conclude_errorPDU,
RejectPDU_rejectReason_initiate_requestPDU  } RejectPDU_rejectReason_Choice;


typedef enum { ServiceError_errorClass_t_vmd_state,
ServiceError_errorClass_t_application_reference, ServiceError_errorClass_t_definition,
ServiceError_errorClass_t_resource, ServiceError_errorClass_t_service,
ServiceError_errorClass_t_service_preempt, ServiceError_errorClass_t_time_resolution,
ServiceError_errorClass_t_accesst, ServiceError_errorClass_t_initiate,
ServiceError_errorClass_t_conclude, ServiceError_errorClass_t_otherst  }
ServiceError_errorClass_t_Choice;



typedef enum { ConfirmedServiceResponse_read, ConfirmedServiceResponse_write,
ConfirmedServiceResponse_initiateDownloadSequence, ConfirmedServiceResponse_downloadSegment,
ConfirmedServiceResponse_terminateDownloadSequence, ConfirmedServiceResponse_start,
ConfirmedServiceResponse_stop  } ConfirmedServiceResponse_Choice;

typedef enum { ConfirmedServiceRequest_read, ConfirmedServiceRequest_write,
ConfirmedServiceRequest_initiateDownloadSequence, ConfirmedServiceRequest_downloadSegment,
ConfirmedServiceRequest_terminateDownloadSequence, ConfirmedServiceRequest_start,
ConfirmedServiceRequest_stop  } ConfirmedServiceRequest_Choice;

typedef enum { MMSpdu_Unconfirmed_RequestPDU, MMSpdu_confirmed_RequestPDU,
MMSpdu_confirmed_ResponsePDU, MMSpdu_confirmed_ErrorPDU, MMSpdu_reject_PDU,
MMSpdu_initiate_RequestPDU, MMSpdu_initiate_ResponsePDU, MMSpdu_initiate_ErrorPDU,
MMSpdu_conclude_RequestPDU, MMSpdu_conclude_ResponsePDU, MMSpdu_conclude_ErrorPDU  }
MMSpdu_Choice;

typedef struct {
    ObjectName_Choice designator;
    union {
        Identifier ObjectName_vmd_specific;
        struct {
            Identifier domainId;
            Identifier itemId;
        } ObjectName_domain_specific;
    } u;
}  ObjectName;

typedef Null  Stop_Response;

typedef struct {
    Identifier programInvocationName;
}  Stop_Request;
```

```
typedef Null  Start_Response;

typedef struct {
    Identifier programInvocationName;
}  Start_Request;

typedef struct {
    VariableSpecification_Choice designator;
    union {
        ObjectName VariableSpecification_name;
    } u;
}  VariableSpecification;

typedef struct {
    VariableSpecification variableSpecificatn;
}  Values_VariableAccessSpecification_listOfVariable_t;

typedef struct {
    VariableAccessSpecification_Choice designator;
    union {
        struct {
            unsignedlong length;
            Values_VariableAccessSpecification_listOfVariable_t data[max_listOfVariable];
         }  VariableAccessSpecification_listOfVariable;
        ObjectName VariableAccessSpecification_variableListName;
    } u;
}  VariableAccessSpecification;

typedef struct {
    Boolean specificationWithResult;
    VariableAccessSpecification variableAccessSpecificatn;
}  Read_Request;

typedef Integer  DataAccessError;

typedef struct {
    Values_Write_Response_Choice designator;
    union {
        DataAccessError Values_Write_Response_failure;
        Null Values_Write_Response_success;
    } u;
}  Values_Write_Response;

typedef struct {
    unsignedlong length;
    Values_Write_Response *data;
 }   Write_Response;

typedef struct {
    Data_Choice designator;
    union {
        Boolean Data_booleant;
        Integer Data_integert;
        Integer Data_unsigned;
        OctetString Data_octet_string;
        VisibleString Data_visible_string;
    } u;
}  Data;

typedef struct {
    VariableAccessSpecification variableAccessSpecificatn;
    struct {
        unsignedlong length;
        Data *data;
     }  listOfData;
}  Write_Request;

typedef struct {
    AccessResult_Choice designator;
    union {
        DataAccessError AccessResult_failure;
        Data AccessResult_success;
    } u;
}  AccessResult;
```

```
typedef struct {
    VariableAccessSpecification variableAccessSpecificatn;
    int variableAccessSpecificatn_present;
    struct {
        unsignedlong length;
        AccessResult *data;
     } listOfAccessResult;
} Read_Response;

typedef Null  TerminateDownloadSequence_Response;

typedef struct {
    OctetString loadData;
    Boolean moreFollows;
} DownloadSegment_Response;

typedef Null  InitiateDownloadSequence_Response;

typedef struct {
    Identifier domainName;
    Boolean discard;
} TerminateDownloadSequence_Request;

typedef Identifier  DownloadSegment_Request;

typedef struct {
    Identifier domainName;
    struct {
        unsignedlong length;
        VisibleString *data;
     } listOfCapabilities;
    Boolean sharable;
} InitiateDownloadSequence_Request;

typedef struct {
    Unsigned32 originalInvokeID;
    int originalInvokeID_present;
    struct {
        RejectPDU_rejectReason_Choice designator;
        union {
            Integer RejectPDU_rejectReason_confirmed_requestPDU;
            Integer RejectPDU_rejectReason_confirmed_responsePDU;
            Integer RejectPDU_rejectReason_confirmed_errorPDU;
            Integer RejectPDU_rejectReason_pdu_error;
            Integer RejectPDU_rejectReason_conclude_requestPDU;
            Integer RejectPDU_rejectReason_conclude_responsePDU;
            Integer RejectPDU_rejectReason_conclude_errorPDU;
            Integer RejectPDU_rejectReason_initiate_requestPDU;
        } u;
    } rejectReason;
} RejectPDU;

typedef struct {
    struct {
        ServiceError_errorClass_t_Choice designator;
        union {
            Integer ServiceError_errorClass_t_vmd_state;
            Integer ServiceError_errorClass_t_application_reference;
            Integer ServiceError_errorClass_t_definition;
            Integer ServiceError_errorClass_t_resource;
            Integer ServiceError_errorClass_t_service;
            Integer ServiceError_errorClass_t_service_preempt;
            Integer ServiceError_errorClass_t_time_resolution;
            Integer ServiceError_errorClass_t_accesst;
            Integer ServiceError_errorClass_t_initiate;
            Integer ServiceError_errorClass_t_conclude;
            Integer ServiceError_errorClass_t_otherst;
        } u;
    } errorClass;
    Integer additionalCode;
    int additionalCode_present;
    VisibleString additionalDescription;
    int additionalDescription_present;
```

```
} ServiceError;

typedef ServiceError  Conclude_ErrorPDU;

typedef Null  Conclude_ResponsePDU;

typedef Null  Conclude_RequestPDU;

typedef ServiceError  Initiate_ErrorPDU;

typedef struct {
    Integer32 negociatedMaxSegmentSize;
    int negociatedMaxSegmentSize_present;
    Integer16 negociatedMaxServOutstandingCalling;
    Integer16 negociatedMaxServOutstandingCalled;
    Integer8 negociatedDataStructureNestingLevel;
    int negociatedDataStructureNestingLevel_present;
} Initiate_ResponsePDU;

typedef struct {
    Integer32 proposedMaxSegmentSize;
    int proposedMaxSegmentSize_present;
    Integer16 proposedMaxServOutstandingCalling;
    Integer16 proposedMaxServOutstandingCalled;
    Integer8 proposedDataStructureNestingLevel;
    int proposedDataStructureNestingLevel_present;
} Initiate_RequestPDU;

typedef struct {
    ConfirmedServiceResponse_Choice designator;
    union {
        Read_Response ConfirmedServiceResponse_read;
        Write_Response ConfirmedServiceResponse_write;
        InitiateDownloadSequence_Response ConfirmedServiceResponse_initiateDownloadSequence;
        DownloadSegment_Response ConfirmedServiceResponse_downloadSegment;
        TerminateDownloadSequence_Response ConfirmedServiceResponse_terminateDownloadSequence;
        Start_Response ConfirmedServiceResponse_start;
        Stop_Response ConfirmedServiceResponse_stop;
    } u;
} ConfirmedServiceResponse;

typedef struct {
    ConfirmedServiceRequest_Choice designator;
    union {
        Read_Request ConfirmedServiceRequest_read;
        Write_Request ConfirmedServiceRequest_write;
        InitiateDownloadSequence_Request ConfirmedServiceRequest_initiateDownloadSequence;
        DownloadSegment_Request ConfirmedServiceRequest_downloadSegment;
        TerminateDownloadSequence_Request ConfirmedServiceRequest_terminateDownloadSequence;
        Start_Request ConfirmedServiceRequest_start;
        Stop_Request ConfirmedServiceRequest_stop;
    } u;
} ConfirmedServiceRequest;

typedef struct {
    Unsigned32 invokeID;
    ServiceError serviceError;
} Confirmed_ErrorPDU;

typedef struct {
    Unsigned32 invokeID;
    ConfirmedServiceResponse field_2;
} Confirmed_ResponsePDU;

typedef struct {
    Unsigned32 invokeID;
    ConfirmedServiceRequest field_2;
} Confirmed_RequestPDU;

typedef struct {
    MMSpdu_Choice designator;
    union {
        Confirmed_RequestPDU MMSpdu_confirmed_RequestPDU;
        Confirmed_ResponsePDU MMSpdu_confirmed_ResponsePDU;
```

```
        Confirmed_ErrorPDU MMSpdu_confirmed_ErrorPDU;
        RejectPDU MMSpdu_reject_PDU;
        Initiate_RequestPDU MMSpdu_initiate_RequestPDU;
        Initiate_ResponsePDU MMSpdu_initiate_ResponsePDU;
        Initiate_ErrorPDU MMSpdu_initiate_ErrorPDU;
        Conclude_RequestPDU MMSpdu_conclude_RequestPDU;
        Conclude_ResponsePDU MMSpdu_conclude_ResponsePDU;
        Conclude_ErrorPDU MMSpdu_conclude_ErrorPDU;
    } u;
} MMSpdu;

#endif
```

11.11 MMSTYPES.H

```
/*BEGIN MY TYPES*/
typedef char tTrama[TRAMA_LEN];                 /*Comando eventualmente MMS
typedef tTrama tStackTrama[MAX_COMMAND];        /*Stack de comandos
*/
typedef struct {
    int                 Usado;
    tTrama              Trama;
}TxTrama;
typedef TxTrama TxBuffer[LEN_CHAN];             /*Stack de comandos
*/
/*END MY TYPES*/


/*BEGIN system types */
#define Boolean_c 4       /*Cuantos caracteres equivale el tipo*/
#define Integer_c 4
#define num_c 1
#define Null_c 1
#define VisibleString_c 8
#define OctetString_c 8
#define max_listOfVariable 1

typedef int Boolean;      /*tipos no naturales al sys*/
typedef int Integer;
typedef char Null;
typedef char *VisibleString;
typedef char *OctetString;
typedef int unsignedlong;
/*END system types */

/*BEGIN MMS TYPES*/
typedef Integer  Unsigned32;
typedef Integer  Integer32;
typedef Integer  Integer16;
typedef Integer  Integer8;
typedef VisibleString  Identifier;

/*END MMS TYPES*/
```

11.12 MMSCTES.H

```
/* BEGIN MY CONSTANTS*/
/**********OJOOJOOJOOJOOJOOJOOJOOJOOJO:cambiar en ECHO.occ LA CTE CORRES*/
#define TRAMA_LEN 50            /*Longitud de la Trama a tx               */
#define FALSE 0                  /*boleano falso                           */
#define TRUE 1                   /*boleano verdero                         */
#define MAX_COMMAND 15           /*Numero máximo de comando a tx           */
#define LEN_CHAN 3               /*Longitud de Canal                       */
#define TRX_WTIME 2              /*Tiempo en ticks de espera en la emula-  */
                                 /*cion de canal de longitud LEN_CHAN      */
```

```
#define CENT_WTIME 20*TRX_WTIME  /*Tiempo en ticks de espera la espera del */
                                 /*proceso centinela                       */
#define sFIN "FIN"               /*Mensaje para teminar la Tx              */
#define FROMSERVER  "S"                          /**/
#define FROMSMMSPM  "M"                          /**/
#define FROMCLIENT  "C"                          /**/
#define FROMCMMSPM  "P"                          /**/
#define VALIDO      "V"                             /**/
#define INVALIDO    "I"                             /**/

/* END MY CONSTANTS*/

/* BEGIN MMS CONSTANTS*/
/* Constants for type DataAccessError */
#define DataAccessError__object_invalidated 0
#define DataAccessError__hardware_fault 1
#define DataAccessError__temporarly_unavailable 2
#define DataAccessError__object_access_denied 3
#define DataAccessError__object_undefined 4
#define DataAccessError__invalid_address 5
#define DataAccessError__type_unsupported 6
        #define DataAccessError__type_inconsistent 7
#define DataAccessError__object_attribute_inconsistent 8
#define DataAccessError__object_access_unsupported 9
#define DataAccessError__object_non_existent 10

/* Constants for type RejectPDU_rejectReason_confirmed_requestPDU_t */
#define RejectPDU_rejectReason_confirmed_requestPDU_t__other 0
#define RejectPDU_rejectReason_confirmed_requestPDU_t__unrecognized_service 1
#define RejectPDU_rejectReason_confirmed_requestPDU_t__unrecognized_modifier 2
#define RejectPDU_rejectReason_confirmed_requestPDU_t__invalid_invokeID 3
#define RejectPDU_rejectReason_confirmed_requestPDU_t__invalid_argument 4
#define RejectPDU_rejectReason_confirmed_requestPDU_t__invalid_modifier 5
#define RejectPDU_rejectReason_confirmed_requestPDU_t__max_serv_outstanding_exceeded 6
#define RejectPDU_rejectReason_confirmed_requestPDU_t__max_segment_length_exceeded 7
#define RejectPDU_rejectReason_confirmed_requestPDU_t__max_recursion_exceeded 8
#define RejectPDU_rejectReason_confirmed_requestPDU_t__value_out_of_range 9
#define RejectPDU_rejectReason_confirmed_requestPDU_t__noassociation 10

/* Constants for type RejectPDU_rejectReason_confirmed_responsePDU_t */
#define RejectPDU_rejectReason_confirmed_responsePDU_t__other 0
#define RejectPDU_rejectReason_confirmed_responsePDU_t__unrecognized_service 1
#define RejectPDU_rejectReason_confirmed_responsePDU_t__invalid_invokeID 2
#define RejectPDU_rejectReason_confirmed_responsePDU_t__invalid_result 3
#define RejectPDU_rejectReason_confirmed_responsePDU_t__max_segment_length_exceeded 4
#define RejectPDU_rejectReason_confirmed_responsePDU_t__max_recursion_exceeded 5
#define RejectPDU_rejectReason_confirmed_responsePDU_t__value_out_of_range 6

/* Constants for type RejectPDU_rejectReason_confirmed_errorPDU_t */
#define RejectPDU_rejectReason_confirmed_errorPDU_t__other 0
#define RejectPDU_rejectReason_confirmed_errorPDU_t__unrecognized_service 1
#define RejectPDU_rejectReason_confirmed_errorPDU_t__invalid_invokeID 2
#define RejectPDU_rejectReason_confirmed_errorPDU_t__invalid_serviceError 3
#define RejectPDU_rejectReason_confirmed_errorPDU_t__value_out_of_range 4

/* Constants for type RejectPDU_rejectReason_pdu_error_t */
#define RejectPDU_rejectReason_pdu_error_t__unknown_pdu_type 0
#define RejectPDU_rejectReason_pdu_error_t__invalid_pdu 1

/* Constants for type RejectPDU_rejectReason_conclude_requestPDU_t */
#define RejectPDU_rejectReason_conclude_requestPDU_t__other 0
#define RejectPDU_rejectReason_conclude_requestPDU_t__invalid_argument 1

/* Constants for type RejectPDU_rejectReason_conclude_responsePDU_t */
#define RejectPDU_rejectReason_conclude_responsePDU_t__other 0
#define RejectPDU_rejectReason_conclude_responsePDU_t__invalid_result 1

/* Constants for type RejectPDU_rejectReason_conclude_errorPDU_t */
#define RejectPDU_rejectReason_conclude_errorPDU_t__other 0
#define RejectPDU_rejectReason_conclude_errorPDU_t__invalid_serviceError 1
#define RejectPDU_rejectReason_conclude_errorPDU_t__value_out_of_range 2

/* Constants for type RejectPDU_rejectReason_initiate_requestPDU_t */
#define RejectPDU_rejectReason_initiate_requestPDU_t__other 0
```

```
#define RejectPDU_rejectReason_initiate_requestPDU_t__erroneousinitiate 1

/* Constants for type ServiceError_errorClass_t_vmd_state_t */
#define ServiceError_errorClass_t_vmd_state_t__other 0
#define ServiceError_errorClass_t_vmd_state_t__vmd_state_conflict 1
#define ServiceError_errorClass_t_vmd_state_t__vmd_operational_problem 2
#define ServiceError_errorClass_t_vmd_state_t__domain_transfer_problem 3
#define ServiceError_errorClass_t_vmd_state_t__state_machine_id_invalid 4

/* Constants for type ServiceError_errorClass_t_application_reference_t */
#define ServiceError_errorClass_t_application_reference_t__other 0
#define ServiceError_errorClass_t_application_reference_t__aplication_unreachable 1
#define ServiceError_errorClass_t_application_reference_t__connection_lost 2
#define ServiceError_errorClass_t_application_reference_t__application_reference_invalid 3
#define ServiceError_errorClass_t_application_reference_t__context_unsupported 4

/* Constants for type ServiceError_errorClass_t_definition_t */
#define ServiceError_errorClass_t_definition_t__other 0
#define ServiceError_errorClass_t_definition_t__object_undefined 1
#define ServiceError_errorClass_t_definition_t__invalid_address 2
#define ServiceError_errorClass_t_definition_t__type_unsupported 3
#define ServiceError_errorClass_t_definition_t__type_inconsistent 4
#define ServiceError_errorClass_t_definition_t__object_exists 5
#define ServiceError_errorClass_t_definition_t__object_attribute_inconsistent 6

/* Constants for type ServiceError_errorClass_t_resource_t */
#define ServiceError_errorClass_t_resource_t__other 0
#define ServiceError_errorClass_t_resource_t__memory_unavailable 1
#define ServiceError_errorClass_t_resource_t__processor_resource_unavailable 2
#define ServiceError_errorClass_t_resource_t__mass_storage_unavailable 3
#define ServiceError_errorClass_t_resource_t__capability_unavailable 4
#define ServiceError_errorClass_t_resource_t__capability_unknown 5

/* Constants for type ServiceError_errorClass_t_service_t */
#define ServiceError_errorClass_t_service_t__other 0
#define ServiceError_errorClass_t_service_t__primitives_out_of_sequence 1
#define ServiceError_errorClass_t_service_t__object_sate_conflict 2
#define ServiceError_errorClass_t_service_t__pdu_size 3
#define ServiceError_errorClass_t_service_t__continuation_invalid 4
#define ServiceError_errorClass_t_service_t__object_constraint_conflict 5

/* Constants for type ServiceError_errorClass_t_service_preempt_t */
#define ServiceError_errorClass_t_service_preempt_t__other 0
#define ServiceError_errorClass_t_service_preempt_t__timeout 1
#define ServiceError_errorClass_t_service_preempt_t__deadlock 2
#define ServiceError_errorClass_t_service_preempt_t__cancel 3

/* Constants for type ServiceError_errorClass_t_time_resolution_t */
#define ServiceError_errorClass_t_time_resolution_t__other 0
#define ServiceError_errorClass_t_time_resolution_t__unsupportable_time_resolution 1

/* Constants for type ServiceError_errorClass_t_accesst_t */
#define ServiceError_errorClass_t_accesst_t__other 0
#define ServiceError_errorClass_t_accesst_t__object_access_unsupported 1
#define ServiceError_errorClass_t_accesst_t__object_non_existent 2
#define ServiceError_errorClass_t_accesst_t__object_access_denied 3
#define ServiceError_errorClass_t_accesst_t__object_invalidated 4

/* Constants for type ServiceError_errorClass_t_initiate_t */
#define ServiceError_errorClass_t_initiate_t__other 0
#define ServiceError_errorClass_t_initiate_t__version_incompatible 1
#define ServiceError_errorClass_t_initiate_t__max_segment_insufficient 2
#define ServiceError_errorClass_t_initiate_t__max_services_outstanding_calling_insufficient 3
#define ServiceError_errorClass_t_initiate_t__max_services_outstanding_called_insufficient 4
#define ServiceError_errorClass_t_initiate_t__service_CBB_insufficient 5
#define ServiceError_errorClass_t_initiate_t__parameter_CBB_insufficient 6
#define ServiceError_errorClass_t_initiate_t__nesting_level_insufficient 7

/* Constants for type ServiceError_errorClass_t_conclude_t */
#define ServiceError_errorClass_t_conclude_t__other 0
#define ServiceError_errorClass_t_conclude_t__further_communication_required 1
/* END MMS CONSTANTS*/

/*BEGIN MMS RETURN CODES*/
```

```
#define OK                  0
#define ERROR               -1
#define REJECT_LOCAL        -2
#define REJECT_REMOTE       -7
#define ABORT_REMOTE        -8
#define ABORT_LOCAL         -9
#define NO_PDU              -10
#define INITIATE_ERROR_PDU -11
#define CONCLUDE_ERROR_PDU -12
#define LISTEN_ERROR        -13
#define NO_CONNECT          -14
/*END MMS RETURN CODES*/
```

11.13 MMSPRIMI.H

```
typedef int Primitive_Type;

/* ***************   PRIMITIVAS  *************** */
#define A_PrimitivePrefix  70000
#define DUMMY_Primitive         A_PrimitivePrefix + 0
#define A_ASSOCIATE_request     A_PrimitivePrefix + 1
#define A_RELEASE_request       A_PrimitivePrefix + 2
#define A_ABORT_request         A_PrimitivePrefix + 3
#define A_ASSOCIATE_indication A_PrimitivePrefix + 4
#define A_RELEASE_indication    A_PrimitivePrefix + 5
#define A_ABORT_indication      A_PrimitivePrefix + 6
#define A_PABORT_indication     A_PrimitivePrefix + 7
#define A_RETURN_indication     A_PrimitivePrefix + 8
#define A_ASSOCIATE_response    A_PrimitivePrefix + 9
#define A_RELEASE_response      A_PrimitivePrefix + 10
#define A_ASSOCIATE_confirm     A_PrimitivePrefix + 11
#define A_RELEASE_confirm       A_PrimitivePrefix + 12

#define LOGIN_Request           A_PrimitivePrefix + 21
#define LOGIN_Indication        A_PrimitivePrefix + 22
#define LOGIN_Confirm           A_PrimitivePrefix + 23
#define LOGIN_ConfirmErr        A_PrimitivePrefix + 24
#define LOGOUT_Request          A_PrimitivePrefix + 25
#define LOGOUT_Indication       A_PrimitivePrefix + 26
#define LOGOUT_Confirm          A_PrimitivePrefix + 27

/*primitives*/
#define A_MMS_Request            A_PrimitivePrefix + 31
#define A_MMS_Indication         A_PrimitivePrefix + 32
#define A_MMS_Response           A_PrimitivePrefix + 33
#define A_MMS_Confirm            A_PrimitivePrefix + 34
/*primitives*/

/* ***************   MMMS PRIMITIVAS  *************** */

#define PrimitivePrefix  73000

/* Peticiones */
#define M_initiate_request     PrimitivePrefix + 0
#define M_conclude_request     PrimitivePrefix + 1
#define M_read_request         PrimitivePrefix + 2
#define M_write_request        PrimitivePrefix + 3
#define M_start_request        PrimitivePrefix + 4
#define M_stop_request         PrimitivePrefix + 5
#define MMS_Request            PrimitivePrefix + 6

/* Indicaciones */
#define M_initiate_indication PrimitivePrefix + 7
#define M_conclude_indication PrimitivePrefix + 8
#define M_read_indication      PrimitivePrefix + 9
#define M_write_indication     PrimitivePrefix + 10
#define M_start_indication     PrimitivePrefix + 11
#define M_stop_indication      PrimitivePrefix + 12

/* Respuestas */
```

```
#define M_initiate_response    PrimitivePrefix + 15
#define M_conclude_response    PrimitivePrefix + 16
#define M_read_response        PrimitivePrefix + 17
#define M_write_response       PrimitivePrefix + 18
#define M_start_response       PrimitivePrefix + 19
#define M_stop_response        PrimitivePrefix + 20
#define MMS_Response           PrimitivePrefix + 21

/* Confirmaciones */
#define  M_initiate_confirm    PrimitivePrefix + 22
#define  M_conclude_confirm    PrimitivePrefix + 23
#define  M_read_confirm        PrimitivePrefix + 24
#define  M_write_confirm       PrimitivePrefix + 25
#define  M_start_confirm       PrimitivePrefix + 27
#define  M_stop_confirm        PrimitivePrefix + 28
```

11.14 ECHO.OCC

```
PROTOCOL TRX IS [50]BYTE:
PROC echo (CHAN OF TRX input,output)
  PROC wait (VAL INT num.sec)
    --VAL TicksPerSec IS 15625:
    VAL TicksPerSec IS 16:
    TIMER clock :
    INT time :
    SEQ
      clock ? time
      time := time PLUS (num.sec * TicksPerSec)
      clock ? AFTER time
  :
  [50]BYTE string:
  INT i:
  SEQ
    i := 0
    WHILE TRUE
      SEQ
        input ? string
        wait (10)
        output ! string
```

11.15 ROOT_O.OCC

```
--Interface Occam-C para el programa del root
#INCLUDE "hostio.inc"
PROTOCOL TRX IS ANY:

PROC root.occ(CHAN OF SP fs,ts,
              CHAN OF TRX Rx1MmspmFrClnt,MmspmToClnt,      --4 Ch's Smmspm
                          FeederToClnt,                    --Ch para tx datos al servidor
cliente
                          ClientToMon,CmmspmToMon)         --Ch para monitorear al Client
  #USE "hostio.lib"
  #USE "centry.lib"
  [40000]INT stack.and.heap :
  [1]INT dummy :
  VAL flag IS 1 : -- This means combined stack and heap
  [4]INT chan.out:
  [5]INT chan.in:
  SEQ
    LOAD.INPUT.CHANNEL  (chan.in  [2], Rx1MmspmFrClnt)
    LOAD.INPUT.CHANNEL  (chan.in  [3], ClientToMon)
    LOAD.INPUT.CHANNEL  (chan.in  [4], CmmspmToMon)
    LOAD.OUTPUT.CHANNEL (chan.out [2], MmspmToClnt)
    LOAD.OUTPUT.CHANNEL (chan.out [3], FeederToClnt)
```

```
    PROC.ENTRY(fs, ts, flag, stack.and.heap, dummy, chan.in, chan.out)
    so.exit (fs, ts, sps.success)
```

11.16 CLIENT_O.OCC

```
 -Interface Occam-C para el programa del Cliente
#INCLUDE "hostio.inc"
PROTOCOL TRX IS ANY:
PROC client.occ(CHAN OF TRX Rx1MmspmFrServer,MmspmToServer,    --4 Ch's Cmmspm
                            ClntFrFeeder,                      --2 Ch's para el Proc Cliente
                            ClientToMon,CmmspmToMon)           --Ch para monitorear al Client
  #USE "centry.lib"
  [40000]INT stack.and.heap :
  [1]INT dummy :
  VAL flag IS 1 : -- This means combined stack and heap
  [4]INT chan.in :
  [5]INT chan.out
  SEQ
    LOAD.INPUT.CHANNEL (chan.in [2], Rx1MmspmFrServer)
    LOAD.OUTPUT.CHANNEL(chan.out[2], MmspmToServer)
    LOAD.INPUT.CHANNEL (chan.in [3], ClntFrFeeder)
    LOAD.OUTPUT.CHANNEL(chan.out[3], ClientToMon)
    LOAD.OUTPUT.CHANNEL(chan.out[4], CmmspmToMon)
    PROC.ENTRY.RC(flag,stack.and.heap,dummy, chan.in, chan.out)
```

12 APÉNDICE D

FUENTES DE LA APLICACIÓN WRCMDS Y RDCMDS.

12.1 RDCMDS.C

```
#include "mmsprim1.h"   /*      Primitivas MMS                          */
#include "mmsctes.h"    /*      Constante MMS                           */
#include "mmstypes.h"   /*      Tipos MMS                               */
#include "mmspdu.h"     /*      Definicion PDU MMS                      */
#include "mmstype2.h"   /*      Tipos MMS                               */
#include "mmspdutr.h"   /*      Fns de Conversion de MMSPDU a un string */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
#define sStop "&"
FILE *FCmdCtrl;
char sCmdCtrl[9];
tTrama sCmd;
tTrama sTempo;
aleaf WrtAleaf;
int j,nTop,nCountAc,nCount=0;
   printf("\nArchivo de Comandos del Cliente: ");
   gets(sCmdCtrl);
   if ( (FCmdCtrl=fopen(sCmdCtrl,"r")) == NULL )
   {
      printf("\nImposible Abrir Archivo: %s",sCmdCtrl);
      exit(EXIT_FAILURE);
   }
   fseek(FCmdCtrl,0,SEEK_SET);
   while(!feof(FCmdCtrl))
   {
     nCountAc=0;
     while(nCountAc<TRAMA_LEN){
       if(fscanf(FCmdCtrl,"%s%n",sTempo,&nCount)<=0){
         nCountAc=-1;
         nCount=0;
         break;
       }
       else{
         if(nCountAc+nCount<TRAMA_LEN)
           nTop=nCountAc+nCount;
         else
           nTop=TRAMA_LEN;
         for(j=nCountAc;j<nTop;j++){
           sCmd[j]=sTempo[j-nCountAc];
         }
         printf("\ncount:%d\tcountac:%d\tTop%d",nCount,nCountAc,nTop);
         nCountAc=nCountAc+nCount;
       }
     }
     if (nCountAc<0)
       break;
     printf("\tTrama: ");
     for(j=0;j<TRAMA_LEN;j++){
         printf("%c",sCmd[j]);
     }
     UnpackTrama(&WrtAleaf,sCmd);
     printf("\nNbRx: %d Prim: %d Pid: %d PIdRmt: %d Ass: %d AssRmt: %d PDU: %d",
     nCountAc,WrtAleaf.Primitive,
     WrtAleaf.Id.PId,WrtAleaf.Id.PIdRmt,
     WrtAleaf.Id.AssId,WrtAleaf.Id.AssIdRmt,
     WrtAleaf.MmsPdu.designator);
```

```
    }
    fclose(FCmdCtrl);
    printf("\nArchivo Cerrado\n");
  exit(EXIT_SUCCESS);
}
```

12.2 WRCMDS.C

```
#include "mmsprimi.h"    /*      Primitivas MMS                              */
#include "mmsctes.h"     /*      Constante MMS                               */
#include "mmstypes.h"    /*      Tipos MMS                                   */
#include "mmspdu.h"      /*      Definicion PDU MMS                          */
#include "mmstype2.h"    /*      Tipos MMS                                   */
#include "mmspdutr.h"    /*      Fns de Conversion de MMSPDU a un string */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int iPid=0;

int Menu(char *sCmd){
/*Regresa 1 Si es un PDU valido, 0 si no para que termine de escribir*/
char sAnswer[20];
int iAnswer;
aleaf WrtAleaf;

  for(;;){
    printf("\n                                                ");
    printf("\n                                                ");
    printf("\n************************************************");
    printf("\n                                                ");
    printf("\n                                                ");
    printf("\n     Manufacturing Message Specification        ");
    printf("\n      Generador de Archivos de Comandos         ");
    printf("\n                                                ");
    printf("\n                                                ");
    printf("\n   (  1 ) Peticion de Asociación.               ");
    printf("\n   (  2 ) Peticion de Liberación.               ");
    printf("\n   (  3 ) Peticion de Servicio Confirmado.      ");
    printf("\n   (  4 ) Peticion de Serv. No Confirmado.      ");
    printf("\n   (  9 ) DummyPrimitiva.                       ");
    printf("\n   (  0 ) Salir.                                ");
    printf("\n                                                ");
    printf("\n                                                ");
    printf("\n************************************************");
    printf("\n                                                ");
    printf("\n                                                ");
    printf("\n                                                ");
    printf("\n                                                ");
    printf("\nSu Opcion es: ");
    gets(sAnswer);
    iAnswer=sAnswer[0];
    switch(iAnswer){
      case 48:
        return(FALSE);
        break;
      case 49:
        WrtAleaf.Primitive=A_ASSOCIATE_request;
        WrtAleaf.Id.AssId=0;
        WrtAleaf.Id.AssIdRmt=0;
        WrtAleaf.Id.PId=iPid;
        WrtAleaf.Id.PIdRmt=0;
        WrtAleaf.MmsPdu.designator=MMSpdu_initiate_RequestPDU;
        PackTrama(&WrtAleaf,sCmd);
        return(TRUE);
        break;
      case 50:
        WrtAleaf.Primitive=A_RELEASE_request;
        WrtAleaf.Id.PId=iPid;
        WrtAleaf.Id.PIdRmt=0;
        WrtAleaf.Id.AssId=1;
```

```
        WrtAleaf.Id.AssIdRmt=0;
        WrtAleaf.MmsPdu.designator=MMSpdu_conclude_RequestPDU;
        PackTrama(&WrtAleaf,sCmd);
        return(TRUE);
        break;
      case 51:
        WrtAleaf.Primitive=A_MMS_Request;
        WrtAleaf.Id.PId=iPid;
        WrtAleaf.Id.PIdRmt=0;
        WrtAleaf.Id.AssId=1;
        WrtAleaf.Id.AssIdRmt=0;
        WrtAleaf.MmsPdu.designator=MMSpdu_confirmed_RequestPDU;
        PackTrama(&WrtAleaf,sCmd);
        return(TRUE);
        break;
      case 52:
        WrtAleaf.Primitive=A_MMS_Request;
        WrtAleaf.Id.PId=iPid;
        WrtAleaf.Id.PIdRmt=0;
        WrtAleaf.Id.AssId=1;
        WrtAleaf.Id.AssIdRmt=0;
        WrtAleaf.MmsPdu.designator=MMSpdu_Unconfirmed_RequestPDU;
        PackTrama(&WrtAleaf,sCmd);
        return(TRUE);
        break;
      case 57:
        WrtAleaf.Primitive=DUMMY_Primitive;
        WrtAleaf.Id.PId=iPid;
        WrtAleaf.Id.PIdRmt=0;
        WrtAleaf.Id.AssId=-1;
        WrtAleaf.Id.AssIdRmt=-1;
        PackTrama(&WrtAleaf,sCmd);
        return(TRUE);
        break;
    }
  }
}

int main()
{
#define sStop "&"
FILE *FCmdCtrl;
char sCmdCtrl[9];
tStackTrama sCmdBuff;
aleaf WrtAleaf;
int i=0,iCount=0;
   if(MAX_COMMAND<2){
     printf("\nNumero de Comandos de Archivo Insuficiente!!!");
     exit(0);
   }
   ClrPdu(&WrtAleaf);
   for(i=0;i<MAX_COMMAND;i++){
     PackTrama(&WrtAleaf,sCmdBuff[i]);
   }
   i=0;
   printf("\nArchivo de Comandos del Cliente: ");
   gets(sCmdCtrl);
   printf("\nNumero de Proceso: ");
   scanf("%d",&iPid);
   WrtAleaf.Primitive=LOGIN_Request;
   WrtAleaf.Id.PId=iPid;
   WrtAleaf.Id.PIdRmt=0;
   WrtAleaf.Id.AssId=0;
   WrtAleaf.Id.AssIdRmt=0;
   PackTrama(&WrtAleaf,sCmdBuff[i]);
   i=i+1;
   iCount=i;
   while(i<MAX_COMMAND-2){
     if(Menu(sCmdBuff[i])){
       i=i+1;
       iCount=i;
     }
     else
       break;
```

```
    }
    WrtAleaf.Primitive=LOGOUT_Request;
    WrtAleaf.Id.PId=iPid;
    WrtAleaf.Id.PIdRmt=0;
    WrtAleaf.Id.AssId=0;
    WrtAleaf.Id.AssIdRmt=0;
    PackTrama(&WrtAleaf,sCmdBuff[i]);
    i=i+1;
    iCount=i;
    if ((FCmdCtrl=fopen(sCmdCtrl,"w")) == NULL )
    {
       printf("\nImposible Abrir Archivo: %s",sCmdCtrl);
       exit(EXIT_FAILURE);
    }
    fseek(FCmdCtrl,0,SEEK_SET);
    for(i=0;i<iCount;i++){
          if (fwrite(sCmdBuff[i],TRAMA_LEN,1,FCmdCtrl)==0){
            printf("\nError al escribir en el Archivo: %s",sCmdCtrl);
            exit(EXIT_FAILURE);
          }
          if (fwrite(sFIN,1,1,FCmdCtrl)==0){
            printf("\nError al escribir en el Archivo: %s",sCmdCtrl);
            exit(EXIT_FAILURE);
          }
          if (fprintf(FCmdCtrl,"\n")==EOF){
            printf("\nError al escribir en el Archivo: %s",sCmdCtrl);
            exit(EXIT_FAILURE);
          }
    }
    fclose(FCmdCtrl);
    printf("\nArchivo %s Cerrado!!\nNúmero de PDU's: %d",sCmdCtrl,iCount);
  exit(EXIT_SUCCESS);
}
```

13 APÉNDICE E

ARCHIVOS DE CONFIGURACIÓN DE LAS APLICACIONES

13.1 MMS.PGM

```
#INCLUDE "occonf.inc"
PROTOCOL TRX IS ANY:
VAL number.of.T805 IS 1:
[number.of.T805]NODE T805:
NODE t225:
ARC Hostlink:
NETWORK
  DO
    DO i=0 FOR number.of.T805
      SET T805[i] (type, memsize := "T805",2*M)
    SET t225 (type, memsize := "T225",16*K)
    CONNECT T805[0][link][0] TO HOST WITH Hostlink
    CONNECT T805[0][link][2] TO t225[link][1]
    --CONNECT t225[link][2]    TO T805[1][link][1]
:
NODE root.p:
NODE client.p:
NODE echo1.p:
NODE echo2.p:
MAPPING
  DO
    MAP root.p ONTO T805[0]
    MAP client.p ONTO T805[0]
    MAP echo1.p ONTO t225
    MAP echo2.p ONTO t225
:
#INCLUDE "hostio.inc"
#USE "root.lku"
#USE "client.lku"
#USE "echo.lku"
CONFIG
  CHAN OF SP fs, ts :
  PLACE fs, ts ON Hostlink :
  --Relacionados con el Servidor
  CHAN OF TRX Echo.to.Rx1Smmspm,Smmspm.to.Echo:
  --Relacionados con el Cliente
  CHAN OF TRX Echo.to.Rx1Cmmspm,Cmmspm.to.Echo:      --Ch's Cliente
  --Relacionados con el Feeder
  CHAN OF TRX Feed.to.Client:                        --Ch's feeder
  --Relacionados con el Monitor
  CHAN OF TRX Cmmspm.to.Mon,Client.to.Mon:
  PAR
    PROCESSOR root.p
      root.occ(fs, ts,
        Echo.to.Rx1Smmspm,Smmspm.to.Echo,
        Feed.to.Client,Client.to.Mon,Cmmspm.to.Mon)
    PROCESSOR client.p
      client.occ(Echo.to.Rx1Cmmspm,Cmmspm.to.Echo,
        Feed.to.Client,Client.to.Mon,Cmmspm.to.Mon)
    PROCESSOR echo1.p
      echo(Smmspm.to.Echo,Echo.to.Rx1Cmmspm)
    PROCESSOR echo2.p
      echo(Cmmspm.to.Echo,Echo.to.Rx1Smmspm)
```

13.2 RDCMDS.CFS

```
T805 (memory = 2M) Single;

connect Single.link[0] to host;

process (stacksize = 4K,
         heapsize = 50K,
         interface (input FromHost, output ToHost)
        ) Simple;

input HostInput;
output HostOutput;

connect Simple.FromHost to HostInput;
connect Simple.ToHost to HostOutput;

/*  Mapping description */

use "rdcmds.lku" for Simple;
place Simple on Single;

place HostInput on host;
place HostOutput on host;
```

13.3 WRCMDS.CFS

```
T805 (memory = 2M) Single;

connect Single.link[0] to host;

process (stacksize = 4K,
         heapsize = 50K,
         interface (input FromHost, output ToHost)
        ) Simple;

input HostInput;
output HostOutput;

connect Simple.FromHost to HostInput;
connect Simple.ToHost to HostOutput;

/*  Mapping description */

use "wrcmds.lku" for Simple;
place Simple on Single;

place HostInput on host;
place HostOutput on host;
```

14 APÉNDICE F

COMO GENERAR Y CORRER LAS APLICACIONES

14.1 GENERANDO MMS.BTL

Para generar esta aplicación se requiere tanto del paquete de generación de Código tool C como de Occam; para su correcto uso se deben establecer las siguientes variables de ambiente según corresponda.

La aplicación puede generarse siguiendo estos pasos:

Inicialice las variables de ambiente en Occam.
Compilar el programa echo.occ: oc echo.occ /t225 /i.
Ligar el objeto echo.tco para el transputer T222: ilink echo.tco /f occama.lnk /t225.
Compile la interfaz Occam-C para el programa root: oc root_o.occ /t805.
Compile la interfaz Occam-C para el programa cliente: oc client_o.occ /t805.
Inicialice las variables de ambiente para C.
Compilar el programa root.c: icc root.c -t805.
Ligar el programa root.tco, usando el archivo de referencias root.lnk: ilink /f root.lnk t805.
Compilar el programa client.c: icc client.c -t805.
Ligar el programa root.tco, usando el archivo de referencias root.lnk: ilink /f client.lnk -t805.
Inicialice las variables de ambiente para Occam.
Configurar los objetos ligados según la red de transputers que se indica en el archivo mms.pgm: occonf mms.pgm.
Generar el archivo ejecutable: icollect mms.cfb.
Correr la aplicación generada: iserver /sb mms.btl /si.

Opcionalmente puede usar los siguientes archivos por lotes para crear la aplicación:
CO.BAT realiza los pasos 1 a 5 de la lista anterior.
CC.BAT realiza los pasos 7 a 10 de la lista anterior.
CP.BAT realiza los pasos 12 a 14 de la lista anterior.

14.2 GENERANDO WRCMDS.BTL

Esta aplicación es un sencilla y sólo se requiere de C para generarla. Los pasos que se debe seguir para la obtención del archivo ejecutable en el procesador transputer son:

Establecer las variables de ambiente para C.
Compilar el programa wrcmds.c: icc wrcmds.c /t805.
Ligar el objeto tco para un transputer T805: ilink wrcmds.tco /f cstartup.lnk /t805.
Configurar el objeto ligado para una red de un solo transputer según se indica en el archivo wrcnds.cfs: icconf wrcmds.cfs.
Generar el ejecutable: icollect wrcmds.cfb.
Correr el programa: irun -sl B008 wrcmds.btl -znt

Opcionalmente puede usar el archivo por lotes CW.BAT para realizar los pasos 2-6.

14.3 GENERANDO RDCMDS.BTL

Para generar Rdcmds.btl es suficiente contar con el paquete Tool C y seguir los siguientes pasos:

Establecer las variables de ambiente para C.

Compilar el programa rdcmds.c: icc rdcmds.c /t805.
Ligar el objeto tco para un transputer T805: ilink wrcmds.tco /f cstartup.lnk /t805.
Configurar el objeto ligado para una red de un solo transputer según se indica en el archivo wrcnds.cfs: icconf wrcmds.cfs.
Generar el ejecutable: icollect wrcmds.cfb.
Correr el programa: irun -sl B008 wrcmds.btl -znt.

Opcionalmente puede usar el archivo por lotes CR.BAT para realizar los pasos 2-6.

14.4 EJECUTAR LA APLICACIÓN MMS.BTL.

Si se han establecido las variables de ambiente para C teclee la siguiente línea en el prompt de DOS para correr el programa mms.btl:

```
irun -sl B008 mms.btl -znt
```

Si se prefiere correr el programa con la utilería iserver de Occam teclee:

```
iserver /sb mms.btl /si
```

El programa solicita el archivo de comandos MMS a procesar, mostrando las siguientes líneas:

```
C:\TESIS\PROGS\MMS7\PROTO2>iserver /sb mms.btl /si
Archivo de Comandos del Cliente:
```

Teclee el nombre del archivo de comandos y pulse intro, por ejemplo:

```
Archivo de Comandos del Cliente: 987
```

El programa procesa el archivo de acuerdo a los mencionado en el Capítulo 6; la salida de esta ejecución se puede dividir en dos secciones, la primera muestra las comandos del archivo leído en código ascii, ininteligible para el usuario ya que sólo con el formato general MMS puede ser decodificado. La segunda sección muestra los mensajes recibidos por cada proceso para dar la idea de lo que esta sucediendo dentro del sistema. El texto de salida de la ejecución de mms.btl es análogo al siguiente:

```
Archivo de Comandos del Cliente: 987
Comando  es :à  _      F      _
Comando  es :q  _             _      F
Comando  es :r  _             _   F
Comando  es :Á  _             _      F
Comando  es :Á  _             _      F
Comando  es :q  _             _      F
Comando  es :Á  _             _      F
Comando  es :Á  _             _      F
Comando  es :Á  _             _      F
Comando  es :r  _             _   F
Comando  es :p  _             _
```

```
. . .
+Cmmspm[CL]: A_MMS_Request      Id= 987 RId= 0  As= 1   RAs= 0
+Cmmspm[CL]: A_RELEASE_request  Id= 987 RId= 0  As= 1   RAs= 0
+Smmspm[CM]: A_MMS_Indication   Id= 1546        RId= 987        As= 1   RAs= 1
+Smmspm[CM]: A_RELEASE_indication       Id= 1546        RId= 987        As= 1
RAs= 11546,987,A_RELEASE_indication,1,1
+Server[SM]: A_MMS_Indication   Id= 1546        RId= 987        As= 1   RAs= 1
+Server[SM]: A_RELEASE_indication       Id= 1546        RId= 987        As= 1
RAs= 11546,987,A_RELEASE_indication,1,1
+Smmspm[SR]: A_RELEASE_response Id= 1546        RId= 987        As= 1   RAs= 1
+Cmmspm[SM]: A_RELEASE_confirm  Id= 987 RId= 1546       As= 1   RAs= 1
+Client[CM]: A_RELEASE_confirm  Id= 987 RId= 1546       As= 1   RAs= 1
+Client[OP]: DUMMY_Primitive    Id= 987 RId= 0
+Client[OP]: LOGOUT_Request     Id= 987 RId= 0
+Cmmspm[CL]: DUMMY_Primitive    Id= 987 RId= 0
+Cmmspm[CL]: LOGOUT_Request     Id= 987 RId= 0
+Smmspm[CM]: DUMMY_Primitive    Id= 1546        RId= 987
+Smmspm[CM]: LOGOUT_Indication  Id= 1546        RId= 987
+Server[SM]: DUMMY_Primitive    Id= 1546        RId= 987
+Cmmspm[SM]: LOGOUT_Confirm     Id= 0   RId= 1546
+Client[CM]: LOGOUT_Confirm     Id= 0   RId= 1546
+Smmspm[SR]: LOGOUT_Request     Id= 1546        RId= 987
-Server[SM]: LOGOUT_Confirm     Id= 1546        RId= 987
Server[??]:FIN
 Fin
C:\TESIS\PROGS\MMS7\PROTO2>
```

14.5 EJECUTAR LA APLICACIÓN WRDCMDS.BTL.

Esta aplicación genera un archivo de comandos con el formato MMS para que las aplicación mms.btl pueda procesarlo; para ejecutarlo puede teclearse la siguiente línea en el prompt de DOS si se han establecido las variables de ambiente para C.

```
irun -sl B008 wrcmds.btl %1%
```

También se puede ejecutar en el ambiente Occam, tecleando:

```
iserver /sb wrcmds.btl /si
```

Al ejecutarse, el programa solicita el nombre del archivo que se desea generar y el número de proceso que tendrá el cliente; Como ejemplo se designó el nombre 987 al archivo y 512 para el identificador del proceso.

```
C:\TESIS\PROGS\MMS7\PROTO2>iserver /sb wrcmds.btl /si
Archivo de Comandos del Cliente: 956
```

```
Numero de Proceso:14
```

Una vez alimentados estos datos, el sistema muestra el Menú general de los comandos disponibles, como se muestra a continuación:

```
*****************************************
     Manufacturing Message Specification
      Generador de Archivos de Comandos
   (  1 ) Peticion de Asociaci%n.
   (  2 ) Peticion de Liberaci%n.
   (  3 ) Peticion de Servicio Confirmado.
   (  4 ) Peticion de Serv. No Confirmado.
   (  9 ) DummyPrimitiva.
   (  0 ) Salir.
*****************************************
Su Opcion es: 0
```

El programa solicita el tipo de mensaje que se quiere escribir en el archivo; Existen varios tipos de mensajes según MMS, sin embargo, para propósito de demostración del protocolo MMS se han agrupado estos tipos según el tipo de que se trate, por ejemplo, existe el mensaje READ, WRITE sin embargo su tipo puede ser Servicio Confirma o Servicio Sin Confirmación.

Después de haber seleccionado el tipo de mensaje que se desea grabar, el sistema vuelve a presentar el Menú principal para que un nuevo comando sea grabado; la operación anterior se repite hasta que el usuario Teclea 0, después de lo cual, el sistema reporta cuantos mensajes se escribieron en total, en una pantalla parecida a la siguiente:

```
Archivo 987 Cerrado!!
N·mero de PDU's: 14
C:\TESIS\PROGS\MMS7\PROTO2>
```

14.6 EJECUTAR LA APLICACIÓN RDCMDS.BTL.

Esta aplicación es usada opcionalmente para leer los comando grabados con la aplicación wrcmds. Al igual que mms.btl lee un archivo pero sólo para desplegar su contenido; para ejecutarlo puede teclearse la siguiente línea en el prompt de DOS si se han establecido las variables de ambiente para C.

```
irun -sl B008 rdcmds.btl -znt
```

También se puede ejecutar en el ambiente Occam, tecleando:

```
iserver /sb rdcmds.btl /si
```

Al ejecutarse, el programa muestra el contenido del archivo de comandos en una pantalla parecida a:

```
C:\TESIS\PROGS\MMS7\PROTO2>iserver /sb rdcmds.btl /si
Archivo de Comandos del Cliente: 956
. . .
```

```
count:11        countac:0       Top11
count:41        countac:11      Top30   Trama:  F       Á       +
NbRx: 52 Prim: 17920 Pid: 16777489 PIdRmt: 0 Ass: -1 AssRmt: -1879048193 PDU: -9
39524096
count:11        countac:0       Top11
count:41        countac:11      Top30   Trama:  F       q       +
NbRx: 52 Prim: 17920 Pid: 273 PIdRmt: 0 Ass: 1 AssRmt: 1895825408 PDU: -93952409
6
count:11        countac:0       Top11
count:41        countac:11      Top30   Trama:  F       r       +
NbRx: 52 Prim: 17920 Pid: 16777489 PIdRmt: 0 Ass: 0 AssRmt: 1912602624 PDU: -939
524096
count:11        countac:0       Top11
count:41        countac:11      Top30   Trama:  F       Á       +
NbRx: 52 Prim: 17920 Pid: 16777489 PIdRmt: 0 Ass: 1 AssRmt: -1895825408 PDU: -93
9524096
count:11        countac:0       Top11
count:21        countac:11      Top30   Trama:  F       é       +
NbRx: 32 Prim: 17920 Pid: 273 PIdRmt: 0 Ass: 1 AssRmt: -1996488704 PDU: -9395240
96
count:31        countac:0       Top30   Trama:  F       +
NbRx: 31 Prim: 17920 Pid: 456 PIdRmt: 0 Ass: 0 AssRmt: 0 PDU: 0
Archivo Cerrado
C:\TESIS\PROGS\MMS7\PROTO2>
```

15 APÉNDICE G

SECUENCIA.EXE

Como se mencionó en el punto 6.1.1.3, los procesos comunicantes envían al proceso monitor los mensajes que han recibido para que este a su vez los imprima en la pantalla para proporcionar una visión del intercambio de mensajes generado. Esta salida se puede apreciar en el Apéndice F.

Para facilitar la depuración del protocolo y poder apreciar detalladamente la secuencia de mensajes, se diseño la aplicación Secuencia.exe.

Este aplicación fue desarrollada en Visual Basic 5 y consta de una sola interfaz gráfica, la cual se muestra en la Figura 67.

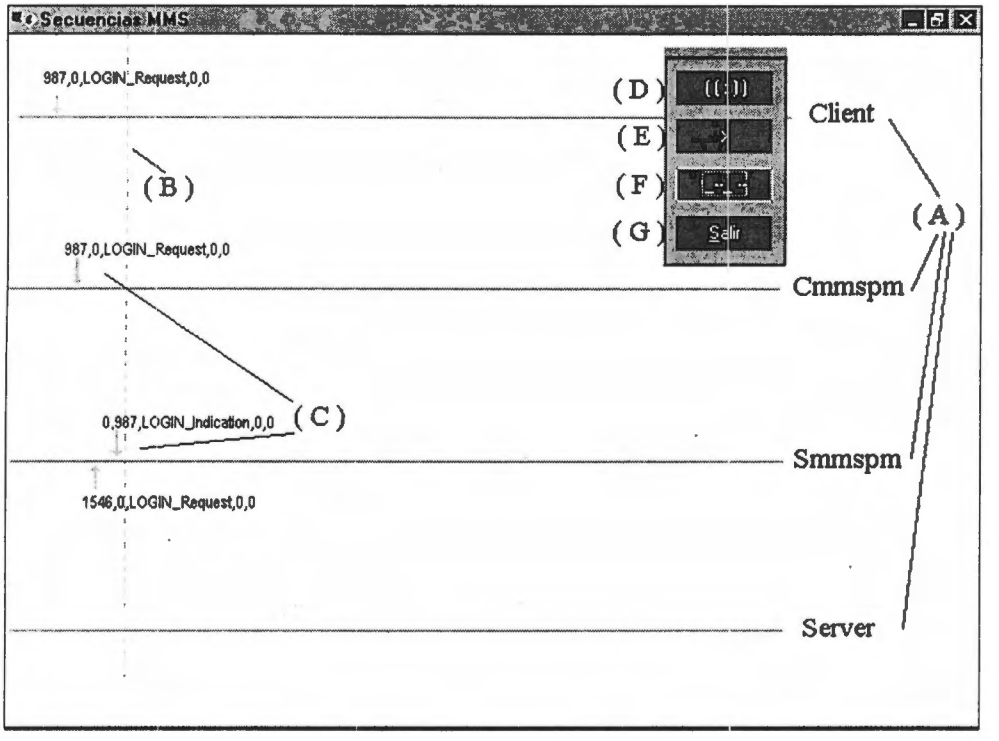


Figura 67 Aplicación Secuencia.exe

La aplicación simplemente lee el archivo de valores separados por coma que genera la aplicación mms.btl y va mostrando en forma gráfica como se fueron generando estos mensajes; el usuario puede controlar la ejecución de la secuencia según se requiera.

La ventana de la aplicación muestra cuatro lineas verdes horizontales **(A)** que representan los cuatro procesos principales involucrados en el protocolo mms: client, cmmspm, smmspm y server; una línea vertical **(B)** que marca la posición actual en el tiempo (eje horizontal) y los mensajes que se van generando **(C)**. Así mismo, se presenta una barra simple de herramientas, que contiene los botones siguientes:

Cargar.- **(D)** Este botón sirve para que la aplicación cargue los datos a memoria del archivo generado por la aplicación mms. Este archivo tiene el nombre output.csv y debe estar en la ruta donde esta ejecutándose la aplicación Secuencia.

Ejecutar.- **(E)** Botón para iniciar la secuencia.

Paso a paso.- **(F)** Botón para ejecutar la secuencia paso a paso. Se pueden alternar este botón con el botón Ejecutar para situarse en el "tiempo" en el que se quiera hacer un análisis más detallado.

Salir.- **(G)** Botón para abandonar la aplicación.

El código de esta aplicación se agrega a continuación.

```
Option Explicit
Dim fi_offset(5) As Integer
Dim fi_nivelneg(5) As Integer
Dim fi_nivelpos(5) As Integer
Dim fi_gap As Integer
Dim fi_gapTime As Integer
Dim FI_HASTA As Integer
Dim FI_DESDE As Integer
Dim FI_PID As Integer
Dim FI_RPID As Integer
Dim FI_PRIMITIVA As Integer
Dim FI_AS As Integer
Dim FI_RAS As Integer
Dim fi_IdSec As Integer
Dim FI_MAXITEMS As Integer
Dim FI_WIDTHOFF As Integer
Dim FI_1stShowItem As Integer
Dim FI_LastShowItem As Integer
Dim fi_rowpinta As Integer
Dim fi_LastX As Integer


Public Function fw_PintaSec()
Dim li_Col As Integer
Dim li_Hasta As Integer
Dim li_Desde As Integer
Dim li_Pid As Integer
Dim li_RPid As Integer
Dim ls_Primitva As String
Dim li_AS As Integer
Dim li_RAS As Integer
Dim li_YTmp As Integer
Dim li_Y2Tmp As Integer

    If fi_rowpinta >= grd_SecTmp.Rows - 1 Then
        fw_PintaSec = False
        Exit Function
    End If
    grd_SecTmp.Row = fi_rowpinta
    grd_SecTmp.col = FI_HASTA
    li_Hasta = grd_SecTmp.Text
    grd_SecTmp.col = FI_DESDE
    li_Desde = grd_SecTmp.Text
    grd_SecTmp.col = FI_PID
    li_Pid = grd_SecTmp.Text
    grd_SecTmp.col = FI_RPID
    li_RPid = grd_SecTmp.Text
    grd_SecTmp.col = FI_PRIMITIVA
    ls_Primitva = grd_SecTmp.Text
    grd_SecTmp.col = FI_AS
    li_AS = grd_SecTmp.Text
    grd_SecTmp.col = FI_RAS
    li_RAS = grd_SecTmp.Text
    If li_Hasta < li_Desde Then
       pic_In(fi_rowpinta).Picture = pic_up.Picture
       li_YTmp = Line1(li_Hasta).Y1 + 20
       grd_IndexUp.col = li_Hasta
       grd_IndexUp.Row = li_Desde
       li_Y2Tmp = Val(grd_IndexUp.Text)
       grd_IndexUp.Text = (li_Y2Tmp + 1) Mod FI_MAXITEMS
       li_Y2Tmp = li_Y2Tmp * fi_gap + pic_In(fi_rowpinta).Height + Label1(fi_rowpinta).Height -
20
    Else
       pic_In(fi_rowpinta).Picture = pic_down.Picture
       li_YTmp = Line1(li_Hasta).Y1 - pic_In(fi_rowpinta).Height - 20
       grd_IndexDown.col = li_Hasta
       grd_IndexDown.Row = li_Desde
       li_Y2Tmp = Val(grd_IndexDown.Text)
       grd_IndexDown.Text = (li_Y2Tmp + 1) Mod FI_MAXITEMS
       li_Y2Tmp = li_Y2Tmp * fi_gap * (-1) + 20
    End If
```

```
    Label1(fi_rowpinta) = CStr(li_Pid) & "," & CStr(li_RPid) & "," & ls_Primitva & "," &
CStr(li_AS) & "," & CStr(li_RAS)
    pic_In(fi_rowpinta).Move fi_LastX + fi_gapTime, li_YTmp
    Label1(fi_rowpinta).Move pic_In(fi_rowpinta).Width + fi_LastX, li_YTmp + li_Y2Tmp -
Label1(fi_rowpinta).Height
    pic_In(fi_rowpinta).Visible = True
    Label1(fi_rowpinta).Visible = True
    If fi_rowpinta < FI_LastShowItem Then
        Line2.X1 = pic_In(fi_rowpinta).Left + pic_In(fi_rowpinta).Width
        Line2.X2 = Line2.X1
    Else
        rw_MueveIzq
    End If
    fi_LastX = pic_In(fi_rowpinta).Left
    fi_rowpinta = fi_rowpinta + 1
    fw_PintaSec = True
End Function

Public Sub rw_DatosIni()
Dim li_counter As Integer
Dim li_Hasta As Integer
    fi_IdSec = 0
    Line2.X1 = FI_WIDTHOFF + pic_In(0).Width
    Line2.X2 = Line2.X1
    Timer1.Interval = 500
    Timer1.Enabled = False
    FI_1stShowItem = 0
    FI_LastShowItem = 20

    Me.BackColor = QBColor(0)
    FI_MAXITEMS = 3
    FI_HASTA = 0
    FI_DESDE = 1
    FI_PID = 2
    FI_RPID = 3
    FI_PRIMITIVA = 4
    FI_AS = 5
    FI_RAS = 6
    FI_WIDTHOFF = 200
    fi_gap = Label1(0).Height * 0.7
    fi_gapTime = 200
    Line1(0).Visible = False
    For li_counter = 1 To 4
        Line1(li_counter).Visible = True
    Next li_counter
    For li_Hasta = 0 To 4
        fi_offset(li_Hasta) = Line1(li_Hasta).Y1
    Next li_Hasta
End Sub

Private Sub btn_Inicia_Click()
    If btn_Inicia.Caption = ">" Then
        If Not fw_PintaSec Then
            MsgBox "El último Mensaje ya ha sido desplegado", vbExclamation, Me.Caption
            Exit Sub
        Else
            Timer1.Enabled = True
            btn_Inicia.Caption = "| |"
            btn_Inicia.ToolTipText = "Detener"
        End If
    Else
        Timer1.Enabled = False
        btn_Inicia.Caption = ">"
        btn_Inicia.ToolTipText = "Iniciar"
    End If
End Sub

Private Sub btn_Paso_Click()
    If Not fw_PintaSec Then
        MsgBox "El último Mensaje ya ha sido desplegado", vbExclamation, Me.Caption
        Exit Sub
    End If
    Timer1.Enabled = False
```

```
    btn_Inicia.Caption = ">"
End Sub

Private Sub btn_Refresh_Click()
    Timer1.Enabled = False
    rw_CargaDatosDisco App.Path & "\output.csv"
    btn_Inicia.Enabled = True
    btn_Paso.Enabled = True
End Sub

Private Sub btn_salir_Click()
    End
End Sub

Private Sub Form_Load()
    rw_DatosIni
End Sub


Public Sub rw_MueveDer()
Dim li_counter As Integer

    For li_counter = 1 To pic_In.Count - 1
        pic_In(li_counter).Left = pic_In(li_counter).Left + FI_WIDTHOFF + fi_gapTime
        Label1(li_counter).Left = Label1(li_counter).Left + FI_WIDTHOFF + fi_gapTime
    Next li_counter
End Sub
Public Sub rw_MueveIzq()
Dim li_counter As Integer

    For li_counter = 0 To pic_In.Count - 1
        pic_In(li_counter).Left = pic_In(li_counter).Left - fi_gapTime
        Label1(li_counter).Left = Label1(li_counter).Left - fi_gapTime
    Next li_counter
End Sub


Sub rw_CargaDatosDisco(ls_Archivo As String)
Dim li_i As Integer
Dim fs_Hasta As String
Dim fl_Desde As String
Dim fs_Pid As String
Dim fs_rPid As String
Dim fs_Primitiva As String
Dim fs_As As String
Dim fs_rAs As String

Dim li_PseudoRow As Integer
Dim MyCol  As Integer
Dim li_Row As Integer
Dim li_Desde As Integer
Dim li_Hasta As Integer
Dim li_counter As Integer

    Screen.MousePointer = vbHourglass
    On Error GoTo ErrorCarga

    For li_counter = 0 To pic_In.Count - 1
        pic_In(li_counter).Visible = False
        Label1(li_counter).Visible = False
    Next li_counter
    For li_Hasta = 0 To grd_IndexUp.Cols - 1
        grd_IndexUp.col = li_Hasta
        grd_IndexDown.col = li_Hasta
        For li_Desde = 0 To grd_IndexUp.Rows - 1
            grd_IndexUp.Row = li_Desde
            grd_IndexUp.Text = 0
            grd_IndexDown.Row = li_Desde
            grd_IndexDown.Text = 0
        Next li_Desde
    Next li_Hasta

    fi_IdSec = 0
    Line2.X1 = FI_WIDTHOFF + pic_In(0).Width
```

```
    Line2.X2 = Line2.X1
    fi_rowpinta = 0
    fi_LastX = FI_WIDTHOFF
    grd_SecTmp.Cols = FI_RAS + 1
    grd_SecTmp.Rows = 2
    r_LimpiaGrid grd_SecTmp

    grd_SecTmp.Rows = 1
    li_i = 0
    Open ls_Archivo For Input As #1
    Do While Not EOF(1) ' Repite el bucle hasta el final del archivo.
        Input #1, fs_Hasta, fl_Desde, fs_Pid, fs_rPid, fs_Primitiva, fs_As, fs_rAs
        If li_i > grd_SecTmp.Rows - 1 Then
            grd_SecTmp.Rows = grd_SecTmp.Rows + 1
        End If
        grd_SecTmp.Row = li_i
        grd_SecTmp.col = FI_HASTA
        grd_SecTmp.Text = fs_Hasta
        grd_SecTmp.col = FI_DESDE
        grd_SecTmp.Text = fl_Desde
        grd_SecTmp.col = FI_PID
        grd_SecTmp.Text = fs_Pid
        grd_SecTmp.col = FI_RPID
        grd_SecTmp.Text = fs_rPid
        grd_SecTmp.col = FI_PRIMITIVA
        grd_SecTmp.Text = fs_Primitiva
        grd_SecTmp.col = FI_AS
        grd_SecTmp.Text = fs_As
        grd_SecTmp.col = FI_RAS
        grd_SecTmp.Text = fs_rAs
        If li_i >= pic_In.Count - 1 Then
            MsgBox "No serán Mostados todos los mensajes, El número es mayor al permitido por
esta aplicación", vbExclamation, Me.Caption
            Exit Do
        End If
        li_i = li_i + 1
    Loop
    Close #1 'Termino Ok la lectura
    Screen.MousePointer = vbDefault
    Exit Sub

ErrorCarga:
    Select Case Err
        Case 57
            MsgBox "Disquette dañado o sin formatear", vbCritical + vbOKOnly, Me.Caption
        Case 70
            MsgBox "Acceso Denegado", vbCritical + vbOKOnly, Me.Caption
        Case 71
            MsgBox "Inserte el disquette", vbCritical + vbOKOnly, Me.Caption
        Case 53
            '------------------------------------
            'no encontro el archivo de estatus
            MsgBox "No existe el archivo", vbCritical + vbOKOnly, Me.Caption
        Case 75
            '------------------------------------
            'no encontro el archivo de estatus
            MsgBox "Ruta Incorrecta", vbCritical + vbOKOnly, Me.Caption
        Case Else
            MsgBox Error$, vbCritical + vbOKOnly, Me.Caption
    End Select
    Screen.MousePointer = vbDefault
    Close #1
    Exit Sub
End Sub

Public Sub rw_CargaDatos()
Dim li_PseudoRow As Integer
Dim MyCol  As Integer
Dim li_Row As Integer
Dim li_Desde As Integer
Dim li_Hasta As Integer
Dim li_counter As Integer

    For li_counter = 0 To pic_In.Count - 1
```

```
        pic_In(li_counter).Visible = False
        Label1(li_counter).Visible = False
    Next li_counter
    For li_Hasta = 0 To grd_IndexUp.Cols - 1
        grd_IndexUp.col = li_Hasta
        grd_IndexDown.col = li_Hasta
        For li_Desde = 0 To grd_IndexUp.Rows - 1
            grd_IndexUp.Row = li_Desde
            grd_IndexUp.Text = 0
            grd_IndexDown.Row = li_Desde
            grd_IndexDown.Text = 0
        Next li_Desde
    Next li_Hasta

    fi_IdSec = 0
    Line2.X1 = FI_WIDTHOFF + pic_In(0).Width
    Line2.X2 = Line2.X1
    fi_rowpinta = 0
    fi_LastX = FI_WIDTHOFF
    grd_SecTmp.Cols = FI_RAS + 1
    grd_SecTmp.Rows = 2
    r_LimpiaGrid grd_SecTmp
   For li_Row = 0 To 10 'Data1.Recordset.RecordCount
        li_PseudoRow = li_Row Mod 5 '(grd_sec.VisibleRows - 1)
        If li_Row Mod 5 Then '(grd_sec.VisibleRows - 1)) = 0 And li_Row > 0 Then
            'grd_sec.Scroll 0, grd_sec.VisibleRows - 1
        End If
        If li_Row > grd_SecTmp.Row - 1 Then
            grd_SecTmp.Rows = grd_SecTmp.Rows + 1
        End If
        grd_SecTmp.Row = li_Row
        For MyCol = 0 To grd_SecTmp.Cols - 1
            grd_SecTmp.col = MyCol
            grd_SecTmp.Text = "" 'grd_sec.Text
            If grd_SecTmp.col = FI_DESDE Then
                If grd_SecTmp.Text = "-1" Then
                    Exit For
                End If
            End If
        Next MyCol
        If li_Row >= pic_In.Count - 1 Then
            MsgBox "No serán Mostados todos los mensajes, El número es mayor al permitido por
esta aplicación", vbExclamation, Me.Caption
            Exit For
        End If
    Next li_Row
End Sub

Public Sub rw_Menu(btn_push As Object)
    If btn_push.Caption = "btn_Refresh" Then
        Timer1.Enabled = False
    Else
        If btn_push.Caption = "btn_Inicia" Then
        End If
    End If
End Sub

Private Sub Timer1_Timer()
    If fi_rowpinta < grd_SecTmp.Rows - 1 Then
        fw_PintaSec
    Else
        Timer1.Enabled = False
        btn_Inicia.Caption = ">"
    End If
End Sub
```

2163598