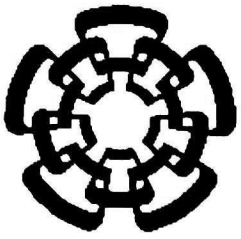


CINVESTAV

Centro de Investigación y de Estudios Avanzados del IPN
Unidad Guadalajara

DISEÑO DE UN PROVEEDOR DE SERVICIOS DE TELEFONÍA PARA EL CONTROL DE LA LLAMADA USANDO TÉCNICAS DE DESCRIPCIÓN FORMAL

TESIS QUE PRESENTA
LUIS PRABÚ XIMÉNEZ CÁRDENAS

PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS

EN LA ESPECIALIDAD DE
INGENIERÍA ELÉCTRICA

Guadalajara, Jal., Agosto de 2001.

DISEÑO DE UN PROVEEDOR DE SERVICIOS DE TELEFONÍA PARA EL CONTROL DE LA LLAMADA USANDO TÉCNICAS DE DESCRIPCIÓN FORMAL

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

por:

Luis Prabú Ximénez Cárdenas

Licenciado en Informática
Universidad de Guadalajara, 1993-1997

Becario del CONACYT, expediente no. 121120

Director de Tesis:

Dr. Deni Librado Torres Román

CINVESTAV del IPN Unidad Guadalajara, Agosto de 2001

Agradecimientos

A mis padres por su cariño, sus sacrificios y por haberme enseñado el camino del estudio.

A mi asesor Dr. Deni Torres Román por el tiempo que invirtió en el desarrollo de este trabajo y por su infinita paciencia con mis tropiezos.

A mi amor, Rosa, por haberme apoyado durante los momentos más difíciles.

Al Dr. Félix Ramos por sus muy acertados regaños.

A CONACYT por el apoyo económico que me permitió realizar esta maestría.

Luis Prabú Ximénez Cárdenas.

... no sé cómo he sido visto por el mundo, pero en lo que a mi concierne, me parece haber sido solo un niño, jugando en la playa y maravillándome al encontrar de vez en cuando una piedra más lisa o una concha más hermosa de lo habitual, mientras el gran océano de la verdad yacía misterioso ante mí ...

Issac Newton.

Indice General

Indice de Figuras

Indice de Especificaciones

1

Introducción

- Por favor su majestad, ¿por dónde debo empezar?
- Comienza por el principio, dijo el rey muy seriamente, y sigue hasta que llegues al final. Entonces detente.

Lewis Carroll. "Alicia en el país de las maravillas".

1.1 Motivación

Aproximadamente cada 10 años, la telefonía experimenta cambios importantes[*Schulzrinne99*]. En los años 50's, la introducción de cables coaxiales trasatlánticos permitió la marcación directa internacional; en los 60's las tecnologías digitales de transmisión mejoraron drásticamente la calidad del audio; durante los años 70's los conmutadores programables permitieron la marcación por tonos y servicios locales como *llamada en espera*[1]; y en los 80's los sistemas de señalización fuera de banda hicieron posible los servicios 800 primero y la red inteligente después. En los 90's hemos sido testigos de la convergencia entre la telefonía y las computadoras con el desarrollo de conceptos como CTI[2] Desde finales de los 90's y hasta nuestros días, el potencial de Internet ha tenido una gran influencia en el desarrollo de tecnologías como la Telefonía sobre IP.

Debido a esta convergencia, el potencial para el desarrollo de sistemas telefónicos que proporcionen nuevos y más variados servicios ha aumentado considerablemente. Sin embargo, el desarrollo de estos sistemas telefónicos se ha vuelto una tarea compleja. El gran número de requerimientos y la naturaleza altamente concurrente de ellos y la posible influencia que tendrán sobre nuestro estilo de vida, convierten su desarrollo e investigación en un problema interesante desde dos puntos de vista: el económico y el científico.

[1] *Call Waiting*
[2] *Computer Telephone Integration*

Desde el punto de vista económico porque grandes empresas realizan inversiones millonarias para mantenerse a la vanguardia telefónica. Un estudio reciente llevado a cabo por la empresa Dataquest indica que entre los años 1996 y 2000 el ritmo de crecimiento anual del mercado CTI ha sido del 25% (sólo en Norteamérica). Con este ritmo de crecimiento ellos predicen que los ingresos generados por este mercado llegarán a los 6.1 billones de dólares al inicio de esta década[*Dataquest99*].

Desde el punto de vista científico el desarrollo de sistemas telefónicos modernos presenta problemas especialmente difíciles de resolver. Entre ellos, el problema de interacción entre facilidades es quizá el más interesante. Las facilidades corresponden al comportamiento observable de un sistema telefónico y por lo tanto la interacción entre ellas es un problema de especificación de requerimientos[*Zave93*].

Existen en la literatura muchas definiciones para el problema de interacción entre facilidades: el conjunto de todas las interacciones que interfieren con la operación normal de una facilidad[*Cameron93*]; la situación en la cual el comportamiento global del sistema (especificado como un conjunto de facilidades) no satisface el comportamiento individual de cada facilidad[*Gibson97*]; y el estado en el cual la activación de dos funciones (facilidades) produce un resultado impredecible[*Frappier97*]. El uso de lenguajes de especificación formal ha sido propuesto como una manera de diseño y detección de interacciones. De esta manera, los errores pueden ser encontrados antes de su implementación, eliminando con ello los elevados costos asociados con el rediseño.

Como podemos apreciar, la especificación y desarrollo de sistemas telefónicos presenta retos y oportunidades interesantes. El impacto que tienen sobre nuestro estilo de vida es tal que su estudio se justifica ampliamente. Estas razones han sido el origen y motivación del presente trabajo, con el cual se busca la incorporación de nuevas ideas al conocimiento existente sobre el tema.

1.2 Objetivo

En este trabajo se presenta el diseño de un proveedor de servicios de telefonía para un sistema CTI. Para esto fueron utilizadas técnicas de descripción formal para especificar sin ambiguedades su comportamiento y garantizar su correcto funcionamiento.

Dicho proveedor de servicios permite el manejo de las facilidades SWC[3] (llamadas telefónicas originadas desde software) y HWC[4] (llamadas telefónicas originadas desde hardware) presentes en el sistema CTI[*Walters97*].

Para la especificación se utilizó el lenguaje de descripción formal LOTOS[*ISO89*], basado en ordenamiento temporal del comportamiento observable. La especificación resultante fue implementada como un proveedor de servicios de telefonía para Windows®.

[3] *Software Telephone Calls*
[4] *Hardware Telephone Calls*

1.3 Estructura de la tesis

En el capítulo 2 se presenta una descripción general de los sistemas telefónicos. Los capítulos 3 y 4 describen los conceptos de telefonía en Windows® y proveedores de servicio de telefonía para Windows® respectivamente. Una especificación informal del proveedor de servicios es presentada en el capítulo 5. En el capítulo 6 se desarrolla la especificación formal en LOTOS que incorpora una solución para el problema de la interacción entre SWC y HWC. Finalmente, el capítulo 7 expone las conclusiones y el trabajo futuro del presente trabajo.

1.4 Trabajo previo

La especificación y diseño de sistemas telefónicos complejos mediante técnicas formales ha llamado fuertemente la atención de la comunidad científica. Una gran variedad de técnicas formales han sido utilizadas. Zave escribió uno de los primeros artículos presentando un formalismo para describir sistemas distribuidos, protocolos y sistemas telefónicos[*Zave85*]. Análisis basados en el uso de redes de Petri[*Nakamura97*] y redes de Petri coloreadas[*Jensen87*] también han sido empleados. El uso de tipos de datos abstractos es mostrado en[*Biebow85*] y [*Stepien95*], extendiéndose estos conceptos a la programación orientada a objetos en[*Iraqui97*] y [*Prehofer97*]. También han sido empleados la lógica temporal[*Blom94*] y la teoría de control de eventos discretos[*Chen94*]. Finalmente, resulta interesante la exploración del problema hecha en[*Charnois97*] en donde la teoría de procesamiento de lenguaje natural es utilizada.

2

Sistemas telefónicos

En teoría, no hay diferencia entre la teoría y la práctica. En la práctica, sí la hay.

Chuck Reid.

2.1 Introducción

Los sistemas de telecomunicaciones han evolucionado de dos campos tecnológicos: la computación y las comunicaciones[*Spragins91*]. Estos sistemas se encuentran entre las aplicaciones de software y hardware más complejas. Los factores que influyen en esta complejidad son tanto la gran variedad de arquitecturas, dispositivos y protocolos que intervienen en su construcción; como su naturaleza altamente concurrente y distribuida.

Los sistemas de telecomunicaciones son un caso particular de sistemas distribuidos. Entidades en diferentes lugares(posiblemente alejadas geográficamente) usan interfaces para comunicarse entre sí. El conjunto de interfaces involucradas en la comunicación es llamado sistema de interacción.

En las secciones siguientes se revisarán los conceptos básicos de sistemas distribuidos. Se presentará un panorama general de los sistemas telefónicos basado en referencias clásicas como[*Tanenbaum97*]. Y finalmente, se hará una descripción general de la metodología utilizada en el diseño de este tipo de sistemas.

2.2 Sistemas distribuidos

Un sistema distribuido consiste en una colección de elementos autónomos que interactúan, como una única entidad, con el ambiente(usuarios) y equipado con software para sistemas distribuidos[*Coulouris96*]. El software para sistemas distribuidos habilita a los elementos para coordinar sus actividades y compartir recursos del sistema y datos. Los usuarios de un sistema distribuido deben percibir un simple e integrado servicio aunque éste se constituya por muchos elementos(posiblemente) separados geográficamente.

De esta manera podemos identificar características que definen a los sistemas distribuidos[*Mullender95*]:

- Múltiples dispositivos
- Sistemas de interconexión
- Estado compartido

La manera más abstracta de visualizar un sistema distribuido es como una caja negra(Figura 2-1). En este enfoque, no es posible distinguir la estructura interna del sistema, en lugar de ello, sólo aparecen definidas las interfaces de entrada/salida con el ambiente. Analizando un sistema distribuido desde esta perspectiva, todo el énfasis es puesto en las interacciones entre el sistema y el ambiente.

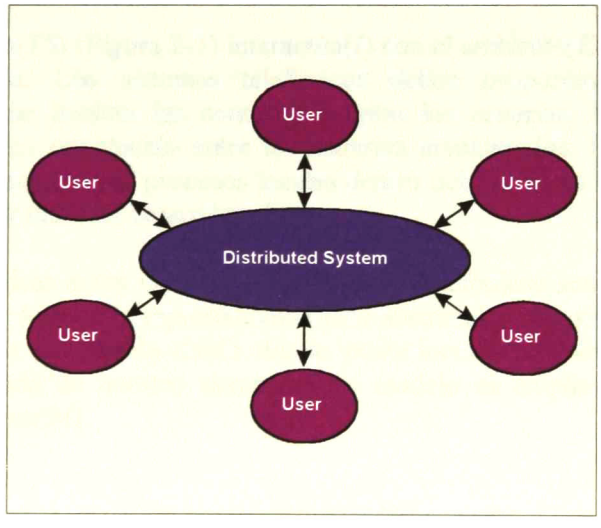


Figura 2-1. Sistema distribuido visto como una caja negra

Los elementos que constituyen un sistema distribuido pueden comunicarse entre sí para interactuar y formar un todo unificado. Estos elementos son llamados subsistemas(Sb). La función de cada uno de los subsistemas interactuando, vistos como una caja blanca, integran la función que el sistema distribuido ofrece a sus usuarios(Figura 2-2).

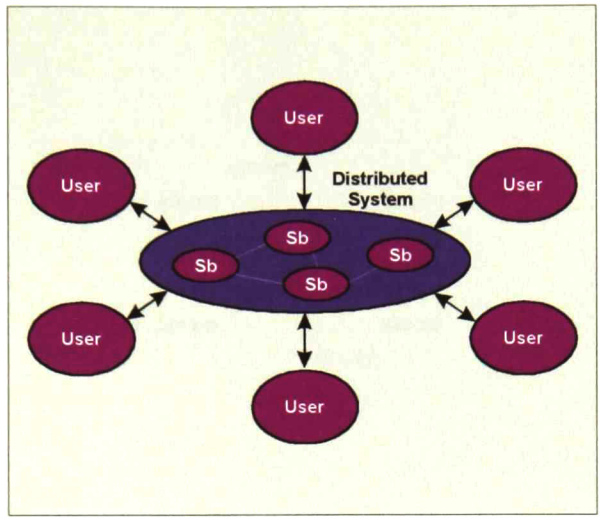


Figura 2-2. Sistema distribuido visto como caja blanca

2.3 Arquitectura general de los sistemas telefónicos

Un sistema telefónico(*TS*) (Figura 2-3) interactúa(*I*) con el ambiente(*ES*) constituido por los usuarios del sistema. Los sistemas telefónicos deben proporcionar un servicio de comunicación(*CS*) que habilita las conexiones entre los usuarios. Una conexión es un camino virtual o físico establecido entre los usuarios involucrados. Estas conexiones son manejadas simultáneamente por procesos locales dentro del *CS*. Cada uno de esos procesos se encarga de manejar una sola conexión a la vez.

Dadas las características antes descritas, los sistemas telefónicos son claramente sistemas distribuidos. Este modelo es la representación más abstracta de un sistema telefónico. Por lo tanto es el punto de partida desde donde posteriores refinamientos producirán una especificación detallada de nuestro sistema. Este modelo es ampliamente aceptado y es descrito en[*Boumezbeur91*].

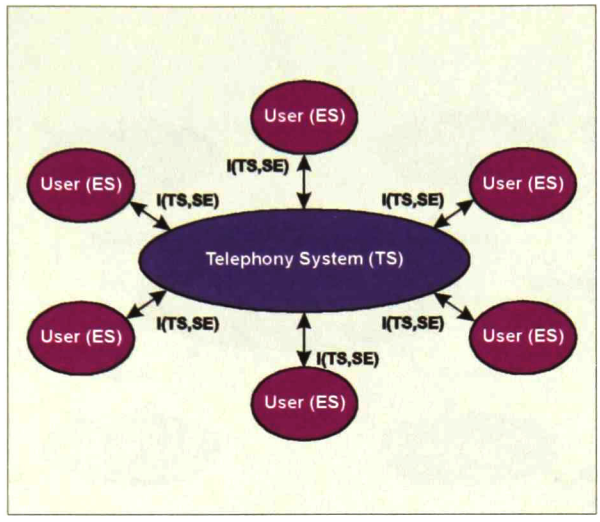


Figura 2-3. Representación general de los sistemas telefónicos

2.3.1 Servicio de comunicación

El servicio de comunicación es el proceso más importante en un sistema telefónico(Figura 2-4). Dicho servicio es el responsable de mantener todas las conexiones entre los usuarios del sistema independientemente de su situación espacial(posición y distancia entre ellos).

El *CS* interactúa con todos los usuarios del sistema y tiene la capacidad(y el deber) de mantener actualizado el estatus de todas las líneas conectadas al sistema. Además, se encarga de contestar todas las peticiones de los usuarios mediante las señales adecuadas. Una vez que una conexión es terminada por alguno de los usuarios involucrados en ella, el *CS* desconecta sus líneas y las pone nuevamente disponibles.

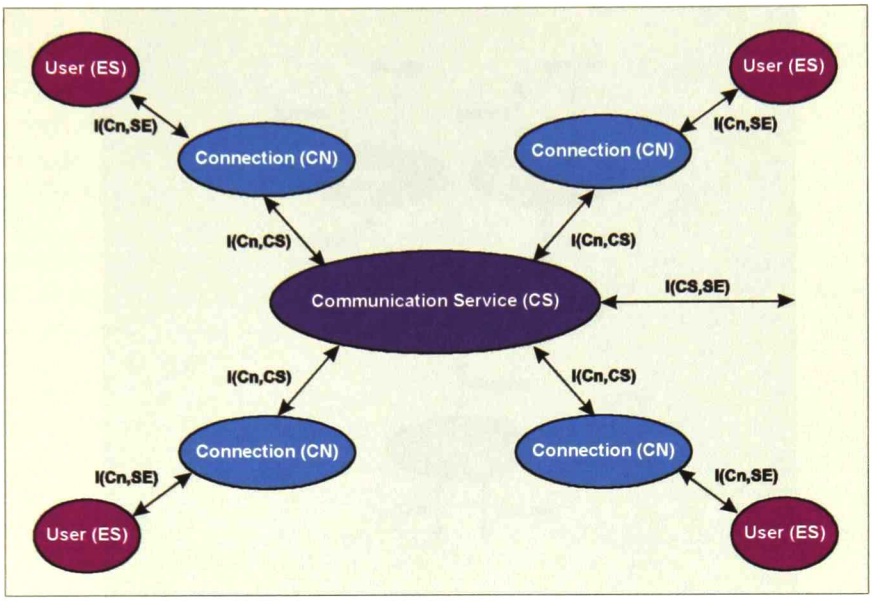


Figura 2-4 Servicio de comunicación en los sistemas telefónicos

2.3.2 Conexiones

Dentro de un *TS*, las conexiones consisten de un transmisor que inicia una llamada y posiblemente de un receptor(si la llamada llega exitosamente a su destino y es contestada)(Figura 2-5).

Una vez que una conexión es establecida, inicia el intercambio de datos entre los usuarios. Estos datos son en general voz, pero es posible transmitir cualquier otro tipo de información. En el caso más general, una vez que se establece una conexión, otros usuarios pueden ser añadidos a la misma. La comunicación entre usuarios en una conexión es manejada por un proceso especial llamado manejador de conexión.

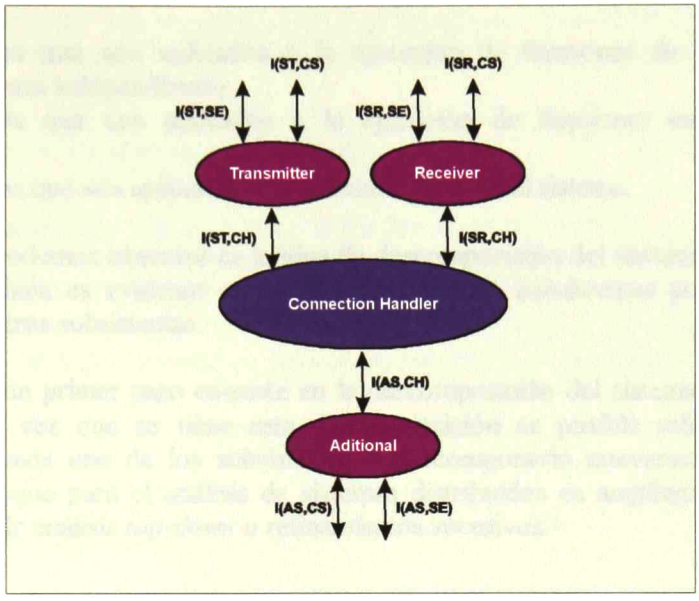


Figura 2-5 Esquema de una conexión

2.4 Diseño de sistemas telefónicos

El diseño de sistemas telefónicos es difícil dadas sus características. Por lo tanto, se vuelve necesario el uso de metodologías y herramientas adecuadas. En particular, el uso de técnicas de descripción formal[1] facilitan el diseño de este tipo de sistemas.

El uso de técnicas de descripción formal está bien justificado para sistemas que tienen las siguientes características[*Turner93*]:

- Complejos
- Concurrentes
- Críticos
- Seguros
- Estandarizados

Las técnicas de descripción formal producen especificaciones libres de ambigüedad, completas y consistentes. Las especificaciones son la base de donde se derivan los diseños.

[1] Formal Description Techniques

De manera general, podemos distinguir tres clases de requerimientos en un sistema telefónico:

1) Requerimientos que son aplicados a la ejecución de funciones de los subsistemas (vistos de manera independiente).
2) Requerimientos que son aplicados a la ejecución de funciones entre subsistemas relacionados.
3) Requerimientos que son aplicados a la ejecución de todo el sistema.

Lo primero que podemos observar es la idea de descomposición del sistema en subsistemas funcionales. También es evidente que cada uno de estos subsistemas puede o no estar relacionado con otros subsistemas.

De esta manera, un primer paso consiste en la descomposición del sistema en subsistemas funcionales. Una vez que se tiene esta descomposición es posible refinarla, es decir, podemos tomar cada uno de los subsistemas y descomponerlo nuevamente(Figura 2-6). Este tipo de enfoque para el análisis de sistemas distribuidos es ampliamente utilizado y recibe el nombre de análisis *top-down* o refinamientos sucesivos.

2.4.1 Análisis top-down

Siguiendo este enfoque, primero se describe el sistema como un conjunto de procesos que representan diferentes objetos, luego cada proceso resultante es descompuesto en subprocesos. El proceso de refinamiento del sistema es repetido hasta que terminamos con entidades simples que no pueden ser descompuestas. Cada paso del proceso de refinamiento representa un nivel de abstracción de la descripción formal del sistema.

Este método de refinamientos sucesivos ayuda al diseñador a lograr una descripción más clara del comportamiento de cada componente por separado, y a mostrar de una manera más estructurada el comportamiento de todo el sistema.

El proceso de diseño consiste en un conjunto de fases. Este proceso empieza con la descripción más abstracta del sistema y termina cuando el sistema ha sido construido. Los pasos que definen el proceso son identificados como:

- Fase de captura de requerimientos: produce un documento de requerimientos
- Fase arquitectural: define y describe formalmente la arquitectura de sistemas y subsistemas considerados
- Fase de implementación: produce una descripción formal de la implementación
- Fase de realización: produce el sistema
- Fase de pruebas: verifica las propiedades y funcionamiento del sistema

Un paso de refinamiento transforma una descripción del sistema en una descripción más detallada mediante decisiones de diseño. Cada una de esas descripciones representa un diferente nivel de abstracción. Un ejemplo podemos observarlo en el diseño de una red, en

el cual los pasos de refinamiento consisten en especificar el servicio provisto por una capa, para ser implementado por un protocolo más el servicio subyacente[*Vissers95*]. Este proceso es llevado de manera iterativa hasta que todas las capas han sido diseñadas.

2.4.2 Estructura de un paso de refinamiento

Una descripción precisa de una fase de refinamiento es necesaria antes de dar una guía de cómo llevar a cabo el proceso de desarrollo. El modelo de una fase de refinamiento que se da aquí es concebido con suficiente generalidad para permitir diferentes métodos. Es compatible con diferentes modelos de desarrollo de software incluyendo el modelo de cascada, prototipeo rápido y modelo en espiral[*Boehm88*][*Davis88*][*ESA87*].

Una fase de refinamiento es descompuesta en tareas concretas:

- Diseño formal
- Valoración
- Implementación y prototipeo

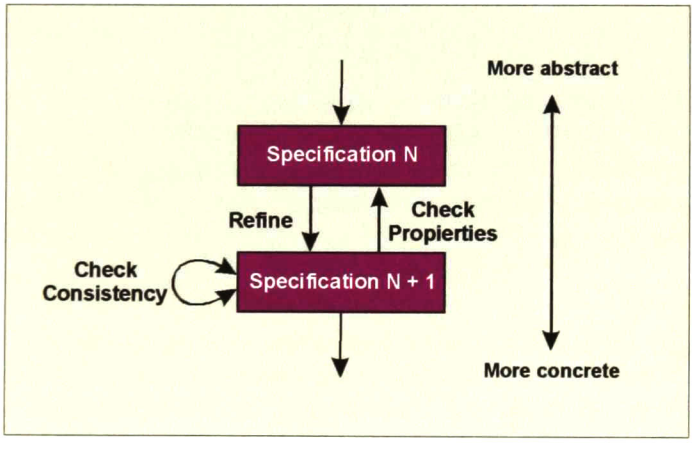


Figura 2-6. Enfoque de refinamientos sucesivos para el análisis de sistemas distribuidos

2.5 Comentarios finales

Como puede observarse los sistemas telefónicos presentan una gran complejidad. Debido al número de elementos que interactúan, es necesario el uso de metodologías de diseño que hagan manejable dicha complejidad.

El enfoque top-down utiliza el método de refinamientos sucesivos para producir especificaciones que describen el comportamiento de cada componente del sistema, al

mismo tiempo que muestra estructuradamente su comportamiento global. En el siguiente capítulo abordaremos el tema de la telefonía en Windows®, describiendo su arquitectura y la función de cada uno de sus componentes.

3

Arquitectura de Telefonía en Windows®

Una mente que se expande por una idea nueva, nunca retorna a su dimensión original.

Oliver Wendell Holmes.

3.1 Introducción

Convencionalmente las aplicaciones telefónicas se desarrollan para piezas particulares de hardware. Las aplicaciones interactúan con dicho hardware mediante interfaces propietarias[*ALGO97*]. Esto es, interfaces que dependen de las características y capacidades específicas de cada pieza de hardware. La naturaleza propietaria de estas interfaces dificulta la tarea de los desarrolladores de software ya que a éstos les interesa que sus aplicaciones funcionen en la mayor diversidad de hardware posible.

El modelo de proveedores de servicio como intermediarios, es una de las propuestas mejor aceptada y más ampliamente utilizada para eliminar el problema de las interfaces propietarias. Una implementación de este modelo que goza de mucha difusión es la Telefonía en Windows®. En este capítulo se presentarán sus conceptos básicos de una manera muy general. Estos conceptos son importantes ya que de ellos se desprenden consideraciones estructurales que serán manejadas en el resto de la presentación.

3.2 Arquitectura

La telefonía en Windows® fue creada por Microsoft e Intel como una extensión del sistema operativo Windows®, siguiendo el modelo de servicios abiertos WOSA[1][*Microsoft98a*].

El modelo WOSA proporciona un marco de trabajo en el cual las aplicaciones de telefonía pueden acceder de manera transparente a los recursos telefónicos disponibles en el

[1] Windows Open Service Architecture

sistema(Figura 3-1). Para ello, WOSA define una arquitectura genérica que se compone de tres capas estandarizadas de software:

- La interfaz de programación TAPI (Telephony Application Programming Interface) que define como una aplicación puede acceder a la información, recursos y servicios telefónicos del sistema.
- La interfaz de proveedores de servicio TSPI (Telephony Service Provider Interface) que especifica como el sistema operativo puede acceder a los manejadores de dispositivo propietarios.
- La interfaz TAPI/TSPI conocida como "*thunk layer*" que se encarga de mapear dinámicamente las solicitudes de servicios de las aplicaciones en solicitudes para los manejadores de dispositivo propietarios.

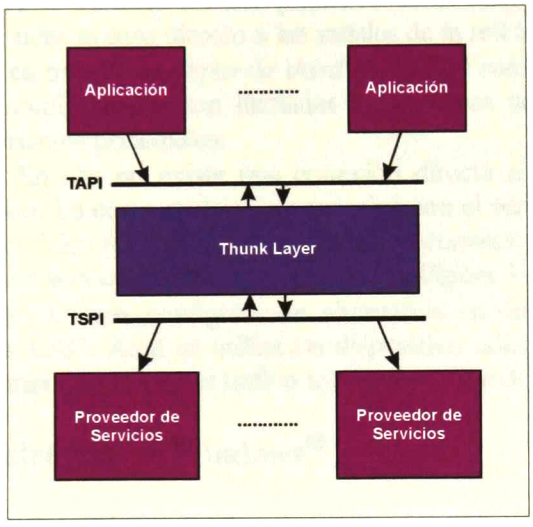


Figura 3-1. Arquitectura de servicios telefónicos basada en WOSA

El uso de dichas capas estandarizadas de software proporciona ventajas a los desarrolladores tanto de software como de hardware.

- Para los desarrolladores de software, porque sus aplicaciones no se enfocan a un modelo de hardware particular. Por el contrario éstas funcionarán con cualquier hardware compatible con TAPI. De esta manera se tiene una independencia entre el software, el harware específico y sus manejadores.
- Para los desarrolladores de hardware, porque sus productos funcionarán con todo el software compatible con TAPI que se desarrolle. Así, no tienen que persuadir a los desarrolladores de software de escribir aplicaciones especialmente para su interfaz propietaria.

3.3 Escenarios de aplicación de TAPI

Existen diferentes configuraciones en las que puede ser usado TAPI. Estas dependen de la manera como se conecta la computadora a la red telefónica. TAPI soporta dos tipos de configuración[*UNISYS97*]:

- La primera se ajusta al concepto de *first party call control*. Tiene dos variantes principales:
  - Centrado en el teléfono: Es este escenario, el teléfono es conectado a la línea telefónica y a la computadora. Aquí, el aparato telefónico es el dispositivo primario en el sistema(Figura 3-2.a).
  - Centrado en la computadora: En este escenario, la computadora es conectada a la red y al aparato telefónico. La computadora es el dispositivo primario de estos sistemas ya que tiene acceso directo a las señales de la red telefónica(Figura 3-2.b).
- La segunda se acerca más al concepto de *third party call control*. Las aplicaciones que se ajustan a esta configuración son llamadas aplicaciones servidoras y funcionan en redes. Tiene dos variantes principales:
  - *Switch-to-host*: En ella no existe una conexión directa entre la computadora y el aparato telefónico. La computadora se comunica con el servidor a través de la LAN[2] para solicitar servicios telefónicos. El servidor se conecta con la red telefónica y se encarga de dar los servicios que le son solicitados(Figura 3-2.c).
  - Servidor de voz: Es una configuración alternativa en donde el tráfico de voz es llevado sobre la LAN. Aquí se utiliza un dispositivo adicional llamado servidor de voz que se encarga de inyectar el tráfico telefónico a la red(Figura 3-2.d).

3.4 Servicios de telefonía en Windows®

Las facilidades telefónicas que define TAPI son clasificadas en cuatro niveles de servicio(Figura 3-3):

- Servicios de telefonía asistida: Este nivel de servicio permite incorporar facilidades de marcación a aplicaciones que no fueron diseñadas con esa capacidad, como procesadores de texto y hojas de cálculo.

[2] Local Area Network

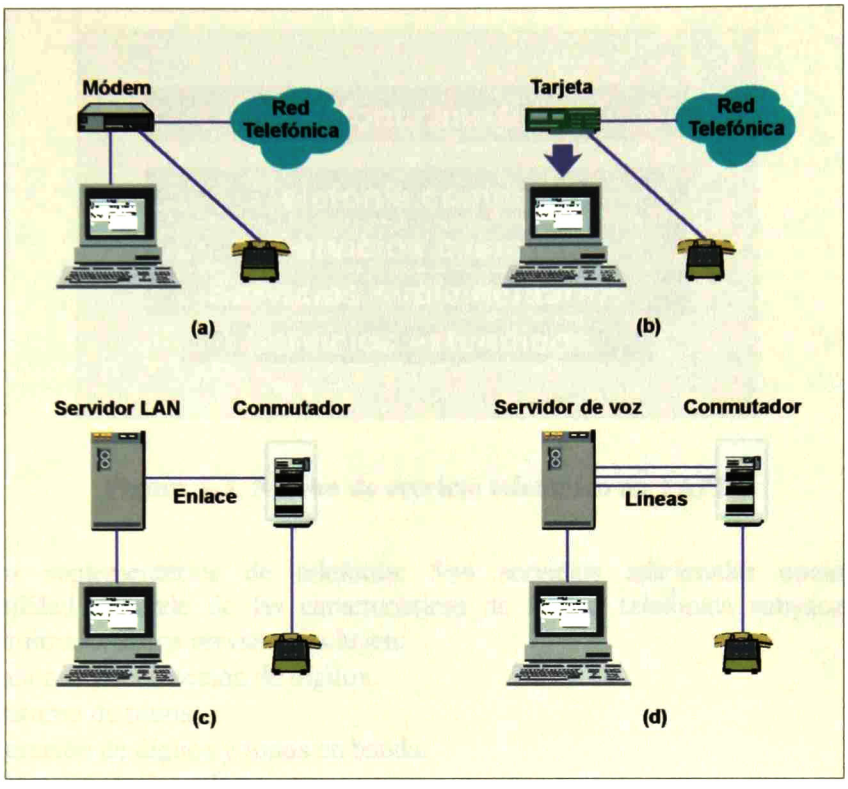


Figura 3-2. Ambientes telefónicos soportados por TAPI

- Servicios básicos de telefonía: Los servicios de telefonía básica corresponden con las funciones tradicionales de POTS[3]. Estos servicios deben ser provistos por todos los proveedores de servicio telefónico e incluyen:
  - Inicialización y terminación.
  - Negociación de versiones de TAPI.
  - Apertura y cierre de líneas.
  - Reporte de capacidades.
  - Notificación del status de los dispositivos.
  - Configuración.
  - Traducción de direcciones.
  - Realizar llamadas.
  - Responder llamadas entrantes.
  - Notificación de estados de la llamada y eventos telefónicos.
  - Terminación de llamadas.

[3] Plain Old Telephone System

Figura 3-3. Niveles de servicio telefónico en TAPI

- Servicios suplementarios de telefonía: Son servicios adicionales opcionales, su disponibilidad depende de las características de la red telefónica subyacente y del equipo utilizado. Estos servicios incluyen:
  - Monitoreo y recolección de dígitos.
  - Monitoreo de tonos.
  - Generación de dígitos y tonos en banda.
  - Redirección de llamadas.
  - Características encontradas en los sistemas PBX[4] tales como *hold*, *transfer*, *conference* y *park*.
- Servicios extendidos de telefonía: Corresponden a aquellas extensiones a la API hechas para un dispositivo específico. TAPI sólo define los mecanismos mediante los cuales se pueden hacer extensiones, por lo que la definición del comportamiento se especifica completamente por el proveedor de servicios telefónicos.

3.5 Conceptos básicos

TAPI fue diseñado basado en tres conceptos básicos: las clases de dispositivos, las direcciones y las llamadas[*D'Hooge95*]. Estos conceptos se encuentran íntimamente relacionados con el poder y versatilidad de TAPI para adaptarse a diferentes ambientes (Figura 3-4). En las siguientes secciones se explicarán con detalle cada uno de ellos.

[4] Private Branch Exchange

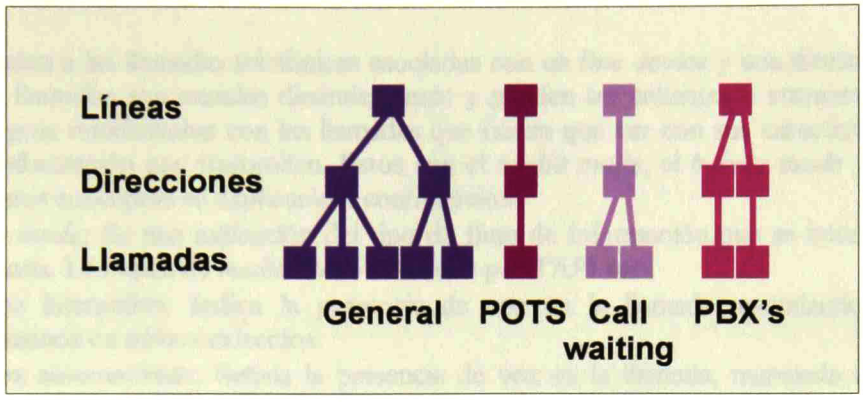


Figura 3-4. Relaciones entre las líneas, las direcciones y las llamadas

3.5.1 Clases de dispositivos

Las clases de dispositivos son grupos de dispositivos físicos relacionados, a través de los cuales las aplicaciones envían y reciben la información que constituye una llamada. Cada clase de dispositivo tiene asociado un nombre que la identifica, una interfaz de programación y una serie de comandos que pueden ser usados para comunicarse con otros dispositivos pertenecientes a la clase.
TAPI define dos clases de dispositivos:

- *Line Devices*: Es la abstracción de un dispositivo de línea física, tal como un módem, una tarjeta de fax o una tarjeta ISDN[5] que se encuentran conectados a una línea telefónica.
- *Phone Devices*: Es la abstracción de un aparato telefónico. Puede ser usado para crear teléfonos "virtuales" utilizando una variedad de dispositivos como micrófonos y bocinas.

3.5.2 Direcciones

Cada *line device* tiene asociada una o más direcciones. Una dirección corresponde a un número en el directorio telefónico. Una vez que se han asociado direcciones a los *line devices* correspondientes, TAPI asigna un identificador de dirección a cada una de éstas. Usualmente existe una dirección por línea, con las siguientes excepciones:

- Múltiples direcciones en POTS: Aquí la asignación de direcciones múltiples a una sola línea existe únicamente en sistemas que soportan timbrado distintivo(*distinctive ringing*) o que están conectados a una troncal DID(*Direct Inward Dialing*).
- Múltiples direcciones en ISDN: ISDN fue diseñado para permitir la existencia de múltiples direcciones de manera simultánea proporcionando múltiples canales.

[5] Integrated Services Digital Network

3.5.3 Llamadas

Corresponden a las llamadas telefónicas asociadas con un *line device* y una dirección en esa línea. Las llamadas son creadas dinámicamente y pueden ser salientes o entrantes. Existen tres conceptos relacionados con las llamadas que tienen que ver con sus características y la clase de información que transmiten. Estos son el *media mode*, el *bearer mode* y el *media stream*. Estos conceptos se explicarán a continuación:

- *Media mode*: Es una indicación del tipo de flujo de información que se intercambia en la llamada. Los tipos de *media mode* definidos por TAPI son:
  - Voz interactiva: Indica la presencia de voz en la llamada, asumiendo que hay humanos en ambos extremos.
  - Voz automatizada: Señala la presencia de voz en la llamada, manejada localmente por una aplicación automatizada.
  - Datamodem: Se utiliza cuando se transfieren datos a través de un modem.
  - Fax G3: Un fax del grupo 3 es enviado o recibido.
  - Fax G4: Un fax del grupo 4 es enviado o recibido.
  - Datos digitales: Datos digitales son enviados o recibidos.

- *Bearer mode:* Es una indicación de la calidad del servicio que se espera de la conexión telefónica proporcionada por la red. Los tipos de *bearer mode* básicos definidos por TAPI son:
  - Voz: Indica un servicio de voz analógica de 3.1 KHz. Soporta los *media modes* de fax y modem.
  - *Speech*: Corresponde a la transmisión de *speech* G.711[*ITUG711*] en la llamada.
  - Datos: Se refiere a la transferencia de datos sin restricción de formato en la llamada.

- *Media stream:* Es el flujo actual de información que se intercambia en una llamada. TAPI no provee de manera directa acceso al flujo de información. Para acceder al flujo de información, las aplicaciones deben utilizar otro grupo de APIs como la *Media Control Interface API*[*Microsoft98*].

3.6 Comentarios finales

Como pudo observarse, la Telefonía en Windows® define una arquitectura en la cual se hace una clara separación de responsabilidades. Esta separación permite resolver el problema de las interfaces propietarias.

Para este trabajo resulta de particular importancia la interfaz TSPI. Dicha interfaz es implementada como un *proveedor de servicios*. Estos proveedores son los encargados de proporcionar los servicios telefónicos que serán accedidos por las aplicaciones. En el siguiente capítulo se hace una presentación de su estructura y sus características más importantes.

4

Interfaz de Programación para Proveedores de Servicios de Telefonía

El lector avanzado que salta partes que le son demasiado elementales puede perder más que el lector menos avanzado que salta partes que le son demasiado complejas.

George Pólya. Mathematics and Plausible Reasoning.

4.1 Introducción

La telefonía en Windows® es una propuesta para resolver el problema de compatibilidad originado por las interfaces propietarias en sistemas telefónicos. Uno de los componentes que intervienen en la resolución de dicho problema es el proveedor de servicios telefónicos. Los proveedores de servicio permiten al sistema operativo acceder a los servicios telefónicos propietarios para a su vez ofrecerlos a las aplicaciones de una manera estándar.

En este capítulo se presentará la interfaz para proveedores de servicio de telefonía en Windows® Abordaremos los conceptos principales de la interfaz, así como detalles de su arquitectura y la comunicación entre componentes.

4.2 Conceptos generales

TSPI es la interfaz de programación para proveedores de servicio de telefonía(*Telephony Service Providers o TSP's*). TSPI es una especificación de cómo puede Windows® acceder a los manejadores de dispositivo propietarios para hardware de telefonía.

TSPI define un conjunto de funciones, estructuras y eventos utilizados para modelar e implementar los TSP's[*Microsoft98b*]. En las siguientes secciones se explicarán a detalle estos conceptos.

4.2.1 Funciones

Los TSP's implementan funciones para atender las solicitudes de las aplicaciones. El significado y el modo de operación de estas funciones es definido estrictamente por TSPI. Esto quiere decir que cada TSP puede implementar las funciones de diferente manera pero siempre apegándose a la definición de TSPI. Dependiendo de la solicitud, las funciones pueden operar de dos formas diferentes: síncrona y asíncronamente. A continuación se explicarán las características de cada una de ellas.

4.2.1.1 Funciones síncronas

Las funciones síncronas ejecutan completamente todas las operaciones necesarias para atender una solicitud antes de retornar el control a la aplicación(Figura 4-1).

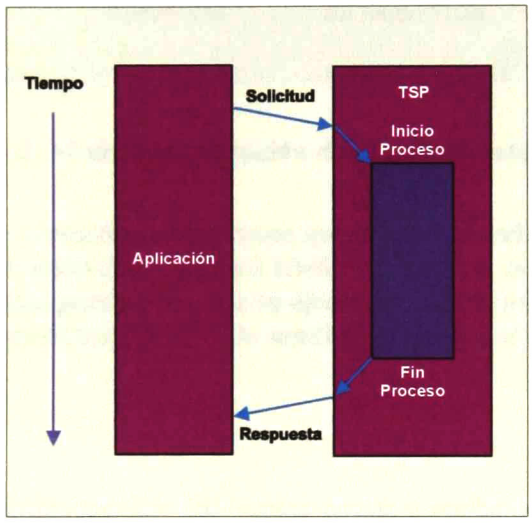


Figura 4-1. Modelo de ejecución de funciones síncronas

En general, las funciones síncronas no demoran mucho tiempo ejecutando su proceso ya que su finalidad es regresar el control a la aplicación lo más rápido posible. Ejemplos de estas funciones son aquéllas destinadas a obtener información sobre las capacidades de una línea o el estado de una llamada.

4.2.1.2 Funciones asíncronas

Las funciones asíncronas regresan el control a la aplicación antes de ejecutar completamente las operaciones para atender una solicitud, notificando después la terminación de las mismas(Figura 4-2).

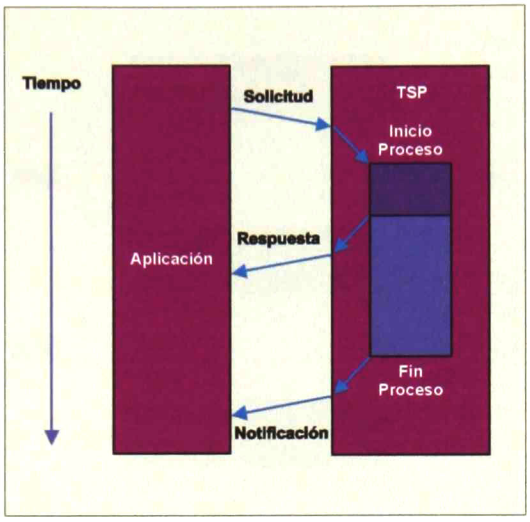


Figura 4-2. Modelo de ejecución de funciones asíncronas

Este tipo de funciones se especializa en procesos que demoran mucho tiempo ejecutándose, como por ejemplo la marcación de un número telefónico. En este caso se regresa el control a la aplicación para que ésta pueda continuar su ejecución normal mientras el TSP realiza la marcación; una vez terminada ésta, el TSP lo notifica a la aplicación para que lleve a cabo las acciones pertinentes.

4.2.2 Estructuras

Las estructuras son utilizadas por los TSP's para enviar y recibir los datos asociados con una solicitud de servicios telefónicos. TSPI define tanto el significado de cada miembro de las estructuras, como la responsabilidad de actualizar el valor de los mismos. Dicha responsabilidad se divide entre TAPI y los TSP's, es decir, algunos de los miembros de las estructuras son actualizados automáticamente por TAPI mientras que otros son actualizados por los TSP's.

Frecuentemente los miembros de las estructuras varían en número y longitud en dependencia de las solicitudes. Entonces, las estructuras consisten de un conjunto de miembros de longitud fija seguidos de cero o más miembros de longitud variable.

Cuando una aplicación solicita información telefónica, por ejemplo las capacidades de una línea, el llenado de la estructura con la información se lleva a cabo en diferentes niveles:

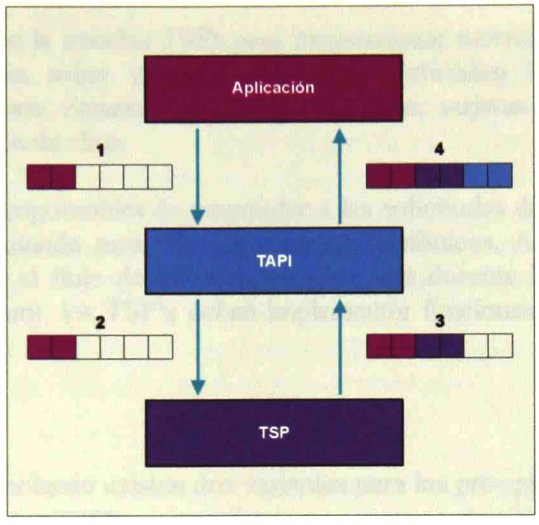


Figura 4-3. Llenado de estructuras durante una solicitud de información telefónica

La aplicación crea una instancia de la estructura adecuada para almacenar dicha información(Figura 4-3.1).

- El TSP recibe la estructura de TAPI y actualiza los miembros que le corresponden con la información solicitada(Figura 4-3.1 y Figura 4-3.2).
- TAPI completa la estructura con los miembros que le corresponden y la envía a la aplicación(Figura 4-3.3).

4.2.3 Eventos

Los eventos son acciones que modifican el estado de los recursos telefónicos: dispositivos, direcciones y llamadas.

TSPI define dos clases de eventos:

- Eventos solicitados: Son generados por la aplicación que controla los recursos telefónicos. En general, son resultado directo de una solicitud.
- Eventos no solicitados: No son resultado directo de solicitudes de las aplicaciones. Estos son provocados por cambios en conmutadores, la red telefónica o por acciones de abonados remotos.

4.3 Proveedores de servicio

Los TSP's implementan la interfaz TSPI para proporcionar servicios a las aplicaciones que requieran comunicación sobre una red telefónica. Extienden los servicios telefónicos soportados por hardware especializado, como módems, tarjetas ISDN, conmutadores o incluso una combinación de ellos.

Todos los TSP's son responsables de responder a las solicitudes de servicios telefónicos de las aplicaciones, controlando para ello los recursos telefónicos. Además, son responsables de controlar y accesar el flujo de información generado durante la llamada. Para manejar dicho flujo(media stream), los TSP's deben implementar funciones adicionales no definidas por TSPI.

4.3.1 Arquitectura

Arquitectónicamente hablando existen dos variantes para los proveedores de servicio:

- Centralizados: Los TSP's centralizados constan de una librería de enlace dinámico(DLL) que implementa las funciones TSPI y controla el hardware que soporta los servicios(Figura 4-4.a).
- Descentralizados: Los TSP's descentralizados también se implementan como librerías de enlace dinámico pero se apoyan de un módulo adicional para el control del hardware conocido como VxD(Virtual Device Driver)(Figura 4-4.b). La razón principal de este enfoque es puramente técnica, ya que los VxD's permiten el control en tiempo real del hardware mientras que las librerías de enlace dinámico no.

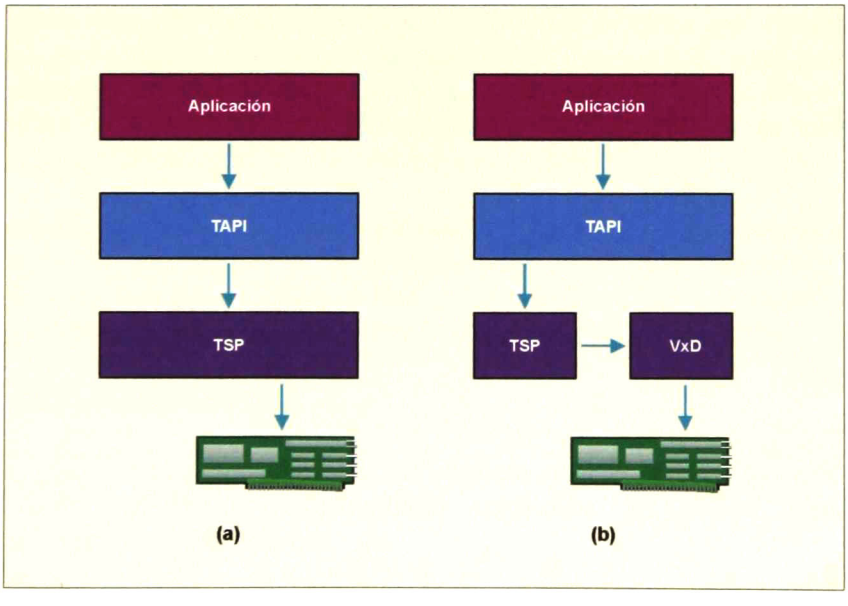


Figura 4-4. Arquitecturas centralizada y descentralizada para proveedores de servicio

4.4 Comentarios finales

El proveedor de servicios de telefonía es un módulo de mucha importancia dentro de la arquitectura de telefonía sobre Windows® Este módulo permite acceder los servicios provistos por los dispositivos telefónicos. En este capítulo se presentó una descripción de los conceptos principales involucrados en el desarrollo de proveedores de servicio. Estos conceptos son de gran importancia para la implementación final del proveedor de servicios en un ambiente Windows® Arquitectónicamente se definieron dos clases de proveedores de servicios: los centralizados y los descentralizados; esto es de gran importancia ya que tomaremos el modelo descentralizado como base para la especificación.

En el siguiente capítulo describiremos el funcionamiento, susceptible de formalizar, del proveedor de servicios de telefonía diseñado para este trabajo.

5

Descripción Informal del Proveedor de Servicios de Telefonía

Un buen modelo representa una abstracción bien balanceada de una situación práctica; no muy lejos pero tampoco muy cerca de la realidad.

Arto Salomaa. Computation and Automata.

5.1 Introducción

Como se vio en los capítulos anteriores, la arquitectura impuesta por la telefonía sobre Windows presenta algunas diferencias con respecto al modelo tradicional. Estas diferencias provienen de la inclusión de módulos de software que habilitan a una computadora para actuar como una terminal telefónica. Una vez que habilitamos a la computadora de esta manera tenemos la posibilidad de utilizar su capacidad para proveer facilidades extendidas a los usuarios. En este trabajo especificaremos el comportamiento de uno de dichos módulos: el proveedor de servicios para telefonía.

En este capítulo iniciaremos con la especificación del proveedor de servicios de telefonía. Presentaremos su arquitectura y la relación que existe con el modelo tradicional. Definiremos cada uno de los componentes y las relaciones entre cada uno de ellos. Y finalmente comenzaremos con el análisis funcional, susceptible de formalizar.

5.2 Modelo arquitectónico

El objetivo del proveedor de servicios de telefonía es implementar las funciones TSPI necesarias para soportar los requerimientos telefónicos del sistema operativo y las aplicaciones. Estos servicios incluyen:

- El registro e inicialización del proveedor de servicios.
- Petición de información acerca del estado de los recursos telefónicos.
- Ejecución de operaciones telefónicas.

De ahora en adelante, nos referiremos al proveedor de servicios como *CHSP*(Communication Helper Service Provider). Esta denominación se debe a que el *CHSP* forma parte del sistema *Communication Helper*. El *Communication Helper* es una aplicación CTI que consta de varios componentes(Figura 5-1): una aplicación de usuario descrita en[*Carranza01*] y en[*Padilla00*], el *CHSP* descrito en este trabajo, y finalmente, el hardware del sistema y el manejador que lo controla denominado *VHELPERD*[*Figueroa01*].

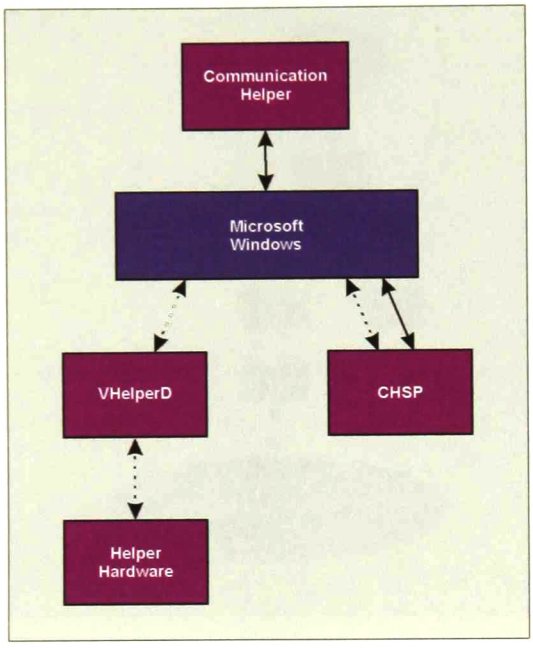


Figura 5-1. Relación entre el *CHSP* y los otros módulos del sistema *Communication Helper*

Arquitectónicamente hablando, el *CHSP* se apega al modelo descentralizado de proveedores de servicio. Este esquema implica que las señales generadas por el sistema telefónico(*TS*) no serán enviadas directamente a los teléfonos de los usuarios(*ES*). En su lugar, éstas serán interceptadas por el *VHELPERD* para luego ser transformadas en mensajes apropiados para el *CHSP*(líneas punteadas). Así mismo, es responsabilidad del *VHELPERD* mandar las señales generadas por los *ES* al sistema telefónico. Una vez que el *CHSP* ha recibido los mensajes provenientes del *VHELPERD*, los procesa siguiendo su propio modelo de control de la llamada. Este procesamiento origina una serie de mensajes y notificaciones tanto para el *VHELPERD* como para la aplicación de usuario(líneas sólidas). Los mensajes representan las peticiones de servicio telefónico que se envían de las aplicaciones hacia el *CHSP* y del *CHSP* hacia el *VHELPERD*. Las notificaciones por su parte, son enviadas por el *CHSP* hacia las aplicaciones para actualizar el estado de la

llamada. Las secuencias de mensajes y notificaciones necesarias para la ejecución de operaciones telefónicas serán descritas con detalle en las secciones siguientes. De esta manera quedan identificadas las interfaces de comunicación del *CHSP*: por un lado la interfaz hacia la aplicación de usuario, y por el otro la interfaz hacia el *VHELPERD*. La Figura 5-2 muestra una visión global del modelo tradicional(descrito en el capítulo 2) interactuando con la arquitectura de telefonía del *Communication Helper*(presentada en la Figura 5-1).

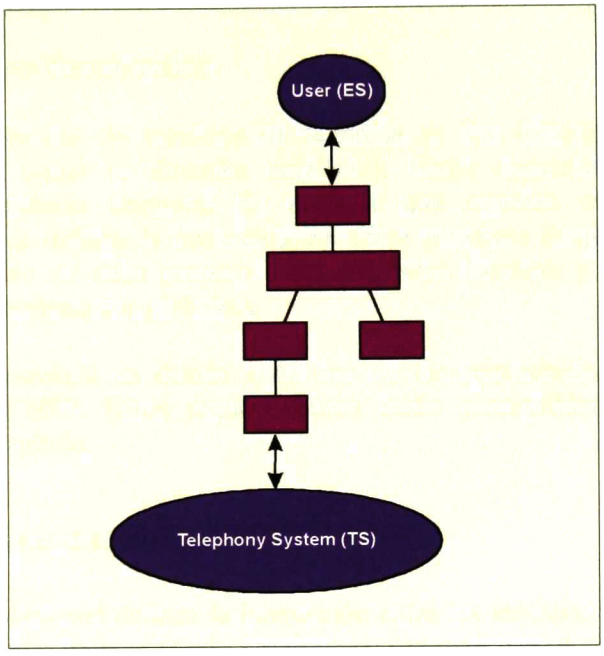


Figura 5-2. Visión del modelo tradicional de telefonía interactuando con el esquema de telefonía de Windows

Como puede observarse, es una arquitectura compleja en donde pueden generarse un gran número de interacciones y secuencias. Con la finalidad de hacer más manejable el problema, es necesario hacer algunas simplificaciones. En la siguiente sección estableceremos la arquitectura simplificada final que tomaremos como base para nuestra especificación.

5.3 Un modelo simplificado

En este trabajo se presenta la descripción de los servicios o facilidades que ofrece el *CHSP* Esto quiere decir que nos enfocaremos en él y en las interfaces que tiene hacia otros módulos. Para ello supondremos que el *VHELPERD* toma el lugar del *TS*. Esto significa que para el *CHSP* el *VHELPERD* será en origen y destino de todas las señales telefónicas. Esto puede hacerse sin pérdida de generalidad ya que el *VHELPERD* es en realidad una

interfaz que transforma las señales del *TS* en mensajes de software que son enviados al *CHSP*. De la misma manera la aplicación de usuario tomará el lugar de los *ES*. De nuevo, es posible hacer esto porque podemos ver a la aplicación de usuario como una caja negra a la cual y de la cual recibimos notificaciones y mensajes, que de otra manera recibiríamos de los *ES*.

Una vez definido el contexto en el cual el CHSP existe, podemos pasar a su definición funcional. Para ello presentaremos el conjunto de funciones que debe realizar el *CHSP*.

5.4 Requerimientos funcionales

En el diseño del proveedor de servicios de telefonía se han introducido tres tipos de requerimientos, tal y como se describe en[*Fls91*]. Estos requerimientos definen los principios de ordenamiento temporal de eventos que ocurren en las líneas. Los requerimientos entonces definen la comunicación entre procesos (requerimientos end-to-end); el comportamiento de cada proceso (requerimientos locales); y el comportamiento global del sistema (requerimientos globales).

A continuación se presentará en detalle cada uno de los requerimientos que definen el comportamiento del *CHSP*. Estos requerimientos serán susceptibles de especificación formal en el próximo capítulo.

5.4.1 Requerimientos end-to-end

Las requerimientos end-to-end definen la interacción entre los módulos dentro de la misma conexión. En esta sección se presentarán esas secuencias mediante el uso de escenarios de ejecución.

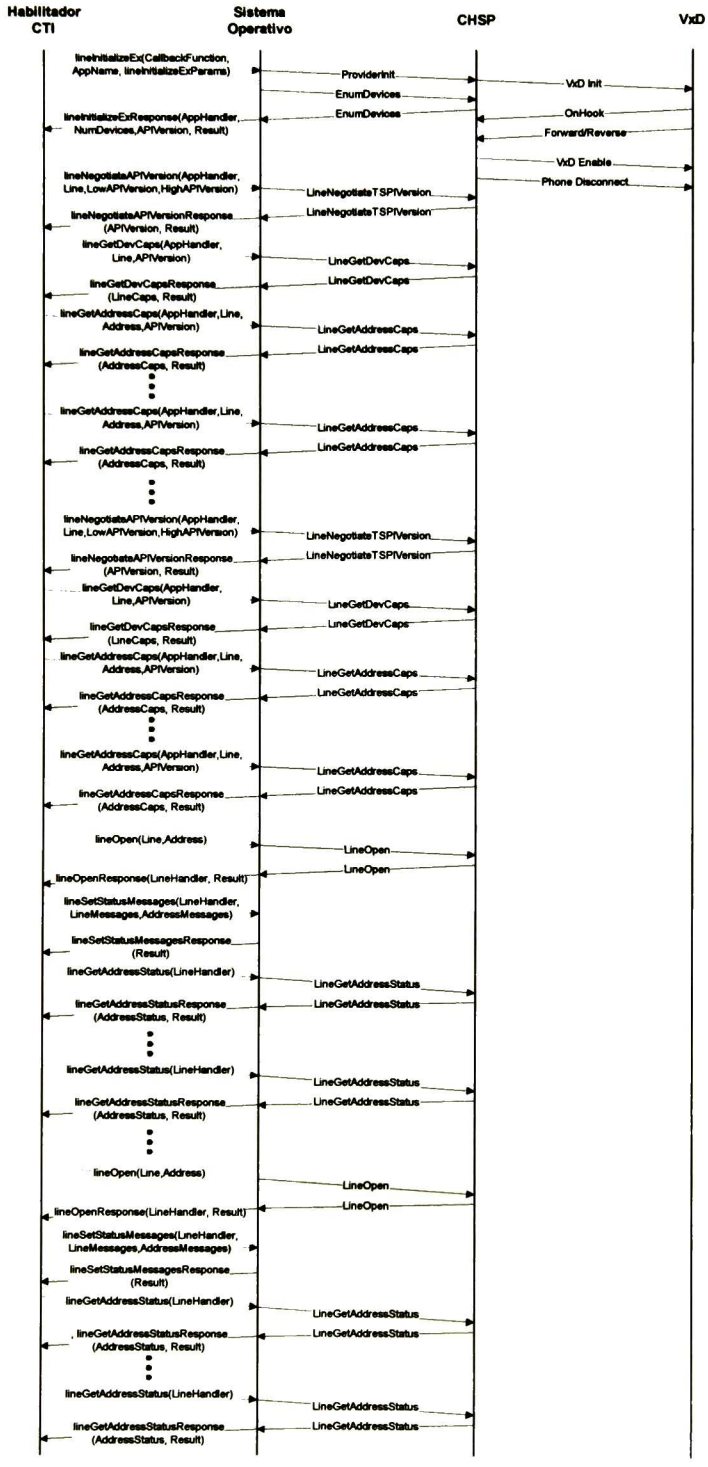
Como se vio anteriormente, el *Communication Helper* está integrado por varios componentes que interactúan entre sí(Figura 5-2). En este trabajo se enfatiza en la especificación del *CHSP*, por lo tanto los escenarios de ejecución se centrarán en ese módulo. Los escenarios de ejecución posibles se presentan en la Tabla 5-1.

Diagrama	Escenarios de ejecución
1	Inicialización del CHSP y del VxD
2	Llamada saliente desde software terminada por software
3	Llamada saliente desde software terminada desde el teléfono
4	Llamada saliente desde el teléfono terminada por software
5	Llamada entrante terminada por software
6	Llamada entrante terminada desde el teléfono

Tabla 5-1. Escenarios de ejecución del CHSP con respecto a otros módulos del Communication Helper

Diagrama 1

Inicialización del CHSP y del VxD (1/1)

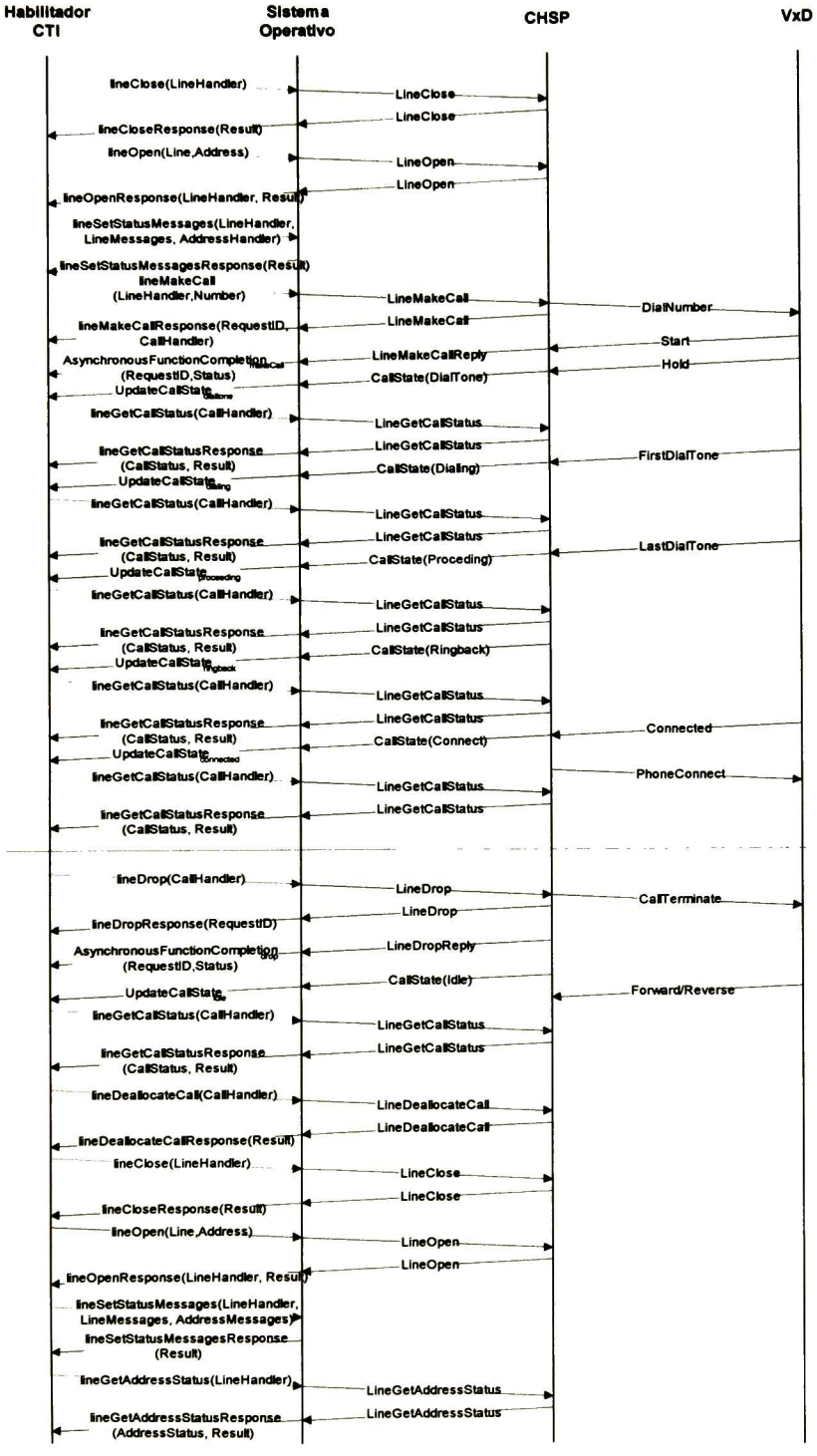


5.4.1.1 Inicialización del CHSP y del VxD

Descripción: El diagrama 1 muestra el escenario donde el Control Administrativo del *Communication Helper* solicita al Habilitador CTI (*HTT*) que se inicialice el uso de los line devices y las direcciones. Luego, el *CHSP* inicializa el *VHELPERD* para que se encargue de la recepción de señales provenientes del TS, e informa al sistema operativo que está listo. El sistema operativo regresa un handler con el que TAPI identifica a la aplicación, el número de line devices disponibles y la versión más actual de TAPI que soporta la aplicación. Enseguida, el *HTT* indica que para cada line device y cada dirección se obtengan las capacidades telefónicas y que se negocie la versión de TAPI con la que se manejarán. Por último, se configura cada dirección para que pueda recibir llamadas entrantes.

Diagrama 2

Llamada saliente desde software terminada por software (1/1)



5.4.1.2 Llamada saliente desde software terminada por software

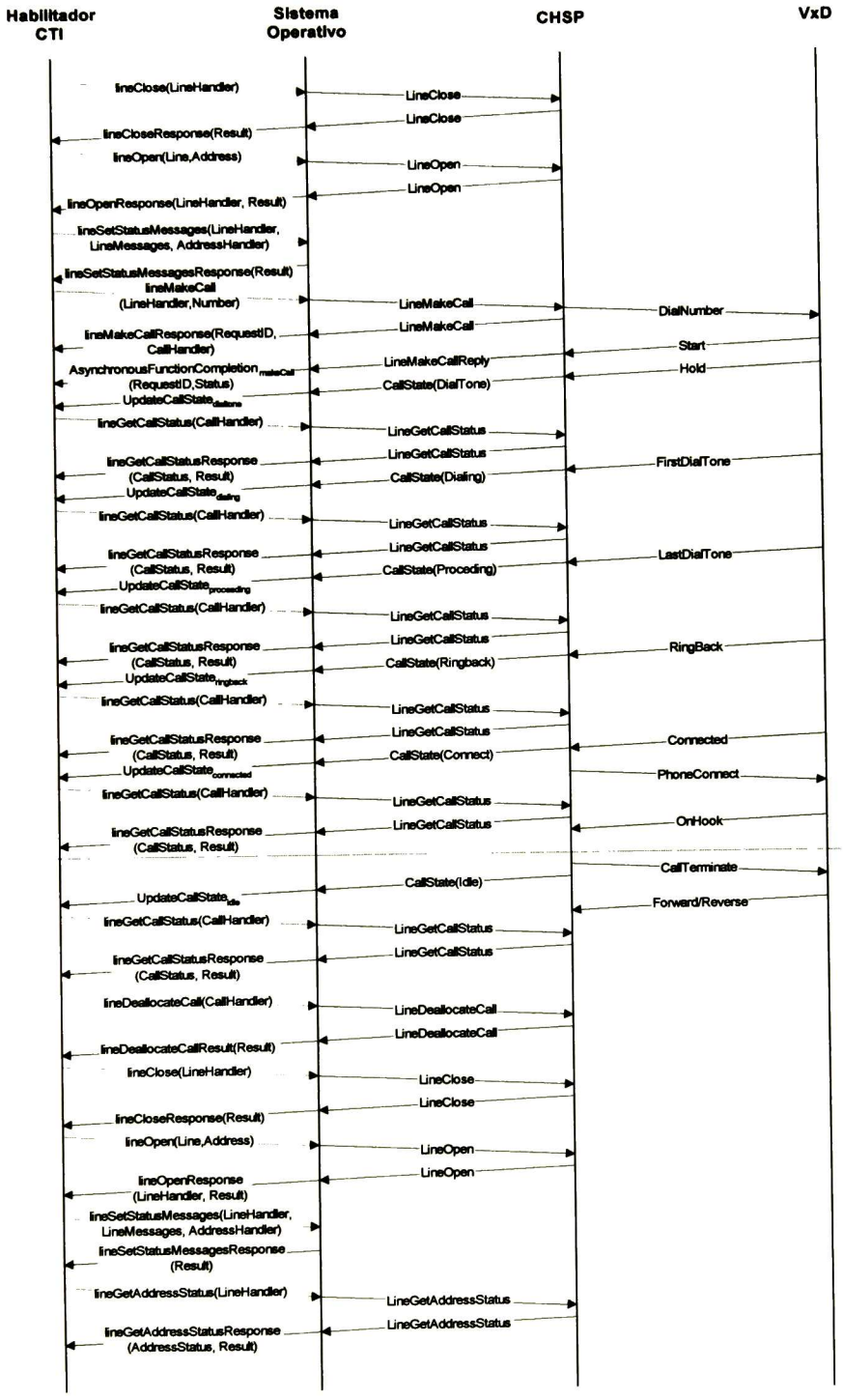
Descripción: La parte del diagrama 2 que se localiza sobre la línea punteada refleja las operaciones del sistema y los eventos que se realizan antes y durante el establecimiento de la llamada. El Control Administrativo indica al *HTT* que realice una llamada telefónica saliente y le informa la dirección por la cual realizar la llamada. El *HTT* reconfigura el handler de la dirección correspondiente para llevar a cabo una llamada y enseguida solicita al Sistema Operativo que realice la marcación del número.

El *CHSP* configura la línea para realizar la marcación y envía un mensaje al *VHELPERD* para que realice físicamente la marcación. El *CHSP* notifica al Sistema Operativo los estados por los que atraviesa la llamada. Cuando se llega al estado Connected, el Control Administrativo comienza la tarificación sobre esa llamada.

La parte del diagrama que se localiza bajo la línea punteada muestra lo que sucede cuando el Control Administrativo comunica al habilitador CTI que finalice la llamada telefónica. Este módulo transfiere la petición al Sistema Operativo para que éste la envíe al *CHSP*, revisando previamente si la llamada no ha sido finalizada aún por hardware. Si la llamada no ha sido terminada aún, se envía un mensaje al *VHELPERD* para que termine la conexión por esa línea. La terminación de la llamada se comunica al Sistema Operativo mediante el estado Idle. Enseguida, se libera el handler de TAPI para la llamada y los recursos asociados con ella. Por último, se configura de nuevo la dirección por la que se realizó la llamada para que pueda recibir llamadas entrantes.

Diagrama 3

Llamada saliente desde software terminada desde el teléfono (1/1)



5.4.1.3 Llamada saliente desde software terminada desde el teléfono

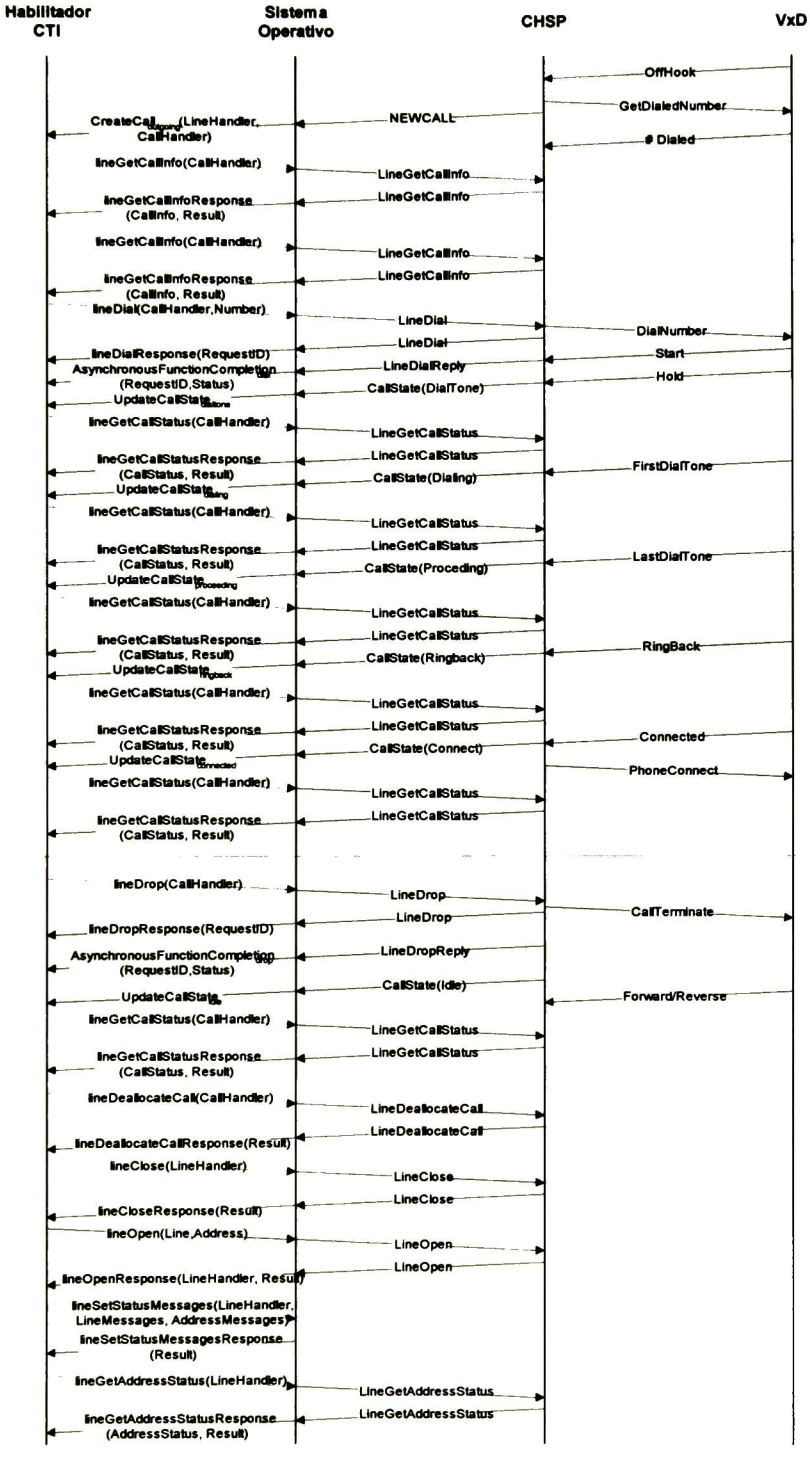
Descripción: El diagrama 3 nos muestra sobre la línea punteada refleja las operaciones del sistema y los eventos que se realizan antes y durante el establecimiento de la llamada. El Control Administrativo indica al *HTT* que realice una llamada telefónica saliente y le informa la dirección por la cual realizar la llamada. El *HTT* reconfigura el handler de la dirección correspondiente para llevar a cabo una llamada y enseguida solicita al Sistema Operativo que realice la marcación del número.

El *CHSP* configura la línea para realizar la marcación y envía un mensaje al *VHELPERD* para que realice físicamente la marcación. El *CHSP* notifica al Sistema Operativo los estados por los que atraviesa la llamada. Cuando se llega al estado Connected, el Control Administrativo comienza la tarificación sobre esa llamada.

La parte del diagrama que se localiza bajo la línea punteada muestra lo que ocurre cuando el *CHSP* notifica al *VHELPERD* que la llamada telefónica ha sido terminada por un teléfono. El *CHSP* lo notifica al Sistema Operativo mediante el estado Idle. El *VHELPERD* a su vez, libera físicamente la línea y la hace disponible para una nueva llamada. El *HTT* envía un mensaje al Control Administrativo para que finalice la tarificación. Enseguida, se libera el handler de TAPI para la llamada y los recursos asociados con ella. Por último, se configura de nuevo la dirección por la que se realizó la llamada para que pueda recibir llamadas entrantes.

Diagrama 4

Llamada saliente desde el teléfono terminada por software (1/1)



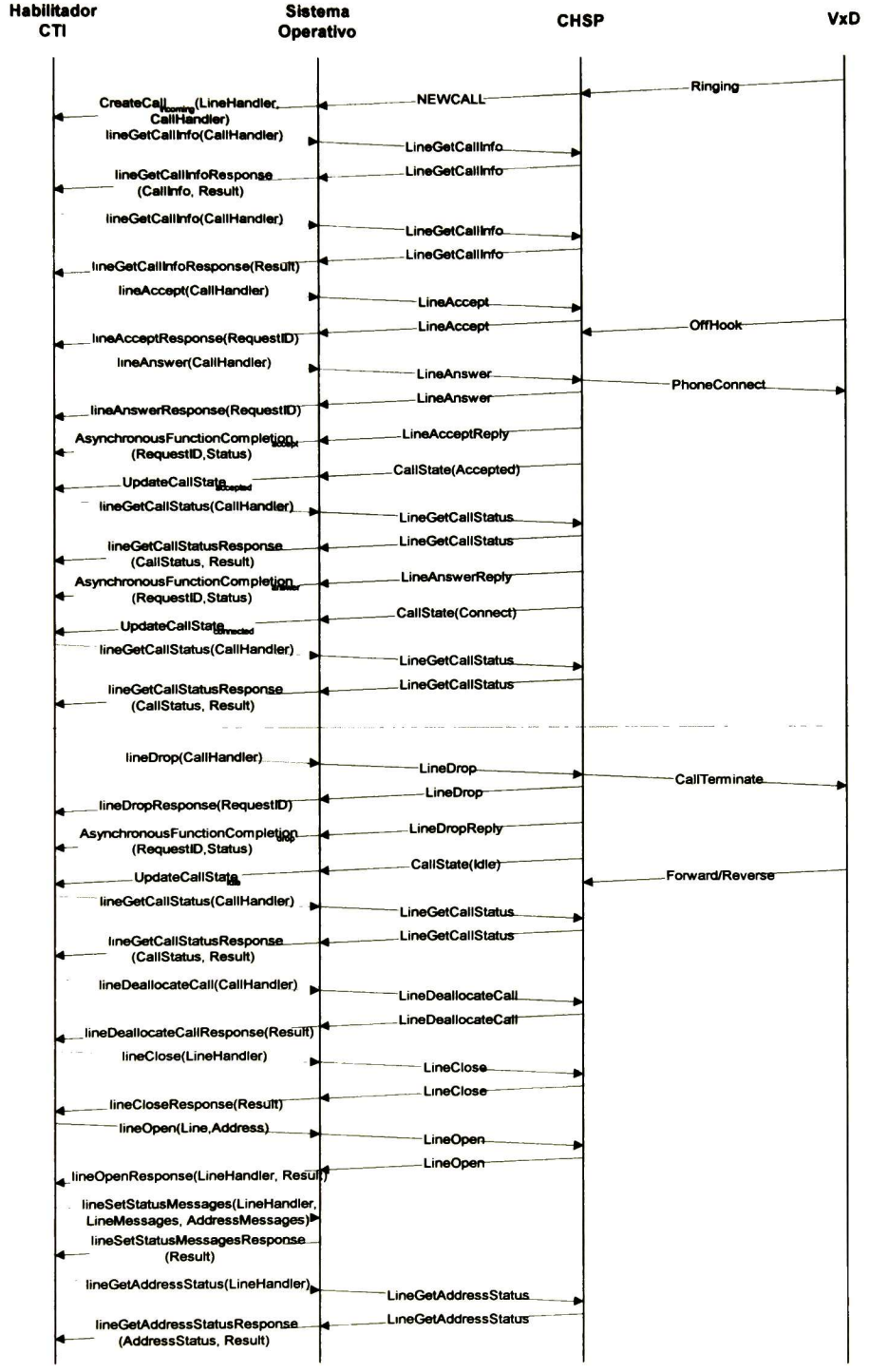
5.4.1.4 Llamada saliente desde el teléfono terminada por software

Descripción: La parte del diagrama 4 que se localiza sobre la línea punteada refleja las operaciones del sistema y los eventos que se realizan antes y durante el establecimiento de la llamada. El *VHELPERD* notifica al *CHSP* que se desea realizar una llamada desde el aparato telefónico. El *CHSP* envía el mensaje de nueva llamada al Sistema Operativo para que éste a su vez le envíe un handler al Habilitador CTI para que pueda procesar cualquier acción posterior sobre la misma. Además se indica el handler de la dirección por la cual se quiere hacer la llamada. El *CHSP* le notifica al Sistema Operativo los números marcados. Una vez que se tiene esta información, el Sistema Operativo los transfiere al *HTT* para que determine si la nueva llamada puede ser realizada. En caso afirmativo, el Habilitador CTI envía los mensajes correspondientes al Sistema Operativo. Cuando la llamada alcanza el estado de Connected, el *CHSP* lo notifica al Sistema Operativo para que el Control Administrativo inicie la tarificación sobre la llamada.

La parte del diagrama que se localiza bajo la línea punteada muestra lo que sucede cuando el Sistema Operativo comunica al *CHSP* que finalice la llamada telefónica. El *CHSP* envía un mensaje al *VHELPERD* para que éste termine la conexión y libere la línea. Cuando la llamada termina, el *CHSP* envía al Sistema Operativo el estado Idle. Enseguida, se libera el handler de TAPI para la llamada y los recursos asociados con ella. Por último, se configura de nuevo la dirección por la que se realizó la llamada para que pueda recibir llamadas entrantes.

Diagrama 5

Llamada entrante terminada por software (1/1)



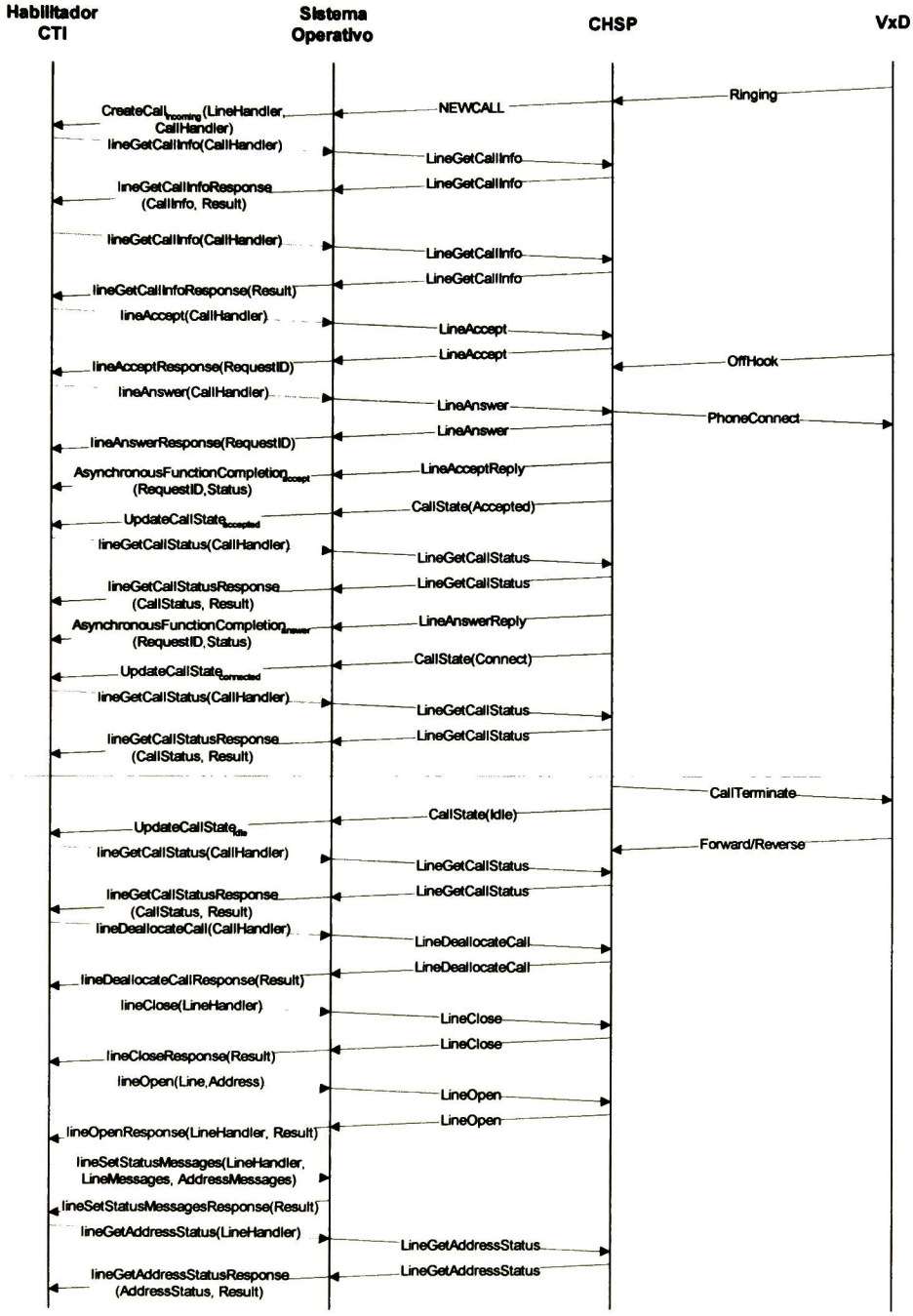
5.4.1.5 Llamada entrante terminada por software

Descripción: Sobre la línea punteada del diagrama 5 se reflejan las operaciones del sistema y los eventos que se realizan antes y durante el establecimiento de la llamada. El *VHELPERD* notifica al *CHSP* que hay una nueva llamada telefónica entrante. El Sistema Operativo recibe el mensaje de nueva llamada y se lo comunica al *HTT* que le solicita información adicional para la llamada, principalmente el callerID. Una vez que se tiene esta información, el *HTT* la sube al Control Administrativo para que determine si la nueva llamada puede ser aceptada y respondida. En caso afirmativo, el *HTT* envía los mensajes correspondientes al Sistema Operativo para que éste a su vez los transfiera al *CHSP*. Después el *CHSP* envía al *VHELPERD* el mensaje para que conecte la línea. Cuando la llamada alcanza el estado Connected, se le notifica al Sistema Operativo para que inicie la tarificación sobre la llamada.

La parte del diagrama que se localiza bajo la línea punteada muestra lo que sucede cuando el Sistema Operativo comunica al *CHSP* que finalice la llamada telefónica. El *CHSP* envía un mensaje al *VHELPERD* para que éste termine la conexión y libere la línea. Cuando la llamada termina, el *CHSP* envía al Sistema Operativo el estado Idle. Enseguida, se libera el handler de TAPI para la llamada y los recursos asociados con ella. Por último, se configura de nuevo la dirección por la que se realizó la llamada para que pueda recibir llamadas entrantes.

Diagrama 6

Llamada entrante terminada desde el teléfono (1/1)



5.4.1.6 Llamada entrante terminada desde el teléfono

Descripción: La parte del diagrama 6 que se localiza sobre la línea punteada refleja las operaciones del sistema y los eventos que se realizan antes y durante el establecimiento de la llamada. El *VHELPERD* notifica al *CHSP* que hay una nueva llamada telefónica entrante. El Sistema Operativo recibe el mensaje de nueva llamada y se lo comunica al *HTT* que le solicita información adicional para la llamada, principalmente el callerID. Una vez que se tiene esta información, el *HTT* la sube al Control Administrativo para que determine si la nueva llamada puede ser aceptada y respondida. En caso afirmativo, el Habilitador CTI envía los mensajes correspondientes al Sistema Operativo para que éste a su vez los transfiera al *CHSP*. Después el *CHSP* envía al *VHELPERD* el mensaje para que conecte la línea. Cuando la llamada alcanza el estado Connected, se le notifica al Sistema Operativo para que inicie la tarificación sobre la llamada.

La parte del diagrama que se localiza bajo la línea punteada muestra lo que ocurre cuando el *CHSP* notifica al *VHELPERD* que la llamada telefónica ha sido terminada por un teléfono. El *CHSP* lo notifica al Sistema Operativo mediante el estado Idle. El *VHELPERD* a su vez, libera físicamente la línea y la hace disponible para una nueva llamada. El *HTT* envía un mensaje al Control Administrativo para que finalice la tarificación. Enseguida, se libera el handler de TAPI para la llamada y los recursos asociados con ella. Por último, se configura de nuevo la dirección por la que se realizó la llamada para que pueda recibir llamadas entrantes.

5.4.2 Requerimientos locales

Las requerimientos locales controlan el ordenamiento temporal de eventos dentro del *CHSP*. En esta sección se presentarán esas secuencias mediante el uso de máquinas de estado. La máquina de estado que describe el comportamiento interno del *CHSP* se compone de las máquinas presentadas en Tabla 5-2.

Máquinas de estado
Inicialización
Establecimiento de llamadas desde software
Establecimiento de llamadas desde el teléfono
Establecimiento de llamadas software-teléfono
Evolución de llamada

Tabla 5-2. Máquinas de estado que conforman la máquina de estado del CHSP

5.4.2.1 Inicialización

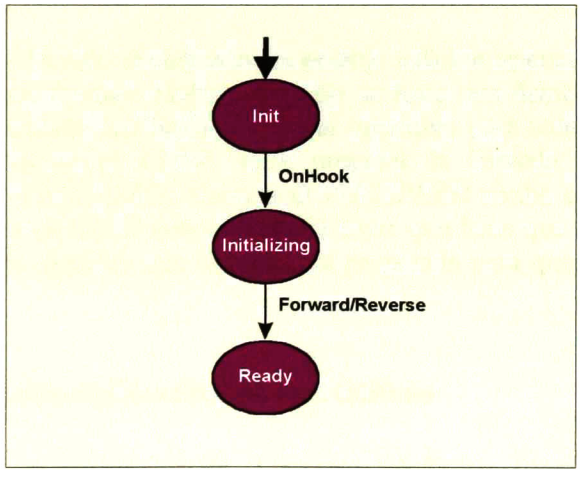


Figura 5-3. Máquina de estados de la inicialización del CHSP

La primera vez que se solicita un servicio telefónico, el Sistema Operativo levanta el proceso del *CHSP*, el cual a su vez corre el proceso del *VHELPERD*. Una vez que se tiene funcionando el *VHELPERD*, éste verifica que la terminal telefónica se encuentre colgada y envía el mensaje *OnHook* hacia el *CHSP*. Posteriormente revisa que la línea telefónica esté en servicio y se lo comunica al *CHSP* mediante el mensaje *Forward/Reverse* (Figura 5-3).

Los mensajes recibidos por el proveedor de servicios generan cambios en su estado interno y lo llevan hasta el estado *Ready*. Cuando el *CHSP* se encuentra en este estado, está listo para ejecutar operaciones telefónicas.

5.4.2.2 Establecimiento de llamadas desde software

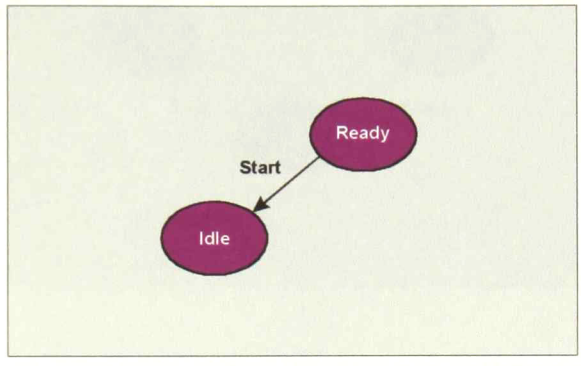


Figura 5-4. Máquina de estados del establecimiento de llamadas desde el software del Communication Helper

Una vez que el *CHSP* ha sido inicializado, es posible solicitar operaciones telefónicas desde el software del *Communication Helper*. Cuando se hace una llamada, el Habilitador CTI envía un mensaje *LineMakeCall* hacia el sistema operativo para solicitar el servicio. Este a su vez envía la petición al *CHSP*. Para preparar la llamada se envía un mensaje *DialNumber* hacia el *VHELPERD*. Cuando el *VHELPERD* recibe este mensaje prepara la línea y se lo comunica al *CHSP* con el mensaje *Start* que hace que el estado cambie hacia *Idle* (Figura 5-4). Esto significa que la línea está preparada para que sobre ella se realice la llamada.

5.4.2.3 Establecimiento de llamadas desde el teléfono

Cuando una llamada es iniciada desde el aparato telefónico, (descolgando la bocina), el *VHELPERD* detecta la acción. Esta acción genera un mensaje *OffHook* hacia el *CHSP* que hace que cambie su estado a *OffHook*. Cuando se encuentra en este estado, el *CHSP* manda un mensaje *GetDialedNumber* al *VHELPERD* para que inicie la recolección de dígitos marcados en el aparato telefónico.

Cuando todos los dígitos son recolectados son enviados al proveedor de servicios mediante el mensaje *#Dialed* con lo cual cambia su estado a *Prepared*. Este estado indica que tiene toda la información necesaria para realizar la marcación. Posteriormente el *VHELPERD* manda el mensaje *Start* que lleva al *CHSP* al estado de *Idle* que significa que la llamada está preparada (Figura 5-5).

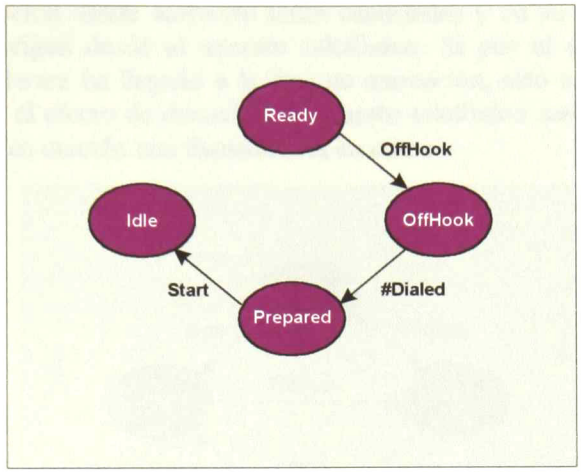


Figura 5-5. Máquina de estados del establecimiento de llamadas desde el aparato telefónico

5.4.2.4 Establecimiento de llamadas software-teléfono

Aquí presentamos la máquina de estado resultante cuando unimos las máquinas para el establecimiento de llamadas desde software y desde el aparato telefónico. Esto debe hacerse por el hecho de que el *CHSP* permite ambas variantes para el establecimiento de llamadas.

Hay dos puntos muy importantes que tenemos que remarcar. El primero de ellos es el hecho de que forzamos un estado conjunto entre las llamadas establecidas por software y por el aparato telefónico. Cuando hacemos que tanto las llamadas generadas por software como por el aparato telefónico lleguen al estado *Idle*, tenemos la ventaja de no tener que definir secuencias de eventos independientes para cada uno de los casos. Esto quiere decir que una vez que se está en *Idle* el mecanismo de control de la llamada será el mismo independientemente del origen de la misma.

El segundo punto que debemos remarcar es el mecanismo de interrupción de llamadas de software. Este mecanismo permite la interacción entre los dos tipos de establecimiento de llamadas. Supongamos que una llamada es establecida desde software, pero antes de que se reserve la línea para empezar con la marcación, el aparato telefónico es descolgado. Aquí se tiene una situación interesante ya que cuando se descuelga el aparato la línea es inmediatamente reservada para la marcación. Un mecanismo posible para solucionar esta interacción negativa es la priorización. En el *CHSP* las llamadas establecidas desde el aparato telefónico tienen mayor prioridad que las llamadas establecidas desde software. De

esta manera, cuando se presente una situación como la descrita anteriormente, las acciones generadas por la petición desde software serán canceladas y en su lugar se generará una nueva llamada con origen desde el aparato telefónico. Si por el contrario, una llamada establecida desde software ha llegado a la fase de marcación, esto es, ha sido reservada la línea para la llamada, el efecto de descolgar el aparato telefónico será el mismo de levantar una extensión telefónica cuando una llamada está en curso.

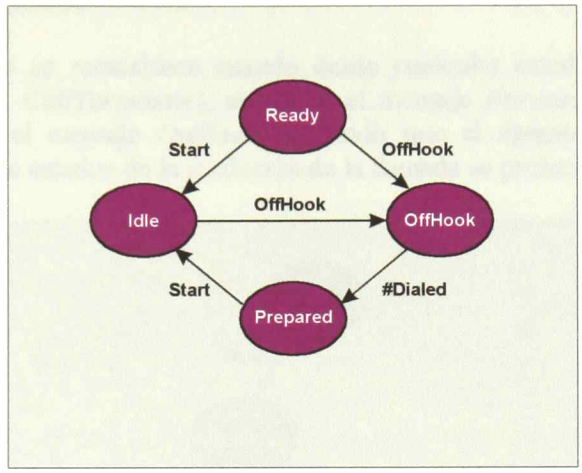


Figura 5-6. Máquina de estados para el manejo conjunto de llamadas establecidas desde software o desde el aparato telefónico

Para garantizar el funcionamiento del mecanismo de interrupción de llamadas desde software, es añadida una nueva transición a la máquina de estados que puede llevarnos del estado *Idle* a *OffHook* mediante el mensaje *OffHook* (Figura 5-6).

5.4.2.5 Evolución de llamada

Una vez que la llamada ha sido establecida, el *CHSP* cambia al estado *DialTone* cuando recibe el mensaje *Hold* del *VHELPERD*; esto le indica que la línea ha sido reservada para la marcación.

Cuando se inicia la marcación el *VHELPERD* lo notifica mediante el mensaje *FirstDialTone* con lo cual el *CHSP* cambia al estado *Dialing*. De manera similar, al terminar la marcación se informa con el mensaje *LastDialTone* que lleva al estado *EndDialing*. Al llegar al estado *EndDialing* la marcación del número ha terminado y una variedad de casos pueden ocurrir.

Si la marcación ha terminado el *VHELPERD* puede informar que ha detectado el número ocupado mediante el mensaje *BusyTone*; que la línea está muerta mediante el mensaje

NoTone; o que ha detectado el timbrado en el número marcado con el mensaje *RingBack*. Si el timbrado se realiza es posible que respondan o no la llamada. En el caso en que contesten el *CHSP* pasa al estado *TimeOut* para detener la llamada. En caso de que respondan la llamada un mensaje *Connect* es enviado al *CHSP*. Este mensaje lleva al estado *Connected* y quiere decir que la llamada ha sido establecida con éxito. Finalmente, la llamada termina satisfactoriamente cuando el aparato telefónico es colgado. Cuando eso sucede es notificado mediante el mensaje *OnHook*.

La máquina de estado se reestablece cuando desde cualquier estado terminal (*NoTone*, *BusyTone*, *TimeOut* o *CallTerminate*), se recibe el mensaje *Forward/Reverse* que indica presencia de línea y el mensaje *OnHook* indicando que el aparato telefónico ha sido colgado. La máquina de estados de la evolución de la llamada se presenta en la Figura 5-7.

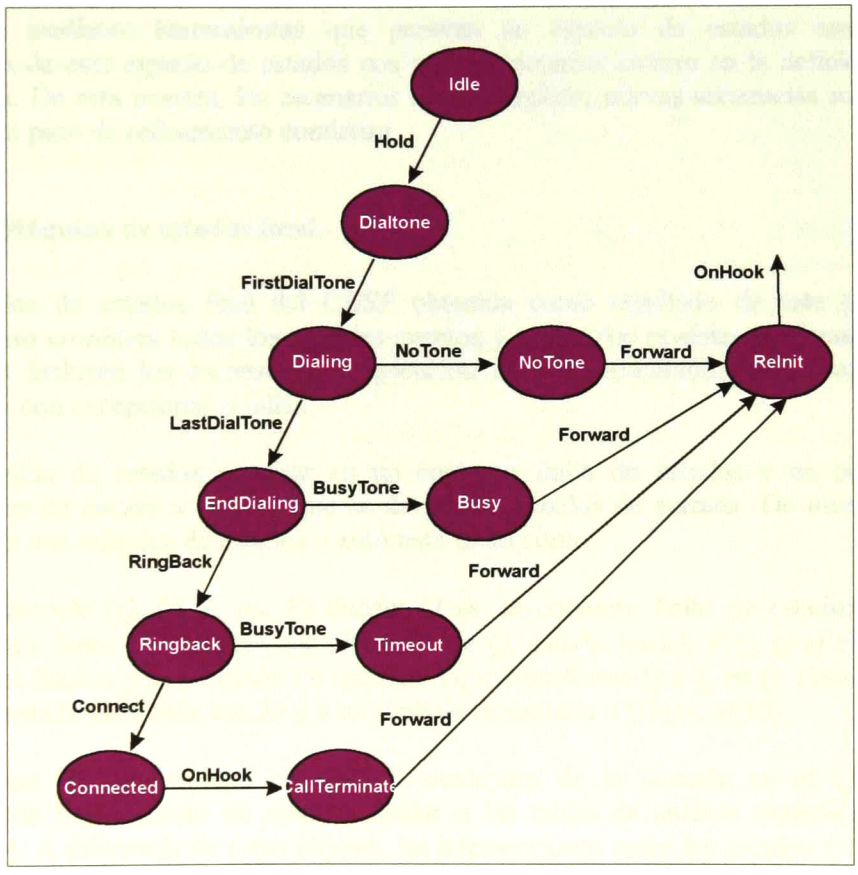


Figura 5-7. Mecanismo de control de llamada del CHSP

5.4.2.6 Proceso de diseño

El primer paso del diseño de la máquina de estados para el control de la llamada es la identificación inicial de escenarios y máquinas de estado que representan las secuencias normales de ejecución. Una vez que se tienen identificadas, iniciamos un ciclo iterativo de refinamiento que tiene por objeto definir todos los posibles escenarios de ejecución mediante el análisis de todas las combinaciones de eventos y estados(Capítulo 2).

El punto central de este proceso de refinamiento es el uso de técnicas de descripción formal para crear especificaciones que sean susceptibles de verificar y que nos permitan encontrar, en los escenarios de ejecución, fallas como ciclos infinitos, estados inválidos o bloqueos. En cada paso de refinamiento, una especificación en LOTOS(presentada en el siguiente capítulo) representa el comportamiento definido hasta el momento. Dicha especificación es verificada mediante herramientas que generan su espacio de estados asociado. La inspección de este espacio de estados nos permite detectar errores en la definición de los escenarios. De esta manera, los escenarios son corregidos, nuevas secuencias son añadidas y un nuevo paso de refinamiento comienza.

5.4.2.7 Máquina de estados final

La máquina de estados final del *CHSP* obtenida como resultado de este proceso de refinamiento considera todos los posibles eventos y escenarios posibles de ejecución. Estos escenarios incluyen los escenarios de ejecución normal presentados anteriormente y los escenarios con excepciones o fallas.

Una máquina de estados consiste en un conjunto finito de estados y un conjunto de transiciones de estado a estado, que se da sobre símbolos de entrada. De manera formal denotamos una máquina de estados o autómata finito como:

- La quíntupla $(Q, \sum, \delta, q_0, F)$ donde: Q es un conjunto finito de estados; $\sum$ es un conjunto finito de entradas; q_0, elemento de Q, estado inicial; $F \subseteq Q$ el conjunto de estados finales; y δ la función de transición que transforma $Q \times \sum$ en Q. Esto es, $\delta(q, a)$ es un estado para cada estado q y un símbolo de entrada a [*Hopcroft93*].

La máquina de estados que controla la evolución de la llamada en el *CHSP*, será representada tabularmente de manera similar a las tablas de análisis sintáctico descritas en[*Aho90*]. A diferencia de éstas últimas, las intersecciones entre los estados y los eventos o mensajes no son acciones sino representaciones del siguiente estado. De esta manera podemos definir la máquina de estados como:

Una tabla m formada por elementos $m_{i,j}$ donde $i \in Q$; y $j \in \sum$
Cada elemento $m_{i,j} = k$ representa el siguiente estado, donde $k = \delta(i, j)$

- La función de transición $\delta(i, j)$ definida mediante el algoritmo:

Sea $k = i = 1$ el estado inicial del *CHSP*
Repetir
Recibir el siguiente evento j del *VHELPERD*
Sea $k = m_{kj}$ el siguiente estado del *CHSP*

La Figura 5-8 presenta la máquina de estados final del *CHSP*. Esta máquina es el resultado del proceso de refinamientos sucesivos descrito en la sección anterior. Dicho proceso produce especificaciones en LOTOS que son utilizadas para depurar las secuencias de la máquina y actualizarla. En esta figura podemos observar todas las transiciones posibles que pueden ocurrir en cada estado de acuerdo a los eventos. Las transiciones que son presentadas en la tabla con un color oscuro representan combinaciones de estados y entradas que no pueden suceder. Sin embargo, estas combinaciones son contempladas para hacer más robusta la máquina, enviándola al estado inicial en caso de algún comportamiento anómalo. La especificación final en LOTOS es presentada en el siguiente capítulo.

5.4.3 Requerimientos globales

Los requerimientos globales son requerimientos que afectan a todo el sistema. En la definición del *CHSP* estos requerimientos definen el uso de líneas por los aparatos telefónicos. Por ejemplo, una línea puede ser usada por un solo teléfono en cualquier momento dado o es imposible que se reciba un ring en un teléfono mientras esta involucrado en alguna conexión.

En este caso, el *CHSP* recibe todos los mensajes del *VHELPERD* que procesa las señales directamente del Sistema Telefónico. Esto quiere decir que podemos observar estos requerimientos como restricciones impuestas por la implementación del mismo.

	Message ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
State ID		GetStatePhone (OnHook)	GetStatePhone (OffHook)	GetStateLine (Forward)	GetStateLine (Reverse)	GetStateLine (LineOutOfService)	GetStateLine (ExtInService)	GetStateLine (Ring)	GetPhoneNumber (# Marcado)	DialNumber (Start)	DialNumber (Hold)	DialNumber (FirstDialTone)	DialNumber (LastDialTone)	CallState (RingBack)	CallState (BussyTone)	CallState (Connect)	CallState (NoConnect)	CallState (NoTone)	GetCallerID (Caller ID Number)	AcceptCall	
1	Init	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
2	Initializing	1	1	3	3	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	
3	Ready	1	5	3	3	15	14	19	1	4	1	1	1	1	1	1	1	1	1	1	
4	Idle	1	5	4	4	15	14	19	1	1	7	1	1	1	1	1	1	1	1	1	
5	OffHook	3	1	5	5	15	14	1	6	1	1	1	1	1	1	1	1	1	1	1	
6	Prepared	3	1	6	6	15	14	1	1	4	1	1	1	1	1	1	1	1	1	1	
7	Dialtone	16	7	7	7	15	7	1	1	1	1	8	1	1	1	1	1	1	1	1	
8	Dialing	16	8	8	8	15	8	1	1	1	1	1	9	1	1	1	1	1	1	1	
9	EndDialing	16	9	9	9	15	9	1	1	1	1	1	1	10	11	12	1	18	1	1	
10	RingBack	17	10	10	10	15	10	1	1	1	1	1	1	10	13	12	13	1	1	1	
11	SubscriberBusy	11	11	16	16	15	14	1	1	1	1	1	1	1	1	1	1	1	1	1	
12	Connected	17	12	12	12	15	12	1	1	1	1	1	1	1	1	1	1	1	1	1	
13	Timeout	1	13	16	16	15	14	1	1	1	1	1	1	1	1	1	1	1	1	1	
14	ExtBusy	14	14	16	16	15	1	1	14	1	1	1	1	1	1	1	1	1	14	14	
15	LineOutOfService	15	15	16	16	1	14	1	1	1	1	1	1	1	1	1	1	1	15	15	
16	Reinit	3	1	16	16	15	14	1	1	1	1	1	1	1	1	1	1	1	16	16	
17	CallTerminate	1	17	16	16	15	14	1	1	1	1	1	1	1	1	1	1	1	1	1	
18	NoTone	18	18	16	16	15	14	1	1	1	1	1	1	1	1	1	1	1	1	1	
19	WaitCallerID	19	19	16	16	15	14	19	1	1	1	1	1	1	1	1	1	1	20	1	
20	WaitAcceptCall	20	20	16	16	15	14	20	1	1	1	1	1	1	1	1	1	1	1	21	
21	Ringing	21	12	16	16	15	14	21	1	1	1	1	1	1	1	1	1	1	1	1	

Figura 5-8. Máquina de Estados Final del CHSP

5.5 Comentarios finales

El funcionamiento del *CHSP* se define mediante tres tipos de requerimientos: end-to-end, locales y globales; éstos, definen los principios de ordenamiento temporal de los eventos telefónicos. Dichos requerimientos son presentados mediante diagramas de secuencias y máquinas de estado finitos. Un proceso de refinamiento que toma especificaciones en LOTOS (basadas en los requerimientos mencionados) fue utilizado. Este proceso se basa en la verificación de las propiedades del espacio de estado de la especificación para encontrar errores en las definiciones. Con ello, los escenarios son corregidos, nuevas secuencias son añadidas y un nuevo paso de refinamiento comienza. En el siguiente capítulo será presentada la especificación formal en LOTOS resultante.

6

Especificación Formal del Proveedor de Servicios de Telefonía

Con mucha frecuencia se dice que una persona no entiende a fondo un tema hasta que se lo enseña a alguien más. De hecho, una persona no entiende a fondo un tema hasta que se lo enseña a una computadora.

Donald Knuth. Computer Science and its Relation to Mathematics.

6.1 Introducción

Una especificación es una descripción abstracta de un sistema. Las especificaciones pueden ser escritas en un lenguaje formal como LOTOS[*ISO89*] o informal como el Español. Las especificaciones informales son más fáciles de leer y escribir, sin embargo, con mucha frecuencia resultan ambiguas.

Para especificar formalmente un sistema es necesario generar una definición precisa del mismo. Esta definición debe tomar en cuenta tanto los elementos que conforman el sistema como la manera como éstos se interconectan. LOTOS permite hacer especificaciones que de manera natural incorpora dichos elementos. Una especificación en LOTOS define los elementos que integran un sistema como procesos cuyo comportamiento se representa con secuencias de eventos. Además, define la comunicación entre dichos procesos mediante la sincronización entre los eventos de cada uno de ellos.

Primero se presentará el origen del lenguaje de especificación formal LOTOS. Se analizarán los diferentes estilos de especificación utilizados para la descripción del proveedor de servicios de telefonía. Tomando como base los requerimientos presentados en el capítulo anterior, especificaremos cada uno de sus componentes y las propiedades generales del sistema. Finalmente veremos como probar la especificación usando un intérprete.

6.2 Origen de LOTOS

LOTOS nació del esfuerzo de la Organización Internacional de Estandarización (ISO[1]) que trabajaba en el Modelo de Interconexión de Sistemas Abiertos (OSI[2]). Cuando se inició este trabajo en los 70's, pronto fue evidente que para que los estándares de OSI fueran efectivos tendrían que ser definidos de manera precisa. El término *Técnicas de Descripción Formal* fue acuñado (FDT[3]) para referirse a las técnicas para la especificación exacta de protocolos y servicios.

Algunas de las características más deseables de los FDTs son su capacidad de abstracción, independencia de la implementación, semántica formal y el soporte para métodos de verificación. Pronto fue evidente que no existía un FDT con tales características por lo que en 1979 se creo el comité conocido como FDT Ad.hoc Group of ISO/TC 97/SC 16/WG 1. Después de algunas reuniones fueron establecidos dos subcomités: el subgrupo B, que trabajó bajo el modelo de Máquinas de Estado Finito Extendidas; y el subgrupo C, que trabajó con la técnica de Ordenamiento Temporal. El subgrupo B eventualmente produciría Estelle[*ISO89a*], mientras que el subgrupo C produjo LOTOS[*ISO89*].

El desarrollo de LOTOS se basó en el Cálculo de Sistemas Comunicantes de Milner (CCS[4]) [*Mil89*], y también en los Procesos Secuenciales Comunicantes de Hoare (CSP[5]) [*Hoa85*]. En 1983 el lenguaje adquirió su nombre, que es un acrónimo de Language Of Temporal Ordering Specifications.

En 1984 se decidió que el formalismo ACT ONE[*Ehr85*] sería usado como la base para la definición de tipos de datos abstractos. La primera versión del lenguaje salió a la luz en 1985 y fue publicado el ISO Draft Proposal 8807. La semántica de LOTOS fue definida en base al lenguaje CCS*, que extendía CCS. En 1986, la Universidad de Otawa produjo el primer intérprete que soportaba el lenguaje completo[*Log88*]. En 1989, LOTOS se convirtió finalmente en estándar internacional.

6.3 Principios de LOTOS

Algunos de los principios que inspiraron el diseño de LOTOS son:

1. Formalismos complementarios para los datos y el control. Los diseñadores de LOTOS sintieron que no existía un formalismo suficientemente general que expresara de manera conveniente los componentes de control y de datos de una especificación. De esta manera el lenguaje fue concebido como la unión de dos formalismos: ACT ONE para la parte de datos y CCS/CSP para la parte de control.

---

[1] *International Organization for Standarization*
[2] *Open Systems Interconnection*
[3] *Formal Description Techniques*
[4] *Calculus of Communicating Systems*
[5] *Communicating Sequential Processes*

2. Definición formal. La sintaxis y semántica del lenguaje son definidas formalmente. Los elementos estáticos de la semántica se describen por una gramática mientras que los elementos dinámicos se describen en términos de reglas de inferencia[*Bol87*].
3. Algebra de procesos. La semántica operacional es definida de manera que provee un conjunto de propiedades de equivalencia. Estas propiedades pueden ser usadas para probar la equivalencia o correctud de especificaciones al igual que para transformar la estructura de una especificación.
4. Concurrencia entrelazada. Los eventos son considerados atómicos, y por lo tanto la ejecución paralela de dos eventos es definida como una situación de decisión. Así, cualquier comportamiento en LOTOS puede ser escrito como una expresión de decisión entre expresiones de comportamiento (teorema de la expansión) [*Mil80*].
5. Ejecutabilidad. Debido a que la semántica de LOTOS es definida operacionalmente, es posible implementar esa semántica en un intérprete. Un intérprete puede, a partir de una expresión de comportamiento, enumerar las posibles acciones y las expresiones de comportamiento resultantes de la ejecución de cada una de ellas. Esto quiere decir que las especificaciones en LOTOS pueden ser escritas de una manera que pueden ser interpretadas o incluso trasladadas hacia un programa[*Man89*].
6. Modularidad y reusabilidad. LOTOS favorece el refinamiento sucesivo de procesos. Por medio del uso de parámetros estos procesos se vuelven reusables.

6.4 Principios de construcción de especificaciones

En las siguientes secciones se presentan algunos principios de diseño a ser respetados por un diseñador para obtener una mejor representación de la descripción del sistema. La estructura de una especificación en LOTOS puede observarse en la Especificación 6-1.

6.4.1 Principios de diseño

El objetivo principal del proceso de diseño es derivar en una implementación. Debido a la complejidad de los sistemas distribuidos, el proceso de diseño se puede convertir en una tarea complicada. Para poder lograr una mejor y más clara representación del sistema, el proceso de diseño debe seguirse en pasos, donde cada paso representa un diferente nivel de abstracción del sistema. Estos niveles reciben el nombre de refinamientos. En LOTOS, la especificación completa del comportamiento de un sistema complejo puede llevarse a cabo mediante refinamientos sucesivos.

En el diseño del proveedor de servicios de telefonía son usadas una variedad de estilos de especificación. El uso de un estilo de especificación debe respetar algunos principios generales de diseño. Estos principios son principalmente Ortogonalidad, Generalidad y Flexibilidad. La ortogonalidad preserva localmente aspectos de los subprocesos, la generalidad sugiere el uso de definiciones de propósito general parametrizadas, y la flexibilidad en el diseño para facilitar la modificación de la funcionalidad del sistema.

6.4.2 Estilos de especificación

En el diseño del proveedor de servicios de telefonía en LOTOS, diferentes estilos de especificación fueron usados. Cada uno de estos estilos tiene su propio rol en la descripción formal del subsistema específico dependiendo de su funcionalidad y del nivel de abstracción[*Vissers88*]. Para lograr un diseño satisfactorio, el uso de diferentes estilos en la etapa de diseño de un sistema debe cumplir con los principios de diseño nombrados arriba.

La descripción formal de un sistema puede tener como objetivo fundamental la descripción de comportamiento observable, como caja negra, o la descripción del comportamiento no observable, como caja blanca. En esta sección se presenta una discusión general de los estilos de especificación y como se relacionan en el diseño del sistema telefónico.

6.4.2.1 Estilo monolítico

El estilo monolítico presenta de manera explícita todas las secuencias posibles de acciones permitidas en la especificación. La especificación en este estilo, representa un árbol de decisiones. Este estilo es útil para depurar especificaciones y generar secuencias de prueba.

6.4.2.2 Estilo orientado a estados

En el estilo orientado a estados, se identifican estados específicos del sistema. Aunque el uso de este estilo puede ser tedioso si se usa en toda la especificación, puede ser muy útil cuando se introducen variables de estado que describen el estado de algún dispositivo. Esto puede llevar a especificaciones muy legibles cuando la descripción informal hace mención explícita del concepto de estado (muy común en el caso de sistemas telefónicos). Además este tipo de especificación puede llevarse directamente a la implementación.

6.4.2.3 Estilo orientado a recursos

En este tipo de especificaciones, los procesos son seleccionados de manera que representen recursos, es decir, módulos susceptibles de implementar. Este estilo es útil para producir especificaciones que representan la implementación del sistema.

6.4.2.4 Estilo orientado a requerimientos

El estilo orientado a requerimientos es el más abstracto de todos debido a que se enfoca en las secuencias de eventos tal y como son vistas externamente. Debe ser usado para

representar sistemas como cajas negras, esto es, sin considerar su estructura interna. Este estilo es usado para generar especificaciones independientes de la implementación.

6.5 Especificación del proveedor de servicios de telefonía

La especificación del proveedor de servicios de telefonía usa una mezcla de diferentes estilos. Debido a que describe un sistema telefónico, la idea de estados está presente y por lo tanto se utiliza el estilo orientado a estados. Además, una de las ideas principales del proveedor de servicios de telefonía es interactuar como módulo con otros módulos para conformar el sistema, por ello el estilo orientado a recursos es también utilizado.

La especificación del sistema telefónico está estructurada como se muestra en la Especificación 6-1:

```
specification Service_Provider [line,bus] :noexit
  < Definiciones de tipos de datos >

behaviour
  < Definiciones de procesos >

endspec
```

Especificación 6-1. Estructura de la especificación del proveedor de servicios de telefonía

Esta especificación define dos puntos de interacción: *line* y *bus*. La compuerta *line* es usada para la transmisión y recepción de señales entre el *CHSP* y el *Habilitador CTI* (*HTT*). La compuerta *bus* es utilizada para sincronizar los eventos originados en el *VHELPERD* (*VxD*). En las siguientes secciones se presentará a detalle la especificación formal en LOTOS del *CHSP*.

6.5.1 Nivel más alto de especificación

El nivel más alto de la especificación del proveedor de servicios de telefonía está compuesto por los procesos *HTT*, *VxD* y *CHSP*. Como se mencionó anteriormente, el *CHSP* interactúa tanto con el proceso *HTT* como con el proceso *VxD*.

En este nivel, la especificación consta de los procesos *HTT* y *VxD* compuestos en paralelo. El proceso *CHSP* se sincroniza con ellos mediante las compuertas *line* y *bus* (Especificación 6-2).

```
behaviour

  (
    HTT[Line]

    |||

    VxD[Bus]
  )

  |[Line,Bus]|

  CHSP[Line,Bus](INIT_S)
```

Especificación 6-2. Nivel más general

El proceso *HTT* crea las secuencias de eventos que definen las llamadas originadas desde software. Por su parte, el proceso *VXD* crea las secuencias de eventos que se originan en las señales de la red telefónica. Estas son creadas por ambos procesos son generadas simultáneamente y sin sincronización entre ellas. Esto tiene sentido ya que las señales de la red telefónica son totalmente independientes de los mensajes provenientes del *HTT*.

Todos los eventos de los procesos *HTT* y *VXD* deben sincronizarse con el proceso del proveedor de servicios. La compuerta *Line* es utilizada en la especificación para sincronizarse con todos los eventos del proceso *HTT*; la compuerta *Bus* es utilizada para sincronizarse con los eventos del proceso *VXD*. La representación gráfica se presenta en la Figura 6-1.

Nosotros podemos ocultar explícitamente ambas compuertas ya que en realidad son eventos internos que no modifican el servicio prestado al usuario.

Los procesos de la especificación utilizan dos conjuntos de señales para comunicarse entre sí (Especificación 6-3). Un conjunto *Messages* que define los mensajes enviados desde el *HTT* y el *VxD* y un conjunto *States* que define los estados internos del *CHSP* utilizados para el control de la llamada.

```
$sort Messages is
  (OnHook, OffHook, Forward, Reverse, LineOutOfService, ExtInService,
   Ring, Number, Start, Hold, FirstDialTone, LastDialTone, Ringback,
   BusyTone, Connected, NoConnected,NoTone, CallerID, CallAccept,
   StandardConnect, ExtendedConnect,StandardResponse,ExtendedResponse,
   SWCall, HWCall)


$sort States is
  (INIT_S, INITIALIZING_S, READY_S, IDLE_S, OFFHOOK_S, PREPARED_S,
   DIALTONE_S, DIALING_S, ENDDIALING_S, RINGBACK_S, SUBSCRIBERBUSY_S,
   CONNECTED_S, TIMEOUT_S, EXTBUSY_S, LINEOUTOFSERVICE_S, REINIT_S,
   CALLTERMINATE_S, NOTONE_S, WAITCALLERID_S, WAITACCEPTCALL_S,
   RINGING_S, ONHOOK_S)
```

Especificación 6-3. Conjuntos de señales y estados

Como se ha mencionado, el proceso *CHSP* tiene como objetivo mantener el control de la llamada al hacerla evolucionar por estados. Por ello, el comportamiento de una conexión es definido dentro de este proceso.

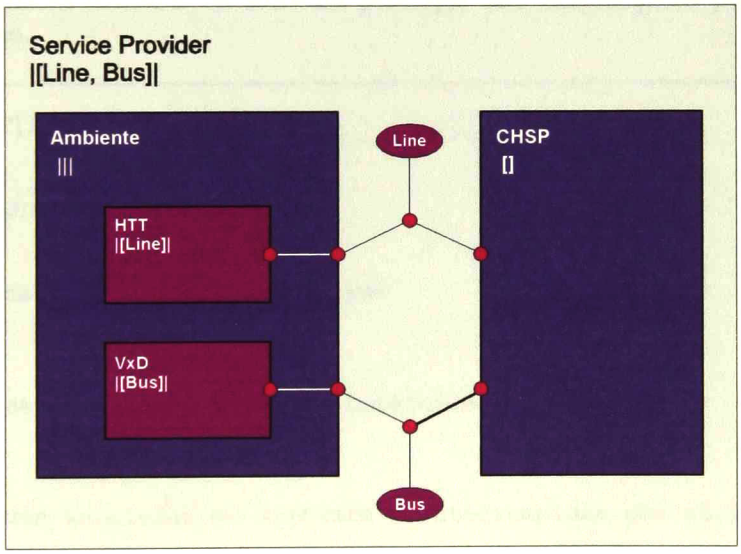


Figura 6-1. Representación gráfica del nivel más general de la especificación

6.5.2 Procesos de soporte

Los procesos *HTT* y *VxD* nos proporcionan el soporte necesario para poder especificar al *CHSP*. Estos procesos se encargan de enviar los mensajes que representan la solicitud y ejecución de procesos telefónicos al *CHSP*. El alcance de este trabajo no incluye la especificación detallada de ambos procesos ya que sus especificaciones son hechas con la finalidad de probar la funcionalidad del *CHSP*. La descripción de las especificaciones de prueba para dichos procesos se presenta en las siguientes secciones.

6.5.2.1 Proceso HTT

El proceso *HTT* representa un módulo de software que debe reaccionar a cualquier evento o notificación generado en el sistema. Por lo tanto, debe contemplar tanto las secuencias de llamadas originadas desde software como su interacción con las llamadas originadas desde hardware. Dichas secuencias son representadas en la especificación con una construcción de selección mediante el operador [] (Especificación 6-4).

De esta manera se generan tres árboles de ejecución que representan las posibles combinaciones en los eventos: llamadas iniciadas en software, llamadas iniciadas en el teléfono, y llamadas iniciadas en software pero que son interrumpidas por una llamada desde el teléfono.

```
process HTT[Line]:exit:=

(
   Line!READY_S;

   (
     <Llamadas iniciadas en software>

     []

     <Llamadas iniciadas desde el teléfono>

     []

     <Llamadas iniciadas en software e interrumpidas por el teléfono>
   )
)
```

Especificación 6-4. Proceso HTT

Cada una de las opciones es definida como una secuencia de eventos en la compuerta *Line*. Esta compuerta es utilizada en la especificación para sincronizar los procesos *HTT* y *CHSP*.

Así, por la compuerta *Line* se sincronizan en el estado READY_S. Esta sincronización será evidente más adelante cuando se presente la especificación del proceso *CHSP*.

6.5.2.1.1 Llamadas iniciadas desde software

La Especificación 6-5 define la secuencia de ejecución para las llamadas iniciadas desde software. Una vez que el *CHSP* y *HTT* se han sincronizado en el estado READY_S, se pasa a la ejecución de los procesos *SWCall* Y *Connection*. Dichos procesos se componen secuencialmente mediante el operador >>.

El proceso *SWCall* define el evento que da inicio a una llamada desde software; el proceso *Connection* define los eventos que representan la evolución de la llamada desde la reservación de la línea hasta el colgado del teléfono después de la llamada (Especificación 6-6).

```
process HTT[Line]:exit:=

(
   Line!READY_S;

   (
     (
       SWCall[Line]

       >>

       Connection[Line]

       >>

       exit
     )

     []




   )
)
```

Especificación 6-5. Llamadas iniciadas desde software

Estas secuencias de eventos fueron definidas mediante las máquinas de estados presentadas en el capítulo anterior.

```
process SWCall[Line]:exit:=

  Line!Start;
  exit

endproc


process Connection[Line]:exit:=

  Line!Hold;
  Line!FirstDialTone;
  Line!LastDialTone;
  Line!Ringback;
  Line!Connected;
  Line!OnHook;
  exit

endproc
```

Especificación 6-6. Procesos SWCall y Connection

6.5.2.1.2 Llamadas iniciadas desde el teléfono

La segunda secuencia que puede ser seleccionada es la de llamadas iniciadas desde el teléfono (Especificación 6-7). Aquí, el proceso *HTT* se sincroniza con los eventos de estado que recibe del *CHSP*. El proceso *Connection* contiene los eventos descritos con anterioridad que definen la evolución de la llamada.

```
(
  Line!OFFHOOK_S;
  Line!PREPARED_S;
  Line!IDLE_S;
  Connection[Line]
)

>>

exit
```

Especificación 6-7. Llamadas iniciadas desde el teléfono

6.5.2.1.3 Llamadas iniciadas en software e interrumpidas desde el teléfono

En esta secuencia, primero se presenta el proceso *SWCall* que inicia con la sincronización de los eventos para una llamada saliente desde software. Sin embargo, aquí se tiene una variación, en la cual una secuencia iniciada desde el aparato telefónico interrumpe la ejecución normal (Especificación 6-8). Esta secuencia es igual a la presentada en la sección anterior y con ella, el *CHSP* se sincroniza con los eventos de una llamada originada desde teléfono.

```
(
  SWCall[Line]

  >>

  (
    Line!OFFHOOK_S;
    Line!PREPARED_S;
    Line!IDLE_S;
    Connection[Line]
  )

  >>

  exit
)
```

Especificación 6-8. Llamadas de software interrumpidas desde el teléfono

6.5.2.2 Proceso VxD

```
process VxD[Bus]:exit:=

  Initialize[Bus]
  >>
  (
    HWCall[Bus]
  )
  >>
  exit

endproc
```

Especificación 6-9. Proceso VxD

El proceso *VxD* define los eventos que representan las señales recibidas de la línea telefónica. Estos eventos son sincronizados con el *CHSP* mediante la compuerta *Bus*. Como puede observarse en la Especificación 6-9, el *VxD* se compone secuencialmente de los procesos *Initialize* y *HWCall*, donde *Initialize* habilita a *HWCall* mediante el operador >>.

De manera similar al proceso *HTT*, *VxD* es un proceso utilizado para la prueba del *CHSP*. Las especificaciones de prueba para los procesos *Initilize* y *HWCall* se presentan a continuación.

6.5.2.2.1 Proceso Initialize

Este proceso es muy sencillo, simplemente genera un evento *OnHook* que indica que el aparato telefónico está colgado y posteriormente un evento *Forward* o *Reverse* en la compuerta *Bus* que representan la presencia de línea(Especificación 6-10). Estos eventos se sincronizan con el *CHSP* y le indican que el aparato telefónico está listo y que la línea telefónica ha sido detectada y que puede procederse con su uso.

```
process Initialize[Bus]:exit:=

  Bus!OnHook;
  (
    Bus!Forward;
    exit

    []

    Bus!Reverse;
    exit
  )

endproc
```

Especificación 6-10. Proceso Initialize

6.5.2.2.2 Proceso HWCall

Este proceso genera los eventos que a través del *CHSP* se sincronizarán con los eventos de estado en el proceso *HTT*. La secuencia está integrada por los eventos *OffHook*, *Number* y *Start* que simbolizan el descolgado del aparato telefónico, el marcado del número y el mensaje de inicio de conexión respectivamente (Especificación 6-11).

```
process HWCall[Bus]:exit:=

  (
    Bus!OffHook;
    Bus!Number;
    Bus!Start;
    exit
  )

endproc
```

Especificación 6-11. Proceso HWCall

6.5.2.3 Proceso CHSP

Una vez que se tienen los procesos auxiliares podemos pasar a la especificación del proveedor de servicios de telefonía. El proceso *CHSP* está construido principalmente como una estructura de selección mediante el operador **[]**.

El *CHSP* debe sincronizarse con los eventos generados tanto en el proceso *HTT* como en el proceso *VxD*, para ello utiliza las compuertas *Line* y *Bus* respectivamente.

```
process CHSP[Line,Bus](St:States):exit:=

  (
    Bus?Ms:Messages;
    SM[Line,Bus](St,Ms)
  )

  []

  (
    Line?Ms:Messages;
    SM[Line,Bus](St,Ms)
  )

endproc
```

Especificación 6-12. Proceso CHSP

Ahora bien, la secuencia de los eventos generados por los procesos auxiliares no es conocida (ni debe ser conocida) por el *CHSP*, ya que ésta es precisamente su función:

mantener el control sobre los eventos telefónicos y hacer evolucionar la llamada de acuerdo a las señales que recibe. Para lograr este objetivo, el CHSP se sincroniza independientemente del mensaje que reciba utilizando la construcción de variables de evento: *Compuerta?Ms:Messages*, en donde *Compuerta* es la compuerta (*Line* o *Bus*) por la cual se sincroniza con el evento.

Por lo tanto, dos opciones son especificadas para la selección: la sincronización con el proceso *HTT* a través de la compuerta *Line*; y la sincronización con el proceso *VHELPERD* a través de la compuerta *Bus*. Una vez que se ha sincronizado el evento en alguna de las opciones, el evento es enviado al proceso *SM* mediante el parámetro *Ms* (Especificación 6-12). El proceso *CHSP* recibe como parámetro el estado en el cual se encuentra, y es especificado explícitamente en el nivel más abstracto, donde se inicializa con el estado INIT_S. El estado es enviado al proceso *SM* mediante el parámetro *St*.

6.5.2.3.1 Proceso SM

El proceso *SM* especifica la máquina de estados que permite el control y evolución de las llamadas. Para representar este proceso se desarrolló una estructura de especificación que permite trasladar de manera directa la máquina de estados presentada al final del capítulo anterior (y cualquier otra máquina de estados determinista).

El primer paso es la identificación del evento que ha sido enviado como argumento al proceso *SM*. Para ello, utilizamos una estructura denominada guarda. En una guarda tenemos una expresión booleana, denominada premisa, que es utilizada para evaluar una condición. Entonces, el proceso *SM* está especificado como una selección de múltiples opciones, donde cada opción es una guarda sobre el evento recibido en el argumento *Ms* (Especificación 6-13).

```
(
  [Ms eq OnHook]->
    (
      ...     <Procesamiento adicional>
    )
  []

  [Ms eq OffHook]->
    (
      ...
    )
  []
  ...     <Guardas sobre eventos adicionales>
)
```

Especificación 6-13. Estructura general del proceso SM

El siguiente paso es el reconocimiento del estado actual en el cual se encuentra el *CHSP*. Como se mencionó anteriormente, dicho estado es enviado en el parámetro *St*. Una extensión lógica del paso anterior es definir una serie de guardas sobre el estado recibido. Las guardas sobre estados son definidas dentro del ámbito de las guardas sobre eventos. Con esta estructura (Especificación 6-14) podemos especificar perfectamente la recepción de eventos en un estado en particular.

```
(
  [Ms eq OnHook]->
    (
        [St eq INIT_S]->
          ...     <Procesamiento adicional>
        []

        [St eq OFFHOOK_S]->
          ..      <Procesamiento adicional>
        []
        ...     <Guardas sobre estados adicionales>
    )
  []

  [Ms eq OffHook]->
    (
        [St eq INIT_S]->
          ...
        []

        [St eq OFFHOOK_S]->
          ...
        []
        ...
    )
  []
  ...     <Guardas sobre eventos adicionales>
)
```

Especificación 6-14. Refinamiento del proceso SM

Una vez que tenemos identificados tanto el evento como el estado actual, necesitamos un mecanismo para hacer evolucionar el estado de la máquina. En este trabajo se logró mediante el uso de recursividad. LOTOS permite el modelado de comportamiento repetitivo mediante la instanciación de procesos. Cuando instanciamos un proceso, una nueva copia del proceso sustituye a la anterior. Entonces, es posible crear una nueva instancia del proceso *CHSP* que reciba como argumento el nuevo estado actual. Este nuevo proceso se comportaría exactamente como el anterior pero en un estado diferente. El nuevo estado (resultante de la interacción de un evento especifico en un estado específico) es mapeado directamente de la máquina de estados final presentada con anterioridad. Con esta

idea (Especificación 6-15) podemos finalizar la estructura para la especificación de la máquina de estados para el proveedor de servicios de telefonía.

```
process SM[Line,Bus](St:States,Ms:Messages):exit:=

  [Ms eq OnHook]->
    (
        [St eq INIT_S]->
          CHSP[Line,Bus](INITIALIZING_S)
        []

        [St eq OFFHOOK_S]->
          CHSP[Line,Bus](READY_S)
        []
        ...     <Guardas sobre estados adicionales>
    )
  []

  [Ms eq OffHook]->
    (
        [St eq INIT_S]->
          CHSP[Line,Bus](INIT_S)
        []

        [St eq READY_S]->
          Line!OFFHOOK_S;     <Eventos de notificación de estados>
          CHSP[Line,Bus](OFFHOOK_S)
        []
        ...
    )
  []
  ...     <Guardas sobre eventos adicionales>
```

Especificación 6-15. Estructura final del proceso SM

Finalmente, podemos observar que es posible añadir todas los eventos de notificación de estado que deseemos. Estos eventos se corresponden con los escenarios de ejecución planteados en el capítulo anterior. La especificación completa es presentada en el Apéndice-A.

6.6 Pruebas a la especificación

Una vez que tenemos la especificación en LOTOS del proveedor de servicios de telefonía, podemos proceder a probarla. El ciclo de pruebas de la especificación es un proceso iterativo que nos permite refinar en cada paso la máquina de estados del *CHSP* Con ello, se genera una nueva especificación de la máquina refinada. Este proceso nos llevó a la especificación final descrita en este capítulo.

Para realizar las pruebas a la especificación, el paquete de herramientas *Ara Tools* [*AATOS94b*] fue utilizado.

Ara Tools es un conjunto de herramientas para analizar el comportamiento de sistemas concurrentes. Estas herramientas pueden ser usadas para localizar bloqueos, errores de sincronización, funcionalidad ilegal y otros errores relacionados con el comportamiento lógico de un sistema.

Las pruebas de *Ara* están basadas en una técnica llamada simulación exhaustiva o análisis de alcanzabilidad[*AATOS94*]. Esto quiere decir, que detecta los errores de un sistema mediante la simulación de su modelo y de todos los estados (y las relaciones entre ellos) que puede alcanzar.

Las pruebas de la especificación del proveedor de servicios de telefonía se realizaron siguiendo los siguientes pasos:

- Creación de la especificación
- Pruebas manuales del comportamiento de la especificación
- Construcción del espacio de estados
- Graficación del espacio de estados de la especificación

6.6.1 Creación de la especificación

El primer paso para el análisis de un sistema es la creación de un archivo que contiene la descripción abstracta del comportamiento del mismo (especificación). *Ara* acepta especificaciones escritas en *ARA LOTOS*. El lenguaje de especificación *Ara LOTOS* contiene *Basic LOTOS*, soporta un subconjunto de *Full LOTOS* e incluye algunas extensiones (Figura 6-2).

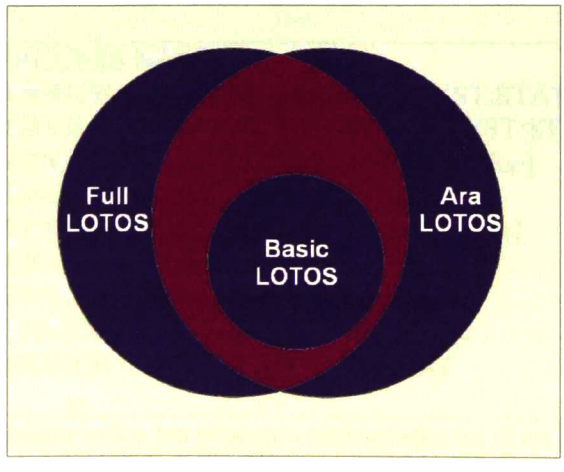


Figura 6-2. Ara LOTOS

Ara LOTOS es suficiente para representar nuestra especificación ya que no se utilizaron en ella los tipos de datos abstractos de *Full LOTOS*. La especificación en *Ara LOTOS* es salvada en un archivo de texto con la extensión *.lot*.

6.6.2 Pruebas manuales del comportamiento de la especificación

Una vez que se tiene la especificación, la mejor manera de comprender su funcionalidad es analizando el comportamiento de los procesos. Para hacer este análisis, *Ara* tiene una herramienta llamada ***Ara Simulator*** que permite simular la ejecución de los procesos de la especificación. Ara Simulator toma el archivo de la especificación como argumento. La Figura 6-3 muestra el inicio de una sesión de análisis con Ara Simulator.

Comando	:\> *arasim chsp.lot* **Enter**
Pantalla	ARA Simulator/Demo v2.01 (c) VTT Electronics 1992-94 file chsp.lot opened file chsp.lot closed >
Comentarios	• Este comando inicia la ejecución del depurador interactivo *Ara Simulator* • El archivo *chsp.lot* es leído y transformado en una estructura de datos interna • El símbolo > denota que el simulador espera por comandos adicionales

Figura 6-3. Simulación de análisis de procesos del proveedor de servicios de telefonía

Comando Arasim	> print-all **Enter**
Pantalla	ID: PROCESS NAME: 1 SERVICE_PROVIDER.SM[line, bus](ST:STATES,MS:MESSAGES) 2 SERVICE_PROVIDER.CHSP[line, bus](ST:STATES) 3 SERVICE_PROVIDER.CONNECTION[line] 4 SERVICE_PROVIDER.INITIALIZE[bus] 5 SERVICE_PROVIDER.HWCALL[bus, test] 6 SERVICE_PROVIDER.SWCALL[line] 7 SERVICE_PROVIDER.VXD[bus, test] 8 SERVICE_PROVIDER.HTT[line, test] 9 SERVICE_PROVIDER[line, test, bus] >
Comentarios	• Enlista todos los procesos encontrados en el archivo *chsp.lot*

Figura 6-4. Listado de procesos disponibles en la especificación

Primero seleccionamos el proceso que nos interesa simular. Para ello, usamos el comando *print-all* para obtener la lista completa de procesos de la especificación del proveedor de servicios (Figura 6-4).

Ahora seleccionaremos un proceso para revisarlo. Para este ejemplo seleccionaremos el número 8 (Figura 6-5) que es el proceso que describe el *HTT*. De esta manera podemos visualizar el comportamiento de los procesos de la especificación cuando generamos algunos eventos.

Comando Arasim	> use 8 **Enter**
Pantalla	* State 1: line!READY_S:STATES; ((i/$delta,$delta/$delta1in SERVICE_PROVIDER.SWCALL ||| $delta; (i/$delta in $delta; ($delta1/$delta in $delta; stop) \|\|\| line!OFFHOOK_S:STATES; line!PREPARED_S:STATES; line!IDLE_S:STATES; SERVICE_PROVIDER.CONNECTION)) [] (i/$delta,$delta/$delta in test!HOPE:SIGNALS; SERVICE_PROVIDER.SWCALL ||| $delta; (i/$delta in SERVICE_PROVIDER.CONNECTION ||| $delta; ($delta1/$delta in $delta; stop))) [] (i/$delta, $delta/$delta1 in $delta; ($delta1/$delta in $delta; stop) ||| line!OFFHOOK_S:STATES; line!PREPARED_S:STATES; line!IDLE_S:STATES; SERVICE_PROVIDER.CONNECTION)) 1 line!READY_S:STATES ->> >
Comentarios	• Seleccionamos el proceso 8 para su inspección • La salida nos muestra que nos encontramos en el estado *1* (* State 1:) que es igual a la expresión de comportamiento mostrada • La expresión de comportamiento representa las secuencias de eventos que son posibles desde el punto inicial del proceso seleccionado • La línea final nos muestra los eventos que pueden generar transiciones válidas (1 line!READY_S:STATES ->), donde *1* es el identificador que se utilizará para continuar con las simulación

Figura 6-5. Selección de un proceso para su inspección

La única transición disponible es la que se genera de la selección del evento *line!READY_S:STATES* con el identificador *1*. Para continuar con nuestra simulación, seleccionaremos el evento disponible (Figura 6-6).

Comando Arasim	> 1 **Enter**
Pantalla	* State 2: (i/$delta, $delta/$delta1 in SERVICE_PROVIDER.SWCALL ||| $delta; (i/$delta in $delta; ($delta1/$delta in $delta; stop) \|\|\| line!OFFHOOK_S:STATES; line!PREPARED_S:STATES; line!IDLE_S:STATES; SERVICE_PROVIDER.CONNECTION)) [] (i/$delta, $delta/$delta1 in test!HOPE:SIGNALS; SERVICE_PROVIDER.SWCALL ||| $delta; (i/$delta in SERVICE_PROVIDER.CONNECTION ||| $delta; ($delta1/$delta in $delta; stop))) [] (i/$delta, $delta/$delta1 in $delta; ($delta1/$delta in $delta; stop) ||| line!OFFHOOK_S:STATES; line!PREPARED_S:STATES; line!IDLE_S:STATES; SERVICE_PROVIDER.CONNECTION) 1 line!START:MESSAGES ->2 test!HOPE:SIGNALS ->3 line!OFFHOOK_S:STATES ->> >
Comentarios	• Una vez que seleccionamos el evento disponible, el nuevo estado en el que se encuentra la simulación es *2* (* State 2:) • La nueva expresión de comportamiento representa las secuencias de eventos disponibles desde el nuevo estado • Tres eventos pueden generar transiciones válidas desde aquí

Figura 6-6. Selección de un evento durante la simulación

Como puede observarse, *Ara Simulator* es una poderosa herramienta que nos permite analizar paso a paso los procesos de una especificación. Sin embargo, también resulta claro que el análisis para la especificación completa genera un gran número de transiciones y por lo tanto su inspección visual no es práctica.

Para el análisis de especificaciones más elaboradas, resulta más conveniente visualizar su espacio de estados completo. El espacio de estados representa todos los estados y

transiciones posibles. *Ara* nos permite generar el espacio de estados como un LTS[6] a partir de una especificación mediante la herramienta *Ara State Space Generator.*

6.6.3 Construcción del espacio de estados

El comportamiento de una especificación es mucho más claro si observamos su LTS correspondiente. *Ara* puede tomar un archivo que contenga una especificación válida y generar el espacio de estados en un archivo con extensión *.lts* (Figura 6-7).

Comando	:\> *arassg chsp.lot chsp.lts* **Enter**
Pantalla	ARA State Space Generator v2.00 (c) VTT Electronics 1992-1994 file chsp.lot opened file chsp.lot closed ....* ...1.... ....2....*....3....*....4....*...5 50 s, 2699 b ....*....6.... ....7....*....8....* 86 states in LTS 126 transitions in LTS >
Comentarios	• El archivo *chsp.lts* contiene el espacio de estados completo de la especificación • El espacio de estados contiene 86 estados y 126 transiciones

Figura 6-7. Generación del espacio de estados

El archivo que contiene el espacio de estados es *chsp.lts* y contiene la definición de todos los estados y transiciones generados. Este archivo puede ser visto en cualquier editor, pero resulta difícil deducir la funcionalidad de una especificación a partir de él. Para facilitar su comprensión, usamos la herramienta de visualización de *Ara.*

6.6.4 Graficación del espacio de estados de la especificación

Mediante *Ara Illustrator* es posible graficar el espacio de estados generado en el paso anterior. Esta herramienta toma como entrada el archivo *chsp.lts* y dibuja el grafo de transiciones correspondiente.

[6] *Labelled Transition Systems*

La Figura 6-8 nos muestra el espacio de estados graficado para la especificación del proveedor de servicios de telefonía. Es importante hacer notar que este grafo depende en gran medida de los procesos auxiliares. Esto se debe a que el proceso CHSP es muy complejo y soporta una gran cantidad de secuencias diferentes, pero los procesos auxiliares generan un número limitado de eventos. Estos eventos y las transiciones que provocan son los que aparecen en dicha gráfica.

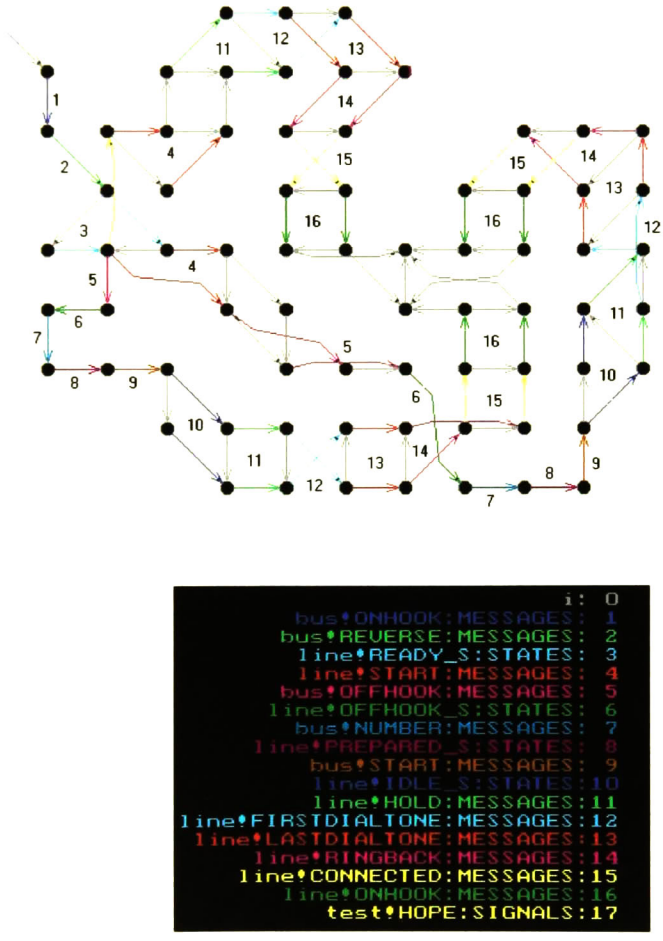
Comando Arasim	:\> araillu chsp.lts **Enter**
Pantalla	
Comentarios	• La figura muestra el espacio de estados completo para la especificación • Los colores representan mensajes específicos

Figura 6-8. Grafo de transiciones del espacio de estados

6.7 Comentarios finales

En este capítulo se presentó la especificación del proveedor de servicios de telefonía. Para ello se empleó el lenguaje de especificación formal LOTOS. La especificación es resultado de un ciclo de refinamientos sucesivos que toma como base la información presentada en el capítulo anterior. Este ciclo es soportado por el paquete *Ara Tools* que nos proporciona las herramientas para analizar la especificación, generar su espacio de estados y graficarlo.

7

Conclusiones y trabajo futuro

No puedo creer que Dios juegue a los dados con el cosmos.

Albert Einstein.

Dios no sólo juega a los dados, además a veces los lanza donde no pueden ser vistos

Stephen Hawking.

7.1 Conclusiones

En esta tesis se presentó el diseño de un proveedor de servicios de telefonía para el control de la llamada. Este proveedor de servicios se denomina *Communication Helper Service Provider* (*CHSP*). El *CHSP* fue definido mediante tres tipos de requerimientos: end-to-end, representados por diagramas de interacción; locales, representados por máquinas de estado finito; y globales, apegándose a la estructura de proveedores de servicio descentralizados definida por TAPI.

El diseño del *CHSP* fue realizado mediante un proceso iterativo de refinamiento basado en requerimientos, a través del cual se definieron todos los posibles escenarios de ejecución. El punto central de este proceso de refinamiento fue el uso de técnicas de descripción formal para crear especificaciones susceptibles de verificar. Durante la verificación se inspeccionó el espacio de estados generado por la especificación para detectar errores en la definición de los escenarios. Una vez que los errores detectados eran corregidos, se añadían nuevas secuencias y se iniciaba otro paso de refinamiento.

La especificación obtenida al final del ciclo de refinamiento tiene las siguientes características:

- *Facilidad de comprensión*
  La especificación muestra claramente la funcionalidad de cada uno de los módulos y los eventos mediante los cuales interactúan cada uno de ellos.

- *Generalidad*
  La estructura desarrollada para la especificación del *CHSP* es la de una máquina de estados general. Esto quiere decir, que puede ser usada para la especificación de cualquier otro tipo de proceso basado en estados.

  Extensibilidad
  Debido a que la estructura de la especificación es totalmente general, es posible añadir fácilmente nuevas extensiones para agregar funcionalidad adicional.

- *Facilidad de implementación*
  Llevar la especificación del *CHSP* a una implementación resulta muy sencillo. Esto se debe a que la gran mayoría de errores ya han sido detectados y corregidos durante el ciclo de refinamiento. Además, la especificación puede mapearse directamente a una estructura de datos que permita el avance por estados de la llamada telefónica.

7.2 Trabajo futuro

La especificación desarrollada contempla los casos de interacción entre las facilidades de llamadas iniciadas desde software y desde el aparato telefónico. Sin embargo, una gran cantidad de facilidades telefónicas han quedado fuera del alcance de este trabajo. Como trabajo futuro puede realizarse investigación en las siguientes áreas:

- Extender la especificación para incluir un mayor número de facilidades telefónicas
- Incorporar restricciones temporales a la ejecución de los servicios telefónicos
- Profundizar en la especificación de los procesos auxiliares mencionados en este trabajo
- Analizar la integración de la telefonía en Internet en la arquitectura presentada

Apéndice A.

A.1 Especificación del proveedor de servicios de telefonía

```
specification Service_Provider[Line,TeSt,Bus]:noexit

  $sort Messages is (OnHook, OffHook, Forward, Reverse, LineOutOfService,
                     ExtInService, Ring, Number, Start, Hold,
                     FirStDialTone, LaStDialTone, Ringback, BusyTone,
                     Connected, NoConnected, NoTone, CallerID,
                     CallAccept, StandardConnect, ExtendedConnect,
                     StandardResponse, ExtendedResponse, SWCall, HWCall)

  $sort States is (INIT_S, INITIALIZING_S, READY_S, IDLE_S, OFFHOOK_S,
                   PREPARED_S, DIALTONE_S, DIALING_S, ENDDIALING_S,
                   RINGBACK_S, SUBSCRIBERBUSY_S, CONNECTED_S, TIMEOUT_S,
                   EXTBUSY_S, LINEOUTOFSERVICE_S, REINIT_S,
                   CALLTERMINATE_S, NOTONE_S, WAITCALLERID_S,
                   WAITACCEPTCALL_S, RINGING_S, ONHOOK_S)

behaviour

  (
    HTT[Line,TeSt]
    |[TeSt]|
    VxD[Bus,TeSt]
  )
  |[Line,Bus]|
  CHSP[Line,Bus](INIT_S)

where

  process HTT[Line,TeSt]:exit:=
    Line!READY_S;
    (
      (
        SWCall[Line]
        >>
        (
          Line!OFFHOOK_S;
          Line!PREPARED_S;
          Line!IDLE_S;
          Connection[Line]
        )
        >>
        exit
      )
      []
      (
```

```
      (
        Line!OFFHOOK_S;
        Line!PREPARED_S;
        Line!IDLE_S;
        Connection[Line]
      )
      >>
      exit
    )
    []
    (
      TeSt!Hope;
      SWCall[Line]
      >>
      Connection[Line]
      >>
      exit
    )
  )

endproc

process VxD[Bus,TeSt]:exit:=
  Initialize[Bus]
  >>
  (
    HWCall[Bus,TeSt]
  )
  >>
  exit
endproc

process SWCall[Line]:exit:=
  Line!Start;
  exit
endproc

process HWCall[Bus,TeSt]:exit:=
  (
    TeSt!Hope;
    exit
  )
  []
  (
    Bus!OffHook;
    Bus!Number;
    Bus!Start;
    exit
  )
endproc
```

```
process Initialize[Bus]:exit:=
  Bus!OnHook;
  (
    Bus!Reverse;
    exit
  )
endproc

process Connection[Line]:exit:=
  Line!Hold;
  Line!FirStDialTone;
  Line!LaStDialTone;
  Line!Ringback;
  Line!Connected;
  Line!OnHook;
  exit
endproc

process CHSP[Line,Bus](St:States):exit:=
  (
    Bus?Ms:Messages;
    SM[Line,Bus](St,Ms)
  )
  []
  (
    Line?Ms:Messages;
    SM[Line,Bus](St,Ms)
  )
endproc

process SM[Line,Bus](St:States,Ms:Messages):exit:=
  [Ms eq OnHook]->
    (
      [St eq INIT_S]->
        CHSP[Line,Bus](INITIALIZING_S)
      []
      [St eq OFFHOOK_S]->
        CHSP[Line,Bus](READY_S)
      []
      [St eq PREPARED_S]->
        CHSP[Line,Bus](READY_S)
      []
      [St eq DIALTONE_S]->
        CHSP[Line,Bus](REINIT_S)
      []
      [St eq DIALING_S]->
        CHSP[Line,Bus](REINIT_S)
      []
      [St eq ENDDIALING_S]->
        CHSP[Line,Bus](REINIT_S)
      []
      [St eq RINGBACK_S]->
        CHSP[Line,Bus](CALLTERMINATE_S)
      []
```

```
      [St eq SUBSCRIBERBUSY_S]->
        CHSP[Line,Bus](SUBSCRIBERBUSY_S)
      []
      [St eq CONNECTED_S]->
        CHSP[Line,Bus](CALLTERMINATE_S)
      []
      [St eq EXTBUSY_S]->
        CHSP[Line,Bus](EXTBUSY_S)
      []
      [St eq LINEOUTOFSERVICE_S]->
        CHSP[Line,Bus](LINEOUTOFSERVICE_S)
      []
      [St eq REINIT_S]->
        CHSP[Line,Bus](READY_S)
      []
      [St eq NOTONE_S]->
        CHSP[Line,Bus](NOTONE_S)
      []
      [St eq WAITCALLERID_S]->
        CHSP[Line,Bus](WAITCALLERID_S)
      []
      [St eq WAITACCEPTCALL_S]->
        CHSP[Line,Bus](WAITACCEPTCALL_S)
      []
      [St eq RINGING_S]->
        CHSP[Line,Bus](RINGING_S)
    )
    []
  [Ms eq OffHook]->
    (
      [St eq INIT_S]->
        CHSP[Line,Bus](INIT_S)
      []
      [St eq INITIALIZING_S]->
        CHSP[Line,Bus](INIT_S)
      []
      [St eq READY_S]->
        Line!OFFHOOK_S;
        CHSP[Line,Bus](OFFHOOK_S)
      []
      [St eq IDLE_S]->
        Line!OFFHOOK_S;
        CHSP[Line,Bus](OFFHOOK_S)
      []
      [St eq DIALTONE_S]->
        CHSP[Line,Bus](DIALTONE_S)
      []
      [St eq DIALING_S]->
        CHSP[Line,Bus](DIALING_S)
      []
      [St eq ENDDIALING_S]->
        CHSP[Line,Bus](ENDDIALING_S)
      []
      [St eq RINGBACK_S]->
```

```
        CHSP[Line,Bus](RINGBACK_S)
      []
      [St eq SUBSCRIBERBUSY_S]->
        CHSP[Line,Bus](SUBSCRIBERBUSY_S)
      []
      [St eq CONNECTED_S]->
        CHSP[Line,Bus](CONNECTED_S)
      []
      [St eq TIMEOUT_S]->
        CHSP[Line,Bus](TIMEOUT_S)
      []
      [St eq EXTBUSY_S]->
        CHSP[Line,Bus](EXTBUSY_S)
      []
      [St eq LINEOUTOFSERVICE_S]->
        CHSP[Line,Bus](LINEOUTOFSERVICE_S)
      []
      [St eq CALLTERMINATE_S]->
        CHSP[Line,Bus](CALLTERMINATE_S)
      []
      [St eq NOTONE_S]->
        CHSP[Line,Bus](NOTONE_S)
      []
      [St eq WAITCALLERID_S]->
        CHSP[Line,Bus](WAITCALLERID_S)
      []
      [St eq WAITACCEPTCALL_S]->
        CHSP[Line,Bus](WAITACCEPTCALL_S)
      []
      [St eq RINGING_S]->
        CHSP[Line,Bus](CONNECTED_S)
    )
    []
  [(Ms eq Forward) or (Ms eq Reverse)]->
    (
      [St eq INITIALIZING_S]->
        Line!READY_S;
        CHSP[Line,Bus](READY_S)
      []
      [St eq READY_S]->
        CHSP[Line,Bus](READY_S)
      []
      [St eq IDLE_S]->
        CHSP[Line,Bus](IDLE_S)
      []
      [St eq OFFHOOK_S]->
        CHSP[Line,Bus](OFFHOOK_S)
      []
      [St eq PREPARED_S]->
        CHSP[Line,Bus](PREPARED_S)
      []
      [St eq DIALTONE_S]->
        CHSP[Line,Bus](DIALTONE_S)
      []
```

```
        [St eq DIALING_S]->
          CHSP[Line,Bus](DIALING_S)
        []
        [St eq ENDDIALING_S]->
          CHSP[Line,Bus](ENDDIALING_S)
        []
        [St eq RINGBACK_S]->
          CHSP[Line,Bus](RINGBACK_S)
        []
        [St eq SUBSCRIBERBUSY_S]->
          CHSP[Line,Bus](REINIT_S)
        []
        [St eq CONNECTED_S]->
          CHSP[Line,Bus](CONNECTED_S)
        []
        [St eq TIMEOUT_S]->
          CHSP[Line,Bus](REINIT_S)
        []
        [St eq EXTBUSY_S]->
          CHSP[Line,Bus](REINIT_S)
        []
        [St eq LINEOUTOFSERVICE_S]->
          CHSP[Line,Bus](REINIT_S)
        []
        [St eq REINIT_S]->
          CHSP[Line,Bus](REINIT_S)
        []
        [St eq CALLTERMINATE_S]->
          CHSP[Line,Bus](REINIT_S)
        []
        [St eq NOTONE_S]->
          CHSP[Line,Bus](REINIT_S)
        []
        [St eq WAITCALLERID_S]->
          CHSP[Line,Bus](REINIT_S)
        []
        [St eq WAITACCEPTCALL_S]->
          CHSP[Line,Bus](REINIT_S)
        []
        [St eq RINGING_S]->
          CHSP[Line,Bus](REINIT_S)
      )
      []
    [Ms eq LineOutOfService]->
      (
        [St eq INIT_S]->
          CHSP[Line,Bus](INIT_S)
        []
        [St eq INITIALIZING_S]->
          CHSP[Line,Bus](INITIALIZING_S)
        []
        [St eq READY_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
```

```
        [St eq IDLE_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq OFFHOOK_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq PREPARED_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq DIALTONE_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq DIALING_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq ENDDIALING_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq RINGBACK_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq SUBSCRIBERBUSY_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq CONNECTED_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq TIMEOUT_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq EXTBUSY_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq REINIT_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq CALLTERMINATE_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq NOTONE_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq WAITCALLERID_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq WAITACCEPTCALL_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq RINGING_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
      )
      []
    [Ms eq ExtInService]->
      (
```

```
[St eq INIT_S]->
  CHSP[Line,Bus](INIT_S)
[]
[St eq INITIALIZING_S]->
  CHSP[Line,Bus](INITIALIZING_S)
[]
[St eq READY_S]->
  CHSP[Line,Bus](EXTBUSY_S)
[]
[St eq IDLE_S]->
  CHSP[Line,Bus](EXTBUSY_S)
[]
[St eq OFFHOOK_S]->
  CHSP[Line,Bus](EXTBUSY_S)
[]
[St eq PREPARED_S]->
  CHSP[Line,Bus](EXTBUSY_S)
[]
[St eq DIALTONE_S]->
  CHSP[Line,Bus](DIALTONE_S)
[]
[St eq DIALING_S]->
  CHSP[Line,Bus](DIALING_S)
[]
[St eq ENDDIALING_S]->
  CHSP[Line,Bus](ENDDIALING_S)
[]
[St eq RINGBACK_S]->
  CHSP[Line,Bus](RINGBACK_S)
[]
[St eq SUBSCRIBERBUSY_S]->
  CHSP[Line,Bus](EXTBUSY_S)
[]
[St eq CONNECTED_S]->
  CHSP[Line,Bus](CONNECTED_S)
[]
[St eq TIMEOUT_S]->
  CHSP[Line,Bus](EXTBUSY_S)
[]
[St eq LINEOUTOFSERVICE_S]->
  CHSP[Line,Bus](EXTBUSY_S)
[]
[St eq REINIT_S]->
  CHSP[Line,Bus](EXTBUSY_S)
[]
[St eq CALLTERMINATE_S]->
  CHSP[Line,Bus](EXTBUSY_S)
[]
[St eq NOTONE_S]->
  CHSP[Line,Bus](EXTBUSY_S)
[]
[St eq WAITCALLERID_S]->
  CHSP[Line,Bus](EXTBUSY_S)
[]
```

```
      [St eq WAITACCEPTCALL_S]->
        CHSP[Line,Bus](EXTBUSY_S)
      []
      [St eq RINGING_S]->
        CHSP[Line,Bus](EXTBUSY_S)
    )
    []
  [Ms eq Ring]->
    (
      [St eq INITIALIZING_S]->
        CHSP[Line,Bus](INITIALIZING_S)
      []
      [St eq READY_S]->
        CHSP[Line,Bus](WAITCALLERID_S)
      []
      [St eq IDLE_S]->
        CHSP[Line,Bus](WAITCALLERID_S)
      []
      [St eq WAITCALLERID_S]->
        CHSP[Line,Bus](WAITCALLERID_S)
      []
      [St eq WAITACCEPTCALL_S]->
        CHSP[Line,Bus](WAITACCEPTCALL_S)
      []
      [St eq RINGING_S]->
        CHSP[Line,Bus](RINGING_S)
    )
    []
  [Ms eq Number]->
    (
      [St eq OFFHOOK_S]->
        Line!PREPARED_S;
        CHSP[Line,Bus](PREPARED_S)
      []
      [St eq EXTBUSY_S]->
        CHSP[Line,Bus](EXTBUSY_S)
    )
    []
  [Ms eq Start]->
    (
      [St eq READY_S]->
        CHSP[Line,Bus](IDLE_S)
      []
      [St eq PREPARED_S]->
        Line!IDLE_S;
        CHSP[Line,Bus](IDLE_S)
    )
    []
  [Ms eq Hold]->
    (
      [St eq IDLE_S]->
        CHSP[Line,Bus](DIALTONE_S)
    )
    []
```

```
[Ms eq FirStDialTone]->
  (
    [St eq DIALTONE_S]->
      CHSP[Line,Bus](DIALING_S)
  )
  []
[Ms eq LaStDialTone]->
  (
    [St eq DIALING_S]->
      CHSP[Line,Bus](ENDDIALING_S)
  )
  []
[Ms eq Ringback]->
  (
    [St eq ENDDIALING_S]->
      CHSP[Line,Bus](RINGBACK_S)
    []
    [St eq RINGBACK_S]->
      CHSP[Line,Bus](RINGBACK_S)
  )
  []
[Ms eq BusyTone]->
  (
    [St eq ENDDIALING_S]->
      CHSP[Line,Bus](SUBSCRIBERBUSY_S)
    []
    [St eq RINGBACK_S]->
      CHSP[Line,Bus](TIMEOUT_S)
  )
  []
[Ms eq Connected]->
  (
    [St eq ENDDIALING_S]->
      CHSP[Line,Bus](CONNECTED_S)
    []
    [St eq RINGBACK_S]->
      CHSP[Line,Bus](CONNECTED_S)
  )
  []
[Ms eq NoConnected]->
  (
    [St eq RINGBACK_S]->
      CHSP[Line,Bus](TIMEOUT_S)
  )
  []
[Ms eq NoTone]->
  (
    [St eq ENDDIALING_S]->
      CHSP[Line,Bus](NOTONE_S)
  )
  []
[Ms eq CallerID]->
  (
    [St eq EXTBUSY_S]->
```

```
          CHSP[Line,Bus](EXTBUSY_S)
        []
        [St eq LINEOUTOFSERVICE_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq REINIT_S]->
          CHSP[Line,Bus](REINIT_S)
        []
        [St eq WAITCALLERID_S]->
          CHSP[Line,Bus](WAITACCEPTCALL_S)
      )
      []
    [Ms eq CallAccept]->
      (
        [St eq EXTBUSY_S]->
          CHSP[Line,Bus](EXTBUSY_S)
        []
        [St eq LINEOUTOFSERVICE_S]->
          CHSP[Line,Bus](LINEOUTOFSERVICE_S)
        []
        [St eq REINIT_S]->
          CHSP[Line,Bus](REINIT_S)
        []
        [St eq WAITACCEPTCALL_S]->
          CHSP[Line,Bus](RINGING_S)
      )
  endproc

endspec
```

Referencias

[*AATOS94*] **AATOS Research Group.** Ara Tools 2.0. User's Manual. VTT Electronics, Finland, 1994.

[*AATOS94b*] **AATOS Research Group.** Ara Tools 2.0. Reference Manual. VTT Electronics, Finland, 1994.

[*Aho90*] **Aho A., Sethi R., Ullman J.** Compiladores. Principios, Técnicas y Herramientas. Addison-Wesley Iberoamericana, 1990.

[*ALGO97*] **ALGO Communications Corporation.** System Arquitecture of Windows Telephony API products. http://www.algocomm.com/ctimio/tapi.htm

[*Biebow85*] **Biebow B., Hagel Stein, J.** Algebraic Specifications of Syncronization and errors: A telephone example. Lecture Notes in Computer Science, Vol 186, 294-308, 1985.

[*Blom94*] **Blom J., Jonsson B., Kempe L.** Using temporal logic for modular specifications of telephone services. Feature Interactions in telecommunication systems, Amsterdam Oxford, Washington DC Tokyo, 1994. IOS Press, 197-216(42)

[*Boehm88*] **Boehm B.** A spiral model of software development and enhancement. IEEE Computer, 1988.

[*Bol87*] **Bolognesi B., Brinksma E.** Introduction to the ISO Specification Language LOTOS. Computer Networks and ISDN Systems 14, 1987.

[Boumezbeur91] **Boumezbeur R.** Design, Specification, and Validation of Telephony Systems in LOTOS. Master degree thesis, Ottawa-Carleton Institute of Computer Science, 1991.

[*Cameron93*] **Cameron E.J., Griffeth N., Linand Y.** A feature-interaction benchmark for IN and beyond. IEEE Communications Magazine, 31(3):64-69, 1993

[*Carranza01*] **Carranza A.** Análisis y Diseño de una Aplicación CTI. Master degree thesis, CINVESTAV-IPN, 2001.

[*Charnois97*] **Charnois T.** A Natural Language Processing Approach for Avoidance of Feature Interactions. Feature interactions in telecommucation systems IV, 1997. IOS Press, 347-363

[*Chen94*] **Chen Y. L., Lafortune S., Lin F.** Study of feature interactions in telecommunication systems using the supervisory control theory of discrete event systems. Technical report CGR-94-14, University of Michigan, November 1994(121)

[*Coulouris96*] **Coulouris G., Dollimore J., Kindberg T.** Distributed Systems. Concepts and Design. Addison-Wesley, Second Edition, 1996.

[*Dataquest99*] **Dataquest, Inc.** Planting Some Stakes - The North American CTI Market, 1999

[*D'Hooge95*] **D'Hooge H., Nixon T.** TAPI Tutorial. CT Expo. 1995

[*Davis88*] **Davis M.** A strategy for comparing alternative software development cycle models. IEEE Trans. On Software Engineering, 1988.

[*Ehr85*] **Ehrig H., Mahr B.** Fundamentals of Algebraic Specification 1. Springer-Verlag, Berlin, 1985.

[*ESA87*] **ESA.** Software engineering standards. Technical Report, European Space Agency, Netherlands, 1987.

[*Figueroa01*] **Figueroa M.** Diseño de Hardware de Telefonía. Master degree thesis, CINVESTAV-IPN, 2001.

[*Frappier97*] **Frappier M., Mili A., Desharnais J.** Detecting Feature Interactions on Relational Specifications. Feature Interactions in Telecommunication and Distributed Systems IV. IOS Press, 1997.

[*Fls91*] **Faci M., Logrippo L., Stepien B.** Formal Specification of Telephone Systems in LOTOS: The Constraint-Oriented Approach. Computer Networks and ISDN Systems, 1991.

[*Gibson97*] **Gibson J. Paul.** Feature Requeriments Models: Understanding Interactions. Feature Interactions in Telecommunication and Distributed Systems IV. IOS Press, 1997.

[*Hoa85*] **Hoare C.** Communicating Sequential Processes. Prentice-Hall, 1985.

[*Hopcroft93*] **Hopcroft J., Ullman J.** Introducción a la teoría de Autómatas, Lenguajes y Computación. CECSA, 1993.

[*Iraqui97*] **Iraqui Y., Erradi M.** An experiment for the processing of Feature Interactions within an object-oriented environment. Feature Interactions and its Experimental Evaluation. Feature interactions in telecommucation systems IV, 1997. IOS Press, 298-312

[*ISO89*] **International Organization for Standardization.** IS 8807: LOTOS: A Formal Description Technique Based on the Temporal Ordering of Observational Behavior, 1989.

[*ISO89a*] **International Organization for Standardization.** IS 9074: Estelle: A Formal Description Technique Based on an Extended State Transition Model, 1989.

[*ITUG711*] **International Telecommunication Union.** Recomendation G.711 (11/88)- Pulse code modulation (PCM) of voice frequencies.
http://www.itu.int/itudoc/itu-t/rec/g/g700-799/g711.html

[*Jensen87*] **Jensen, K.** Coloured Petri nets. Lecture Notes in Computer Science. Vol 254, 1987, 248-299

[*Log88*] **Logrippo L., Obaid A., Briand J.P., Fehri M.C.** An Interpreter for LOTOS, A Specification Language for Distributed Systems. Software-Practice and Experience 18, 1988.

[*Man89*] **Mañas J.A., Miguel-More T.** From LOTOS to C. Formal Description Techniques. North-Holland, 1989.

[*Microsoft98*] **Microsoft Corporation.** Introduction to MCI. Microsoft Corporation.
http://msdn.microsoft.com/LIBRARY/DDKDOC/NTDDK/NATIVE/DDK/SG/SRC/MCIDRV_1.HTM

[*Microsoft98b*] **Microsoft Corporation.** TSPI 2.1 Programmers's Reference. Microsoft Corporation, 1998.

[*Microsoft98a*] **Microsoft Corporation.** Microsoft Windows Architecture Training. Microsoft Press, 1998.

[*Mil80*] **Milner R.** A Calculus of Communicating Systems. Lecture Notes in Computer Science 92. Springer-Verlag, 1980.

[*Mil89*] **Milner R.** Communication and Concurrency. Prentice-Hall, 1989.

[*Mullender95*] **Mullender S.** Distributed Systems. Addison-Wesley, Second Edition, 1995.

[*Nakamura97*] **Nakamura M., Kakuda Y., Kikuno T.** Petri-Net Based Detection Method for Non-Deterministic Feature Interactions and its Experimental Evaluation. Feature interactions in telecommucation systems IV, 1997. IOS Press, 138-152

[*Padilla00*] **Padilla R.** Análisis y Diseño de un Habilitador CTI basado en TAPI. Master degree thesis, CINVESTAV-IPN, 2000.

[*Prehofer97*] **Prehofer C.** An Object-Oriented Approach to Feature Interaction. Feature interactions in telecommucation systems IV, 1997. IOS Press, 313-325

[*Schulzrinne99*] **Schulzrinne H.** Service for Telecom Version 2. IEEE Internet Computing, 1999.

[*Spragins91*] **Spragins J.D.** Telecommunications, Protocols and Design. Addison-Wesley, 1991.

[*Stepien95*] **Stepien B., Logrippo L.** Representing and verifying intentions in telephone features using abstact data types. Feature interactions in telecommucation systems III, Amsterdam Oxford Washington Tokyo, 1995. IOS Press, 141-156(42)

[*Tanenbaum97*] **Tanenbaum A.S.** Computer Networks. Third Edition. Prentice Hall, 1997.

[*Turner93*] **Turner, Kenneth J.** Using Formal Description Techniques. An Introduction to ESTELLE, LOTOS and SDL. Wiley, 1993.

[*UNISYS97*] **UNISYS.** Telephony Application Programming Interface. HOTtech Brief. http://www.pc.unisys.com/tech_papers/tapi.htm

[*Vissers88*] **Vissers C., Scollo G., Van Sinderen M.** Architecture and Specification Style in Formal Descriptions of Distributed Systems. Protocol Specification, Testing, and Verification VIII, North-Holland, 1988.

[*Vissers95*] **Vissers C., Logrippo L.** On the importance of the service concept in the design of data communication protocols. Proc. Of Protocol Specification, Testing and Verification VIII, North-Holland Amsterdam, 1995.

[*Walters97*] **Walters R.** CTI in Action. Communications in Business Series. Wiley, 1997.

[*Zave85*] **Zave P.** The distributed alternative to finite-state-machine specifications. ACM Trans. On prog. Lang. And systems, 7, No 1, Jan 1985, 10-36

[*Zave93*] **Zave P.** Feature Interactions and formal specifications in telecommunications. IEEE Computer magazine, pags. 20-29, August 1993

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó la tesis: "Diseño de un proveedor de servicios de telefonía para el control de la llamada usando técnicas de descripción formal" del Lic. Luis Prabú Ximénez Cárdenas el día 28 de Septiembre de 2001.

Dr. José Luis Leyva Montiel
Investigador Cinvestav 3 B
CINVESTAV DEL IPN
Guadalajara

Dr. Deni Librado Torres Román
Investigador Cinvestav 3 A
CINVESTAV DEL IPN
Guadalajara

Dr. Félix Francisco Ramos Corchado
Investigador Cinvestav 2 A
CINVESTAV DEL IPN
Guadalajara