UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERIA

SERVIDOR DE ENVÍO DE CONTENIDOS VÍA BLUETOOTH BASADO EN LINUX

TESIS QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN TELECOMUNICACIONES

PRESENTA:

ALEJANDRO ISLAS LÓPEZ

DIRECTOR DE TESIS:

DR. LUIS ANDRÉS BUZO DE LA PEÑA

2010

Profesionalmente, han sido muchos los maestros, ingenieros y técnicos tanto en la UNAM como en KB/Tel y Softel a los que tengo que agradecer por todas sus enseñanzas y consejos, definitivamente han sido parte importante en mi desarrollo. En especial, quisiera agradecer al Ingeniero Enrique Rifé Rodríguez, por sembrar en mí desde muy joven el interés por la ingeniería pero sobre todo, por su ejemplo de vida y amistad.

A mi padre, por esos 12 años de enseñanzas que llevaré conmigo siempre.

A Salvador y María, por siempre caminar a mi lado.

A Gaby, Malu, Carlo, Gamo y Muñoz por su amistad y complicidad constante a lo largo de todos estos años.

A Claudia, por la gran historia que viví contigo flaca.

A mi madre, por convertirme en el hombre que soy.

ÍNDICE

INTRODUCCIÓN

El avance de las redes y dispositivos móviles existentes en la actualidad permiten el intercambio de información prácticamente en cualquier tipo de entorno. En especial, la incorporación de dispositivos con tecnología Bluetooth en teléfonos celulares, computadoras personales, etc, facilita dicho intercambio a través de una tecnología inalámbrica de corto alcance ampliamente aceptada en la industria. Las aplicaciones del envío de contenidos vía Bluetooth son variadas e incluyen escenarios como la publicidad o marketing de proximidad y envío de textos informativos en museos, por citar algunos ejemplos.

Sin embargo, la gran variedad de entornos y aplicaciones que pueden hacer uso de redes y dispositivos móviles hacen mandatario entender los requerimientos y pasos a seguir necesarios para lograr de manera exitosa el desarrollo de un sistema que haga uso de estas tecnologías. Por lo tanto, es necesario hacer uso de metodologías de diseño/modelado que nos permitan entender el problema en su totalidad al ver al sistema como un todo formado por la interacción ordenada de cada una de sus partes.

El objetivo de esta tesis es desarrollar un servidor basado en el sistema operativo Linux (software libre) que permita establecer comunicación entre un punto central y dispositivos móviles vía Bluetooth para el intercambio organizado de cualquier tipo de contenidos como pueden ser imágenes, textos, sonidos, etc. El sistema deberá hacer uso de las librerías existentes en Linux para el manejo de dispositivos Bluetooth e intercambio de información entre ellos. Los requerimientos, diseño y alcances del servidor deberán ser definidos siguiendo principios de modelado de comportamiento propuestos por la ingeniería de sistemas y haciendo uso de los lenguajes de modelado UML (Lenguaje Unificado de Modelado) y SysML (Lenguaje de Modelado de Sistemas).

La información contenida en el documento será dividida de la siguiente manera:

-La introducción hará referencia a las características principales del sistema operativo Linux así como de los principios básicos para modelado de sistemas y desarrollo de software.

-El capítulo 1 hará una descripción de la tecnología Bluetooth así como de los perfiles/protocolos involucrados en el intercambio de archivo entre dispositivos que hacen uso de dicha tecnología. También hará referencia a las librerías existentes en Linux para el manejo de conexiones Bluetooth.

-El capítulo 2 se concentrará en los lenguajes gráficos de modelado UML y SysML. Hará una descripción de los conceptos, notaciones gráficas y diagramas definidos en ambos lenguajes que serán utilizados en el modelado del servidor.

-El capítulo 3 estará compuesto por el modelo de comportamiento propuesto para el servidor de envío de contenidos. Dicho modelo contendrá los requerimientos necesarios así como todos los diagramas de comportamiento involucrados en el diseño.

-El capítulo 4 definirá casos de prueba del sistema así como los resultados obtenidos y las conclusiones generales.

Cabe hacer mención que si bien todos los términos utilizados en la tesis serán traducidos al español, entre paréntesis y en cursiva se dejará el nombre original en inglés. Esto debido a que la gran mayoría de la bibliografía se encuentra en ese idioma, por lo que el nombre en inglés es útil para cualquier futura referencia.

LINUX

Linux es miembro de una familia de sistemas operativos basados en Unix, surgidos principalmente en la década de l990, entre los que se encuentran sistemas operativos comerciales como el sistema V Versión 4 (System V Release 4 SVR4) desarrollado por AT&T, BSD 4.4 desarrollado por la Universidad de Berkeley en California, Solaris de Sun Microsystems, AIX de IBM y HP-UX de Hewlett-Packard. Todos los sistemas operativos parecidos a Unix, comerciales o no comerciales, acordaron el uso de ciertos estándares como el de Sistemas Operativos Portables basados en Unix (Portable Operating Systems based on Unix, POSIX) del IEEE. De la misma forma, comparten ideas y características de diseño. Por lo tanto, Linux es comparable, en su esencia, con cualquier otro de los sistemas operativos basados en Unix.

A pesar de compartir características con los sistemas operativos antes mencionados, uno de los atractivos principales de Linux es que no es un sistema operativo con fines comerciales. Al contrario, todo su código fuente está protegido por una licencia pública GNU, lo que permite que el código sea completamente abierto para cualquiera que quiera estudiarlo o modificarlo.

Linux fue desarrollado en 1991 por Linus Torvald como un sistema operativo compatible con las computadores personales IBM basado en el microprocesador 80386 de Intel. Con el paso de los años, diferentes desarrolladores se han encargado de portar Linux a diferentes arquitecturas de hardware, como lo son Alpha, Sparc, PowerPC o Motorola 680x0. Su robustez y adaptabilidad hizo que primeramente fuera utilizado como una plataforma ideal para servidores de alto desempeño. Con el paso de los años cada vez es mayor su uso como sistema operativo para computadoras personales o de oficina (BOVET, 2000). Linux también tiene la funcionalidad de poder ser incorporado de manera directa en microprocesadores para realizar sistemas embebidos (*embedded systems*), los cuales gozan de gran popularidad en la actualidad en todo tipo de aplicaciones.

A continuación mencionaremos las características principales del kernel (núcleo) de Linux, así como variaciones que dicho kernel tiene con respecto a otros sistemas operativos basados en Unix:

- El kernel de Linux es monolítico. Puede ser visto como un programa complejo compuesto por diferentes componentes lógicos. Esta característica es compartida por todos los sistemas operativos basados en Unix.

-Un kernel tradicional de Unix compila y enlaza el código de manera estática. Un kernel más moderno puede cargar y descargar componentes del código del kernel, principalmente para el manejo de dispositivos externos, de manera dinámica. El soporte que Linux ofrece para cargar módulos bajo demanda es muy bueno y es una característica poco común entre otras versiones comerciales de Unix.

-Kernel Multi hilos (*Kernel Threading*). Un hilo (*thread*) del kernel es un contexto de ejecución que puede ser operado de manera independiente, puede estar asociado con un programa de usuario o puede ser que se dedique exclusivamente a ejecturar un conjunto limitado de funciones del kernel. A diferencia de otros sistemas operativos como Solaris, Linux sólo ofrece un comportamiento multi hilos limitado, para uso exclusivo de procesos que ejecuten funciones del kernel de manera periódica.

-Soporte Multi hilos a nivel aplicación. Una aplicación multi hilos puede verse como un conjunto de procesos ligeros (*ligthweight processes*) que comparten un espacio de

direccionamiento físico de memoria, directorios o archivos. Mientras otros sistemas como SVR4 o Solaris hacen uso de hilos del kernel para este tipo de aplicaciones, Linux controla la ejecución de dichos procesos ligeros mediante llamadas continuas de ejecución.
-Soporte Multiprocesador. A partir de su versión 2.2, Linux no sólo soporta un esquema basado en el uso de varios procesadores, sino que también no establece ningún tipo de prioridad sobre ellos, por lo que cualquier procesador disponible puede ejecutar cualquier tarea.
-Sencillez de Hardware. Los requerimientos de Hardware necesitados por Linux son pocos, lo que permite instalar el sistema operativo en procesadores antiguos como lo es el 80386 de Intel con solo 4 MB de memoria RAM.
-Estabilidad. Debido a que su diseño tiene como pilar fundamental la eficiencia, los sistemas basados en Linux suelen ser muy estables con un porcentaje de falla muy bajo y con ciclos de mantenimiento muy esporádicos.
-Compatibilidad. Linux es capaz montar archivos de sistema de diferentes plataformas, incluyendo MS DOS, MS Windows, MAC OS, Solaris, etc. Linux también es capaz de interactuar con diversas capas de red como Ethernet, FDDI o Token Ring. Inclusive, haciendo uso de las librerías adecuadas, Linux es capaz de ejecutar aplicaciones diseñadas para otros sistemas operativos.
-Soporte. A pesar de ser software libre, puede ser mucho más sencillo obtener parches o actualizaciones para Linux que para cualquier otro sistema operativo propietario. El procedimiento a seguir puede reducirse a enviar un correo a la lista de correo o grupo de trabajo adecuado. En cuanto a controladores (*drivers*) para el manejo de nuevos productos de hardware, éstos suelen liberarse sólo unas semanas después de la aparición de dichos productos en el mercado. Al ser código completamente abierto para todo el mundo, la cantidad de personas involucradas en el soporte/mantenimiento de Linux superará normalmente por mucho a las que cualquier otro sistema operativo propietario pueda tener.

Conceptos Básicos del Sistema Operativo Linux
Un sistema de computación debe incluir un conjunto básico de programas llamado sistema operativo, donde aquellos de mayor importancia son denominados kernel o núcleo. El sistema operativo suele ser cargado en memoria RAM cuando el sistema arranca y contiene los procedimientos críticos necesarios para que el sistema pueda operar.

Todo sistema operativo debe cumplir con las siguientes funciones:
-Interactuar con los elementos de hardware existentes en el sistema.
-Proveer un ambiente de ejecución para las aplicaciones de nivel usuario existentes en el sistema.

Algunos sistemas operativos, como MS-DOS, permiten al usuario interactuar directamente con los componentes de hardware del sistema. Por el contrario, Linux esconde a las aplicaciones de nivel usuario todos los detalles de bajo nivel relativos a la organización física. Cuando un programa a nivel aplicación quiera hacer uso de un recurso de hardware, debe hacer una petición al sistema operativo. El kernel evalúa la petición y en caso de proceder, interactúa con el componente de hardware en nombre del usuario.

Como la gran mayoría de sistemas operativos modernos, Linux es un sistema operativo multiusuario, lo que implica que es capaz de ejecutar de manera concurrente e independiente diversas aplicaciones pertenecientes a diferentes usuarios. Como todo sistema operativo multiusuario, Linux debe cumplir con ciertas características:

-Sistema de autentificación para validar la identidad de los usuarios.

-Mecanismos de protección contra programas defectuosos a nivel usuarios que pueden llegar a bloquear otras aplicaciones ejecutándose en el sistema.

-Mecanismos de protección contra programas maliciosos a nivel usuario que puedan interferir o espiar la actividad de otros usuarios.

-Mecanismos de administración de recursos que limitan la cantidad de recursos del sistema que se le pueden asignar a cada usuario.

En Linux, cada usuario tiene un espacio privado en la máquina, es decir es dueño de un espacio de almacenamiento para su uso personal. Es responsabilidad del sistema operativo garantizar que sólo el usuario tenga acceso a la información contenido en su espacio de almacenamiento. Los usuarios son identificados por un número único o User ID (UID). Cada UID tiene asociado un nombre de acceso y una contraseña. De forma que puedan compartir información de manera selectiva con otros usuarios, cada usuario puede ser miembro de uno o más grupos, identificados por un identificador de grupo o Group ID (GID).

Como cualquier otro sistema operativo basado en Unix, Linux tiene un usuario especial llamado supervisor, superusuario o root. El administrador del sistema deberá registrarse al sistema a través de dicho usuario de forma que pueda administrar las cuentas de usuario existentes o realizar tareas de mantenimiento o actualización del kernel del sistema operativo. El usuario root puede hacer prácticamente todo, desde acceso a los componentes de hardware del sistema como accesos a todos los archivos y programas de los usuarios.

Todos los sistemas operativos definen con concepto abstracto fundamental: el proceso. Un proceso puede definirse como una instancia de un programa en ejecución o como el contexto de ejecución de un programa. En sistemas operativos tradicionales, un proceso ejecuta una secuencia de instrucciones dentro de un espacio de direcciones, lo cuál no es más que un conjunto de direcciones de memoria a las cuales el proceso tiene permiso a accesar. En la actualidad, los sistemas operativos permiten la ejecución de procesos con múltiples flujos de ejecución, es decir, múltiples secuencias de instrucciones ejecutadas en el mismo espacio de direcciones. Los sistemas operativos multiusuarios deben proveer un ambiente de ejecución en donde varios procesos pueden competir activamente entre si por los recursos del sistema, principalmente el CPU. Este tipo de comportamiento de los sistemas operativos se define como multiprocesamiento. Como parte del multiprocesamiento, el sistema operativo deberá incluir políticas preventivas que permitan rastrear cuánto tiempo de CPU es utilizado por cada uno de los procesos activos, de forma que pueda equilibrar y administrar el uso de los mismos.

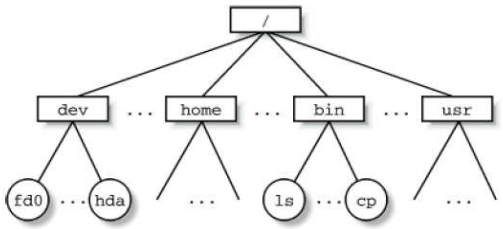
Linux es un sistema operativo multiprocesamiento con procesos preventivos. En realidad, la abstracción proceso es fundamental en todo sistema tipo Unix. Aún cuando no haya ningún usuario registrado en el sistema, varios procesos de sistemas realizan diferentes funciones como es el monitoreo de los componentes de hardware del sistema o esperar por solicitudes de acceso de usuarios en las terminales del sistema. Linux hace uso del modelo proceso/kernel, dónde cada proceso tiene la ilusión de ser el único

proceso activo en el sistema con acceso exclusivo a los servicios. En realidad, el acceso de los usuarios a los recursos es administrado a través de llamados de sistema que ejecutan procesos limitados propios del kernel del sistema.

Sistema de archivos (FileSystem) de Linux

El diseño del sistema operativo Linux está basado en su sistema de archivos. A continuación haremos una breve descripción de sus características principales.

En Linux (o Unix) un archivo es un contenedor de información estructurado como una secuencia de bytes, sin embargo el kernel no interpreta el contenido de dichos archivos, sino que son interpretados por librerías de alto nivel que hacen uso del kernel a través de llamados al sistema. Desde la perspectiva de un usuario, los archivos son organizados en una jerarquía tipo árbol como lo muestra la siguiente figura:



Todos los nodos del árbol, con excepción de las hojas, denotan nombres de directorios. El nombre de un directorio contiene información de los directorios y archivos existentes por debajo de él. Por ejemplo, el directorio dev contiene información sobre todos los componentes (*devices*) de hardware contenidos en el sistema. El directorio correspondiente a la raíz del sistema de archivos es conocido como el directorio raíz y se denota por el símblo /. Linux define 7 diferentes tipos de archivos:

-Archivo regular
-Archivo Directorio
-Archivo Enlace Simbólico
-Archivo de dispositivo orientado a bloque
-Archivo de dispositivo orientado a carácter
-Archivo tipo FIFO (First Input First Output)
-Archivo Socket

Los archivos regulares, de directorio o de enlace simbólico son los constituyentes del sistema de archivos Linux. Los archivos de dispositivo están relacionados con los dispositivos de entrada y salida, y sus respectivos controladores, integrados al sistema. Por ejemplo, cuando un proceso quiere interactuar con un componente de hardware, lo hace a través del archivo de dispositivo asociado. Los archivos tipo FIFO y socket son archivos especiales utilizados para comunicación entre procesos.

El potencial de uso que puede llegar a tener un archivo puede caer en una de tres categorías, las cuales son configuradas por banderas de escritura, lectura o ejecución que pueden ser modificadas en todo momento:

-El archivo sólo es usado por el usuario al cual pertenece.
-El archivo es usado por los usuarios miembros del grupo al cuál pertenece el usuario dueño del archivo.
-El archivo puede ser usado por todos los usuarios existentes en el sistema.

MODELADO DE SISTEMAS

Por sistema entendemos un conjunto de partes o elementos organizados y relacionados que interactúan entre sí para lograr un objetivo. Un sistema puede ser físico o concreto (un automóvil, un televisor, etc.) o puede ser abstracto/conceptual (software).

Sin importar el tipo de sistema o el objetivo para el cual fue diseñado, existen tres razones principales por lo que el resultado final puede fallar (HOLT ,2008):

-Complejidad: Muchas veces no es suficiente poder separar un sistema en las partes que forman el todo para entender la complejidad del mismo. Ver a las partes como entes separados nos llevarían a diseños simplistas que darán como resultado sistemas fallidos. La complejidad de un sistema radica en las relaciones producidas por la interacción de sus partes. Si bien es cierto que muchas veces no es posible dimensionar la complejidad que representará el diseño de un sistema en las etapas iniciales, explorar todas las relaciones existentes entre sus partes es una buena práctica para entender la complejidad del mismo.

-Falta de entendimiento: Problemas de entendimiento pueden estar presentes en cualquier etapa del desarrollo de un sistema, sin embargo, suele ser durante las etapas iniciales de diseño donde una falta de entendimiento puede ser desastrosa. Al definir los requerimientos del sistema, una definición ambigua o poco concisa generará problemas a lo largo de todo el proceso de desarrollo que normalmente son difíciles de corregir en etapas posteriores.

-Problemas de comunicación: Debido a que el diseño de un sistema puede involucrar diferentes áreas de conocimiento, es común encontrar equipos multidisciplinarios encargados del diseño del mismo. Por lo tanto, uno de los principales retos a vencer es encontrar un lenguaje común que pueda ser entendido por todos los miembros de equipo, lo cuál permitirá reducir al máximo los posibles problemas de comunicación.

Una manera de evitar estos problemas es a través del uso de modelado. Por modelo entendemos una simplificación de la realidad. En el caso específico del diseño de sistemas, el uso de modelos que simplifiquen la realidad representada por el sistema es un punto de inicio para identificar la complejidad del mismo.

Para poder modelar un sistema de manera eficiente, es necesario tener un lenguaje común que pueda cubrir las necesidades del modelado. Existen diferentes acercamientos al proceso de modelado haciendo unos de herramientas gráficas, matemáticas o textuales. Sin embargo, sin importar el acercamiento que se use, cualquier lenguaje de modelado de sistemas debe cumplir con los siguientes requerimientos:

-Flexibilidad: Al buscar la solución de un problema, es común el uso de diferentes perspectivas que permitan entender la problemática desde diferentes ángulos. Por lo tanto todo lenguaje de modelado deberá ser lo suficientemente flexible como para permitir diferentes representaciones de la misma información de forma que podamos encontrar la solución más adecuada al problema.

-Nivel de abstracción: Un sistema puede ser visto desde diferentes niveles de abstracción. Considerando el ejemplo del diseño de un automóvil, definir los requerimientos del vehículo como una unidad, desde una perspectiva externa, puede ser considerado como un alto nivel de abstracción. Por otro lado, entender el proceso de combustión interna involucrado en el funcionamiento del motor del vehículo es un bajo nivel de abstracción.

Por lo tanto, es mandatario que toda herramienta de modelado tenga la capacidad de representar los diferentes niveles de abstracción presentes en un sistema.
-Conexión con la realidad: Como ya definimos, un modelo es una simplificación de la realidad. Sin embargo, siempre se corre el riesgo de simplificar de más el modelo. Lo que da como resultado que el modelo pierda toda conexión con la realidad que representa y por lo tanto pierda sentido. Es responsabilidad del lenguaje de modelado brindar herramientas que permitan mantener la conexión entre el modelo y la realidad representada.
-Niveles de información: En un equipo de trabajo encargado del desarrollo de un sistema siempre habrá personas con diferentes roles/actividades a realizar. De la misma forma, la información que cada persona necesite siempre será diferente. En el caso de un desarrollo de software, la información que necesite el encargado de la interacción con una base de datos será diferente a la información necesaria por el usuario encargado de las interfases gráficas. Sin embargo, a pesar de ser diferentes puntos de vista, el lenguaje de modelado debe brindar las herramientas necesarias para que la información sea consistente sin importar el punto de vista en el que se presente. Sólo logrando esa consistencia en la información se logrará un entendimiento del sistema a todos los niveles.

Todo modelo de un sistema que quiera reflejar la realidad de manera fidedigna debe considerar dos aspectos fundamentales: la estructura y el comportamiento que tendrán dicho sistema (WEILKIENS, 2007).

El modelado estructural de un sistema se basa en identificar los diferentes bloques (entidad física o lógica que forma parte del sistema, llamados clases en el caso específico de sistemas de software) y las relaciones que llegarán a existir entre dichos bloques. En una primera instancia, tanto los bloques como las relaciones existentes entre ellos pueden definirse de manera muy general, pero a medida que las relaciones se van detallando podrán definirse las propiedades, o atributos, que los bloques deberán tener así como las operaciones que cada uno de ellos deberá realizar para cumplir con las relaciones que tenga con otros bloques. Otro aspecto a resaltar del modelado estructural es poder definir dependencias entre bloques, es decir bloques que comparten algún tipo de propiedad u operación. Esto evitará repetir en el diseño bloques que realicen la misma función y por lo tanto desaparecer relaciones innecesarios, dando como resultado un diseño estructural más eficiente.

A diferencia del modelado estructural, encargado de definir los objetos existentes en el sistema y las relaciones que deberán existir entre ellos, el modelado de comportamiento se encarga de explicar cómo es que dichos bloques cumplirán con las relaciones establecidas. Este comportamiento deberá abarcar diferentes niveles de abstracción, entre los que destacan las interacciones que habrá entre los diferentes bloques como resultado de las relaciones entre ellos, las interacciones entre los bloques dependientes entre sí y las interacciones internas que tendrán las operaciones de cada bloque.

El modelado de comportamiento tiene mucho que ver con el análisis de la complejidad del sistema. A primera instancia, las operaciones que un bloque debe realizar para cumplir con las relaciones demandadas pueden resultar muy complejas y abrumadoras. Sin embargo, al realizar un análisis de comportamiento es posible dividir dichas operaciones en un conjunto de acciones que tendrán misiones (o actividades) muy específicas a realizar. Como resultado, la secuencia ordenada de ejecución de dichas

acciones hará más fácil entender, y por lo tanto ejecutar, la complejidad demandada por las operaciones de cada bloque del sistema.

PROCESOS DE DESARROLLO DE SOFTWARE

Tal y como se mencionará en el Capítulo 2, el Lenguaje de Modelado Unificado (UML) nació de un conjunto de de métodos de análisis y diseño de software, especialmente orientado a objetos. Si bien es cierto que UML fue rápidamente adoptado como un lenguaje de modelado por casi toda la industria del software, son varios los procesos existentes para el desarrollo de software y en todos ellos. A continuación haremos una breve descripción de los procesos más utilizados.

Proceso de cascada (Waterfall)

El proceso en cascada divide un proyecto en etapas dependiendo del tipo de actividad que se va a realizar en cada una de ellas. Por ejemplo, un proyecto de 1 año de duración podría dividirse en 2 meses para la etapa de análisis, 4 meses para la etapa de diseño, 3 meses para la etapa de generación de código y 3 meses para la etapa de pruebas. Es común que exista algún tipo de interacción entre las etapas del proceso. Inclusive, puede darse el caso de tener que regresar a una etapa previa desde etapas posteriores. Tal es el caso de tener que retomar cuestiones de diseño y/o análisis una vez adentrado en la etapa de generación de código. Si bien es cierto que es común tener que revisar cuestiones de análisis o diseño como consecuencia de la generación del código, debe de evitarse al máximo debido a que es señal de un mal entendimiento del problema a resolver.

El proceso de diseño de software en cascada ha sido duramente criticado en los últimos años, sobretodo por la comunidad de diseño orientado a objetos, principalmente por su dificultad de definir si un proyecto va por buen camino siguiendo este proceso por etapas clasificadas por el tipo de actividad (FOWLER,2004). Normalmente, la única forma de poder saber en realidad si el desarrollo del software va por buen camino es a través de pruebas constantes del sistema, cosa que no puede hacerse en el proceso de cascada hasta la etapa final.

Proceso Iterativo

El proceso iterativo divide un proyecto en pequeños grupos definidos por la funcionalidad de cada uno. Por ejemplo, un proyecto que tenga una duración de 1 año, puede dividirse en 4 procesos iterativos de 3 meses cada uno. En la primera iteración, se toma la cuarta parte de los requerimientos y se hace un ciclo completo de desarrollo de software (análisis, diseño, código y pruebas) para cumplir con dichos requerimientos. Al finalizar el ciclo, se tiene un sistema que cumple con la cuarta parte de lo necesitado. Al pasar a la siguiente iteración, se repite otro ciclo completo de desarrollo para un nuevo conjunto de requerimientos. Una vez concluidas todas las iteraciones, obtenemos un sistema completo que cumple con el 100% de los requerimientos necesarios.

Aunque cada iteración debe de producir como resultado software listo para ser liberado a nivel producción, normalmente no es así por lo que es necesario considerar una etapa de estabilización posterior a la finalización de la iteración de forma que puedan corregirse todos los errores que puedan llegar a aparecer. Otro tipo de actividades como lo es la generación de manuales o entrenamiento de los usuarios que harán uso del código

generado por cada iteración también suelen realizarse en etapas posteriores, inclusive al final de realizar todos los procesos iterativos.

A diferencia del proceso en cascada, el proceso iterativo nos permite implementar mejores controles de calidad. Al tener software funcional como resultado de cada iteración, es posible tener pruebas controladas del sistema lo que siempre dará como resultado retroalimentación del funcionamiento del mismo y por lo tanto, permitirá una mayor calidad del producto final. Inclusive, es común que, como resultado de dicha retroalimentación, cada iteración genere más de una versión (*release*) de software.

Proceso Racional Unificado

El Proceso Racional Unificado (*Rational Unified Process RUP*) es, más que un proceso de desarrollo, un marco de referencia debido a que provee tanto un vocabulario como una estructura para describir procesos. En parte, es este el motivo por el que RUP, puede llegar a confundirse con UML a pesar de ser cosas completamente independientes.

Al hacer uso de RUP lo primero que se necesita definir es un caso de desarrollo, o proceso que se utilizará en el proyecto. Los casos de desarrollo pueden variar dependiendo del tipo de desarrollo a realizar. Sin embargo, RUP es esencialmente un proceso iterativo, aunque no es raro encontrar procesos de cascadas inmersos dentro del vocabulario definido por RUP.

Todo proyecto RUP debe cumplir con las siguientes 4 fases:

1.- Comienzo (*Inception*): Realiza una evaluación inicial del proyecto. Tiene por objetivo principal definir los recursos necesarios durante la fase de elaboración.

2.- Elaboración (*Elaboration*): Identifica los casos de usos primarios del desarrollo y construye software por medio de iteraciones para obtener la arquitectura básica del sistema. Al final de la fase de elaboración se deberá tener una idea clara no sólo de los requerimientos, sino un esqueleto del sistema que servirá como punto de partida para la fase de construcción. De manera primordial, durante esta fase deberán identificarse los retos principales a vencer para construir el sistema solicitado.

3.- Construcción (*Construction*): Se trata básicamente de la fase en la que se realizarán todas las actividades (codificación) necesaria para la liberación óptima del sistema.

4.-Transición (*Transition*): incluye todas aquellas actividades que no forzosamente serán realizadas dentro de un proceso iterativo, como lo es el entrenamiento de usuarios, instalación del sistema, documentación, etc.

TECNOLOGÍA BLUETOOTH

ANTECEDENTES

Bluetooth es una tecnología inalámbrica de corto alcance y bajo consumo para conectar dispositivos electrónicos de uso personal y portables. Su bajo costo y razonable tasa de transferencia hace a la tecnología sumamente atractiva como reemplazo de cables en conexiones entre dispositivos de corto alcance. Fue desarrollada para ser utilizada de manera global en la frecuencia de 2.4 GHz del espectro de radio frecuencia.

Bluetooth nace en 1998 por iniciativa de Ericsson, IBM, Intel, Nokia Toshiba. Debe su nombre al rey danés Harald Blatand "Bluetooth", a quien se le da crédito de haber unificado a los pueblos de Escandinavia durante el siglo X (BAKKER,2002).

Son 7 las motivaciones principales tras la definición de la tecnología Bluetooth:

1. Dimensiones mínimas de manera que la tecnología pueda ser suportada por dispositivos electrónicos de uso personal como son teléfonos celulares.
2. Robustez y baja complejidad de diseño.
3. Bajo costo.
4. Bajo consumo de manera que pueda extenderse el ciclo de vida de las baterías de los dispositivos.
5. Interoperabilidad entre diferentes fabricantes y regiones.
6. Soporte para aplicaciones de diferentes tipos.
7. Reglas claras de conectividad de manera que los enlaces entre dispositivos puedan realizarse de manera rápida sin la intervención del usuario.

La primera especificación del estándar Bluetooth (1.0) fue liberada en Julio de 1999 por el Grupo de Interés Especial (SIG por sus siglas en inglés) de Bluetooth. En la actualidad, se espera que para mediados del 2010 sea liberada la versión 4.0 del estándar, cuyo mayor contribución será un mayor ahorro de energía y una considerable reducción de tamaño. La siguiente tabla muestra la evolución del estándar Bluetooth desde su aparición a la fecha.

Versión	Fecha Liberación	Principales Características
1.0	1999	Versión Inicial
1.2	2000	Velocidad transmisión de 1 Mbps
2.0	2004	EDR (*Enhanced Data Rate*) velocidad transmisión de 3Mbps
3.0	2009	HS (*High Speed*) velocidad teórica de transmisión de hasta 24 Mbps al mezclarse con un enlace 802.11

En términos de potencia de transmisión, los dispositivos bluetooth son categorizados tal como lo especifica la siguiente tabla.

Clase	Potencia Máxima Permitida		Rango (aproximado)
	mW	dBm	
1	100	20	100 metros
2	2.5	4	10 metros
3	1	0	1 metro

DESCRIPCIÓN DE LA TECNOLOGÍA

Bluetooth es una tecnología inalámbrica operando en la banda reservada para aplicaciones industriales, científicas y médicas (ISM por sus siglas en inglés) de 2.4-2.485 GHz del espectro de radiofrecuencia capaz de proveer conectividad inalámbrica de dispositivos en un rango de hasta 100 metros, aunque típicamente esta distancia no suele exceder los 10 metros. Ejemplos típicos de aplicaciones con conectividad Bluetooth se muestran en la siguiente figura 1.1.

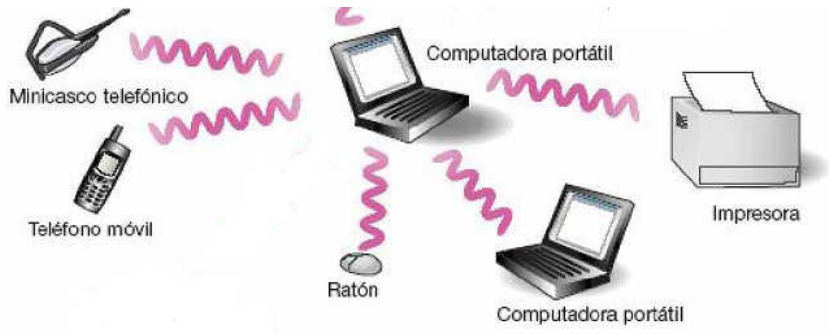


Figura 1.1 Ejemplos aplicaciones Bluetooth

Bluetooth ofrece dos tipos de enlaces. Un enlace síncrono orientado a la conexión (*Syncrhonous connection oriented* SCO) es usado principalmente por aplicaciones de audio en donde se establece una comunicación punto a punto (conmutación de circuitos) entre dos dispositivos a una tasa de transmisión de 64 kbps en modo full duplex. Un enlace asíncrono no orientado a la conexión (*Asyncrhonous Connectionless* ACL) es un esquema de conmutación de paquetes, utilizado principalmente para la transmisión de datos entre dispositivos.

Un sistema Bluetooth consiste principalmente de tres partes: una unidad de radio, una unidad de control del enlace y una unidad que cumple con la doble función de soporte para la gestión del enlace y una interfase con el cliente que hará uso del sistema Bluetooth.

Las diferentes partes de un sistema Bluetooth son realizadas por las diferentes capas que forman la pila (*stack*) del protocolo Bluetooth, mismo que se muestra en la figura 1.2 y se detallará a continuación.

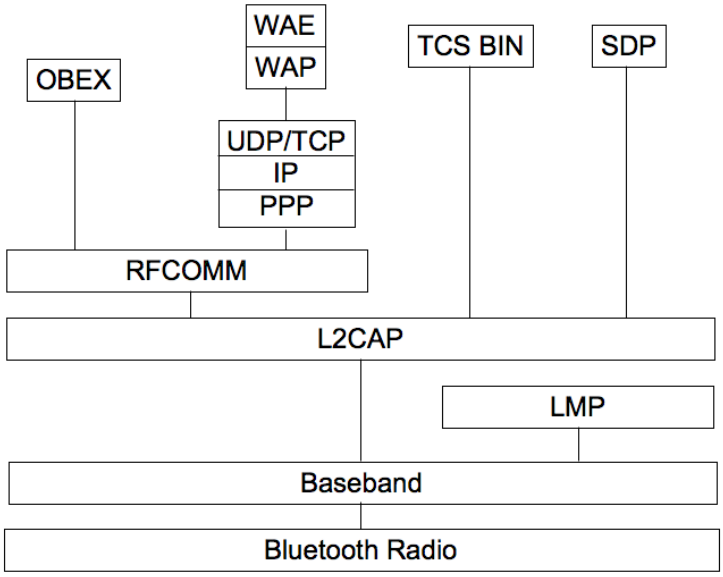


Figura 1.2 Stack Bluetooth

Capa Bluetooth Radio

Bluetooth, a nivel radiofrecuencia, usa Espectro Ensanchado por Salto de Frecuencia (*Frecuency Hopping Spread Spectrum*), full-duplex con una tasa de salto de 1600 saltos por segundo.

Para combatir los efectos de interferencia con otras tecnologías que utilicen la banda de 2.4 GHz, Bluetooth hace uso de la tecnología de salto de frecuencia adaptable (*Adaptive Frequency Hopping AFH*). En ella se hace un uso inteligente de las frecuencias disponibles dentro del espectro al detectar otros dispositivos que estén haciendo uso del espectro y evita utilizar las mismas frecuencias usadas por éstos. Estos saltos adaptables entre 79 frecuencias de 1 MHz (empezando en 2402 GHz, terminando en 2.480 GHz), proveen un alto grado de inmunidad a la interferencia así como un uso eficiente del espectro.

De manera que pueda cumplirse con el requerimiento de minimizar la complejidad del transmisor al máximo, la modulación utilizada es por desplazamiento de frecuencia Gausiana (GFSK), con excepción del modo EDR que hace uso de modulación PSK.

Para poder lograr la transmisión full duplex, la banda de frecuencias es dividida mediante un esquema de multiplexaje por tiempo (*Time Division Multiplexing TDM*). Cada uno de los 79 canales resultantes, numerados de 0 a 78, tiene una longitud de 625 μs. Como resultado, la información es intercambiada a través de paquetes que son transmitidos en diferentes canales y por lo tanto diferentes frecuencias. La sincronización entre los saltos de frecuencia que darán los dispositivos se logra través del uso de un reloj común y un patrón de salto entre las 79 frecuencias, generado por un sólo dispositivo llamado maestro. Todos los demás dispositivos son llamados esclavos. Un maestro puede estar sincronizado con un máximo de 7 esclavos de manera simultánea.

Banda Base (Baseband)

Cada dispositivo Bluetooth tiene dos parámetros que prácticamente están relacionados en todos los aspectos de la comunicación. El primero es una dirección única de 48 bits del tipo IEEE 802 que se le asigna a cada dispositivo a la hora de fabricación, muy parecido a las direcciones MAC que tiene los dispositivos Ethernet. El segundo parámetro es un reloj de 28 bits que genera un pulso cada 312.5 µs, tiempo que corresponde a la mitad del tiempo que el dispositivo permanecerá en cada una de las 79 frecuencias disponibles a un tasa nominal de 1600 saltos por segundo.

Un canal físico es definido por una secuencia pseudo aleatoria de saltos entre canales de RF, un código de acceso y el tiempo de permanencia en cada canal (*slot*).

Dos o más dispositivos compartiendo un mismo canal físico forman una red de tipo Piconet, En ella, un dispositivo forzosamente fungirá como maestro y el resto como esclavos. Como se mencionó con anterioridad, un máximo de 7 esclavos pueden participar de manera activa en la Piconet, aunque un número mayor de dispositivos pueden estar presentes de manera inactiva (estacionaria). Por otro lado, un conjunto de redes piconet forman una red scatternet.La figura 1.3 muestra diferentes configuraciones de redes Piconet (A,B) así como un ejemplo de una red scatternet (C).

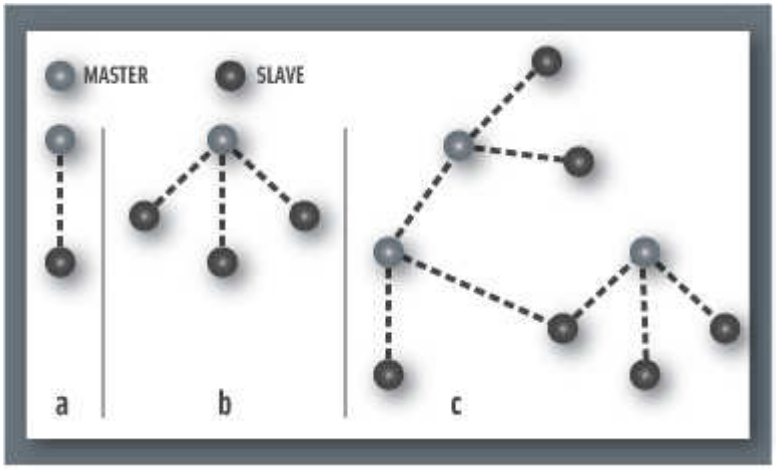


Figura 1.3 Redes Piconet y Scatternet (www.bluetooth.org)

El canal físico de una Piconet es definido por el maestro, el cual se encarga de controlar el tráfico en la Piconet a través de un esquema de poleo. Por definición, el dispositivo que solicita el establecimiento de la conexión (*paging*) es el maestro, pero una vez establecida la red piconet, los roles maestro/esclavo pueden intercambiarse.

La información es transmitida a través del aire por medio de paquetes, Para la tasa básica de transmisión de 1 Mbps el formato del paquete consiste de un código de acceso (*Access Code*), un encabezado (*Header*) y datos (*Payload*) tal como lo muestra la figura 1.4:

Figura 1.4 Encabezado Modo Básico Bluetooth (IDEM)

Para el modo EDR (3 Mbps), no sólo se hace uso de dos tipos de modulaciones para lograr el incremente en la tasa de transmisión, sino que el formato de paquete también sufre modificaciones tal cual lo muestra la figura 1.5:

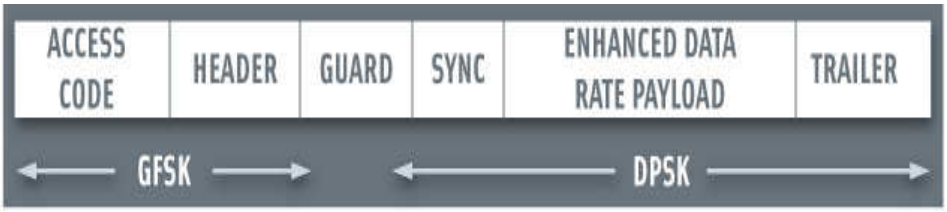


Figura 1.5 Encabezado Modo EDR Bluetooth (IDEM)

Todas las transmisiones sobre el canal físico inician con un código de acceso de 72 bits, el cual puede ser uno de tres tipos:
-Código de Acceso de Dispositvo (*Device Access Code,DAC*): utilizado para el envío de solicitudes de conexión (*Paging Request*) así como las respuestas a las mismas.
-Código de Acceso de Canal (*Channel Access Code,CAC*): utilizado para distinguir a la red piconet.
-Código de Acceso de Consulta (*Inquiry Access Code, IAC*): utilizado para descubrir dispositivos Bluetooth dentro de rango.

El encabezado del paquete de 54 bits contiene información de gestión del enlace con seis campos que incluyen la dirección del dispositivo, tipo de código, control de flujo, indicador de confirmación de recepción, número de secuencia y encabezado de revisión de errores. El campo tipo de código es utilizado para determinar si el paquete pertenece a un enlace de tipo SCO o a uno de tipo ACL.

Por último los datos del paquete es un campo de longitud variable que puede oscilar entre 0 y 2754 bits.

A nivel lógico, existen 5 canales dentro de un sistema Bluetooth:
- LC: canal de control utilizado a nivel control del enlace.
-LM: canal de control utilizado a nivel gestión del enlace.
-UA: transporte información de usuario para enlaces asíncronos.
-UI: transporta información de usuario para enlaces isócronos.
-US: transporta información de usuario para enlaces síncronos.

Con excepción de los canales LC que son contenidos en los encabezados del paquete, los demás tipos de canales son indicados en el encabezado del paquete y contenidos dentro de los datos del mismo. Un canal de tipo US existe en un enlace de tipo SCO mientras que los canales UA y UI existen dentro de los enlaces de tipo ACL.

La figura 1.6 muestra el diagrama de estados para el control del enlace de una conexión Bluetooth:

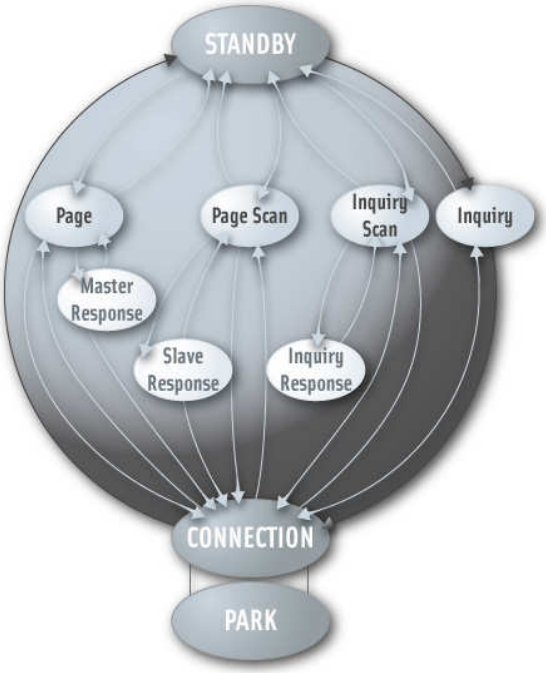


Figura 1.6 Diagrama estados Enlace Bluetooth (IDEM)

Existen dos estados principales, EN ESPERA (*STANDBY*) y CONEXIÓN (*CONNECTION*). Los demás estados son subestados que definen el estado del enlace cada que un esclavo nuevo es añadido a la piconet. Las transiciones entre los estados son disparadas ya sea por comandos de gestión del enlace o debido a señales de eventos internos en los dispositivos.

El estado STANDBY es el estado default de bajo consumo de energía del dispositivo. Es desde éste estado que el dispositivo puede ya sea responder a una solicitud de conexión de otro dispositivo o enviar dicha solicitud en caso de así requerirla y así transitar al estado CONNECTION si el enlace es establecido satisfactoriamente.

Protocolo de Gestión del Enlace (Link Manager Protocol LMP)

LMP es usado para controlar y negociar todos los aspectos de la operación de una conexión Bluetooth entre dos dispositivos. Esto incluye la negociación de la misma y control de los transportes y enlaces lógicos, así como control del enlace físico. LMP es usado para las comunicaciones entre las unidades gestoras del enlace (Link Managers LM) de los dispositivos conectados a través del transporte lógico ACL. LMP también es el encargado del control y negociación del tamaño de los paquetes cuando se transmiten datos así como del control y administración de potencia, direccionamiento y estado de los enlaces en una Piconet. Opciones de seguridad, incluyendo el intercambio de llaves de encripción entre dispositivos también son manejadas por LMP. Debido a su importancia para el control del enlace, los mensajes LMP tienen una prioridad mayor que los datos del usuario.

LMP opera en términos de transacciones, donde por transacción entendemos el intercambio de un conjunto de mensajes cuyo objetivo es cumplir con un propósito específico.

Protocolo de adaptación y Control del Enlace Lógico,(Logical Link Control and Adaptation Protocol , L2CAP)

L2CAP es uno de los dos protocolos de la capa de enlace por encima por el protocolo de banda base. Su función principal es proveer servicios, tanto orientados a la conexión como no orientados a la conexión, a los protocolos de capas superiores. Dichos servicios incluyen capacidades de multiplexaje de protocolos, segmentación y reensamble de paquetes así como información referente a la calidad del servicio.

L2CAP permite a los protocolos de capa superior el intercambio de información a través de paquetes Unidades de Datos de Servicios (Service Data Units SDU) que pueden tener una longitud de hasta 64 kilobytes de longitud.

La operación de L2CAP está basada alrededor del concepto de canales, donde cada extremo del canal es referido por un identificador de canal (CID), como se muestra en la figura 1.7:

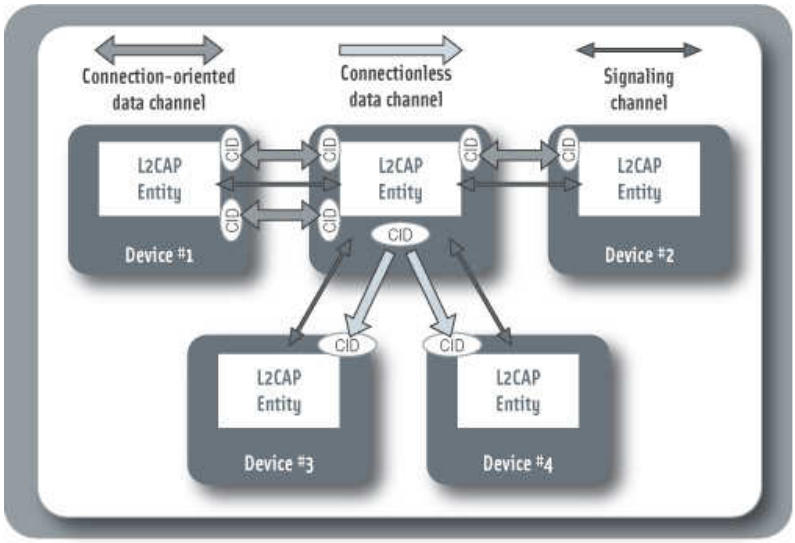


Figura 1.7 Operación L2CAP (IDEM)

Si bien es cierto que la asignación de los CID a utilizar es particular para cada dispositivo en específico, existen ciertos CIDs reservados para propósitos especiales, como lo es el del canal de señalización necesario para negociar y establecer un enlace orientado a la conexión.

En un canal orientado a la conexión, el flujo de datos es bidireccional entre los dispositivos, mientras que un canal no orientado a la conexión, el flujo forzosamente se hará en un solo sentido.

Protocolo para Descubrir Servicios (Service Discovery Protocol SDP)

SDP es un protocolo de tipo cliente/servidor que permite a las aplicaciones clientes descubrir que servidores de servicios se encuentran dentro del rango de la comunicación Bluetooth, así como las características de dichos servicios. Cada dispositivo Bluetooth deberá contar con al menos un servidor SDP para manejar la información de los servicios que ofreces así como todas las peticiones de clientes remotos.

Una entidad SDP puede actuar como servidor o cliente según sea el caso. Las características o los atributos de cada uno de los servicios disponibles en el dispositivo son almacenados en un registro del servicio que contiene el nombre del servicio, la descripción, clase e identificador del servicio y la información de protocolo necesario para hacer uso de ese servicio. Los registros de servicios son almacenados en base a una jerarquía estandarizada que permite su fácil consulta.

La disponibilidad de los servicios depende en gran medida de la proximidad RF existente a los dispositivos que ofrecen dichos servicios. Para dispositivos clientes en busca de servicios específicos, SDP provee funciones de búsqueda y consulta basadas en características comunes entre los servicios.

RFCOMM

RFCOMM es un simple protocolo de transporte con el que puede emularse un puerto de comunicaciones capaz de soportar comunicación serial del tipo RS232 montado sobre el protocolo L2CAP. Es un protocolo basado principalmente en el estándar ETSI TS 07.10, pero el grupo Bluetooth SIG especificó algunas adaptaciones y extensiones para hacerlo compatible con la tecnología Bluetooth., principalmente en cuestiones de acceso al medio y control de flujo.

RFCOMM soporta hasta 60 conexiones simultáneas entre dos dispositivos Bluetooth. Por lo tanto el protocolo RFCOMM es de suma utilidad para todos aquellas protocolos de nivel aplicación que hacen uso de canales seriales, como lo es OBEX.

PERFILES DE BLUETOOTH

El propósito de los perfiles de Bluetooth es describir las distintas aplicaciones existentes en la tecnología así como la implementación de cada una de ellas. El modelo de uso del perfil describe los diferentes escenarios en los que la tecnología Bluetooth puede aplicarse para cumplir con la aplicación mencionada. Cada perfil debe describir el comportamiento necesario para cada una de los protocolos de capas inferiores del stack Bluetooth, así cómo los rangos de parámetros a utilizar por cada uno de ellos.

Los perfiles de Bluetooth son necesarios por cuestiones de interoperabilidad entre fabricantes de dispositivos. Ajustándose a los estándares establecidos por cada uno de los perfiles de las aplicaciones, los diferentes fabricantes pueden asegurar la interoperabilidad de sus dispositivos con los de otros fabricantes no sólo a nivel de radio sin a nivel aplicación.

Cada perfil incluye, como mínimo, información sobre las siguientes cuestiones:

-Dependencia de otros perfiles.

-Propuestas de formato de interfaz de usuario.

-Características concretas de la pila de protocolos Bluetooth utilizada por el perfil. Para realizar su función, cada perfil se sirve de ciertas opciones y parámetros en cada capa de la pila. También se puede incluir un breve resumen de los servicios requeridos si resulta necesario.

La siguiente tabla muestra los perfiles existentes, y debido a que es del interés de esta tesis el perfil genérico de intercambio de objetos (GOEP) será el único que se detallará a continuación.

Perfil de distribución de audio avanzado (A2DP)	El perfil A2DP describe cómo transferir sonido estéreo de alta calidad de una fuente de sonido a un dispositivo receptor.
Protocolo de control de audio y vídeo (AVRCP)	El perfil AVRCP proporciona una interfaz estándar para manejar televisiones, equipos de alta fidelidad o cualquier otro equipo electrónico, y permitir así que un único control remoto, o cualquier otro tipo de mando, controle todo el equipo de audio y vídeo al que el usuario tiene acceso.
Perfil básico de imagen (BIP)	El perfil BIP establece cómo puede controlarse remotamente un dispositivo de imagen, así como la forma de enviarle órdenes de impresión y de transferencia de imágenes a un dispositivo de almacenamiento.
Perfil básico de impresión (BPP)	El perfil BPP permite enviar mensajes de texto, de correo electrónico, tarjetas de visita electrónicas e imágenes, entre otras cosas, a las impresoras disponibles dependiendo de las tareas de impresión.
Perfil de acceso RDSI común (CIP)	El perfil CIP establece cómo se deben transferir las señales RDSI a través de una conexión inalámbrica*Bluetooth*.
Perfil de telefonía inalámbrica (CTP)	El perfil CTP describe la implementación de un teléfono inalámbrico a través de un enlace inalámbrico*Bluetooth*.
Perfil de red de marcado (DUN)	El perfil DUN proporciona un acceso telefónico estándar a Internet y a otros servicios de marcado a través de una

	conexión *Bluetooth*.
Perfil de fax (FAX)	El perfil FAX describe cómo un dispositivo terminal puede utilizar a otro como puerta de enlace para la transmisión de faxes.
Perfil de transferencia de archivos (FTP)	El perfil FTP establece los procedimientos de exploración de carpetas y archivos de un servidor a través de un dispositivo cliente.
Perfil de distribución genérica de audio y vídeo (GAVDP)	El perfil GAVDP sienta las bases de los perfiles A2DP y VDP, pilar de los sistemas diseñados para la transmisión de sonido e imagen mediante la tecnología *Bluetooth*.
Perfil genérico de intercambio de objetos (GOEP)	El GOEP se utiliza para transferir objetos de un dispositivo a otro.
Perfil manos libres (HFP)	El perfil HFP describe cómo un dispositivo que actúa como puerta de enlace puede utilizarse para realizar y recibir llamadas a través de un dispositivo manos libres.
Perfil de sustitución de cable de copia impresa (HCRP)	El perfil HCRP describe cómo imprimir archivos mediante un enlace inalámbrico *Bluetooth* utilizando controladores en el proceso.
Perfil de auricular (HSP)	El HSP describe cómo un auricular con tecnología *Bluetooth* se comunica con otro dispositivo con tecnología*Bluetooth*.
Perfil de dispositivo de interfaz humana (HID)	El perfil HID recoge los protocolos, procedimientos y características empleados por las interfaces de usuario*Bluetooth* tales como teclados, dispositivos punteros, consolas o aparatos de control remoto.
Perfil de intercomunicador (ICP)	El perfil ICP establece cómo conectar dos teléfonos móviles con tecnología *Bluetooth* dentro la misma red sin utilizar la red telefónica pública.
Perfil de introducción de objetos (OPP)	Este perfil distingue entre servidor y cliente de introducción (push) de objetos.
Perfil de redes de área personal (PAN)	El perfil PAN describe cómo dos o más dispositivos con tecnología *Bluetooth* pueden formar una red ad hoc y cómo ese mismo mecanismo permite acceder a la red de forma remota a través de un punto de acceso.
Perfil de aplicación de descubrimiento de servicio (SDAP)	El perfil SDAP detalla cómo una aplicación debe utilizar el perfil SDP para identificar los servicios de un dispositivo remoto.
Perfil de servicio de puerto (SPP)	El perfil SPP describe cómo configurar puertos de serie y conectar dos dispositivos con tecnología *Bluetooth*.
Perfil de sincronización (SYNC)	El perfil SYNC se utiliza junto al GOEP para sincronizar los elementos del administrador de información personal (PIM), como agendas y datos de contacto, entre dispositivos con tecnología *Bluetooth*.
Perfil de distribución de vídeo (VDP)	Este perfil dicta los pasos que deben seguir los dispositivos *Bluetooth* con tecnología *Bluetooth* para la transferencia de flujos de datos de vídeo.

Perfil Genérico de Intercambio de Objetos (Generic Object Exchange Profile GOEP)

Se usa para transferir cualquier tipo de objeto de un dispositivo a otro mediante un enlace Bluetooth. El escenario típico podría ser el envío de archivos de información como puede ser imágenes de un teléfono celular a una PC. GOEP define dos roles:
-Servidor: Dispositivo que contiene un servidor GOEP al cuál se le pueden enviar (*push*) o extraer (*pull*) objetos.
-Cliente: Dispositivo que puede enviar o extraer objetos de un servidor GOEP.

La figura 1.8 muestra los protocolos del stack Bluetooth utilizados por GOEP:

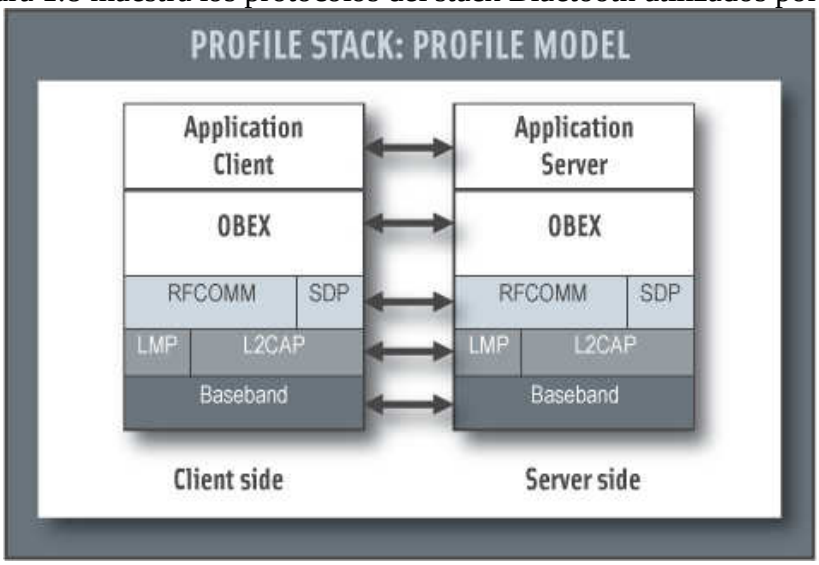


Figura 1.8 Stack Bluetooth para Perfil GOEP

Como puede verse en la figura, la capa de la aplicación es la entidad encargada de transmitir o recibir los objetos haciendo uso del protocolo OBEX.

OBEX (abreviatura de OBject EXchange, intercambio de objetos), es un protocolo de comunicaciones a nivel aplicación que facilita el intercambio de objetos entre dispositivos. Originalmente fue concebido para utilizarse sobre enlace infrarrojos por la Asociación de Datos Infrarrojos (*Infrared Data Association IrDA*), pero poco tiempo después fue adoptado por otras asociaciones como Bluetooth SIG o por la sección SyncML de la Alianza Móvil Abierta (*Open Mobile Alliance*).

OBEX es similar en cuanto a su diseño y funcionalidad al protocolo HTTP, protocolo que se basa principalmente en utilizar un transporte fiable para conectarse a un servidor Web y así poder recibir (*get*) o extrar (*post*) objetos del mismo. Sin embargo, existen algunos puntos en los que OBEX difiere de HTTP:

- Transporte: HTTP funciona sobre un puerto TCP/IP, mientras que en el caso específico de OBEX para enlaces Bluetooth, se utiliza sobre un stack Banda base/LMP/L2CAP/RFCOMM.
- Transmisiones binarias: HTTP utiliza un texto legible para el ser humano, mientras que OBEX utiliza encabezados binarios para intercambiar información sobre una petición o un objeto. Esto debido a la necesidad de implementar el protocolo en dispositivos con características limitadas.

- Soporte para realizar sesiones: Las transacciones HTTP carecen inherentemente de estado. Generalmente, un cliente HTTP establece una conexión, efectúa una sola petición, recibe respuesta y cierra la conexión. En OBEX, una sola conexión de transporte podría llegar a utilizarse para efectuar varias operaciones relacionadas entre sí.

GOEP es un perfil abstracto en el sentido de que existen otros perfiles que hacen uso de GOEP para intercambio de objetos. Tal es el caso de los perfiles para transferencia de archivos (*File Transfer*), sintonización (*Synchronization*) y envío de Objetos (*Object Push*).

Son tres las diferentes operaciones que GOEP ofrece a las aplicaciones de capa superior, todas a través de operaciones de OBEX.

-Establecimiento de una sesión de Intercambio de Objetos
-Envío (*Push*) de un objeto.
-Obtención Remota (*Pull*) de un objeto.

El establecimiento de una sesión entre el cliente y el servidor es necesario para poder hacer cualquier tipo de intercambio de objetos entre ellos. Dicho establecimiento puede darse con autentificación o sin autentificación.

La figura 1.9 muestra el diagrama de secuencia del establecimiento de una sesión OBEX sin autentificación por medio de la operación CONNECT.

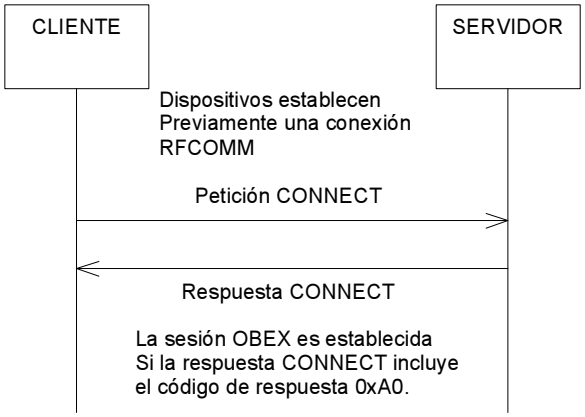


Figura 1.9 Establecimiento sesión OBEX sin autentificación

La figura 1.10 muestra el diagrama de secuencias del establecimiento de una sesión OBEX con autentificación, el cuál es una versión simplificada del esquema de autentificación utlizado por HTTP.

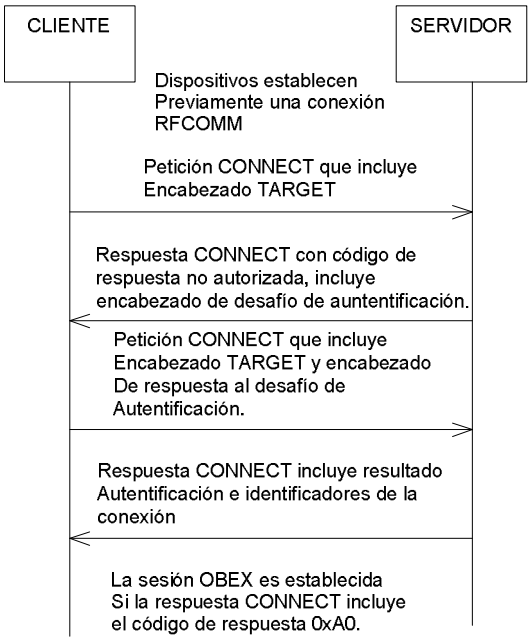


Figura 1.10 Establecimiento sesión OBEX con autentificación

Para enviar (*push*) un objeto a un servidor OBEX desde un dispositivo cliente se hace uso de la operación PUT del protocolo. Dependiendo del tamaño del objeto a enviar, una operación PUT puede ejecutarse en más de un paquete. Sin embargo, debe indicarse en el encabezado del paquete enviado si el paquete a datos por enviar es el último o si el servidor debe esperar por más paquetes. La figura 1.11 muestra el diagrama de secuencias de una operación PUT entre cliente y servidor OBEX.

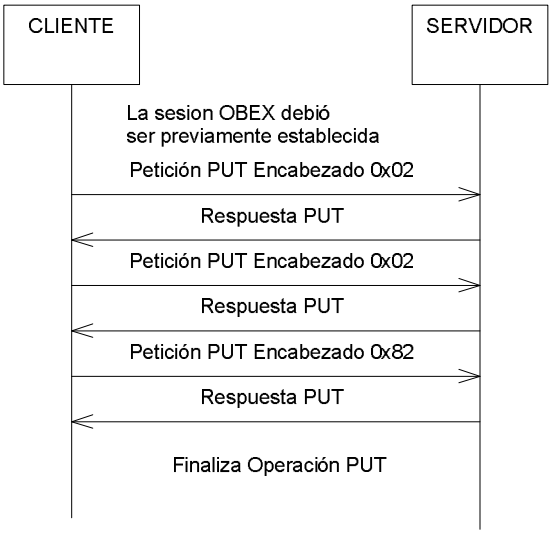


Figura 1.11 Operación PUT

Para obtener (*pull*) un objeto desde servidor OBEX desde un dispositivo cliente se hace uso de la operación GET del protocolo. Dependiendo del tamaño del objeto a obtener, una operación GET puede ejecutarse en más de un paquete. Sin embargo, debe

indicarse en el encabezado del paquete enviado si el paquete a datos por recibir es el último o si el cliente debe esperar por más paquetes. La figura 1.12 muestra el diagrama de secuencias de una operación GET entre cliente y servidor OBEX.

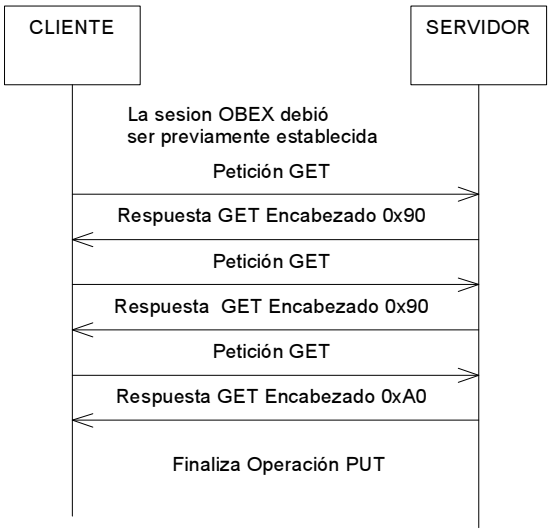


Figura 1.12 Operación GET

Para terminar una sesión OBEX el cliente hace uso de la operación DISCONNECT tal como se muestra en la figura 1.13:

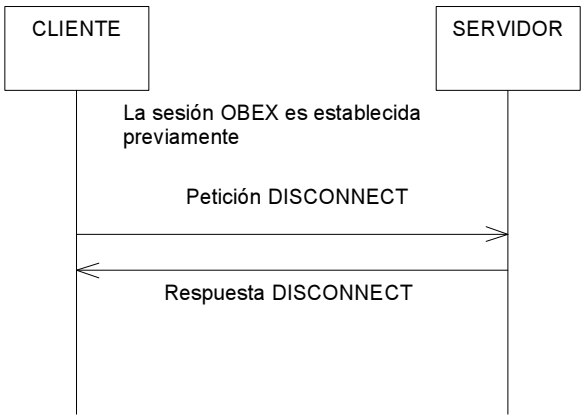


Figura 1.13 Operación DISCONNECT

Debido a que OBEX fue diseñado originalmente para el envío de objetos sobre enlaces infrarrojos, hay algunas consideraciones que deben tomarse en cuanta para hacer uso de OBEX haciendo uso de la capa RFCOMM del stack Bluetooth:

-El dispositivo que soporte OBEX debe de ser capaz de soportar las funcionalidades tanto de cliente como de servidor OBEX.

-Todos los servidores OBEX corriendo en el mismo dispositivo de manera simultánea deberán de utilizar puertos RFCOMM diferentes.

-Todas las aplicaciones (perfiles) que hagan uso de OBEX deberán de ser capaces de registrar su información en la base de datos de servicios disponibles en el dispositivo.

-Antes de poder establecer una sesión OBEX con cualquier otro dispositivo, es mandataria que la aplicación abra el canal RFCOMM que previamente haya registrado en

la base de datos de servicios disponibles. En el caso de un servidor OBEX, dicho canal deberá permanecer permanentemente abierto para poder recibir peticiones de establecimiento de sesión por parte de dispositivos remotos.

STACK LINUX PARA BLUETOOTH, BLUEZ

El stack para soportar enlaces Bluetooth sobre Linux fue originalmente desarrollado por Mark Krasnyansky en Qualcomm. En el año 2001 fue liberado por primera para uso público bajo el esquema de Licencia Pública General (Generic Public License, GPL).Para mediados del 2001, Linus Torvalds decide incluir a Bluez como parte del kernel de Linux lo que permitió que a partir de la liberación 2.4.6 del kernel , Linux tuviera su propio stack para gestionar enlaces Bluetooth (HUANG, 2007).

Para enero del 2004, el mantenimiento de Bluez fue encargado a Marcel Holtmann, poco tiempo después de la liberación del kernel 2.6 de Linux. A partir del 2005, Bluez obtuvo la certificación oficial para ser considerado un subsistema del Bluetooth SIG.

Cabe destacar que Bluez no ha sido el único stack bajo la liciencia GPL desarrollado para Bluetooth. OpenBT fue un stack desarrollado por la empresa Axis Communications, sin embargo su desarrollo fue detenido en 2005 para darle mayor fuerza aún a Bluez.

Bluez provee soporta para todas las capas y protocolos definidos en el stack de Bluetooth. Está desarrollado bajo una filosofía que intenta hacerlo flexible, eficiente y modular. Algunas de sus características principales son:

-Implementación completamente modular
-Procesamiento de datos hecho a través de un esquema multitareas.
-Soporte para un gran número de dispositivos Bluetooth
-Independencia total del hardware.
-Interfaz estándar para los puntos de acceso (*sockets*) existentes entre las diferentes capas.
-Esquemas de seguridad soportados tanto a nivel capas del stack como a nivel dispositivos.

Como ya se mencionó, Bluez está formado por una serie de módulos que operan de manera conjunta:

-Módulo central de interacción con el kernel de Linux.
-Módulo encargado de la capa L2CAP y enlaces de Audio SCO.
-Módulo encargado de la implementación de RFCOMM y de los perfiles BNEP, CMTP y HIDP.
-Módulo encargado del UART HCI y manejo de los dispositivos virtuales.
-Módulo encargado del manejo de la liberarías de SDP y de todos los procesos (*daemons*) de Bluetooth.
-Módulo encargado de la configuración y utilidades para pruebas.
-Módulo encargado para la decodificación del protocolo y herramientas de análisis del mismo.

Las librerías, módulos y utilerias del kernel de Bluez han sido probadas de manera satisfactoria en muchas arquitecturas que soportan Linux, ya sea en esquemas de un solo procesador o multiprocesadores. Algunas de las arquitecturas validadas son:

-Intel y AMD x86
-AMD64 and EM64T (x86-64)
-SUN SPARC 32/64bit
-PowerPC 32/64bit
-Intel StrongARM and XScale
-Hitachi/Renesas SH processors
-Motorola DragonBa

Actualmente existe soporte para Bluez en casi todas las distribuciones de Linux existentes y en general puede decirse que es compatible con todos los sistemas Linux disponibles en el mercado:

-Debian GNU/Linux
-Ubuntu Linux
-Fedora Core / Red Hat Linux
-OpenSuSE / SuSE Linux
-Mandrake Linux

Si bien no es parte del kernel de Bluez, para poder cumplir con el intercambio de objetos vía OBEX, Bluez es 100% compatible con OpenObex , proyecto de código abierto (*open source*) encargado de todo lo referente a gestión de sesiones OBEX.

LENGUAJES DE MODELADO UML/SYSML

UML

Definición

El lenguaje de Modelado Unificado (Unified Modeling Language UML) es un lenguaje y sistema de notaciones gráficas usado para especificar, construir, visualizar y documentar modelos de sistemas, principalmente de software.

Antecedentes

En el período de 1980-1990, un gran número de libros enfocados al desarrollo de software orientado a objetos habían sido publicados, y un gran número de notaciones gráficas fueron introducidas. Pronto surgió la necesidad de unificar todos esos criterios bajo un estándar común (FOWLER, 2004).

En 1995, Grady Booch y Jim Rumbaugh anunciaron la unificación de sus conceptos en lo que llamaron Método Unificado (*Unified Method*). Poco tiempo después su Método Unificado se convirtió en el Lenguaje de Modelado Unificado, término que claramente indica un lenguaje que ofrece una semántica y notificación gráfica uniforme. Sin embargo, UML no debe ser visto como una metodología o un acercamiento al desarrollo de software orientado a objetos. A Booch y Rumbaugh se les unió Ivan Jacobson y al grupo y trabajo que generaron se le conoció como "Three Amigos" (WEILKIENS,2006) en el mundo de ingeniería de software y tecnologías de la información. Debido a la gran cantidad de libros, y aceptación de los mismos, realizado por Booch, Rumbaugh y Jacobson, cuando UML apareció fue rápidamente recibido como un estándar de facto entre los profesionales del software. Eventualmente la versión 1.1 de UML fue enviada al Grupo de Gestión de Objetos (*Object Management Group, OMG*) para su estandarización y fue aceptado en 1997. A partir de ese momento, UML ha sido acogido y desarrollado bajo la supervisión del OMG. Tiempo después, la Organización Internacional para Estandarización (*International Organization for Standarization*) también aceptó a UML como un estándar.

El OMG es un organismo industrial internacional que agrupa a un gran número de compañías de alta relevancia en el campo de tecnologías de la información. Los miembros del OMG son los encargados no sólo de mantener el estándar UML sino también de aumentarlo. Algunos ejemplos de compañías que forman parte del OMG son IBM, Hewlett-Packard, Sun Mycrosystems, Telelogic y Boeing.

La figura 2.1 muestra el historial de versiones de UML desde sus inicios hasta la actualidad.

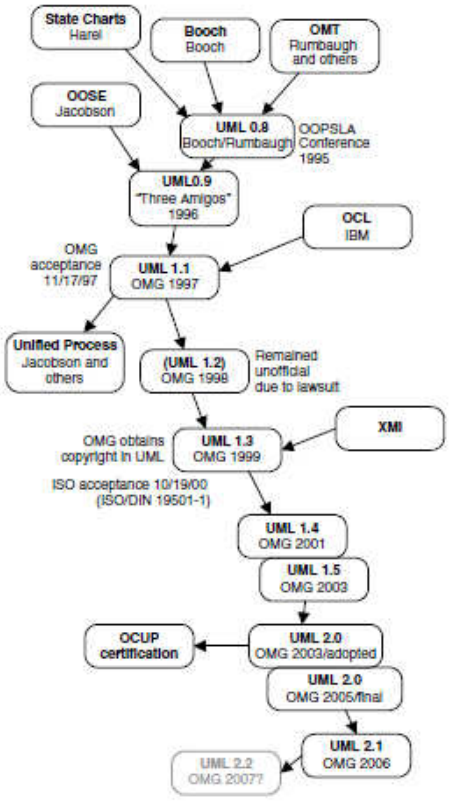


Figura 2.1 Historial de versiones de UML (WEILKIENS,2006)

Hablando específicamente del uso de UML para el desarrollo de software, existen varios modos en los que UML puede ser utilizado, siendo tres los principales: modelado explorativo (*sketching*), modelado intensivo (*blueprinting*) y lenguaje de programación.

El modelado explorativo se refiere al proceso en el que se tiene por objetivo principal definir las ideas o conceptos en torno a los cuales girará el desarrollo del sistema. No es necesario definir todo el código que será necesario generar, sino delimitar claramente los objetivos finales buscados. Bajo este esquema, las sesiones de discusión suelen ser informales y dinámicas, en donde 10 minutos pueden ser suficientes para modelar los trabajos de programación a seguir durante un día. Este tipo de modelado también puede ser útil para procesos de ingeniería inversa, en donde por medio de diagramas se explica el comportamiento básico de un sistema que ya se encuentra operando.

El modelado intensivo tiene por objetivo no sólo definir las ideas o conceptos básicos del sistema, sino detallar la ruta a seguir para lograrlo. Es decir, habrá un diseñador encargado de construir un modelo lo suficientemente detallado como para que un programador pueda codificarlo sin problemas. El modelo deberá incluir qué decisiones deben tomarse para los diferentes escenarios del sistema, lo que debería facilitar la labor del programador en gran manera. En este esquema, la persona encargada de diseñar el modelo basado en UML suele no pertenecer al grupo de programadores encargados de codificar el mismo. Inclusive, el modelo puede ser visto como algo completamente ajeno al lenguaje de programación que se utilizará para codificar el modelo. Sin embargo, siempre es buena práctica tener en cuenta el lenguaje a

programación a utilizar de forma tal que las decisiones y estructuras propuestas en el modelo intensivo se adaptan bien a dicho lenguaje de programación.

Por último, existe el modo en donde UML es visto como un lenguaje de programación, en donde por medio de herramientas sofisticadas los diagramas UML son compilados y código es generado a partir de ellos. Sin embargo, este modo de UML es el menos comúnmente utilizado.

Cabe mencionar que para en esta tesis haremos uso de técnicas de modelado explorativo e intensivo.

Notaciones y Metamodelo

El estándar UML define un sistema de notación específico y un Metamodelo.

El sistema de notación se refiere a todos los componentes gráficos que se ven en los diagramas, es la sintaxis gráfica del lenguaje de modelado. Por ejemplo, la notación de un diagrama de clases define cómo son representados gráficamente objetos y conceptos tales como clases, asociaciones, multiplicidad, etc.

Sin embargo, antes de definir cómo se van a representar conceptos como clases o asociaciones a través de una notación gráfica, es necesario poder definir de una manera formal lo que una clase o una asociación representa. Uno de los problemas existentes con los lenguajes de modelado gráficos previos a UML era la falta de rigor para definir los conceptos básicos a los cuales se les asociarán notaciones gráficas.

La respuesta de UML a éste problema es el Metamodelo, el cuál puede entenderse como un diagrama que tiene la función de definir todos los conceptos utilizados en el lenguaje. El prefijo meta se debe a que la definición del lenguaje reside en un nivel de abstracción por encima de cualquier modelo que un usuario pueda generar.

Una confusión común con el metamodelo es el hecho de que al final el metamodelo es un diagrama de clases UML utilizado para definir UML. En otras palabras, UML es definido dentro de UML. Sin embargo, esta confusión puede aclararse al pensar en UML como un lenguaje similar a un lenguaje humano como puede ser el inglés. Cuando buscamos la definición de una palabra en inglés, dicha definición estará en inglés. Es decir, el lenguaje hace uso de sí mismo para definirse, tal cual pasa con UML. Cada concepto utilizado por UML será definido como una clase dentro del metamodelo de UML. La figura 2.2 muestra una sección simplificada del metamodelo UML. En ella pueden verse tres elementos, clase, propiedad y operación. Al mismo tiempo, puede verse que una clase puede tener un número arbitrario de propiedades y un número arbitrario de operaciones.

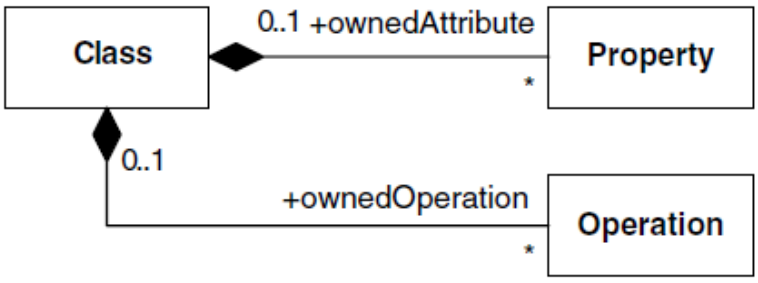


Figura 2.2 Sección Simplificada del metamodelo UML

La gran mayoría de las personas involucradas en el desarrollo constante de UML, invierten su tiempo en el análisis y ampliación del metamodelo, ya que a partir de éste será posible definir de manera puntual y formal, los conceptos asociados a las notaciones gráficas propuestas por el lenguaje.

Tipos de Datos

UML distingue los siguientes tipos de datos:

- Tipo de Datos Simple (*DataType*): Un tipo de dato simple es aquel que tiene valores sin identidad (dinero, información bancaria). Esto significa que dos instancias de un tipo de dato simple con el mismo valor para su atributo son indistinguibles.
- Tipo de Datos Primitivo (*PrimitiveType*): Un tipo de dato primitivo es un tipo de dato simple que no tiene estructura. Son los tipos de datos mínimos que se conocen antes de definir nuestros propios tipos de datos simples al modelar. UML define los siguientes tipos de datos primtivos:
    - Entero: Conjunto infinito de números enteros tanto negativos como positivos, incluye el cero. (…..,-2,-1,0,1,2,….)
    - Booleano: Verdadero, Falso
    - Natural Ilimitado: Conjunto infinito de números naturales (0,1,2,3….)
- Tipo de Dato Ennumerado (*EnumerationType*): Un tipo de dato ennumerado es un tipo de dato simple con valores que se originan de un conjunto limitado de valores. Un ejemplo de este tipo de dato podría ser el estatus civil de una persona.

La figura 2.3 muestra la sección del metamodelo UML referente a los tipos de datos y la figura 2.4 muestra ejemplos de tipos de datos.

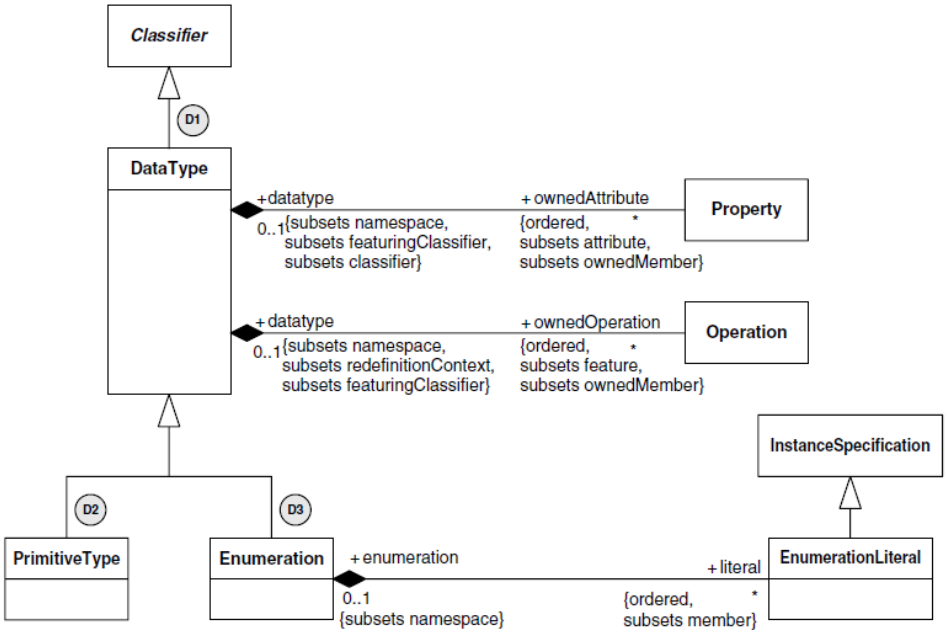


Figura 2.3 Sección Tipos de Datos metamodelo UML (IDEM)

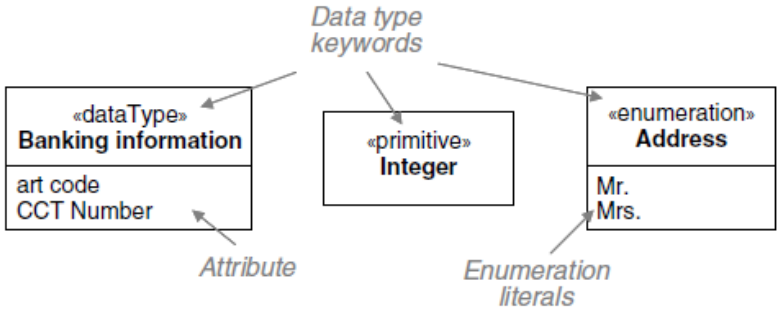


Figura 2.4 Ejemplos de Tipos de Datos

Estereotipos

Los estereotipos son extensiones formales de elementos existentes dentro del metamodelo UML. El elemento existente es directamente influenciado por la semántica definida por la extensión. Más que introducir un elemento nuevo al metamodelo, los estereotipos permite añadir semántica a un elemento existente, lo que permite que el modelo no contenga elementos que puedan considerarse repetidos bajo ciertas circunstancias. Visto de otra manera, los estereotipos clasifican los posibles usos de un elemento del metamodelo.

Los estereotipos se colocan antes o encima del nombre del elemento y se encierran mediante el uso de comillas anguladas (guillemetes). La siguiente tabla muestra algunos de los estereotipos existentes en la especificación UML, así como su descripción y el elemento UML relacionado. Sin embargo, cabe resaltar que a la hora de hacer un modelo UML, el usuario podría crear más estereotipos según sus necesidades de diseño.

Estereotipo	Elemento UML	Descripción
«create»	Dependency	El elemento origen crea instancias en el elemento destino
«send»	Dependency	El elemento origen es una operación y el elemento destino es una señal enviada por la operación
«derive»	Abstraction	El elemento origen puede derivarse del elemento destino mediante un cálculo
«metaClass»	Class	Una clase que contiene instancias que son clases a su vez
«utility»	Class	Las clases de utilidad son colecciones de variables globales y funciones que pueden ser usadas por todas las clases definidas en el modelo

DIAGRAMAS DE UML

La especificación UML 2 reconoce 13 diferentes tipos de diagramas. La siguiente tabla muestra una breve descripción de la funcionalidad de los diferentes diagramas así como a partir de que especificación de UML aparecieron.

Diagrama	**Funcionalidad**	**Aparación UML**
Actividad (*Activity*)	Representa el flujo del comportamiento de procesos del sistema	UML 1
Clases (*Class*)	Diagrama estático que muestra la estructura del sistema. Define sus clases, atributos y relaciones entre ellas.	UML 1
Comunicación (*Communication*)	Modela las interacciones entre objetos en términos de secuencias.	UML 1 antes llamado diagrama de colaboración
Componentes (*Component*)	Divide al sistema en un conjunto de componentes y las relaciones entre ellos	UML 1
Estructura Compuesta (*Composite Structure*)	Define la estructura interna de una clase	UML 2
Despliegue (*Deployment*)	Modela los nodos (hardware) necesario para la implementación del sistema	UML 1
Vista de Interacción (*Interaction Overview*)	Mezcla diagramas de secuencia y actividad	UML 2
Objetos (*Object*)	Muestra la configuración de objetos (clases) para una instancia dada de tiempo	UML 1
Paquetes (*Package*)	Muestra como un sistema se divide en agrupaciones lógicas y las relaciones entre ellas	UML 1
Secuencia (*Sequence*)	Muestra la interacción entre objetos haciendo énfasis en la secuencia de eventos.	UML 1
Estados (*State Machine*)	Muestra como los eventos afectan a los objetos a lo largo de su vida en el sistema	UML 1
Tiempos (*Timing*)	Muestra la interacción entre los objetos haciendo énfasis en el tiempo.	UML 2
Casos de Uso (*Use Case*)	Define la interacción de los usuarios con el sistema	UML 1

Los diagramas UML pueden catalogarse en tres tipos: diagramas de estructura, diagramas de comportamiento y diagramas de interacción. La jerarquía existente entre dichos tipos se muestra en la figura 2.5

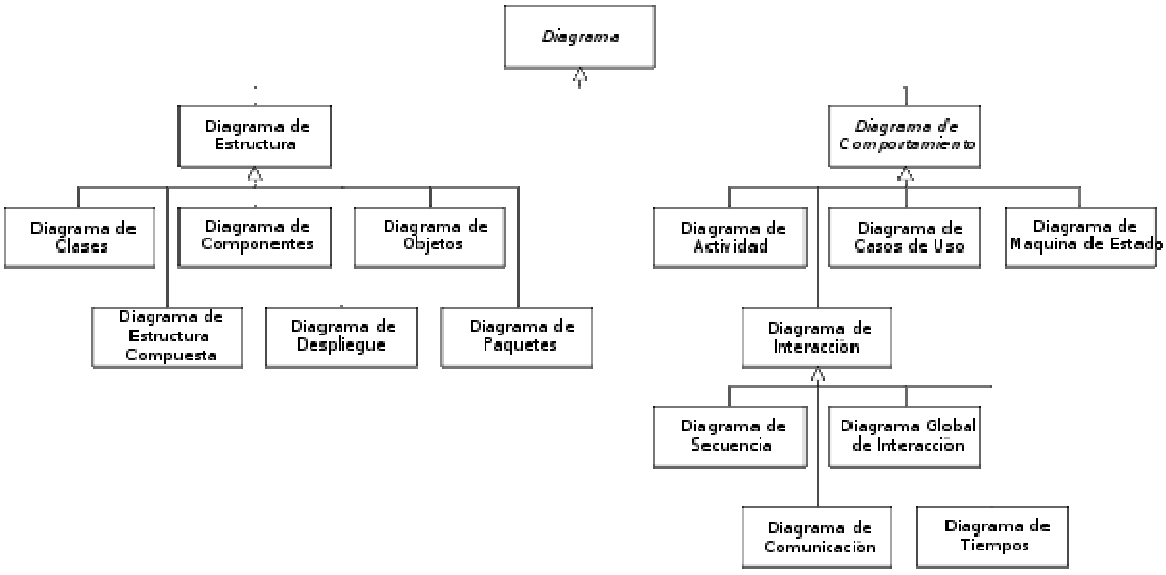


Figura 2.5 Jerarquía tipos de diagramas UML

La definición de los tipos de diagramas no es tan rígida como la definición del metamodelo en el sentido de que no se especifica qué elementos del metamodelo pertenecen a qué tipo de diagramas. Al contrario, es perfectamente normal y esperado el uso de elementos del metamodelo en más de un tipo de diagrama.

Todo diagrama UML deberá contar con un par de notaciones gráficas básicas. El diagrama consiste de un área delimitada dentro de un rectángulo y un encabezado en la esquina superior izquierda. Dicho encabezado debería incluir el tipo de diagrama (opcional), el nombre del diagrama (mandatario) y parámetros del mismo (opcionales). Las figuras 2.6 y 2.7 muestran las notaciones gráficas básicas para todo diagrama y un ejemplo de un diagrama de casos de uso.

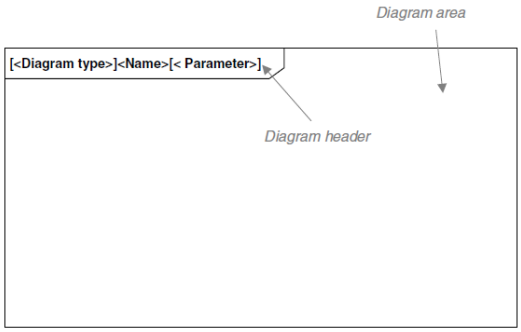


Figura 2.6 Notaciones básicas

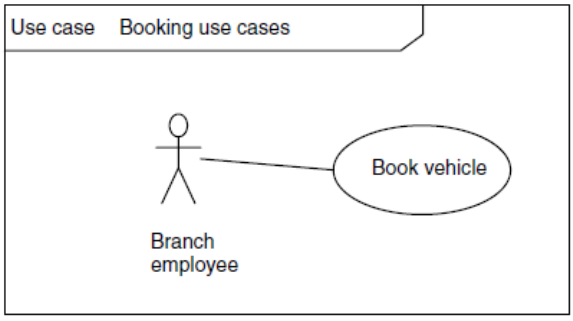


Figura 2.7 Ejemplo diagrama casos de uso

A continuación, haremos una descripción detallada de los diagramas UML que son de más interés para la realización de ésta tesis.

Diagramas de Clases

El diagrama de clases es sin lugar a dudas, el tipo de diagrama UML más utilizado. Es el diagrama básico para modelar la estructura que tendrá nuestro sistema. Un diagrama de clases describe los tipos de objetos (clases) existentes en el sistema y las diferentes relaciones estáticas que existen entre dichos elementos. También sirven para mostrar las propiedades y operaciones que una clase tiene así como las restricciones (*constraints*) que aplican en la manera en la que los objetos se conectan. UML utiliza el término característica como un término general que cubre tanto a las propiedades como a las operaciones de una clase. La figura 2.8 es un ejemplo del contenido de un diagrama de clases básico.

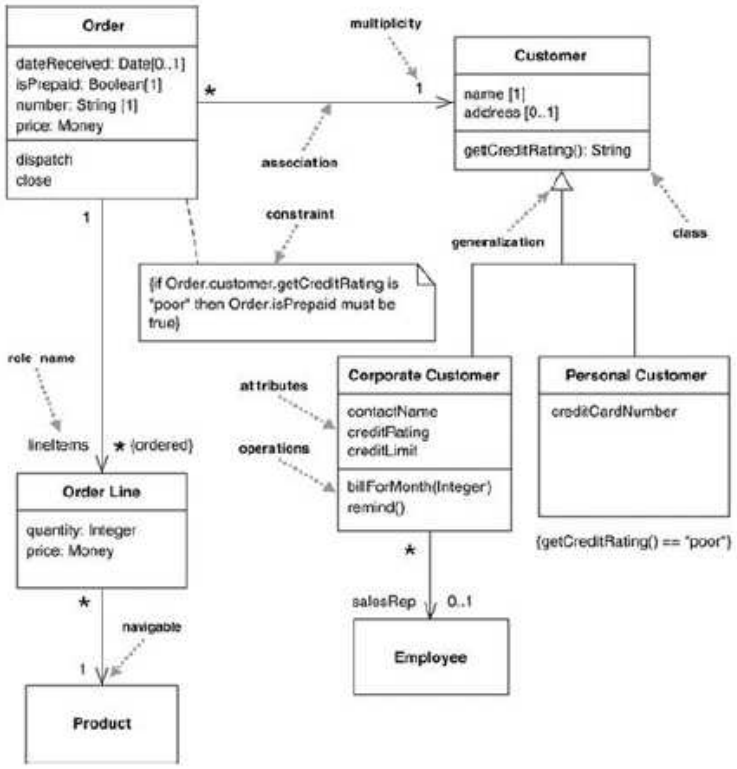


Figura 2.8 Diagrama de Clases (FOWLER,2004)

Los objetos (clases) son representadas por cajas que son divididas en tres compartimientos: el nombre de la clase, sus atributos y sus operaciones.

Propiedades de una clase

Las propiedades representan características estructurales de una clase. Las propiedades de una clase son representadas por dos tipos de notaciones: atributos o asociaciones.

La notación atributo describe una propiedad como una línea de texto dentro de la caja de la clase. El formato de un atributo es de la siguiente forma:

Visibilidad nombre: tipo multiplicidad ValorDefault {PropiedadAdicional}

- El marcador visibilidad indica si el atributo es público(+) o privado (-).
- El nombre del atributo no es otra cosa más que la forma en que la clase llama al atributo. Normalmente ese nombre corresponde al nombre de un campo en un lenguaje de programación.
- El tipo del atributo indica si hay algún tipo de restricción en cuanto al tipo de objeto que puede ser colocado en el atributo.
- La multiplicidad se refiere a cuántos objetos puede llegar a tener el atributo.
- El valor de default es el valor que tendrá un objeto para ese atributo al ser creado en caso de no haber especificado otro valor al momento de la creación.
- La propiedad adicional se usa para aclarar comportamientos extras que tendrá la propiedad.

Un ejemplo de definición de un atributo es el siguiente:

id: String [1] = “Sin nombre” [Sólo lectura]

La figura 2.9 muestra el uso de la notación atributo para representar propiedades.

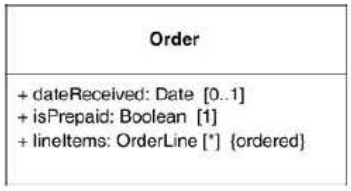


Figura 2.9

La notación asociación describe una propiedad como una línea sólida entre dos clases, con dirección de la clase origen a la clase destino. El nombre de la propiedad se coloca en el extremo final de la asociación. A diferencia de la notación atributo, la multiplicidad de una asociación se representa a ambos extremos de la línea sólida. El tipo de la propiedad es definido por la clase destino de la asociación.

Aunque tanto los atributos como las asociaciones son notaciones válidas para representar las propiedades de una clase y podrían utilizarse indistintamente, las asociaciones suelen usarse para representar propiedades de clases de mayor importancia mientras que los atributos para clases no tan prioritarias en el funcionamiento del sistema. La figura 2.10 muestra el uso de la notación asociación para representar propiedades.

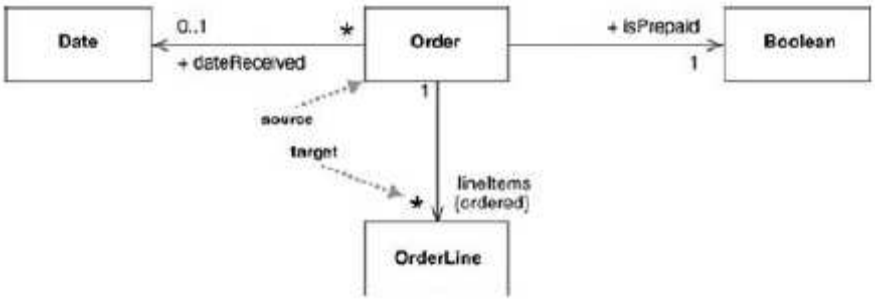


Figura 2.10 (IDEM)

Operaciones de una clase

Las operaciones de una clase son las acciones que dicha clase puede ejecutar. Pensando en términos de programación, las operaciones son los métodos (funciones) contenidos dentro de una clase. La notación utilizada para representar operaciones tiene el siguiente formato:

Visiblidad Nombre (lista parámetros) : ValorRetorno {PropiedadAdicional}

- Los marcadores visibilidad, nombre y propiedad adicional tiene el mismo significado que para la definición de atributos de propiedades.
- Lista de parámetros contiene todos los parámetros necesarios para la ejecución de dicha operación. Dichos parámetros utilizan la misma notación utilizada para atributos.
- El valor de retorno se refiere al tipo de valor que la operación entregará una vez ejecutada.

Un ejemplo para una operación sería el siguiente:

+ Balance (fecha: Fecha) : Dinero

Las operaciones son normalmente utilizadas para indicar las responsabilidades principales de la clase, operaciones que realizan acciones intermediarias de poca importancia no suelen representarse.

Dependencias

Una dependencia existe entre dos elementos cuando un cambio en la definición de un elemento proveedor (*supplier*) causará automáticamente cambios en el otro elemento cliente (*client*). Existen varias razones para la existencia de las dependencias entre clases como pueden ser:

-una clase le envía un mensaje a otra.

-una clase contiene a otra clase como parte de sus datos.

-una clase hace uso de otra clase en una de sus operaciones.

Es importante identificar las dependencias existentes en cualquier tipo de sistema ya que eso permitirá controlarlas y evitar efectos dominó no deseados.

UML permite establecer dependencias entre cualquier tipo de elementos. La notación utilizada es una línea punteada con dirección a la clase objetivo. Cabe señalar que la dependencia siempre tendrá una sola dirección. Es decir, modificaciones que se hagan en la clase cliente no afectarán las características en la clase proveedor. Sin

embargo, cualquier cambio en la clase proveedor cambiaría de manera automática a la clase cliente. La figura 2.11 muestra un ejemplo de dependencias entre clases.

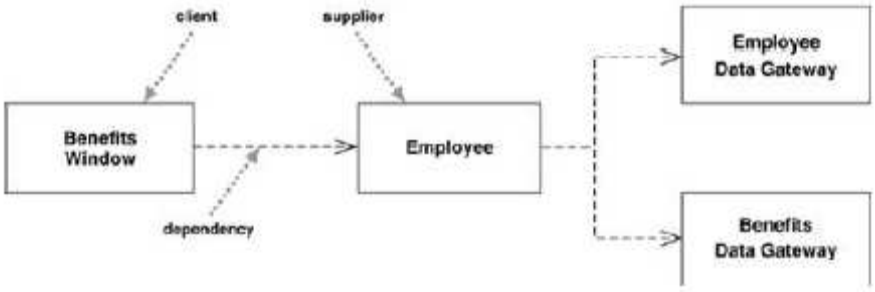


Figura 2.11 (IDEM)

Como regla general, es una buena regla de diseño evitar al máximo las dependencias, sobre todo cuando ocurren a través de grandes porciones del sistema.

Restricciones

Básicamente casi todo lo que se hace al dibujar un diagrama de clases indica restricciones. Al definir las propiedades y operaciones de una clase, establecemos las restricciones que dicha clase tendrá.

Sin embargo, las notaciones gráficas utilizadas para asociaciones, atributos y generalizaciones no pueden especificar todas las restricciones que pueden llegar a existir. Por lo tanto, UML ofrece un formato para capturar dichas restricciones dentro de un diagrama de clases.

Básicamente, la única limitante para capturar restricciones es que sean capturadas entre llaves ({}). Puede utilizarse lenguaje natural, lenguaje de programación o cualquier otro tipo de notación. De manera opcional, una restricción puede ser creada poniendo primero el nombre, seguido por un punto y coma, finalizando con la descripción de la restricción, por ejemplo " { Alumnos avanzados; solo podrán tomar la clases los alumnos inscritos al sexto año}.

Diagramas de Secuencias

El diagrama de secuencias pertenece al segmento de diagramas de interacción dentro de los tipos de diagramas UML. Es decir, ayuda a entender el comportamiento del sistema cuando grupos de objetos colaboran entre sí. El más común de este tipo de diagramas de interacción, es el diagrama de secuencias.

Típicamente, el diagrama de secuencias captura el comportamiento para un escenario en específico. El diagrama muestra un número de objetos y los mensajes que comparten dichos objetos. La figura 2.12 muestra el contenido de un diagrama de secuencias típico, en este caso para calcular el costo total de todos los productos contenidos en una orden.

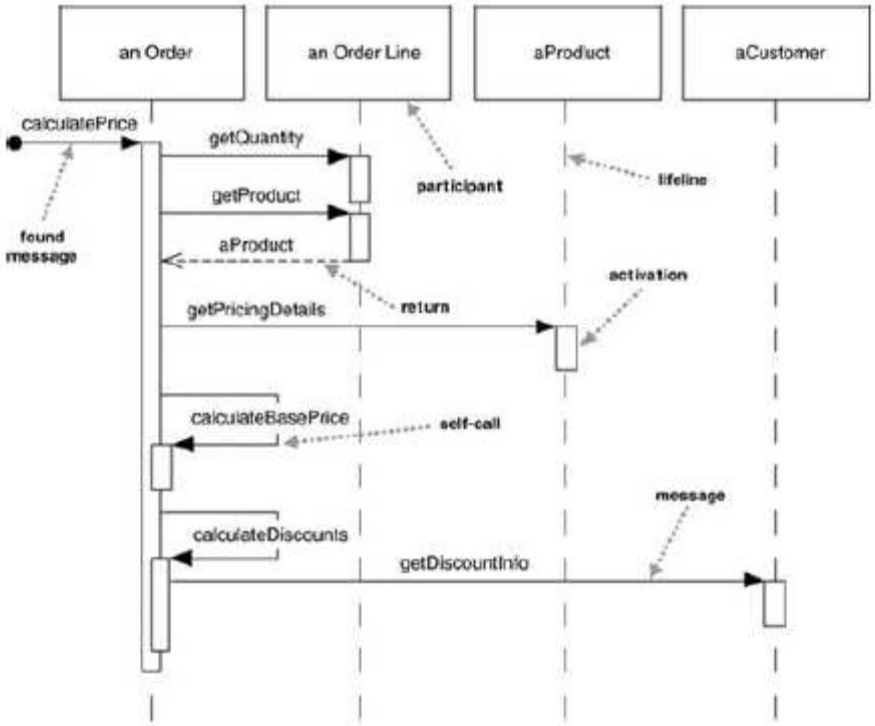


Figura 2.12 Contenido diagrama secuencias

De la figura 2.12 podemos identificar a los diferentes elementos existentes en un diagrama de secuencias:

- Participantes (*participant*): por medio de cajas, representan a los objetos del sistema que tomarán parte en la interacción.
- Línea de vida (*lifeline*): por medio de líneas punteadas representan el paso del tiempo en la interacción representada por el diagrama.
- Barra de activación (*activation*): bloque colocado en algún punto de la línea de vida de cada participante, representa cuándo está activo dicho participante. Pueden entenderse a una barra de activación como la notación gráfica de la ejecución de un método del objeto. A pesar de que las barras de activación son opcionales en UML, son de gran ayuda para clarificar el comportamiento del sistema.

- Mensaje (*message*): Por medio de una línea continua entre los participantes involucrados con dirección al participante destino, representa el envío de información de un participante a otro. Puede ser que un mensaje no represente la interacción entre dos objetos, sino que represente el procesamiento que un participante realiza de cierta información. En ese caso, el origen y destino de la línea continua es el mismo participante.
- Retorno (*return*): Por medio de una línea discontinua entre los participantes involucrados con dirección al participante destino, representa la respuesta a un mensaje previamente definido. Aunque no es obligatorio, por mayor claridad suele nombrarse al retorno de manera similar al mensaje que contesta.
- Mensaje externo (*found message*): Un diagrama de secuencias puede representar la interacción entre objetos tras recibir un mensaje externo, ya sea de otro objeto del sistema o de un usuario externo.

Es claro que una de las grandes virtudes de los diagramas de secuencia es la claridad con la permite representar las diferentes labores e interacciones que los participantes tendrán para cumplir un objetivo común con el paso del tiempo. Este tipo de diagramas no suele ser usados para mostrar detalles de algoritmos o procesos de comportamiento condicionados complejos.

Muchas veces, a lo largo de la ejecución del sistema existirán instancias de objetos que serán creadas bajo circunstancias específicas para realizar una labor en particular y después dejar de existir. La figura 2.13 muestra la notación utilizada por los diagramas de secuencia para la creación y destrucción de participantes.

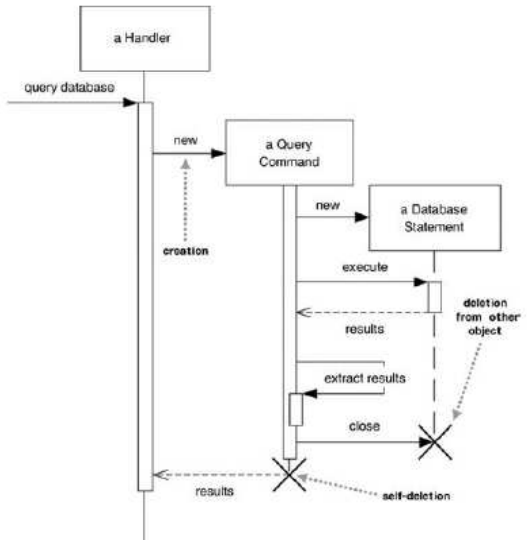


Figura 2.13 Creación y destrucción de Participantes

Para crear un participante, se hace uso de un mensaje que tiene como destino la caja que representa al participante a crear. Una vez creado el participante, puede interactuar como se explicó con anterioridad no sólo con el participante que lo creó, sino con todos los participantes existentes en el diagrama de secuencias.

La destrucción de un participante puede darse en dos formas. Cuando es por medio de un mensaje recibido de un participante, dicho mensaje tiene como destino una X en la línea de vida del participante a destruir. Cuando un participante se destruye a sí mismo, la X aparece sobre su línea de vida sin necesidad de recibir un mensaje que dispare la destrucción.

Un problema frecuente en los diagramas de secuencia es cómo mostrar operaciones iterativas (looping) o comportamiento condicional. Cabe recalcar que este tipo de situaciones no son las que normalmente son representadas mediante diagramas de secuencia, es más común el uso de diagrama de actividad. Esto se debe a que el objetivo de los diagramas de secuencia es poder visualizar la forma en la que los diferentes objetos del sistema interactúan, sin importar mucho el comportamiento lógico tras esa interacción.

Sin embargo, UML ofrece una notación gráfica llamada recuadro de interacción (*interaction frame*), el cuál permite resaltar un sector en específico de un diagrama de secuencia, tal como lo muestra la figura 2.14.

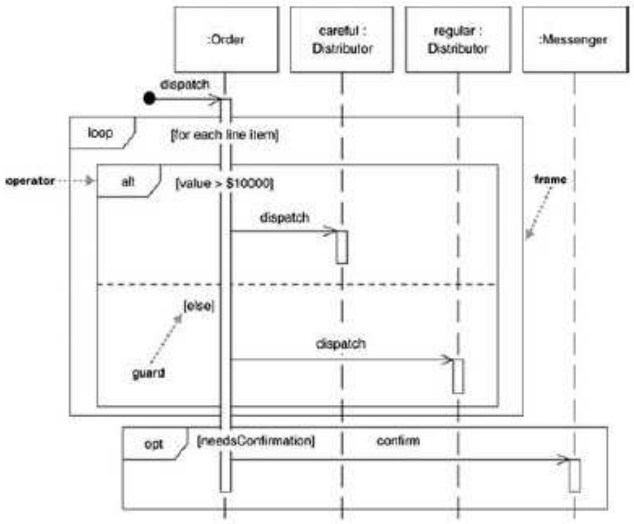


Figura 2.14 Recuadro de interacción

Como puede verse en la figura, un recuadro de interacción consiste en resaltar mediante un recuadro un fragmento de un diagrama de interacción, así como un operador que especifica el tipo de acción asociada a ese recuadro de interacción. Los operadores más comunes son:

- alt: Alternativo (*Alternative*).Utilizada para mostrar comportamiento condicionado donde sólo uno de los fragmentos del recuadro se ejecutará dada una condición en específico.
- Opt: Opcional (*Optional*): El fragmento se ejecuta sólo si se cumple la condición especificada.
- Par: Paralelo (*Parallel*): Cada fragmento es ejecutado en paralelo.
- Loop: Ciclo (*Loop*): El fragmento se ejecutará múltiples veces.

Diagramas de Casos de Uso

Antes de detallar los diagramas de casos de uso, es necesario entender el concepto detrás de un caso de uso. Los casos de uso son una técnica para capturar los requerimientos funcionales de un sistema. Su labor principal es identificar y describir las interacciones típicas que habrá entre el sistema y los usuarios externos que harán uso de él.

Visto de otra manera, los casos de uso describen los servicios que el sistema debe proveer desde un punto de vista externo. Una vez identificados los casos de uso que existirán en un sistema, la suma de todos éstos representará el propósito y significado de la operación total del sistema.

Un caso de uso debe de estar relacionado con al menos un actor, el cual representa el rol que un usuario externo tendrá al interactuar con el sistema. Por ejemplo, en un sistema encargado de la operación de una tienda departamental, los diferentes actores que podríamos llegar a encontrar son los clientes, cajeros, gerente, proveedores, etc. Si bien es lógico que la representación más clara de actores son los humanos que pueden llegar a interactuar con un sistema, todo elemento externo que interactúe con el sistema, como puede ser una máquina u otro sistema, puede ser modelado como un actor.

Cumplir los requerimientos funcionales de un actor es el objetivo principal de un caso de uso, por lo tanto, un caso de uso normalmente englobará un conjunto de escenarios y actividades unidas por el objetivo común especificado por las necesidades del actor. Un actor puede estar relacionado con uno o más casos de uso.

A pesar de que el concepto caso de uso tiene un papel fundamental dentro de UML, no existe nada que describa la manera de representar el conjunto de acciones que un caso de uso deberá cumplir. Sin embargo, una práctica común es realizar una tabla dónde se describe el escenario de éxito al que se quiere llegar así como las interacciones o paso que el sistema tendrá que tener con el actor para lograr dicho éxito.

Una práctica común al diseñar casos de uso es tratar de identificar escenarios que se repitan para cumplir con los requerimientos de más de un actor. Esto nos lleva a casos de uso que incluyen (*include*) a otros casos de uso encargados de realizar las actividades repetitivas. Ésta práctica es clave para la modularidad del sistema.

Un diagrama de casos de uso sirve para mostrar los casos de uso de un sistema así como mostrar la relación que existe no sólo entre los casos de uso y los actores que harán uso del sistema, sino también entre los casos de uso. Los diagramas de casos uso son un poco limitados en cuanto a que no especifican la manera de detallar el contenido de acciones a realizar por un caso de uso. Esto hace que el uso de los diagramas de casos de uso no sea mandatario a la hora de modelar un sistema con UML. En la práctica, los diagramas de casos de uso son utilizados para poder establecer una comunicación estandarizada entre los diseñadores del sistema y los usuarios externos (*stakeholders*) interesados en el funcionamiento del sistema. Un diagrama de casos de uso típico es el mostrado por la figura 2.15

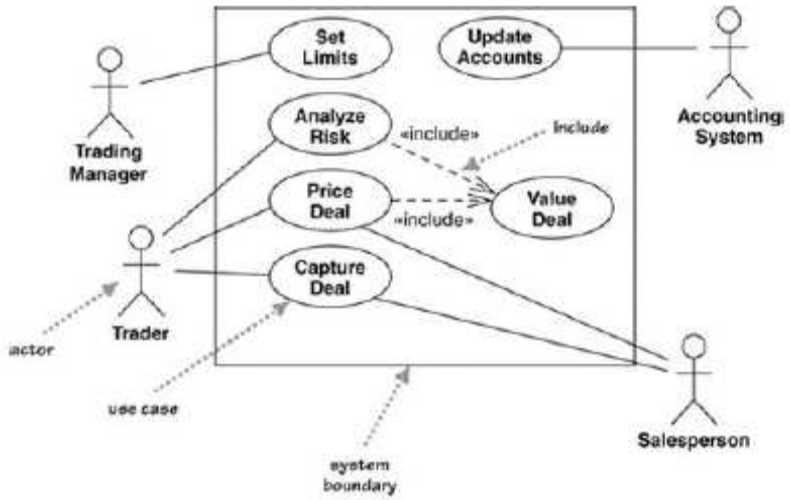


Figura 2.15 Diagrama de casos de uso

Los elementos principales en un diagrama de casos de uso son los siguientes:
-Frontera del Sistema (*System Boundary*): Por medio de un recuadro, engloba todos los caso de uso que conforman al sistema , sirve para delimitar su interfase con el exterior.
-Actor: Representa a los usuarios con los cuales el sistema debe interactuar.
-Casos de Uso: Cada óvalo contenido representa un caso de uso del sistema y por medio de líneas se relaciona con uno o más actores. Cabe resaltar que para representar la relación entre casos de uso, se hace uso de una línea punteada con dirección al caso de uso secundario que el caso de uso primario incluye (*include*).

Diagramas de Máquinas de Estado

Un sistema se encuentra siempre en un estado que no es más que la combinación de los valores que tengan los diferentes componentes del sistema en un instante dado. Los diferentes eventos que lleguen al sistema, provocarán diferentes reacciones que dependerán del estado actual en el que se encuentre el sistema y que, eventualmente, provocarán que el sistema cambie a un estado nuevo. Un concepto básico en el diseño de máquina de estados es la premisa de que un sistema no puede pasar de un estado a otro hasta que se hayan realizado todas las acciones especificados en el estado inicial. El objetivo de un diagrama de máquina de estado es representar de manera gráfico dicho comportamiento. La figura 2.16 muestra un diagrama de máquina de estados típico.

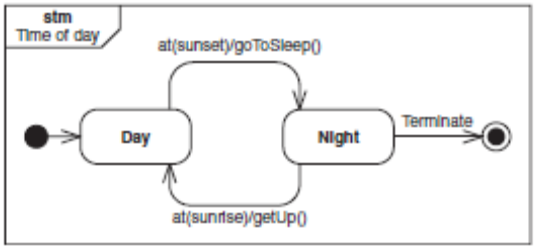


Figura 2.16 Diagrama de Máquina de estados (WEILKIENS,2007)

Básicamente, el diagrama está compuesto por todos los estados existentes así como las transiciones entre los estados y los eventos asociados a dichas transiciones. Normalmente, un diagrama de máquina de estados representa el comportamiento de un objeto o clase del sistema.

La máquina de estados como tal no tiene una representación gráfica, más bien es representada por todo el diagrama en conjunto. El marco externo del diagrama representa la frontera de la máquina de estados del objeto. De la figura 2.16 podemos definir las notaciones gráficas utilizadas para representar a las diferentes partes de un diagrama de máquina de estados.

- Pseudo estado inicial: No es un estado inicial como tal sino un punto de partida representado por un círculo relleno a partir del cual sale una flecha que apunta al estado inicial de la máquina.
- Estado final: por medio de una diana (*bullseye*), representa la total ejecución de la máquina de estados.
- Estado: Cada uno de los estados existentes en la máquina es representado por un caja con las esquinas redondeadas que contiene el nombre del estado.
- Transición: flecha que una los estados involucrados en la transición que tiene como dirección el estado final.

Una transición indica un movimiento de un estado a otro. Cada transición tiene una etiqueta con un formato que consta de tres partes (todas opcionales):

Evento-disparador [condición transición] / Actividad a ejecutar

El evento disparador es normalmente un solo evento capaz de disparar el cambio de estado. La condición de transición, es una condición booleana que debe ser cierta para que la transición se lleve a cabo. La actividad es algún comportamiento a ejecutar durante

la transición. Debido a que no todas las transiciones implican realizar alguna actividad, es común no encontrar actividades en las etiquetas de las transiciones.

No todos los eventos recibidos por un estado implican una transición a otro estado. Existen eventos que pueden generar actividades internas a ejecutar por el estado sin que éstas generen una transición.

Superestados

Es común encontrar estados que comparten tanto transiciones como actividades internas. En estos casos es posible convertir dichos estados en subestados y trasladar el comportamiento común a un superestado, tal como se muestra en la figura 2.17.

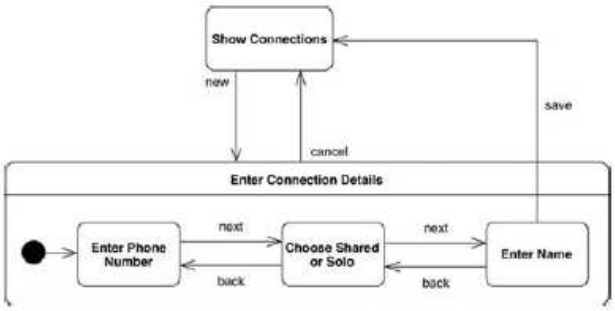


Figura 2.17 Superestado

La ventaja del uso de superestados es evitar la repetición de las transiciones que tienen en común los subestados, como es el caso para las transiciones nueva (*new*) y cancelar (*cancel*) de la figura 2.17.

Estados Ortogonales

Los estados pueden ser descompuestos en varios estados ortogonales que se ejecutan de manera concurrente. La figura 2.18 muestra un ejemplo de una máquina de estados de un radio que por un lado es capaz de reproducir ya sea un cd o una estación de radio y al mismo tiempo es capaz mostrar la hora actual o la hora a la cual tiene programada una alarma.

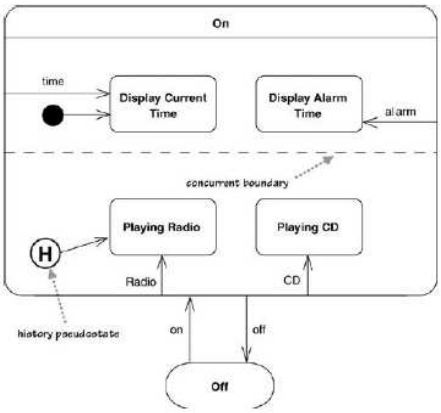


Figura 2.18 Estados ortogonales

Pseudo estados

Un pseudo estado es un elemento de control que influye en la secuencia de eventos en una máquina de estados. No es considerado un estado real, así tampoco representa una combinación de valores de ningún tipo.

UML define 10 diferentes pseudo estados, cada uno encargado de describir una propiedad especial que controla las transiciones de los estados. La siguiente tabla muestra una breve descripción de los pseudo estados existentes.

Pseudo Estado	*Descripción*
Estado inicial (*Initial State*)	Punto de partida de una máquina de estados
Historia Poco Profunda (*Shallow History*)	Representando por una H dentro de un círculo, usado para guardar el último estado que tenía la máquina de estados, de manera que la próxima vez que vuelva a ser ejecutada se recuerde cuál era ese estado.
Historia Profunda (*Deep History*)	Representado por una H seguida de un asterisco dentro de un círculo. Permite recordar no sólo el último estado, sin también el último subestado utilizado, en el caso de que la máquina de estados contenga súper estados.
Empalme (*Junction*)	Representado por un círculo negro, permite conectar dos transiciones e unirlas para formar una sola.
Opción (*Choice*)	Representado por un rombo, permite generar varias transiciones a partir de una. Cada transición saliente deberá tener una condición asociada.
Terminación (*Terminate*)	Representado por una X, implica la terminación de la máquina de estados sin que haya cambio de estado alguno.
Bifurcación (*fork*)	Representado por una línea negra, una bifurcación tiene una transición entrante y varias transiciones salientes que se dirigen a estados diferentes ortogonales entre si. Es decir, todos los estados son activados al mismo tiempo.
Unión (*Join*)	También representado por una línea negra, es el inverso a una bifurcación. Varios transiciones entrantes generadas por diferentes estados, se unen en una sola transición saliente
Punto Entrada (*Entry Point*)	Representado por un círculo vacío. Implica un punto de entrada especial a una estado en específico de la máquina de estados.
Punto Salida (*Exit Point*)	Representado por una X dentro de un círculo. Implica la terminación de la máquina de estados.

Diagramas de Actividad

Los diagramas de actividad son una técnica para describir procedimientos lógicos, procesos de negocios o flujos de trabajo. En muchos sentidos, tienen un rol similar a un diagrama de flujo, con la gran diferencia de que UML permite que los diagramas de actividad soporten comportamiento paralelo.

A lo largo de las versiones existentes de UML, los diagramas de actividad han sufrido grandes cambios. En UML 1, los diagramas de actividad eran vistos como un caso especial de diagramas de estado. Para UML 2, esta noción fue cambiada permitiendo que los flujos de trabajo fueran modelados por los diagramas de actividad de mejor manera.

La figura 2.19 muestra un diagrama de actividad básico.

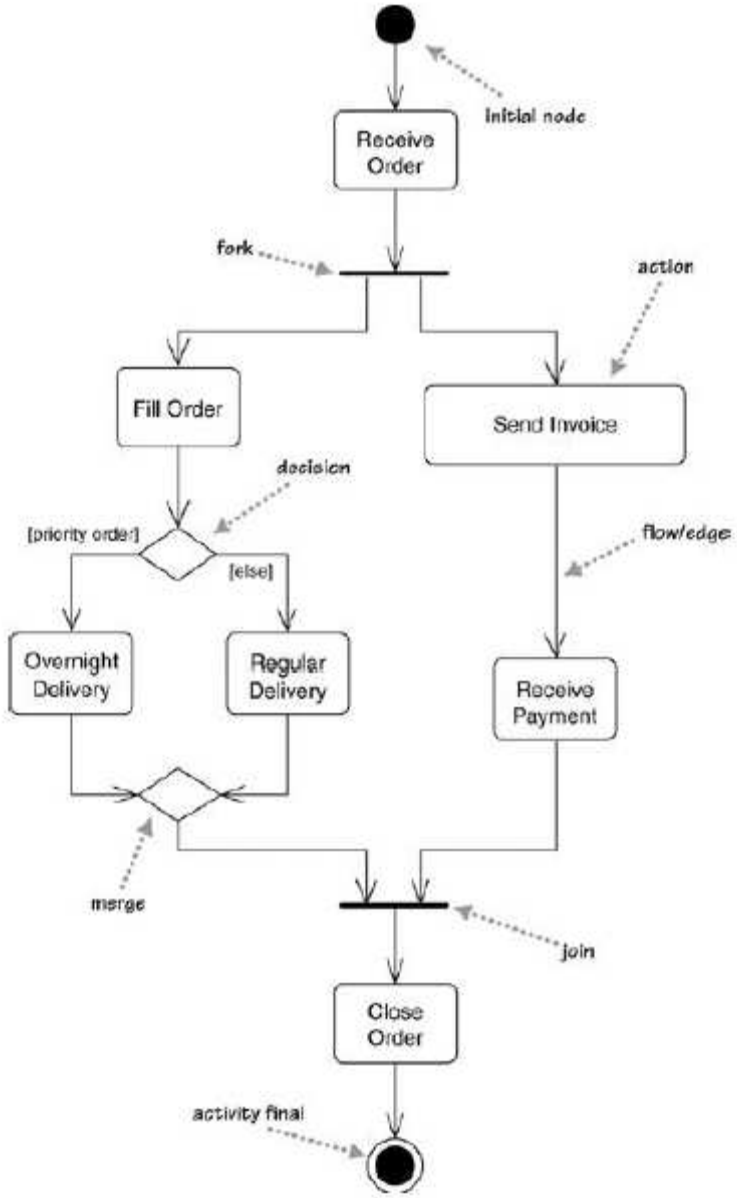


Figura 2.19 Diagrama de Actividad (FOWLER. Op.Cit.)

De la figura anterior, podemos identificar a las notaciones gráficas principales existentes en un diagrama de actividad.

- Nodo Inicial (*Initial Node*): Representado por un círculo lleno, implica el inicio de nuestra actividad.
- Acción (*Action*): Representado por un rectángulo redondeado, implica una acción a realizar durante la actividad.
- Flujo (*flow*): Representado con una flecha con dirección a la acción a realizar.
- Bifurcación (*Fork*): Representado por una línea negra, utilizado para dividir un flujo entrante en varios flujos salientes. Por medio de una bifurcación es como podemos realizar actividades en paralelo.
- Unión (*Join*): Representado por una línea negra, utilizado para unir varios flujos entrantes en un solo flujo saliente. Cabe resaltar que el flujo saliente no se ejecutará hasta que todos los flujos entrantes hayan llegado al punto de unión, lo que permite asegurar la completa ejecución de procesos paralelos previo continuar con la actividad.
- Decisión (*Decision*) : Representado por un rombo, tiene un único flujo de entrada y varios flujos de salida, cada uno con una condición asociada. Cada que el flujo de la actividad llega a una decisión, sólo podrá tomarse un flujo de salida, por lo que las condiciones involucradas deberán ser mutuamente excluyentes.
- Fusión (*merge*): Representado por un rombo, tiene varios flujos de entrada y un solo flujo de salida. Una fusión marca el término del comportamiento condicionado empezado por una decisión.
- Actividad final (*activity final*): Representado por una diana, implica el término del diagrama de actividad.

Con el fin de limitar el tamaño de los diagramas de actividad, las acciones pueden descomponerse en sub actividades tal como lo muestran las figuras 2.20 y 2.21

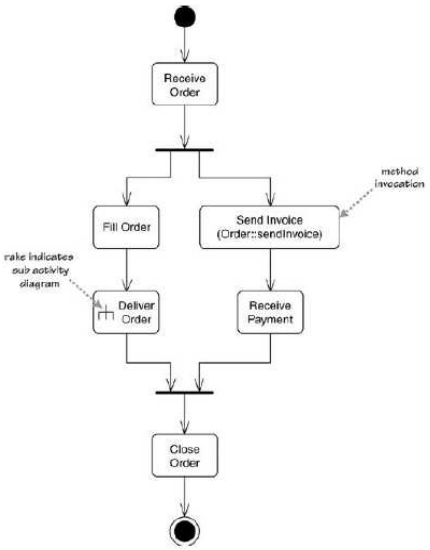


Figura 2.20

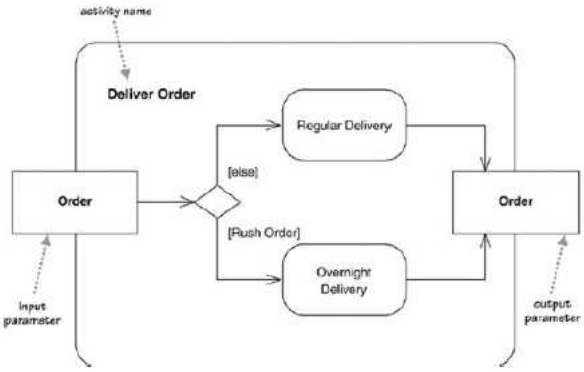


Figura 2.21

Son dos las notaciones principales para representar sub acvtividades en un diagrama de actividad. La primera es por medio de una llamada a un método de una clase (*method invocation*) haciendo uso de la siguiente sintaxis nombre-clase::nombre-método. La segunda notación utilizada para representar sub actividades, y las más común de las dos, es incluyendo un trinche dentro de la acción. Dicha notación hace referencia a que dicha sub actividad será detallada en otro diagrama. En la figura 2.20 podemos ver el uso del trinche en la sub actividad Deliver Order, cuyo diagrama de actividad es detallado en la figura 2.21. De la figura 2.21, cabe resaltar la notación utilizada para representar los parámetros de entrada (*input parameter*) y de salida (*output parameter*) de un actividad, los cuales son representados por cajas colocadas en el lado izquierdo (entrada) y derecho (salida) del recuadro del diagrama de actividad.

Si bien es cierto que un diagrama de actividad por sí sólo puede definir en su totalidad la ejecución de un proceso o una rutina de manera aislada, es importante incorporar notaciones gráficas para representar cómo dichas actividades respondan a señales externas.

Una de las señales más usadas es la señal de tiempo (*time signa*l). Representada por un reloj de arena representa el paso de un intervalo de tiempo específico. Como parte de la etiqueta de dicha señal es necesario indicar el intervalo que dicha señal de tiempo representa.

Muchas veces es necesario esperar a recibir una señal de un proceso externo de forma que pueda continuarse con una actividad. Por medio de la notación gráfica acepta señal (*accept signal*), podemos representar la espera por una señal externa por parte de una actividad. Por lo tanto, el flujo de la actividad no continuará hasta que la señal sea recibida. La espera de una señal externa se representa por medio de un rectángulo con un corte en forma de triángulo en su lado izquierdo.

Así como podemos recibir señales, también existe la notación gráfica para el envío de señales (*send signal*). La notación utilizada para representar el envío es un rectángulo al cual se le añade un triángulo del lado derecho. Tanto el envío como la recepción de señales deberían tener como etiqueta el nombre de la señal involucrada.

Las figuras 2.22 y 2.23 muestran diagramas de actividad que hacen uso de las notaciones aquí descritas para el manejo de señales.

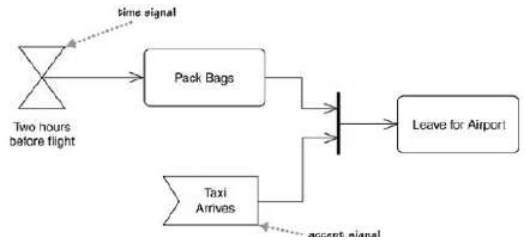


Figura 2.22

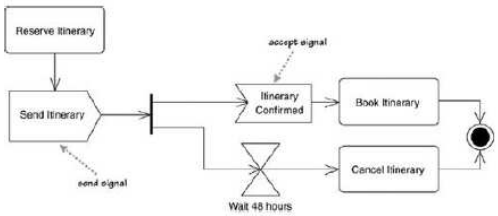


Figura 2.23

UML 2 define diferentes notaciones gráficas para los flujos o conexiones entre dos acciones. La más común es una flecha simple entre dos acciones. Sin embargo, en la medida en que los diagramas de actividad crecen es probable que aparezcan problemas de ruteo con los flujos que unen a las acciones. Éste problema puede resolverse con el uso de conectores (*connector*), los cuales evitan tener que dibujar una flecha para unir las dos acciones. Los conectores deben de existir en pares, uno con un flujo entrante y otro con un flujo saliente. Ambos conectores deben de tener el mismo nombre dentro de un pequeño círculo.

Al igual que los métodos, las acciones pueden tener parámetros. No es muy común mostrar información acerca los parámetros que tienen las acciones en un diagrama de actividad. Pero, en caso de así requerirlo, se puede hacer uso de puertos, que son pequeñas cajas colocadas en los extremos de las acciones. Cuando el puerto se coloca del lado izquierdo representa un parámetro de entrada y si se coloca del lado derecho representa un parámetro de salida. Normalmente suele colocarse el nombre del objeto o clase que la acción espera como parámetro al lado del pin. La figura 2.24 muestra el uso de conectores y pines en un diagrama de actividad.

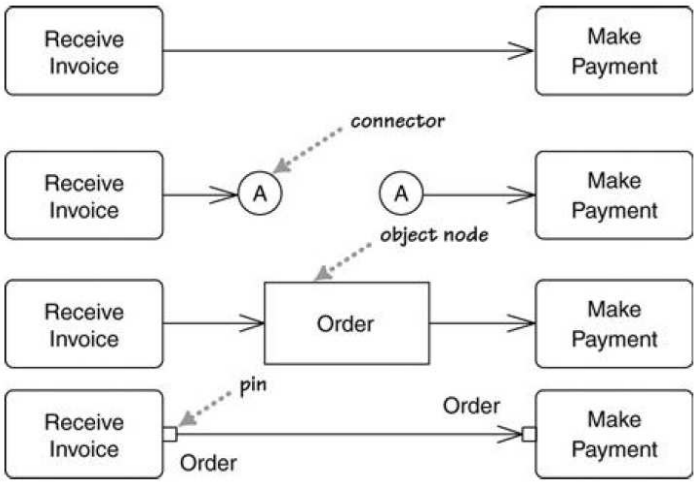


Figura 2.24 Conectores y Puertos (IDEM)

SYSML

Definición

El lenguaje de Modelado de Sistemas (*Systems Modelling Language, SysML*)es un lenguaje de modelado gráfico de propósito general para especificar, analizar, diseñar y verificar sistemas complejos que pueden incluir hardware, software, información o cualquier otro tipo de proceso.

Antecedentes

UML fue diseñado originalmente como un lenguaje de modelado gráfico de software. Gracias a su rápido proliferación y sus mecanismos de extensión integrados (estereotipos), rápidamente fue utilizado en otro tipo de escenarios, en especial la ingeniería de sistemas, que podría ser definida como la rama de la ingeniería dedica en el diseño y aplicación de un todo (sistema) así como de cada una de las partes que lo forman. La ingeniería de sistemas involucra definir un problema a resolver en su totalidad, tomando en cuenta todos los aspectos y todas las variables involucradas, no sólo técnicas sino sociales.

La ingeniería de sistemas no tenía un lenguaje de modelado unificado. Debido a que comúnmente involucra una labor interdisciplinaria, necesitaba de un lenguaje que fuera completamente independiente de ramas específicas como lo son el software, hardware, mecánica, etc. UML 2 fue desde su aparición un buen candidato para solventar este problema. Si bien el lenguaje ya cubría con la mayoría de los requerimientos necesarios para el modelado de sistemas, su mecanismo de extensión permitía casi de manera natural hacer las adaptaciones finales necesarias por la ingeniería de sistemas.

En 2003 el Consejo Internacional de Ingeniería de Sistemas (*International Council of Systems Engineering* INCOSE), se dio a la tarea de establecer a UML como el lenguaje de modelado gráfico estándar a utilizar por la Ingeniería de Sistemas. El lenguaje fue expandido con varios elementos, sobre todo añadiendo la posibilidad de modelar de manera gráfica requerimientos y sistemas continuos. La versión adaptada de UML fue llamada Lenguaje de Modelado de Sistemas (SysML) y fue liberada por primera vez en 2004.

Una de las principales críticas a UML es su extensión (HOLT ,2007). Para evitar que SysML no fuera tan extensivo, varios elementos presentes en UML que no tenían mucha relación con el diseño de sistemas fueron excluidos de SysML. Esto incluye, por ejemplo, componentes diseñados específicamente para modelado de software. El concepto de modelado orientado a objetos pasa a un segundo plano en SysML debido a que se espera que sea un lenguaje utilizado por ingenieros de diferentes áreas como puede ser la electrónica o la mecánica que no forzosamente hacen uso de dicho concepto. La figura 2.25 muestra de manera gráfica la relación existente entre SysML y UML.

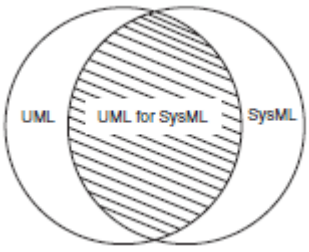


Figura 2.25 Relación UML-SysML

SysML permite a los ingenieros modelar los requerimientos, la estructura y el comportamiento del sistema. El enfoque de los nuevos elementos existentes en SysML tiene una relación estrecha con conceptos básicos de la ingeniería de sistemas como son ingeniería de requerimientos y comportamiento de sistemas. A pesar de que los requerimientos eran modelados tradicionalmente con diagramas de casos de uso, se tenía la gran limitante de considerar los requerimientos desde un punto de vista de comportamiento. La introducción de los diagramas de requerimientos en SysML permiten modelar las relaciones estructurales existentes entre los requerimientos de un sistema.

Las extensiones o cambios más grandes hechos a UML en la especificación de SysML son las siguientes:

-Las clases son llamadas bloques. En SysML el diagrama de clases de UML es llamado diagrama de definición de bloques (*Block Definition Diagram*).

-El diagrama de estructura compuesta de UML es llamado diagrama interno de bloques (*internal block diagram*) en SysML.

-Flujos de datos entre elementos de un diagrama interno de bloques puede ser modelado.

-Dos nuevos tipos de diagramas fueron integrados: diagramas de requerimientos (*Requirements Diagram*) y diagramas paramétricos (*Parametric Diagram*).

A finales del 2006, en la reunión anual de la OMG, SysML fue aceptado como estándar pero no fue hasta 2007 que fue publicada la versión final 1.0 del estándar, después de un período de prueba que le dio a la especificación sus ajustes finales

A pesar de que SysML es una extensión de UML omite algunos elementos de UML. De los 13 diferentes tipos de diagramas existentes en UML, SysML hace uso de 7 de ellos:

-Diagramas de clases, renombrado diagrama de definición de bloque.

-Diagramas de paquete.

-Diagrama de estructura compuesta, renombrado diagrama interno de bloque.

-Diagrama de máquinas de estados.

-Diagramas de actividad, con leves variaciones de sintaxis.

-Diagramas de casos de uso

-Diagramas de secuencia.

Si a estos 7 diagramas se les suman las 2 extensiones de SysML, diagramas de requerimientos y paramétricos, son 9 en total los diagramas ofrecidos por SysML. La figura 2.26 muestra dichos diagramas organizados por las familias de diagramas existentes en SysML, estructurales (*Structural Diagrams*) y de comportamiento (*Behavioural Diagrams*).

El hecho de que son varios los elementos de UML faltantes en SysML, en la práctica no genera ningún tipo de problema. La gran mayoría de herramientas de modelado SysML existentes en el mercado son extensiones a herramientas UML que permiten introducir los conceptos de SysML. Por lo tanto, los elementos nativos de UML siguen presentes. Lo que da como resultado que sea muy común ver modelos de sistemas que no son modelos UML o SysML en el sentido estricto, sino una mezcla de ambos.

Los mecanismos de extensión de UML (estereotipos) también son parte de SysML. Por lo tanto, tal como sucede es UML es posible expandir el vocabulario del

modelo basándonos en los elementos originales del lenguaje, lo que permite adaptarlo a las necesidades de modelado específicas que se tengan.

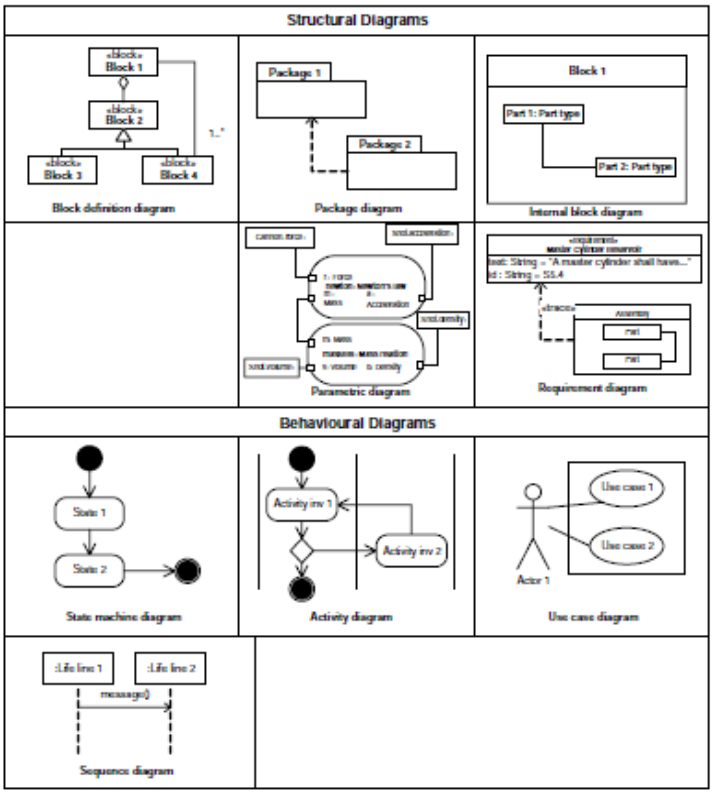


Figura 2.26 Diagramas SysML (HOLT, 2007)

A continuación haremos una descripción detallada de los diagramas de Requerimientos, parte esencial de SysML y que son el único tipo de diagrama específico de SysML utilizado en la realización de ésta tesis.

Diagramas de Requerimientos

A pesar de que en UML es común el uso de casos de uso para modelar requerimientos funcionales, no existe elemento algún para incluir en el modelo requerimientos no funcionales, como pueden ser tamaños, tiempos de respuesta, etc. Es aquí donde los diagramas de requerimientos de SysML rompen con las limitantes de UML en lo que a requerimientos respecta. Por medio de dichos diagramas, SysML permite modelar no sólo los requerimientos del sistema, sino también la relación de dichos requerimientos con los demás elementos del sistema.

Antes de definir las notaciones gráficas usadas en los diagramas de requerimientos, es necesario definir lo que entendemos por un requerimiento: *por requerimiento se entiende la descripción de una propiedad o comportamiento que el sistema debe cumplir en todo momento*(HOLT ,2007). En términos de UML, el requerimiento es un estereotipo («*requirement*») del elemento clase.

Los requerimientos establecen una especie de contrato entre los usuarios solicitantes del sistema (*stakeholders*) y todas las personas involucradas en el diseño y desarrollo del mismo. Los requerimientos deben definir claramente flujos y condiciones que el sistema debe cumplir.

Al ser un requerimiento un estereotipo del elemento clase de UML, mantiene la misma semántica utilizada por la clase modificada por las extensiones de semántica y propiedades en un diagrama de requerimientos típicos como el mostrado en la figura 2.27 y que describiremos a continuación.

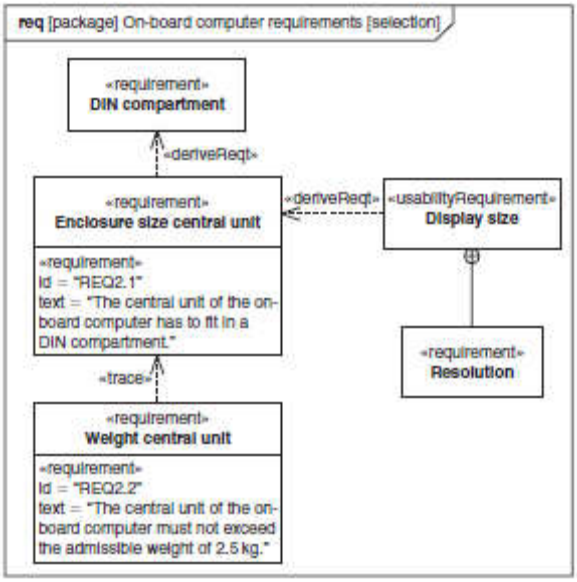


Figura 2.27 Diagrama de Requerimientos (IDEM)

Al igual que las clases, los requerimientos son simbolizados por cuadrados etiquetados por el estereotipo «*requirement*». A diferencia de las clases que se dividen en tres secciones, las cajas que representan requerimientos sólo se dividen en dos secciones. La primera muestra el nombre del requerimiento y la segunda las propiedades del mismo.

Dos propiedades elementales de un requerimiento son un identificador único (ID) y un texto descriptivo (*text*). El ID es simplemente una cadena que identifica al requerimiento dentro de todos los existentes en el sistema. A pesar de que debe de ser único, SysML no especifica un formato o método para validar que cada ID existente lo sea, es responsabilidad única y exclusiva del modelador. El texto descriptivo consiste en una oración breve que hace una descripción puntual del requerimiento. A diferencia de las clases, SysML no permite que se le asocien a un requerimiento atributos u operaciones.

Debido a que los requerimientos son elementos que tienen la única función de especificar un comportamiento o propiedad a cumplir, no tiene ningún sentido la existencias de instancias de requerimientos. Esto provoca que los requerimientos sean considerados elementos 100% abstractos.

Muchas veces la necesidad de cubrir un requerimiento implica la especificación de otro requerimiento derivado por el primero. Por ejemplo, consideremos el caso de un sistema de reservaciones que tiene como requerimiento que la experiencia del usuario al hacer uso de él sea lo más satisfactoria posible. Como resultado de este requerimiento podemos definir otro requerimiento que especifique que el tiempo de respuesta del sistema de reservación sea mínimo. Para mostrar la relación entre un requerimiento derivado de otro, SysML ofrece el elemento requerimiento derivado (*derived requirement*, «deriveReqt»), la cual es un estereotipo del elemento UML abstracción.
La notación gráfica utilizada para representar un requerimiento derivado es una flecha de línea punteada que se lee en la dirección marcada por la fecha. Es decir, el requerimiento del que parte la fecha es derivado del requerimiento señalado por la flecha. La figura 2.28 muestra un ejemplo de un requerimiento derivado.

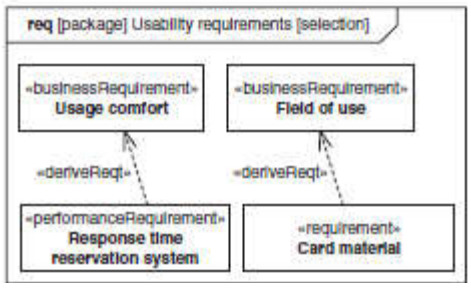


Figura 2.28 Requerimiento derivado

Es común establecer niveles de jerarquía entre los diferentes requerimientos. Requerimientos muy generales, utilizados casi como encabezados, suelen contener un conjunto de requerimientos mucho más detallados. El elemento de SysML para mostrar este tipo de relaciones es un espacio de contención de nombres (*namespace containment*). La notación gráfica utilizada por este elemento es un círculo con una cruz dentro a partir de cual salen línea hacia cada uno de los requerimientos contenidos por el espacio, tal como lo muestra la figura 2.29.

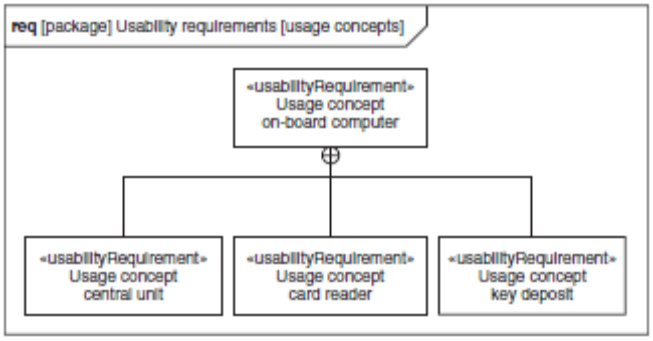


Figura 2.29

Como ya se mencionó con anterioridad, una de las principales innovaciones existentes en los diagramas de requerimientos propuestos por SysML, es la capacidad de vincular conceptos abstractos como son los requerimientos con componentes del sistema en un solo diagrama. Para lograrlo, los diagramas de requerimientos ofrecen una relación de satisfacción la cual se usa para describir a un elemento del diseño del sistema que satisface un requerimiento específico. El elemento ofrecido por SysML para una relación de satisfacción es el estereotipo «satsify».

Al modelar la relación, es posible analizar los efectos que los cambios en el diseño del elemento involucrado afectan al requerimiento solicitado y, de la misma forma, es posible analizar qué cambios son necesarios hacer en el elemento de diseño al cambiar el requerimiento. El elemento de diseño involucrado en la relación de satisfacción puede ser de cualquer tipo, aunque es muy común el uso de paquetes o bloques.

La notación gráfica usada para una relación de satisfacción es una flecha de línea punteada con la etiqueta «*satsify*», tal como lo muestra el diagrama de la figura 2.30

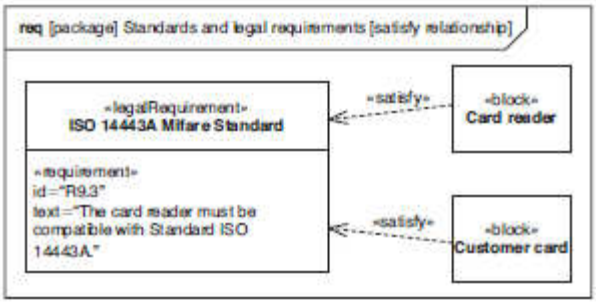


Figura 2.30 Relación de Satisfacción

Una cuestión importante a resaltar con las relaciones de satisfacción es que no indican en qué porcentaje un elemento de diseño satisface a un requerimiento. Por lo que es común encontrar relaciones de satisfacción entre un requerimiento y varios elementos de diseño.

Como parte del modelado integral de todo sistema, es necesario plantear escenarios de prueba que permitan constatar que los elementos de diseño del sistema cumplan de manera adecuada con lo planteado por los requerimientos.

SysML ofrece varios elementos que ayudan a modelar la prueba de sistemas. El primero de ellos es el caso de pruebas, el cual define un flujo encargado de revisar si el

sistema cumple o no con un requerimiento. Es caso de prueba es un estereotipo, «*testCase*», de todos los elemento UML de tipo comportamiento como lo son actividades, interacciones, máquinas de estados, etc. Cualquiera de estos elementos puede ser utilizado para una descripción detallada del flujo (*script* de pasos) representado por un caso de prueba. La notación gráfica utilizada para representar un caso de prueba es un rectángulo que contiene el nombre del caso de prueba y la etiqueta «*testCase*».

Una vez definido el caso de prueba, es necesario hacer uso de otro elemento para poder relacionar dicho caso de prueba con un requerimiento. SysML contiene al elemento relación de verificación (*verify relationship*) para realizar dicho cometido. La notación gráfica utilizada es una flecha de línea punteada con la etiqueta «verify». La figura 2.31 muestra un diagrama que incluye casos de prueba y relaciones de verificación con requerimientos.

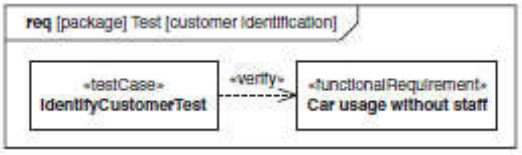
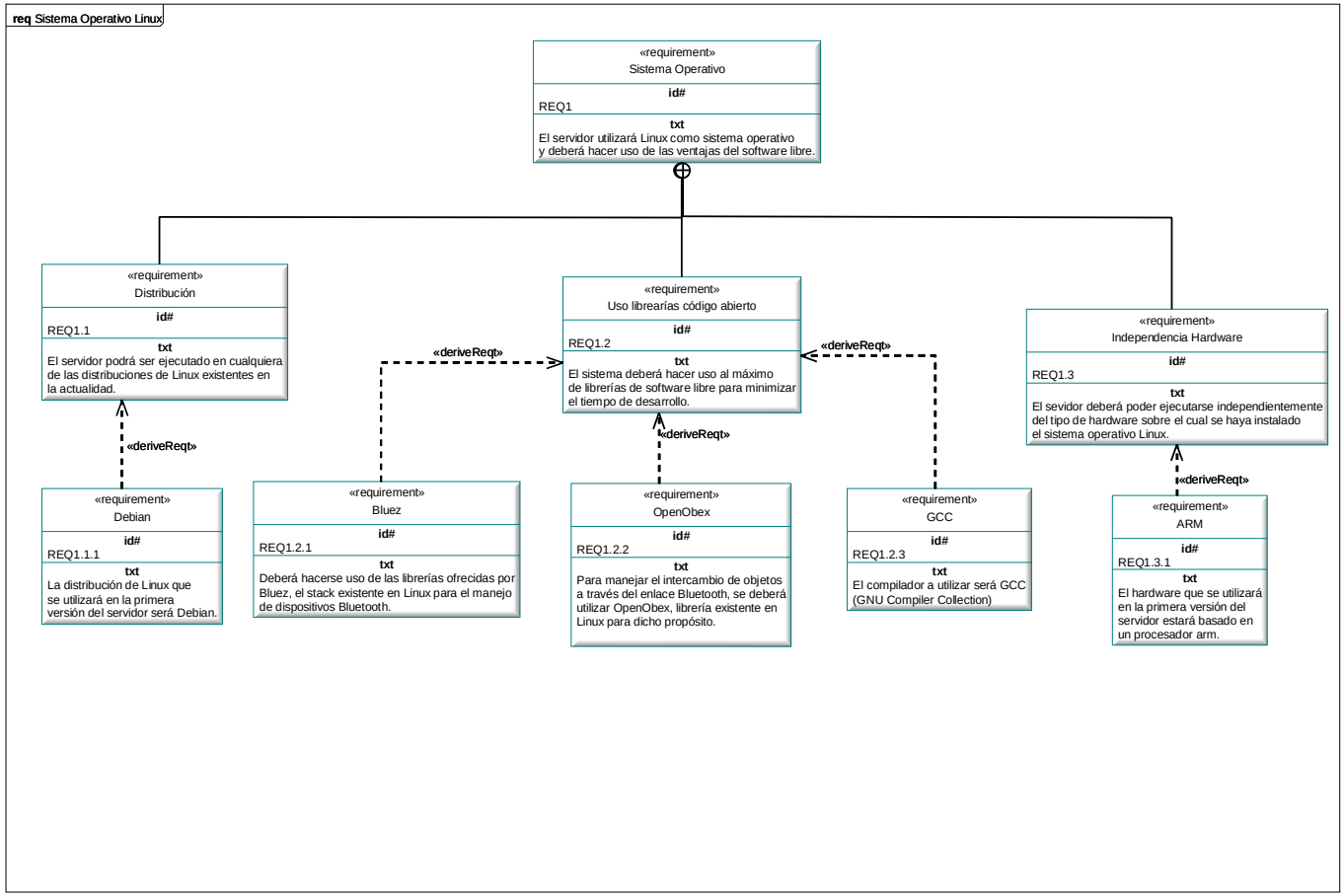


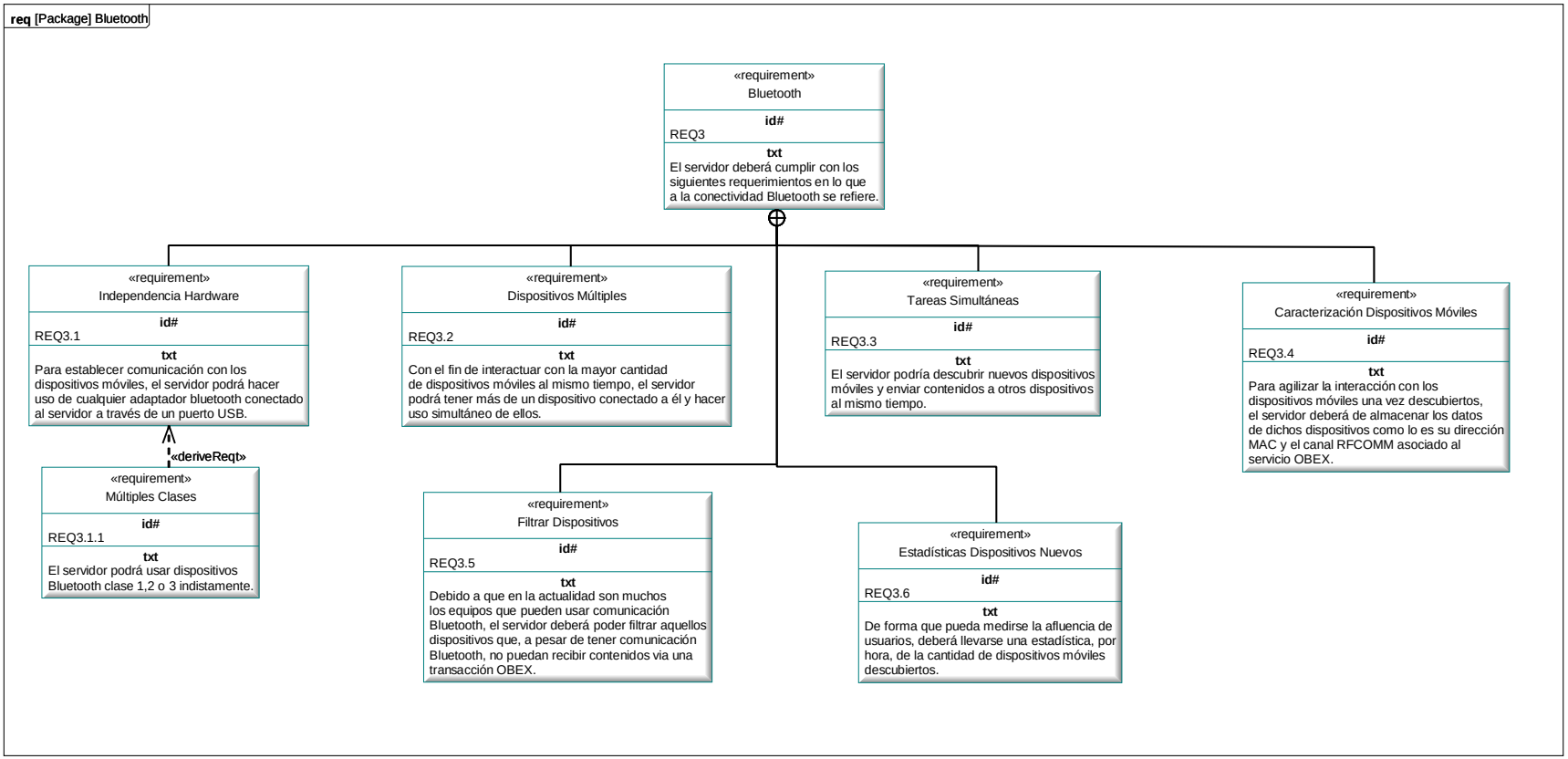
Figura 2.31 Casos de prueba
y relación verificación.

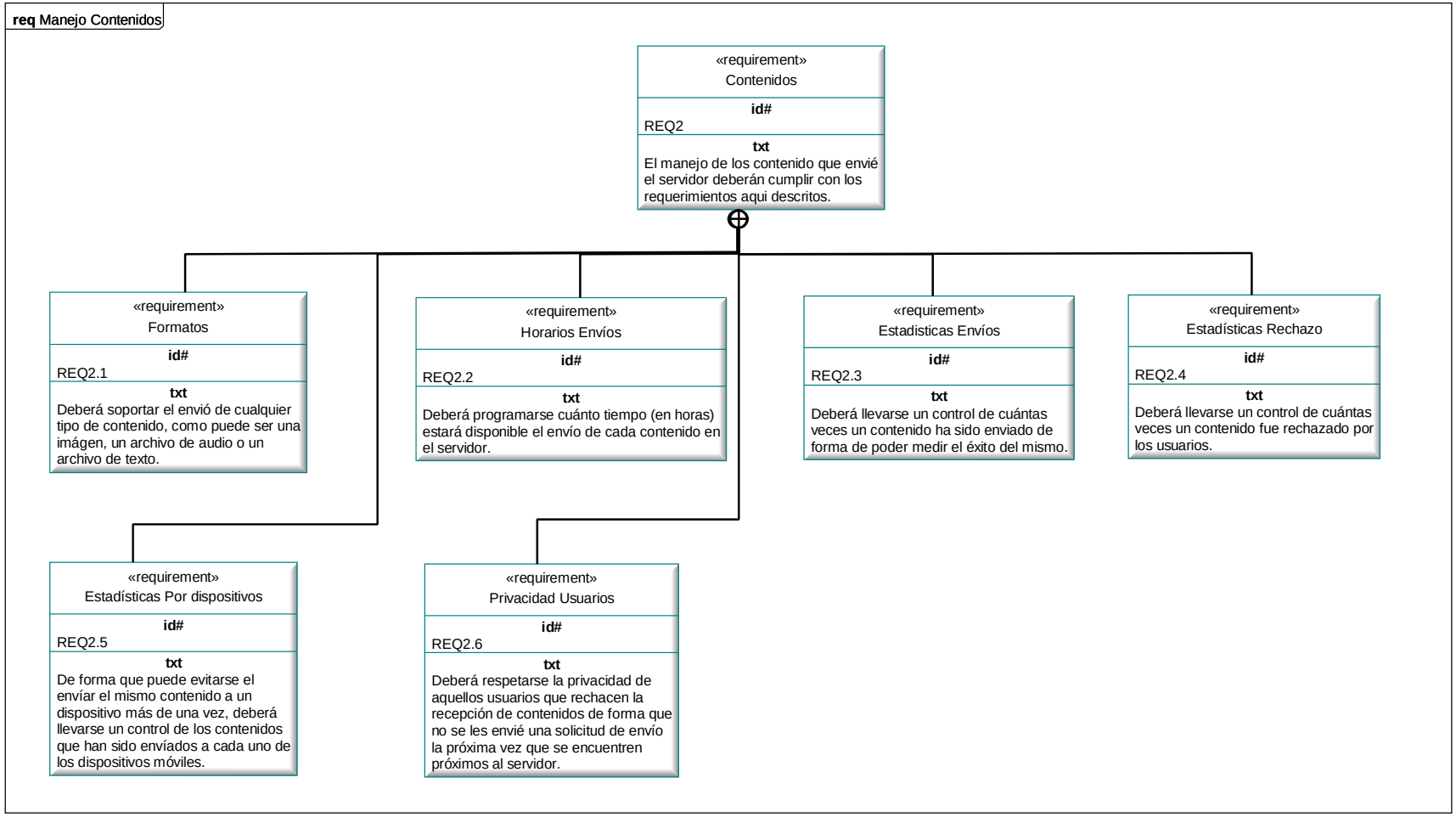
MODELO SERVIDOR ENVÍO CONTENIDOS

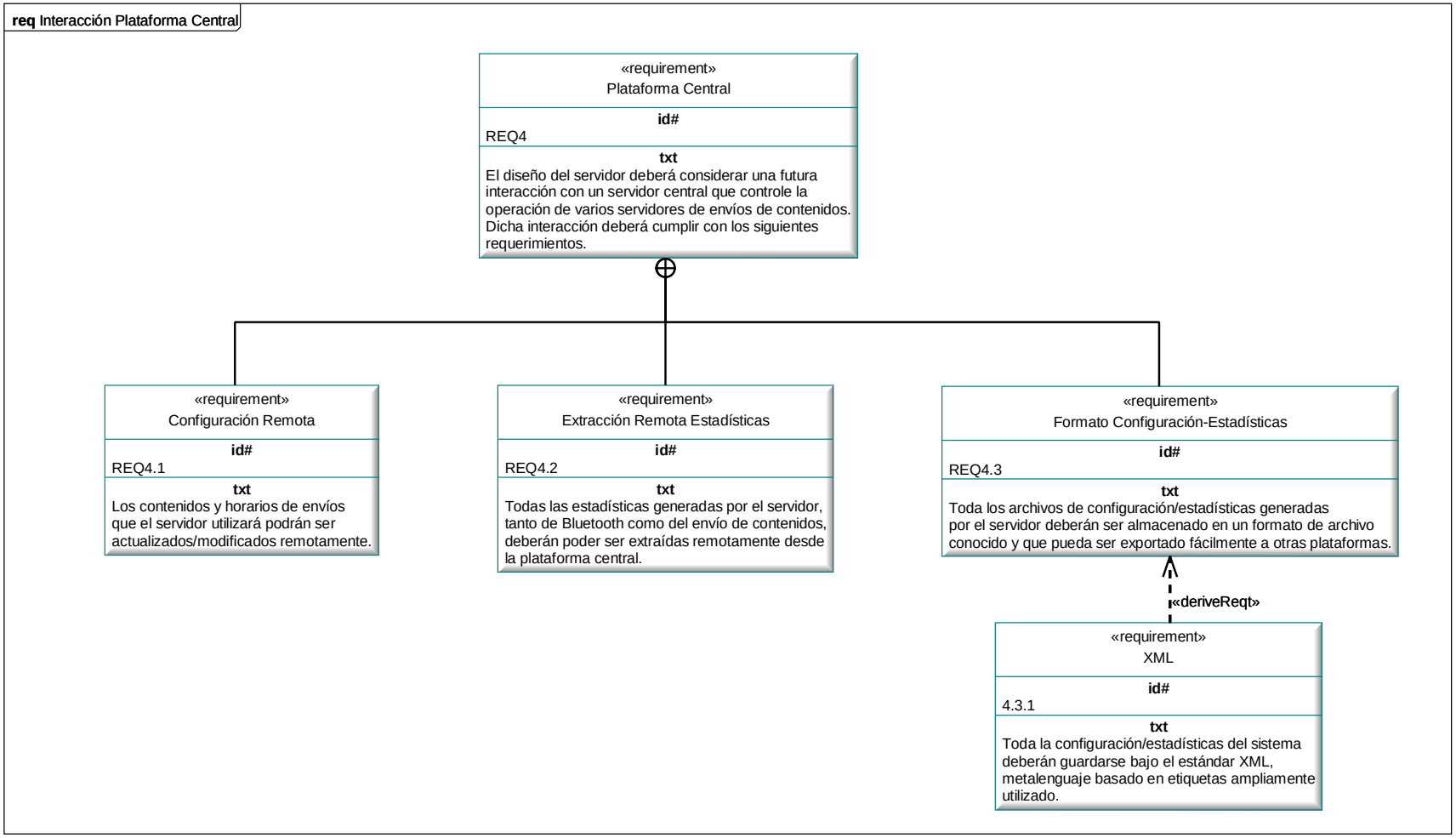
REQUERIMIENTOS

Diagramas de Requerimientos



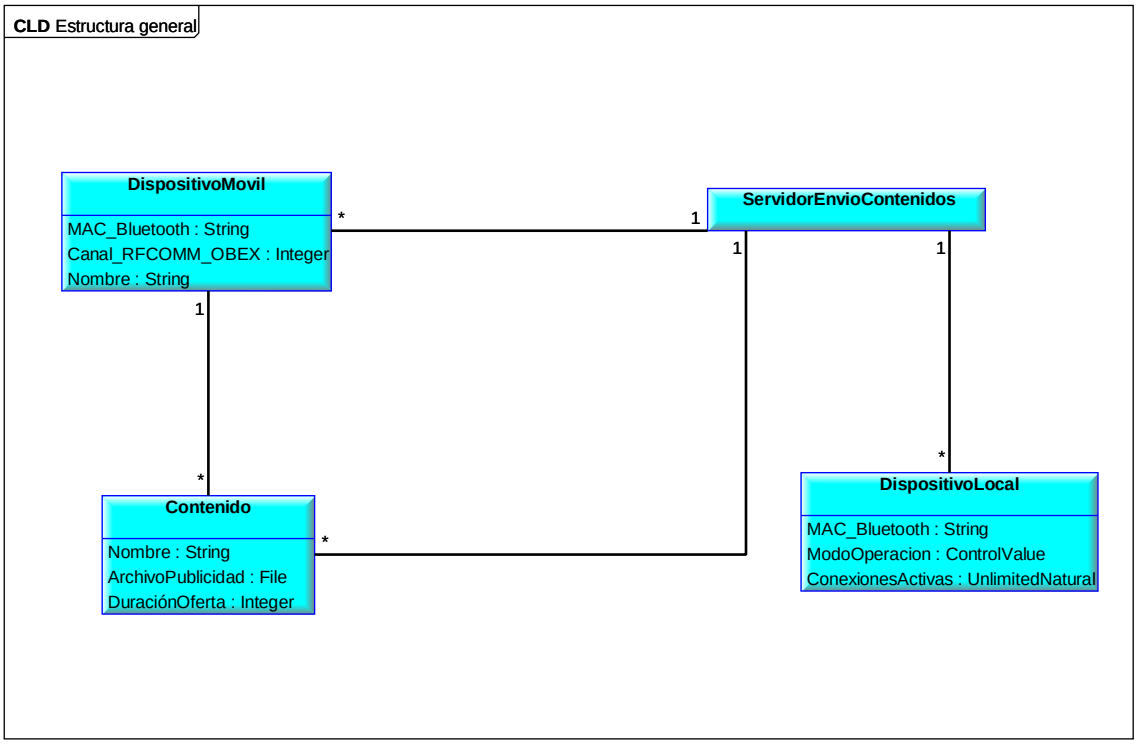

req [Package] Bluetooth
«requirement»
Bluetooth
id#
REQ3
txt
El servidor deberá cumplir con los siguientes requerimientos en lo que a la conectividad Bluetooth se refiere.
«requirement»
Independencia Hardware
id#
REQ3.1
txt
Para establecer comunicación con los dispositivos móviles, el servidor podrá hacer uso de cualquier adaptador bluetooth conectado al servidor a través de un puerto USB.
«deriveReqt»
«requirement»
Múltiples Clases
id#
REQ3.1.1
txt
El servidor podrá usar dispositivos Bluetooth clase 1,2 o 3 indistamente.
«requirement»
Dispositivos Múltiples
id#
REQ3.2
txt
Con el fin de interactuar con la mayor cantidad de dispositivos móviles al mismo tiempo, el servidor podrá tener más de un dispositivo conectado a él y hacer uso simultáneo de ellos.
«requirement»
Tareas Simultáneas
id#
REQ3.3
txt
El servidor podría descubrir nuevos dispositivos móviles y enviar contenidos a otros dispositivos al mismo tiempo.
«requirement»
Caracterización Dispositivos Móviles
id#
REQ3.4
txt
Para agilizar la interacción con los dispositivos móviles una vez descubiertos, el servidor deberá de almacenar los datos de dichos dispositivos como lo es su dirección MAC y el canal RFCOMM asociado al servicio OBEX.
«requirement»
Filtrar Dispositivos
id#
REQ3.5
txt
Debido a que en la actualidad son muchos los equipos que pueden usar comunicación Bluetooth, el servidor deberá poder filtrar aquellos dispositivos que, a pesar de tener comunicación Bluetooth, no puedan recibir contenidos via una transacción OBEX.
«requirement»
Estadísticas Dispositivos Nuevos
id#
REQ3.6
txt
De forma que pueda medirse la afluencia de usuarios, deberá llevarse una estadística, por hora, de la cantidad de dispositivos móviles descubiertos.

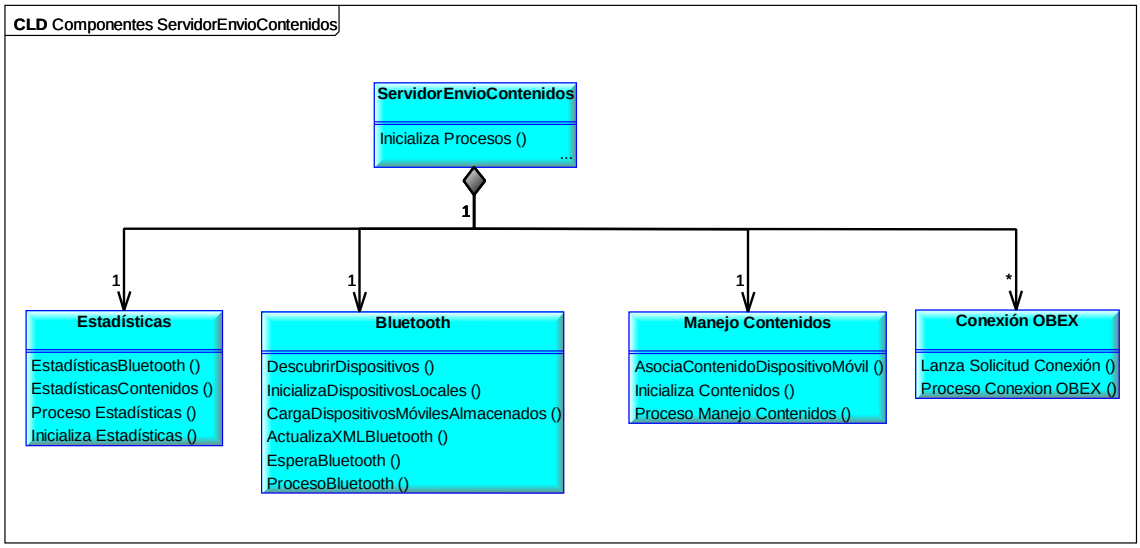

req Manejo Contenidos
«requirement»
Contenidos
id#
REQ2
txt
El manejo de los contenido que envié el servidor deberán cumplir con los requerimientos aqui descritos.
«requirement»
Formatos
id#
REQ2.1
txt
Deberá soportar el envió de cualquier tipo de contenido, como puede ser una imágen, un archivo de audio o un archivo de texto.
«requirement»
Horarios Envíos
id#
REQ2.2
txt
Deberá programarse cuánto tiempo (en horas) estará disponible el envío de cada contenido en el servidor.
«requirement»
Estadisticas Envíos
id#
REQ2.3
txt
Deberá llevarse un control de cuántas veces un contenido ha sido enviado de forma de poder medir el éxito del mismo.
«requirement»
Estadísticas Rechazo
id#
REQ2.4
txt
Deberá llevarse un control de cuántas veces un contenido fue rechazado por los usuarios.
«requirement»
Estadísticas Por dispositivos
id#
REQ2.5
txt
De forma que puede evitarse el envíar el mismo contenido a un dispositivo más de una vez, deberá llevarse un control de los contenidos que han sido envíados a cada uno de los dispositivos móviles.
«requirement»
Privacidad Usuarios
id#
REQ2.6
txt
Deberá respetarse la privacidad de aquellos usuarios que rechacen la recepción de contenidos de forma que no se les envié una solicitud de envío la próxima vez que se encuentren próximos al servidor.

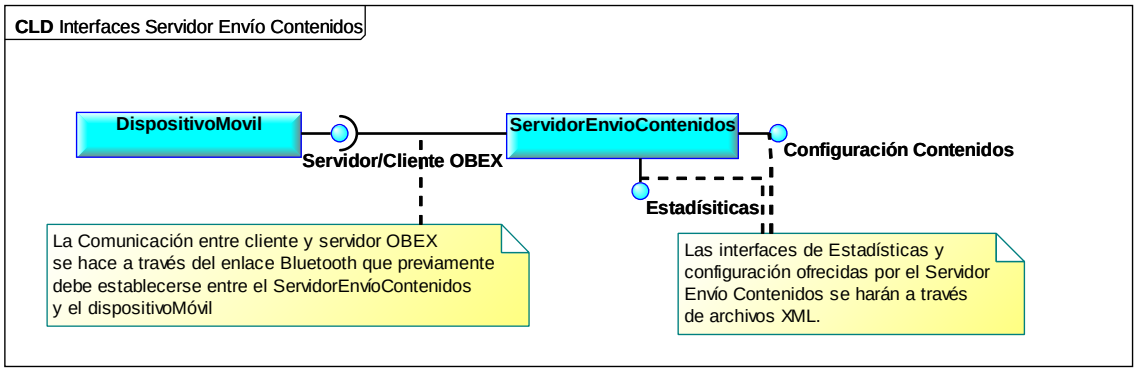

req Interacción Plataforma Central
«requirement»
Plataforma Central
id#
REQ4
txt
El diseño del servidor deberá considerar una futura interacción con un servidor central que controle la operación de varios servidores de envíos de contenidos. Dicha interacción deberá cumplir con los siguientes requerimientos.
«requirement»
Configuración Remota
id#
REQ4.1
txt
Los contenidos y horarios de envíos que el servidor utilizará podrán ser actualizados/modificados remotamente.
«requirement»
Extracción Remota Estadísticas
id#
REQ4.2
txt
Todas las estadísticas generadas por el servidor, tanto de Bluetooth como del envío de contenidos, deberán poder ser extraídas remotamente desde la plataforma central.
«requirement»
Formato Configuración-Estadísticas
id#
REQ4.3
txt
Toda los archivos de configuración/estadísticas generadas por el servidor deberán ser almacenado en un formato de archivo conocido y que pueda ser exportado fácilmente a otras plataformas.
«deriveReqt»
«requirement»
XML
id#
4.3.1
txt
Toda la configuración/estadísticas del sistema deberán guardarse bajo el estándar XML, metalenguaje basado en etiquetas ampliamente utilizado.

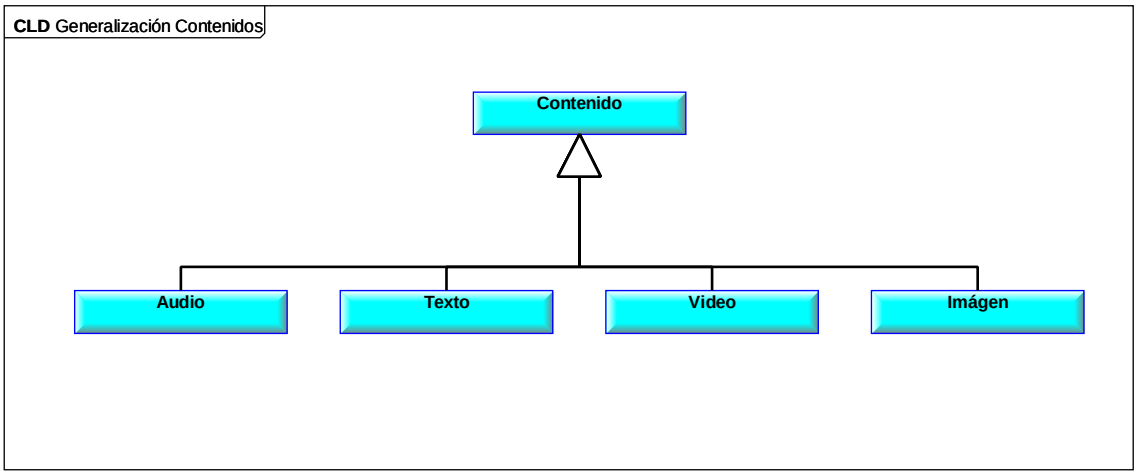
ESTRUCTURA

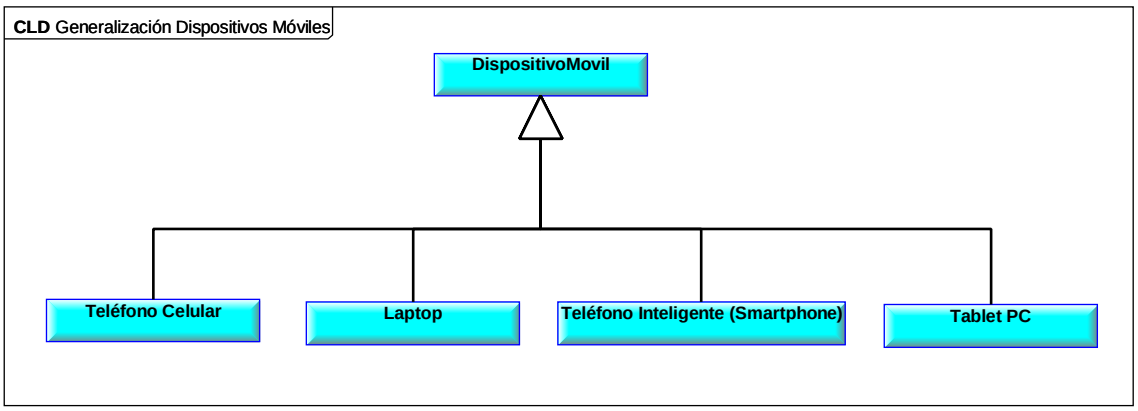
Diagramas de clases





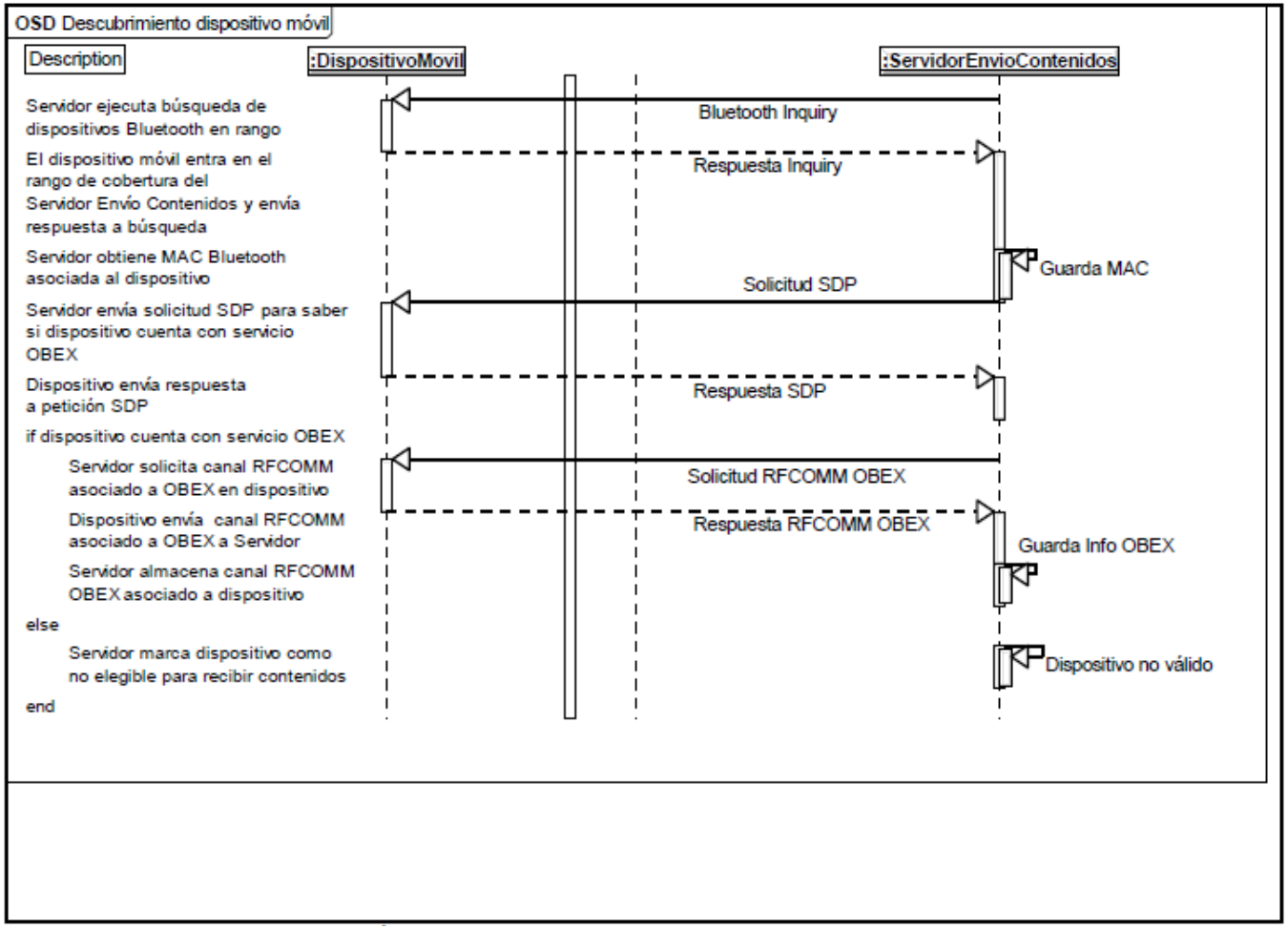

CLD Interfaces Servidor Envío Contenidos
DispositivoMovil
Servidor/Cliente OBEX
ServidorEnvioContenidos
Configuración Contenidos
Estadísiticas
La Comunicación entre cliente y servidor OBEX se hace a través del enlace Bluetooth que previamente debe establecerse entre el ServidorEnvíoContenidos y el dispositivoMóvil
Las interfaces de Estadísticas y configuración ofrecidas por el Servidor Envío Contenidos se harán a través de archivos XML.

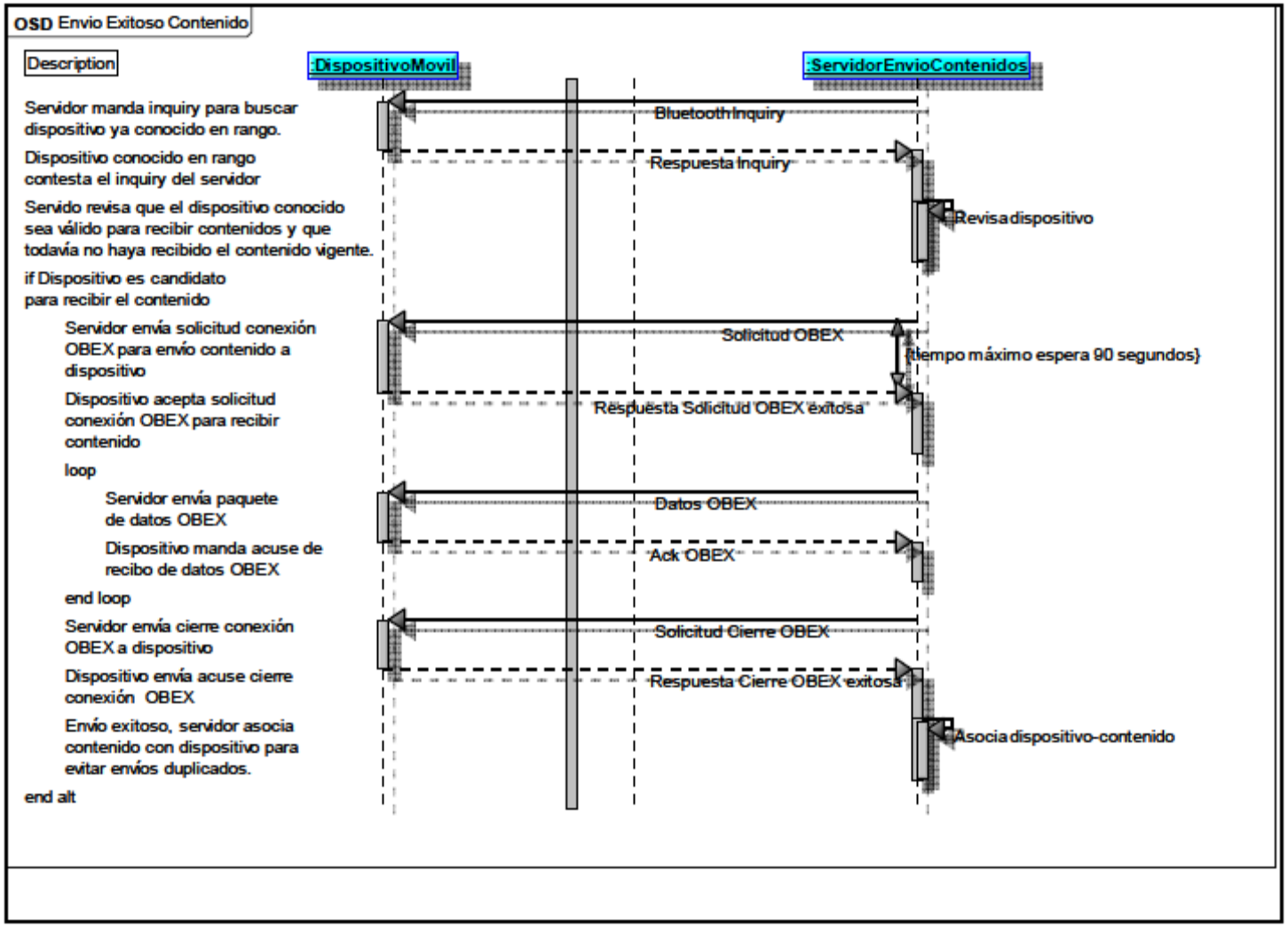

CLD Generalización Contenidos
Contenido
Audio
Texto
Video
Imágen

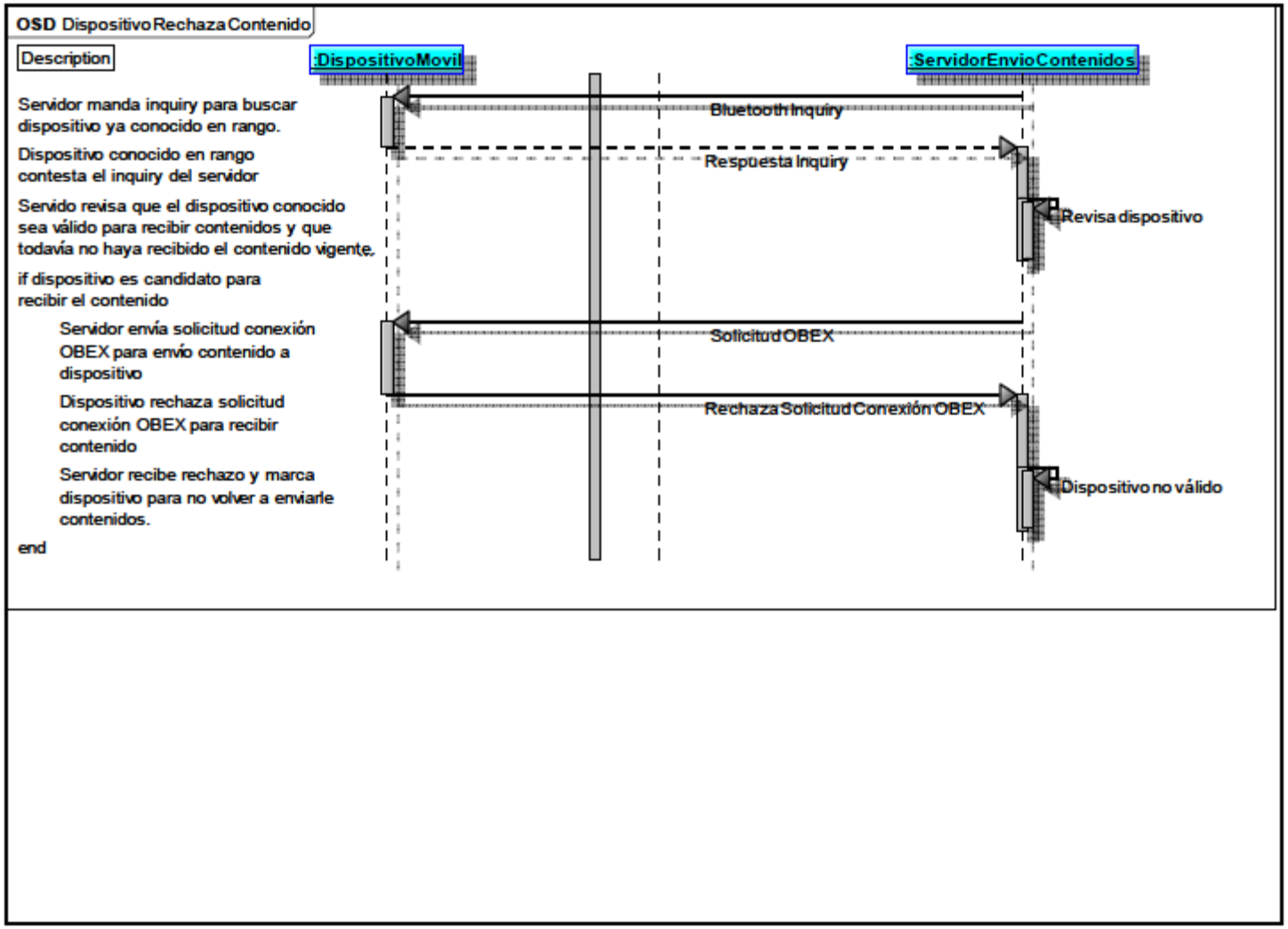

CLD Generalización Dispositivos Móviles
DispositivoMovil
Teléfono Celular
Laptop
Teléfono Inteligente (Smartphone)
Tablet PC

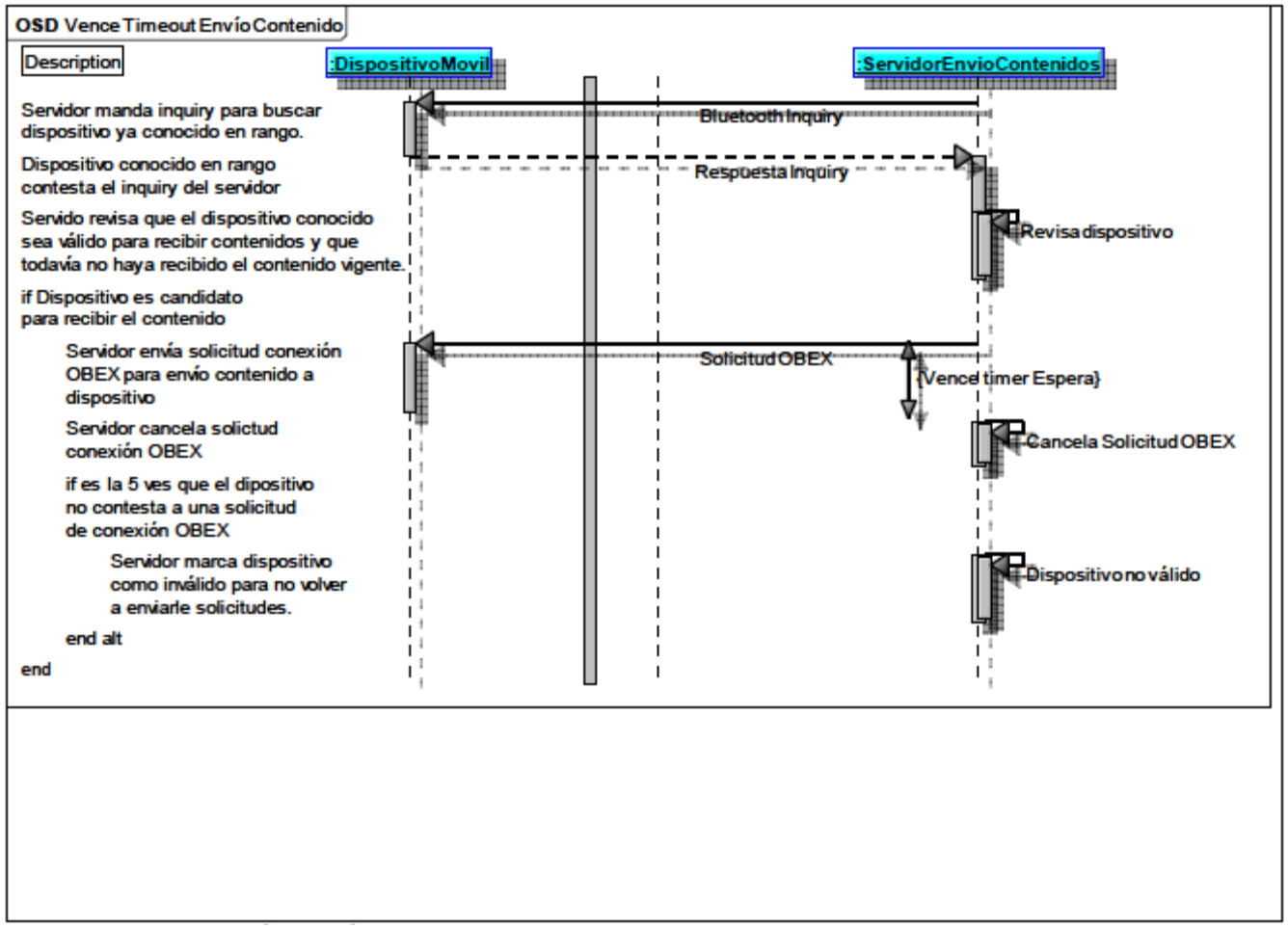
COMPORTAMIENTO

Diagramas de secuencias

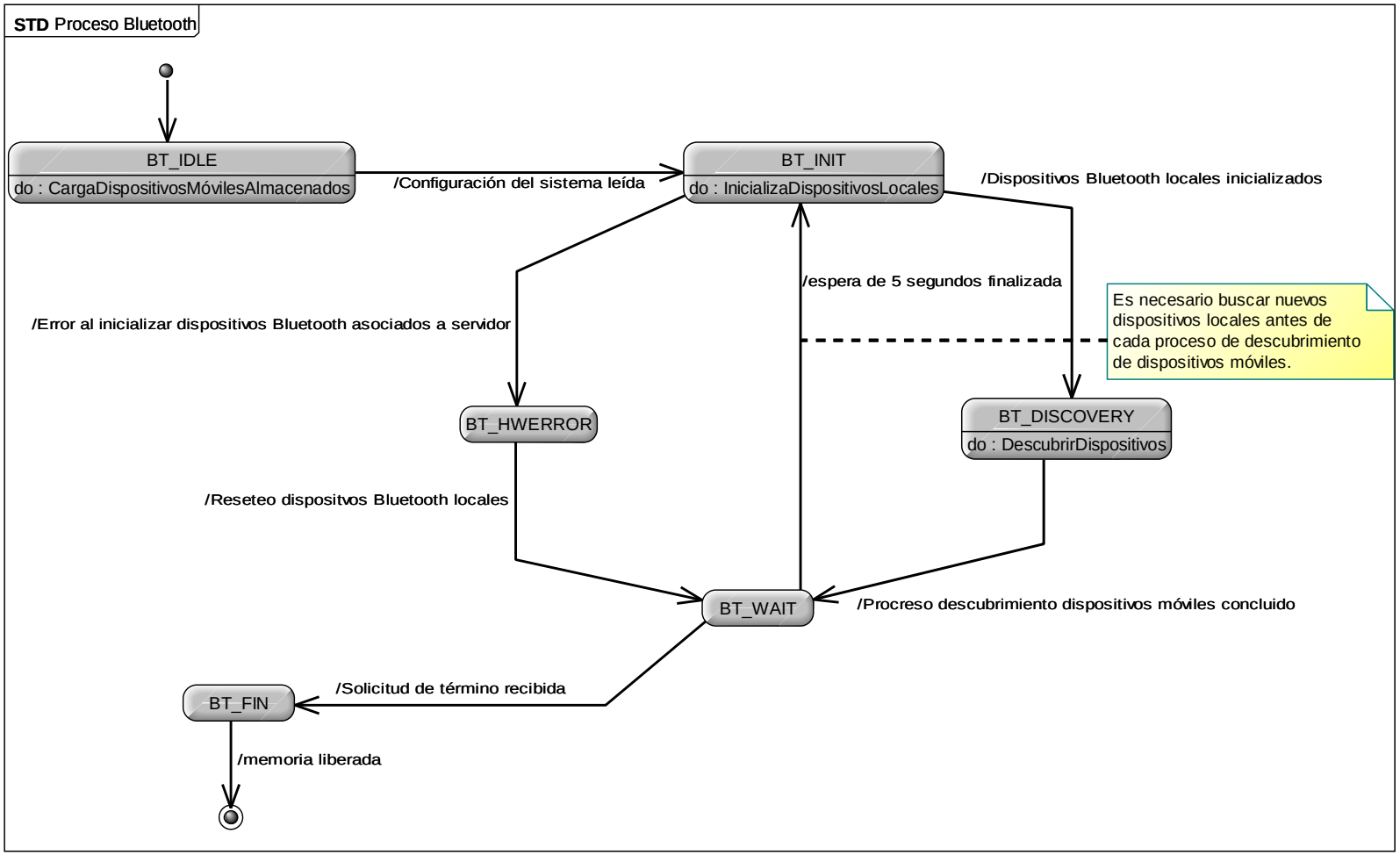


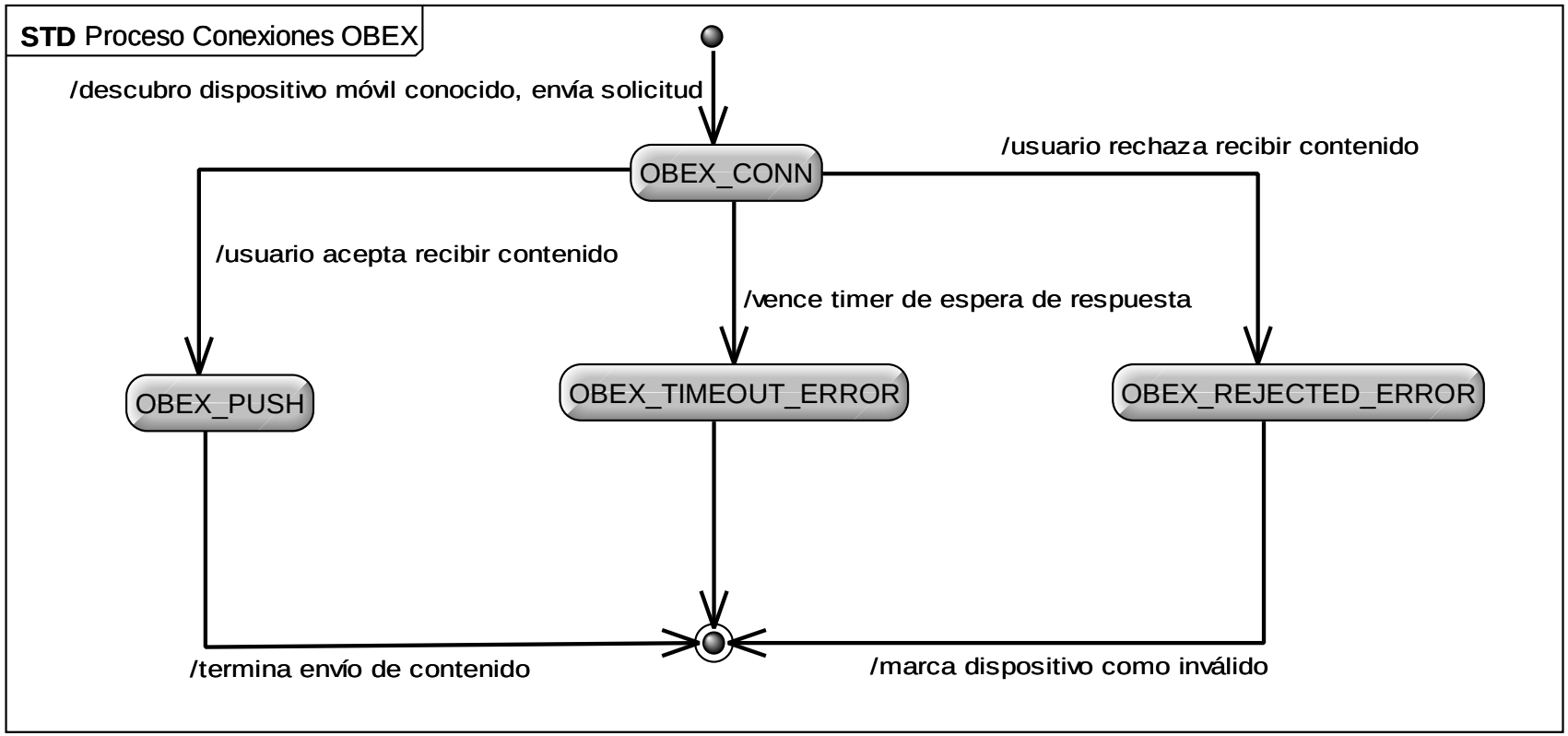

OSD Envio Exitoso Contenido
Description
:DispositivoMovil
:ServidorEnvioContenidos
Servidor manda inquiry para buscar dispositivo ya conocido en rango.
BluetoothInquiry
Dispositivo conocido en rango contesta el inquiry del servidor
Respuesta Inquiry
Servido revisa que el dispositivo conocido sea válido para recibir contenidos y que todavía no haya recibido el contenido vigente.
Revisa dispositivo
if Dispositivo es candidato para recibir el contenido
Servidor envía solicitud conexión OBEX para envío contenido a dispositivo
Solicitud OBEX
{tiempo máximo espera 90 segundos}
Dispositivo acepta solicitud conexión OBEX para recibir contenido
Respuesta Solicitud OBEX exitosa
loop
Servidor envía paquete de datos OBEX
Datos OBEX
Dispositivo manda acuse de recibo de datos OBEX
Ack OBEX
end loop
Servidor envía cierre conexión OBEX a dispositivo
Solicitud Cierre OBEX
Dispositivo envía acuse cierre conexión OBEX
Respuesta Cierre OBEX exitosa
Envío exitoso, servidor asocia contenido con dispositivo para evitar envíos duplicados.
Asocia dispositivo-contenido
end alt

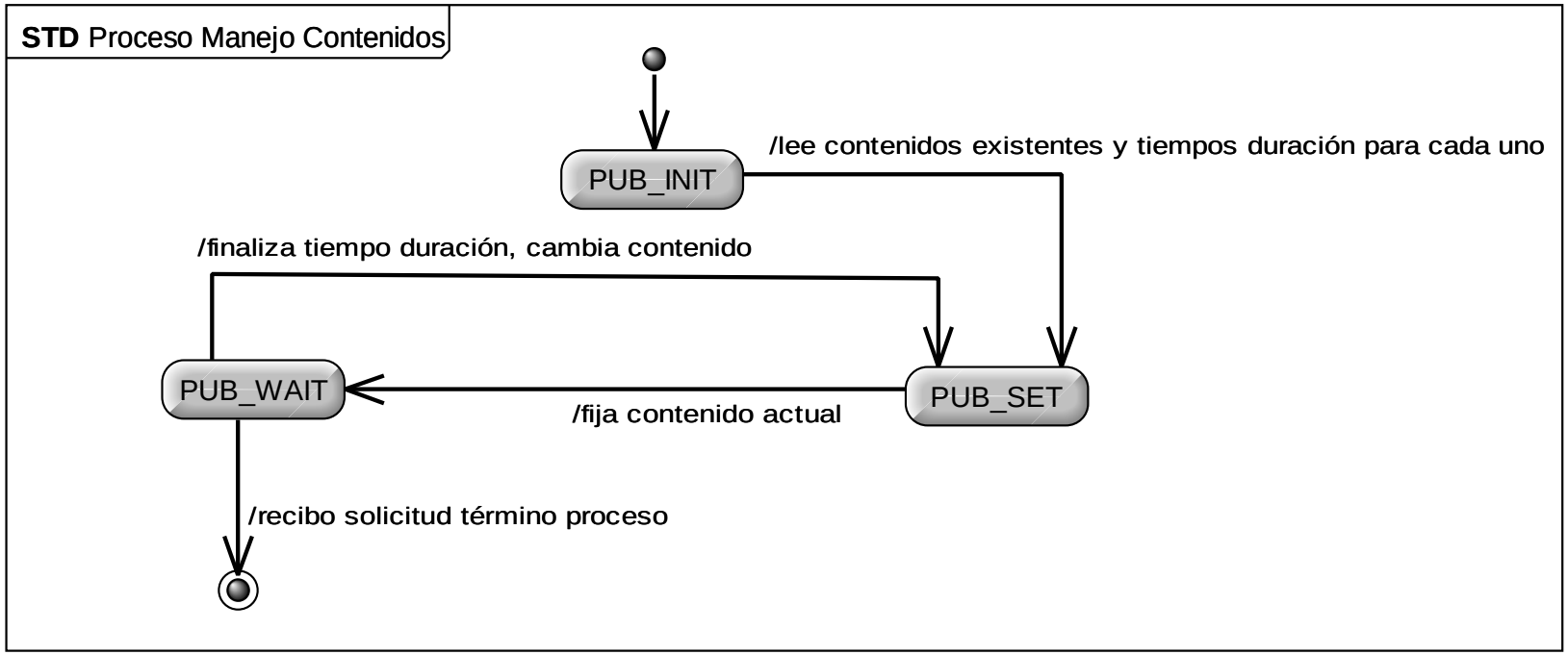

OSD DispositivoRechazaContenido
Description
:DispositivoMovil
:ServidorEnvioContenidos
Servidor manda inquiry para buscar dispositivo ya conocido en rango.
Dispositivo conocido en rango contesta el inquiry del servidor
Servido revisa que el dispositivo conocido sea válido para recibir contenidos y que todavía no haya recibido el contenido vigente.
if dispositivo es candidato para recibir el contenido
Servidor envía solicitud conexión OBEX para envío contenido a dispositivo
Dispositivo rechaza solicitud conexión OBEX para recibir contenido
Servidor recibe rechazo y marca dispositivo para no volver a enviarle contenidos.
end
Bluetooth Inquiry
Respuesta Inquiry
Revisa dispositivo
Solicitud OBEX
Rechaza Solicitud Conexión OBEX
Dispositivo no válido

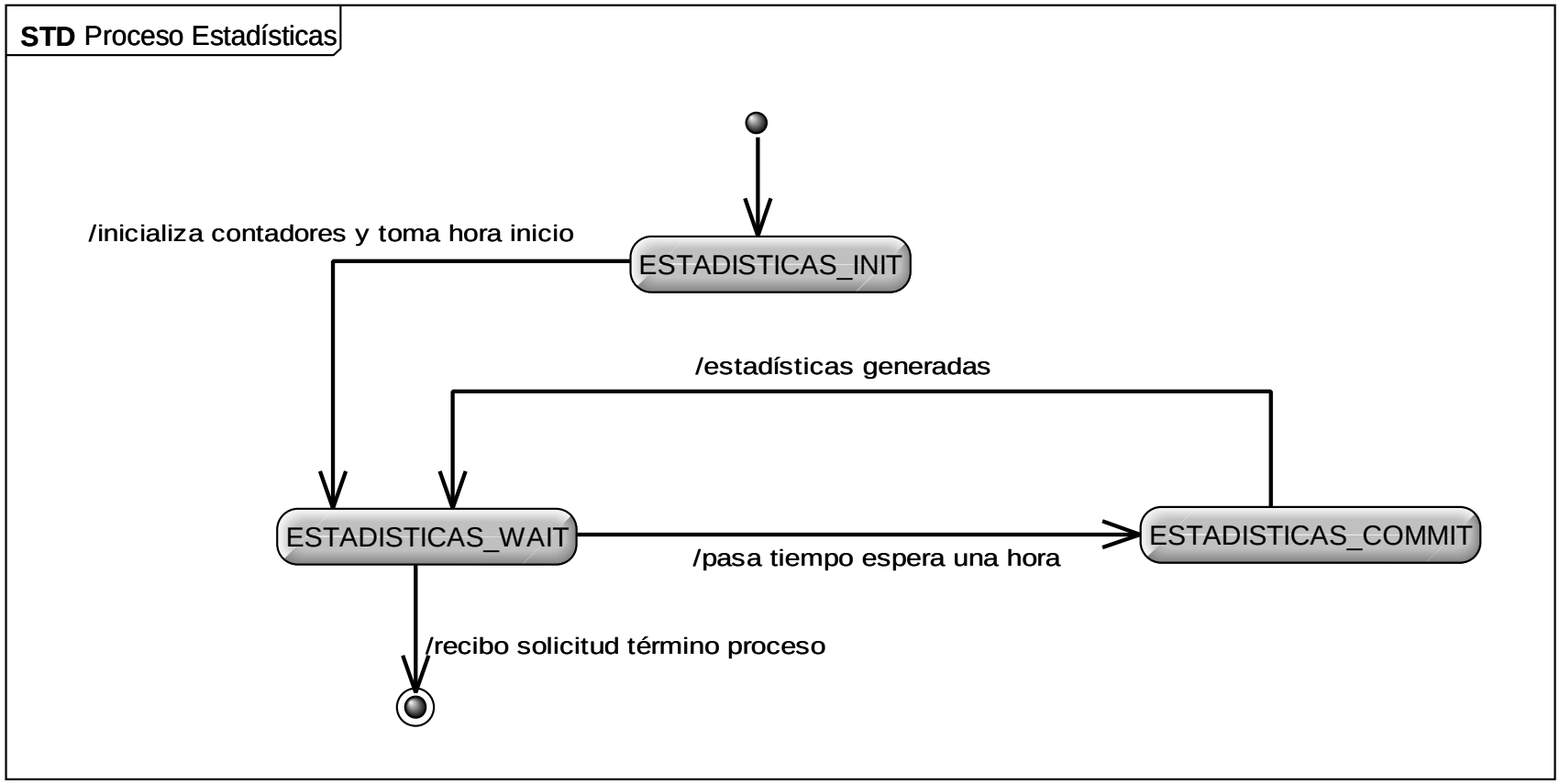

OSD Vence Timeout Envío Contenido
Description
:DispositivoMovil
:ServidorEnvioContenidos
Servidor manda inquiry para buscar dispositivo ya conocido en rango.
Bluetooth Inquiry
Dispositivo conocido en rango contesta el inquiry del servidor
Respuesta Inquiry
Servido revisa que el dispositivo conocido sea válido para recibir contenidos y que todavía no haya recibido el contenido vigente.
Revisa dispositivo
if Dispositivo es candidato para recibir el contenido
Servidor envía solicitud conexión OBEX para envío contenido a dispositivo
Solicitud OBEX
{Vence timer Espera}
Servidor cancela solictud conexión OBEX
Cancela Solicitud OBEX
if es la 5 ves que el dipositivo no contesta a una solicitud de conexión OBEX
Servidor marca dispositivo como inválido para no volver a enviarle solicitudes.
Dispositivo no válido
end alt
end

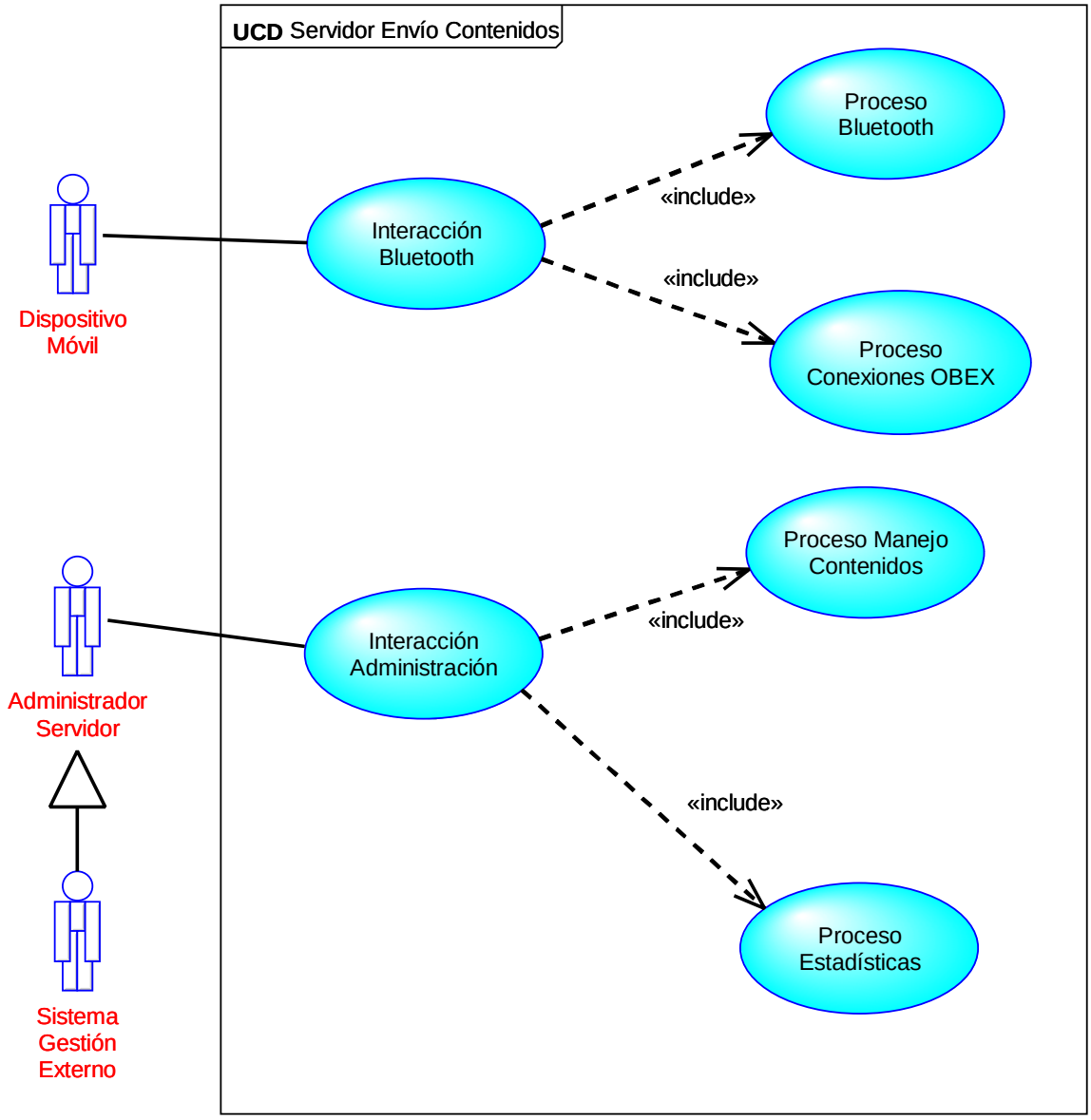
Diagramas de máquinas de estados




STD Proceso Conexiones OBEX
/descubro dispositivo móvil conocido, envía solicitud
OBEX_CONN
/usuario rechaza recibir contenido
/usuario acepta recibir contenido
/vence timer de espera de respuesta
OBEX_PUSH
OBEX_TIMEOUT_ERROR
OBEX_REJECTED_ERROR
/termina envío de contenido
/marca dispositivo como inválido

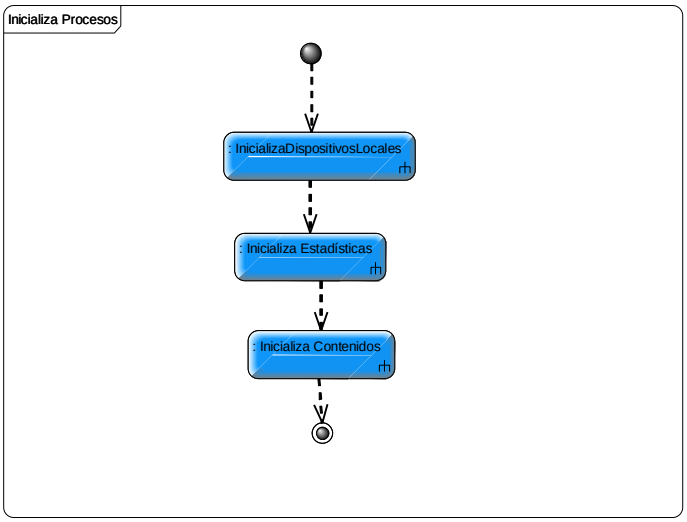

STD Proceso Manejo Contenidos
PUB_INIT
/lee contenidos existentes y tiempos duración para cada uno
/finaliza tiempo duración, cambia contenido
PUB_WAIT
PUB_SET
/fija contenido actual
/recibo solicitud término proceso


STD Proceso Estadísticas
/inicializa contadores y toma hora inicio
ESTADISTICAS_INIT
/estadísticas generadas
ESTADISTICAS_WAIT
ESTADISTICAS_COMMIT
/pasa tiempo espera una hora
/recibo solicitud término proceso

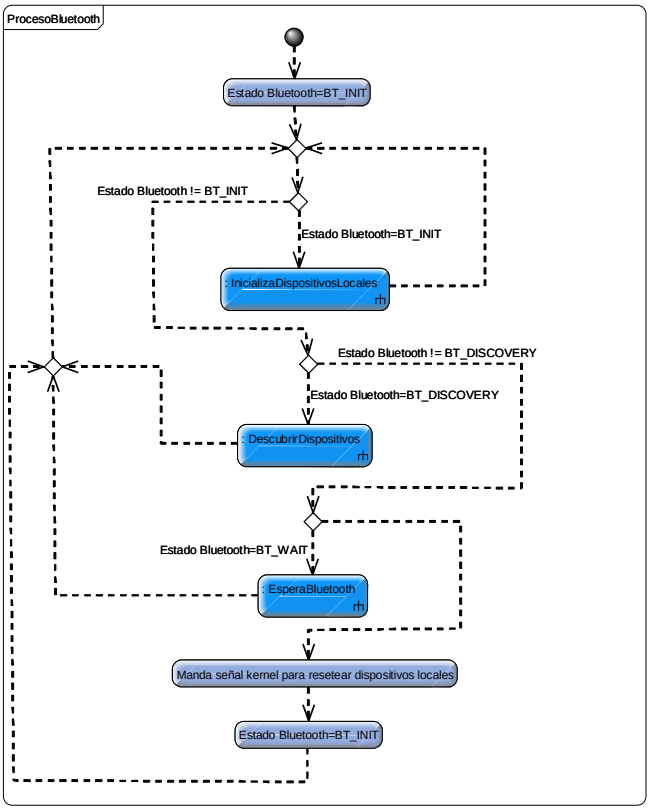

UCD Servidor Envío Contenidos
Proceso Bluetooth
Interacción Bluetooth
«include»
«include»
Proceso Conexiones OBEX
Dispositivo Móvil
Proceso Manejo Contenidos
Interacción Administración
«include»
«include»
Proceso Estadísticas
Administrador Servidor
Sistema Gestión Externo

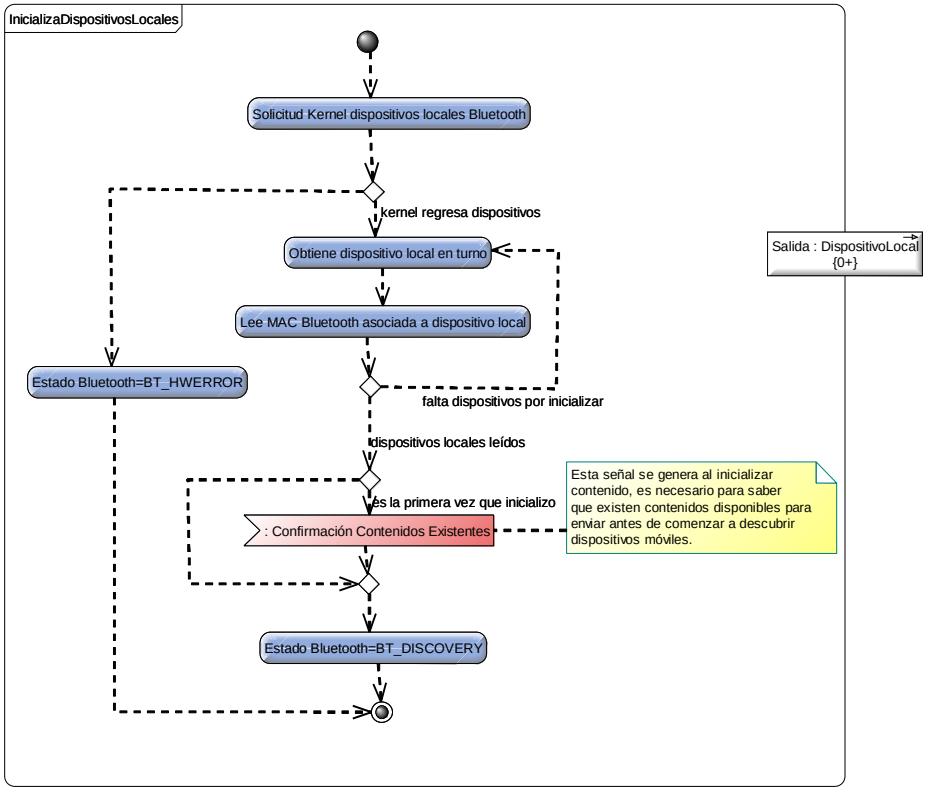
Diagramas de Actividad

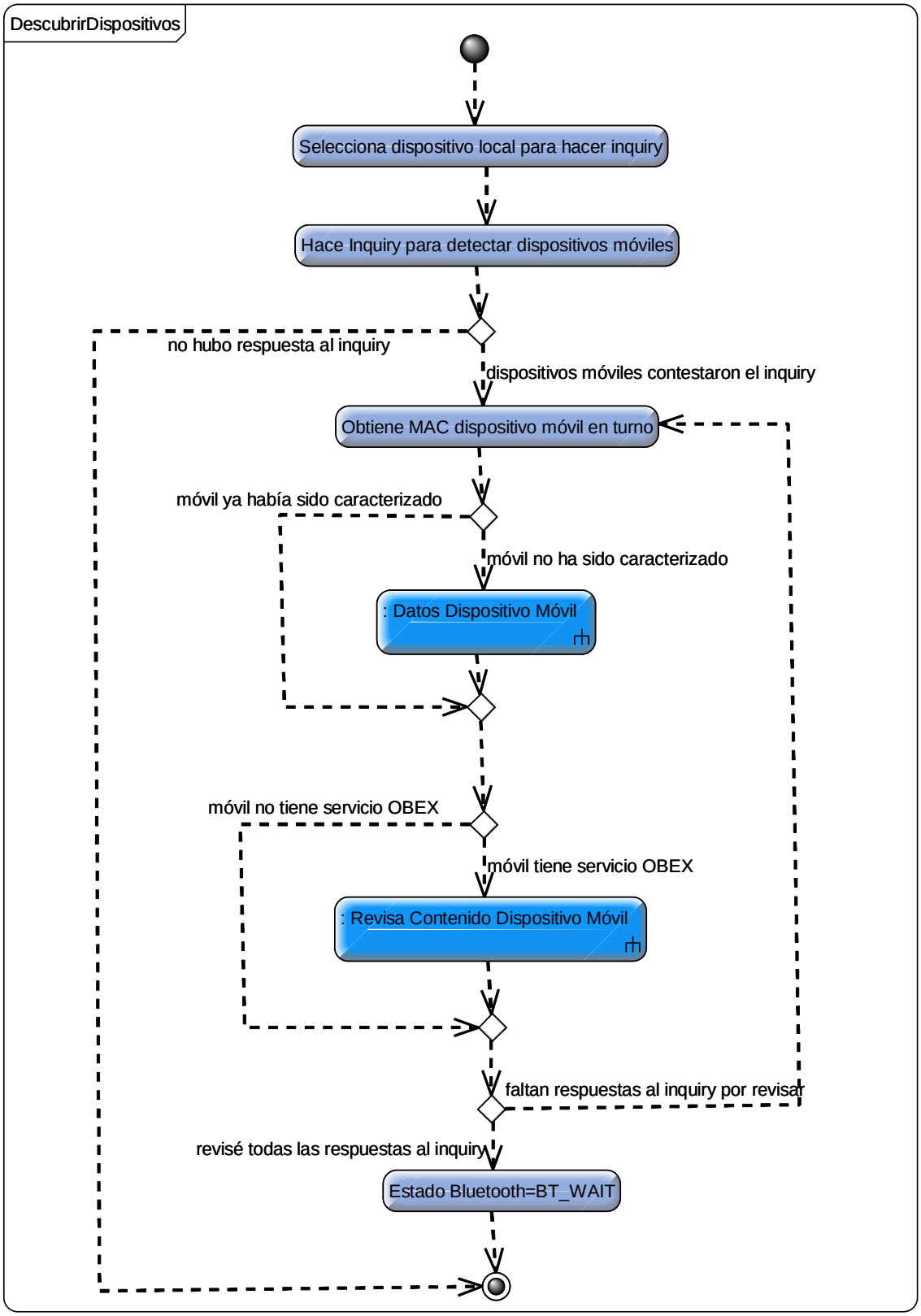
Caso de Uso Interacción Administración

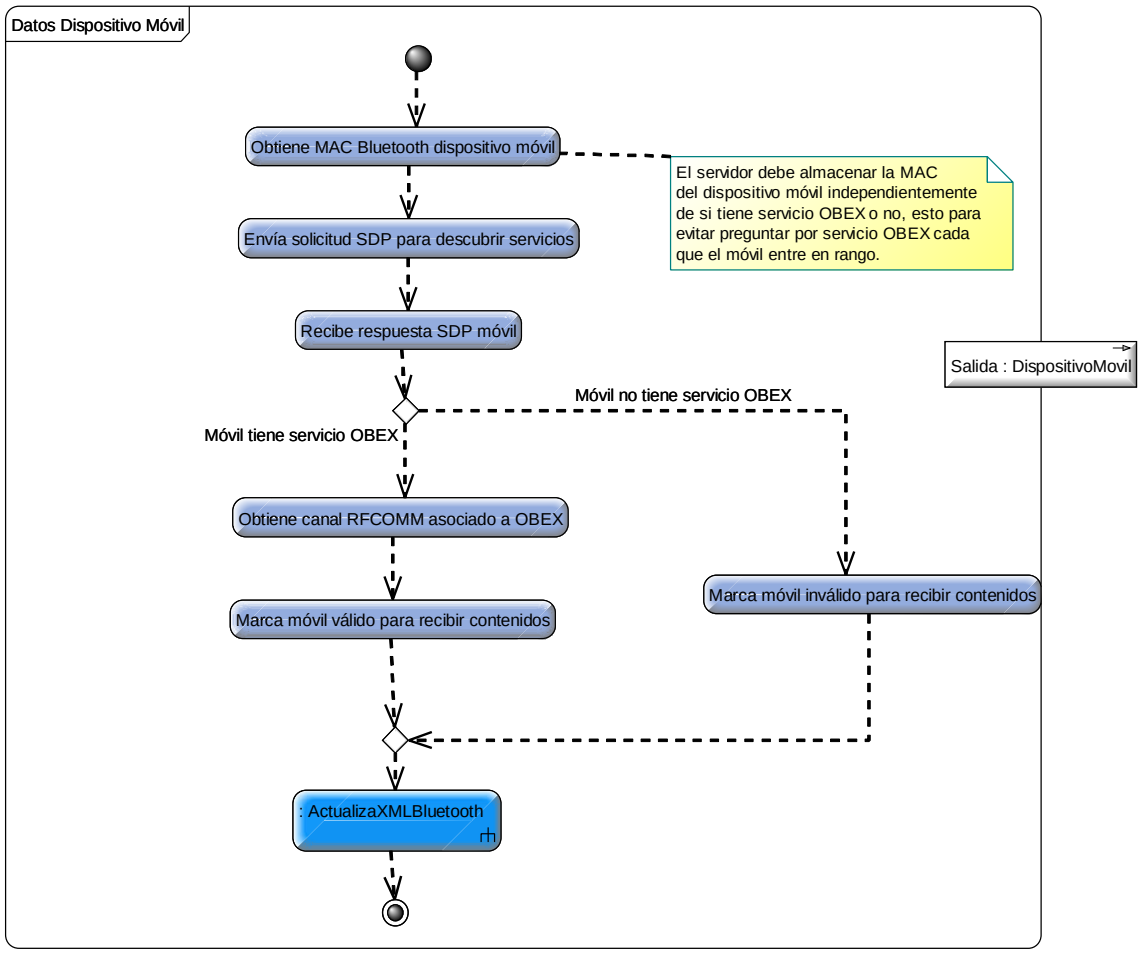


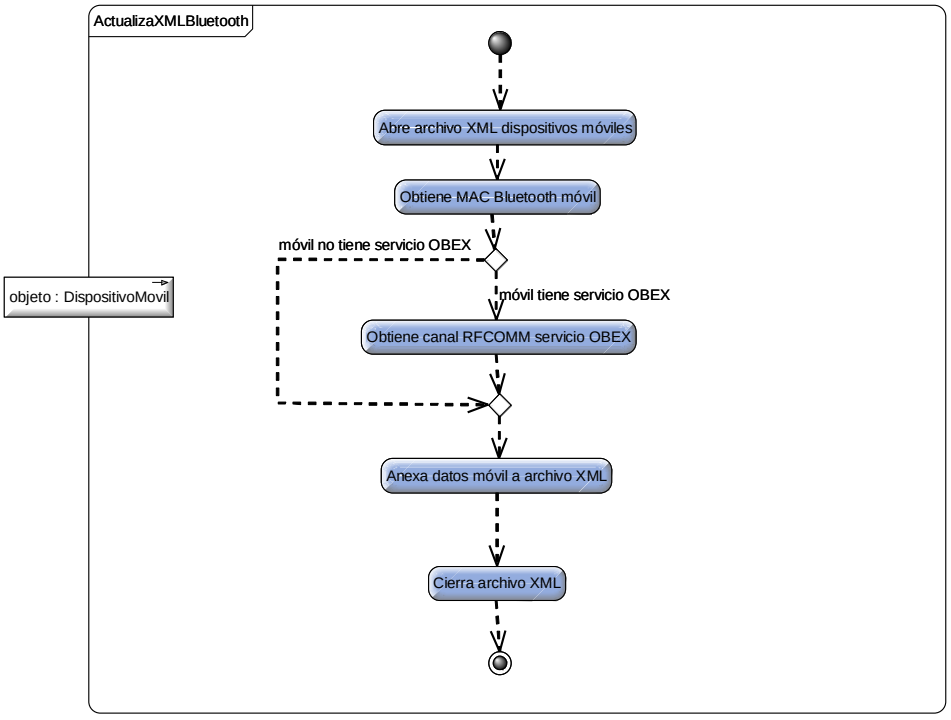
Caso de Uso Proceso Bluetooth

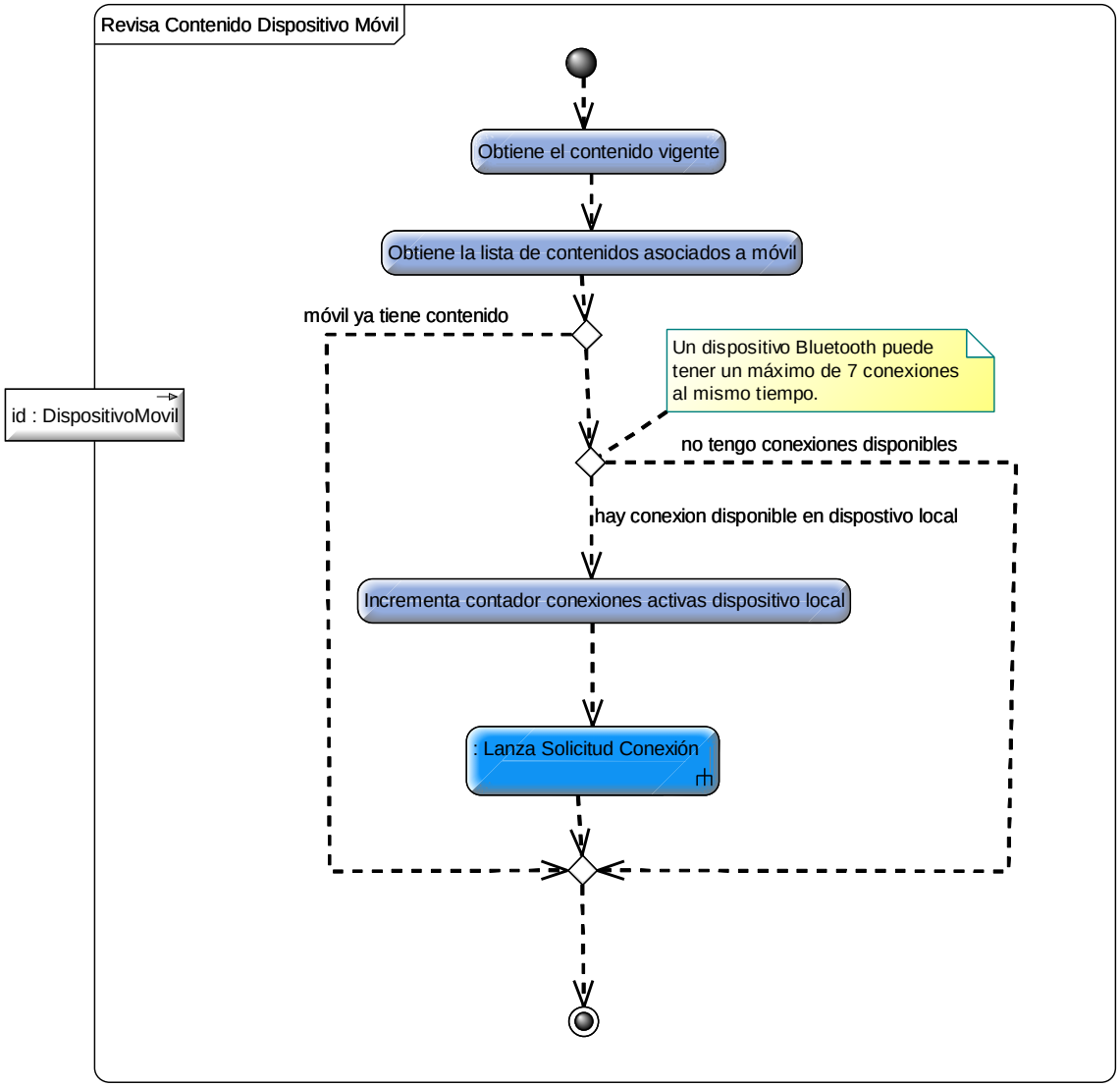


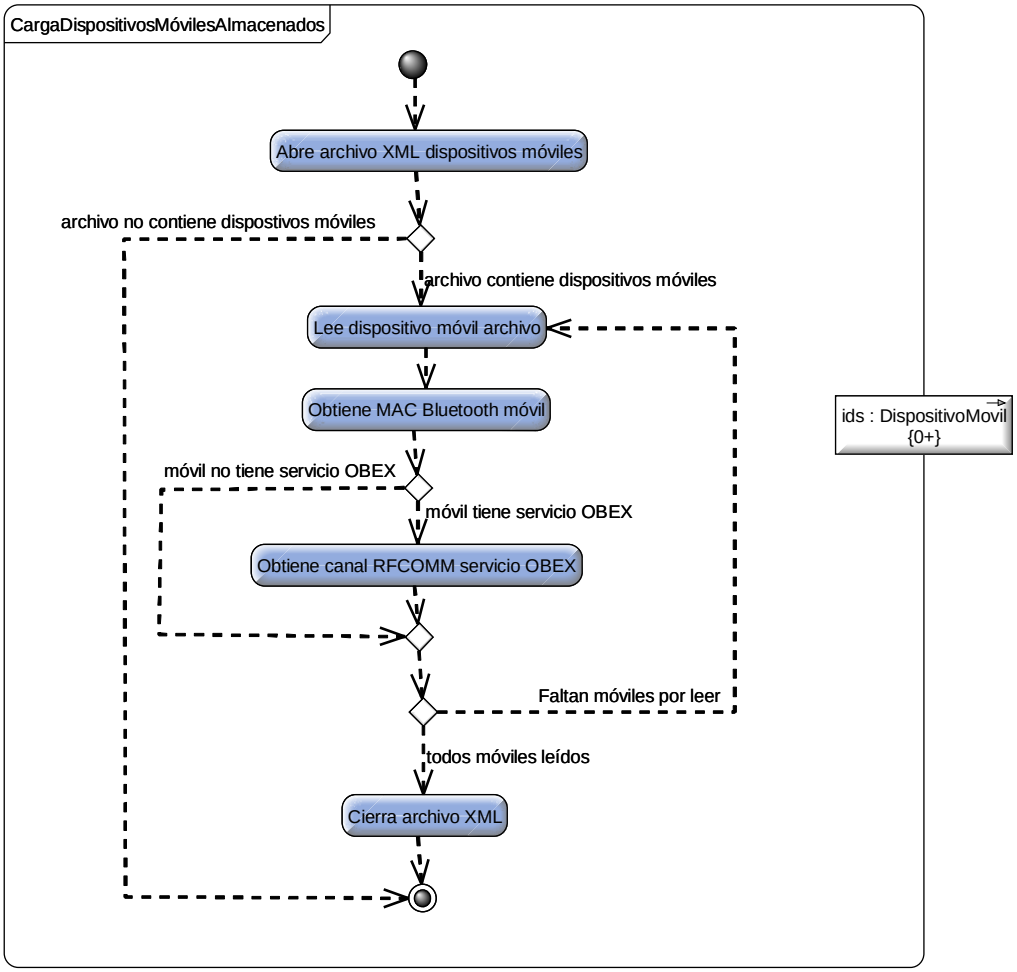

InicializaDispositivosLocales
Solicitud Kernel dispositivos locales Bluetooth
kernel regresa dispositivos
Obtiene dispositivo local en turno
Salida : DispositivoLocal
{0+}
Lee MAC Bluetooth asociada a dispositivo local
Estado Bluetooth=BT_HWERROR
falta dispositivos por inicializar
dispositivos locales leídos
es la primera vez que inicializo
: Confirmación Contenidos Existentes
Esta señal se genera al inicializar contenido, es necesario para saber que existen contenidos disponibles para enviar antes de comenzar a descubrir dispositivos móviles.
Estado Bluetooth=BT_DISCOVERY

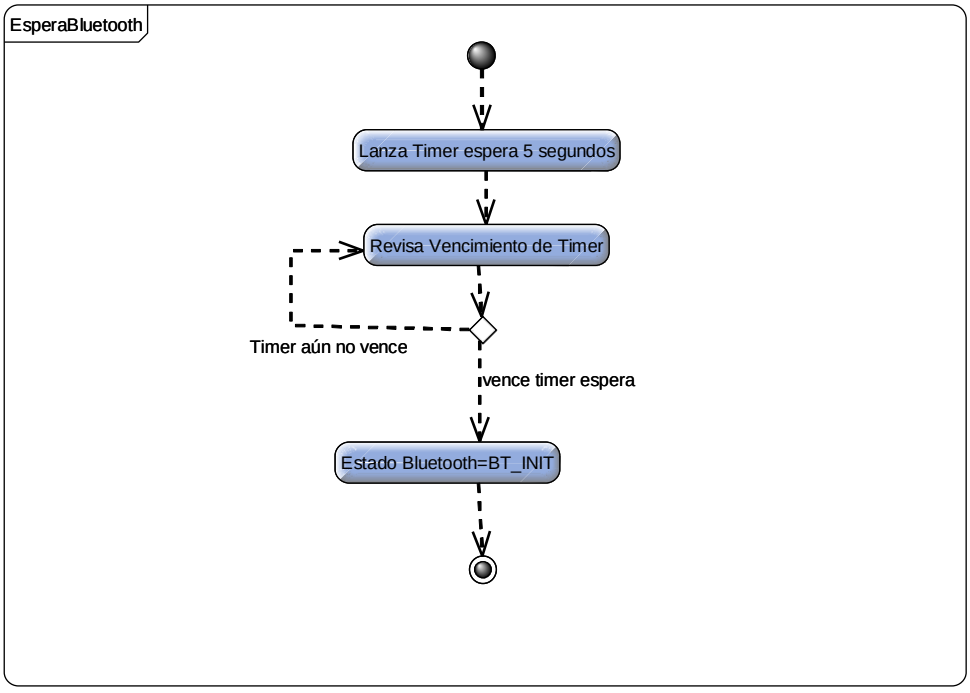

DescubrirDispositivos
Selecciona dispositivo local para hacer inquiry
Hace Inquiry para detectar dispositivos móviles
no hubo respuesta al inquiry
dispositivos móviles contestaron el inquiry
Obtiene MAC dispositivo móvil en turno
móvil ya había sido caracterizado
móvil no ha sido caracterizado
: Datos Dispositivo Móvil
móvil no tiene servicio OBEX
móvil tiene servicio OBEX
: Revisa Contenido Dispositivo Móvil
faltan respuestas al inquiry por revisar
revisé todas las respuestas al inquiry
Estado Bluetooth=BT_WAIT


Datos Dispositivo Móvil
Obtiene MAC Bluetooth dispositivo móvil
El servidor debe almacenar la MAC del dispositivo móvil independientemente de si tiene servicio OBEX o no, esto para evitar preguntar por servicio OBEX cada que el móvil entre en rango.
Envía solicitud SDP para descubrir servicios
Recibe respuesta SDP móvil
Salida : DispositivoMovil
Móvil no tiene servicio OBEX
Móvil tiene servicio OBEX
Obtiene canal RFCOMM asociado a OBEX
Marca móvil inválido para recibir contenidos
Marca móvil válido para recibir contenidos
: ActualizaXMLBluetooth

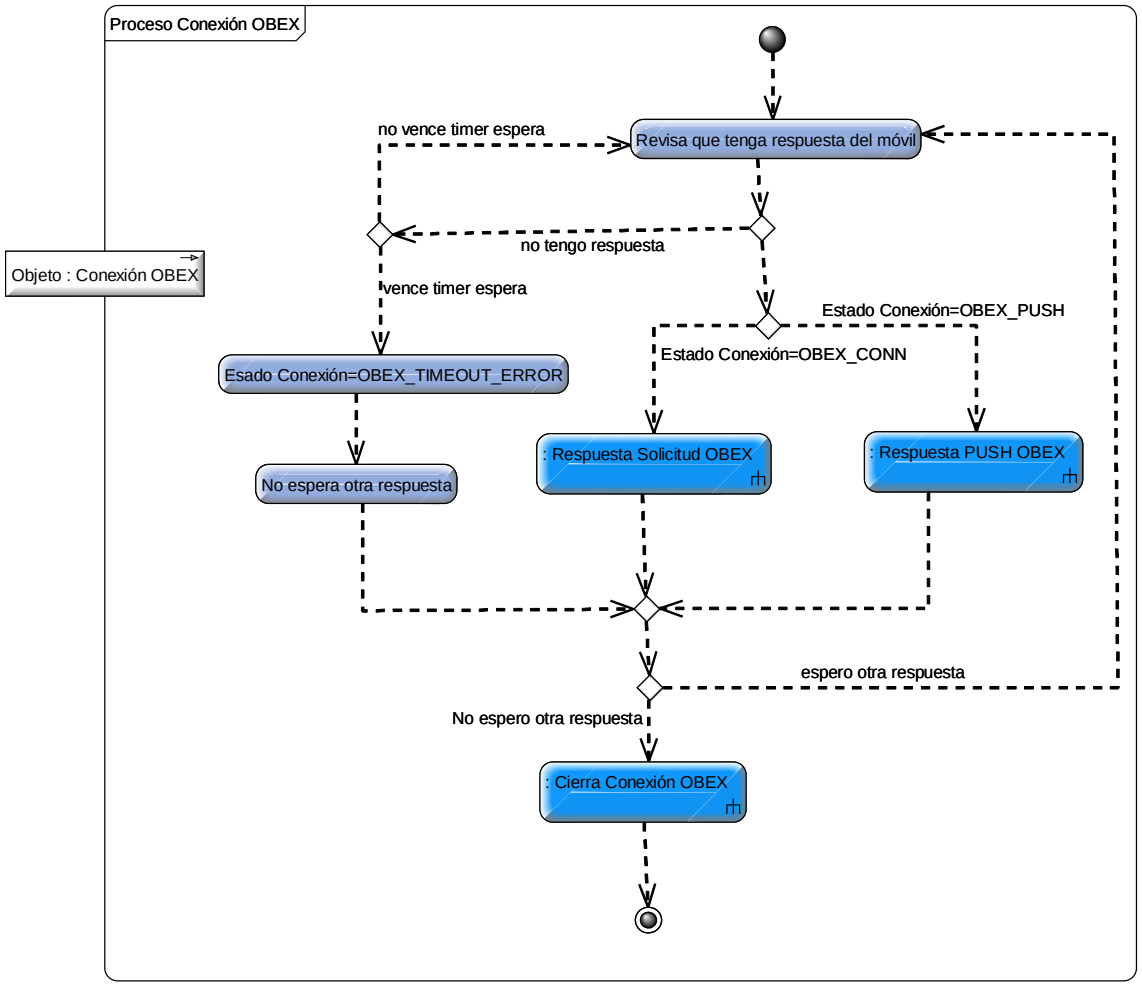

ActualizaXMLBluetooth
Abre archivo XML dispositivos móviles
Obtiene MAC Bluetooth móvil
móvil no tiene servicio OBEX
objeto : DispositivoMovil
móvil tiene servicio OBEX
Obtiene canal RFCOMM servicio OBEX
Anexa datos móvil a archivo XML
Cierra archivo XML

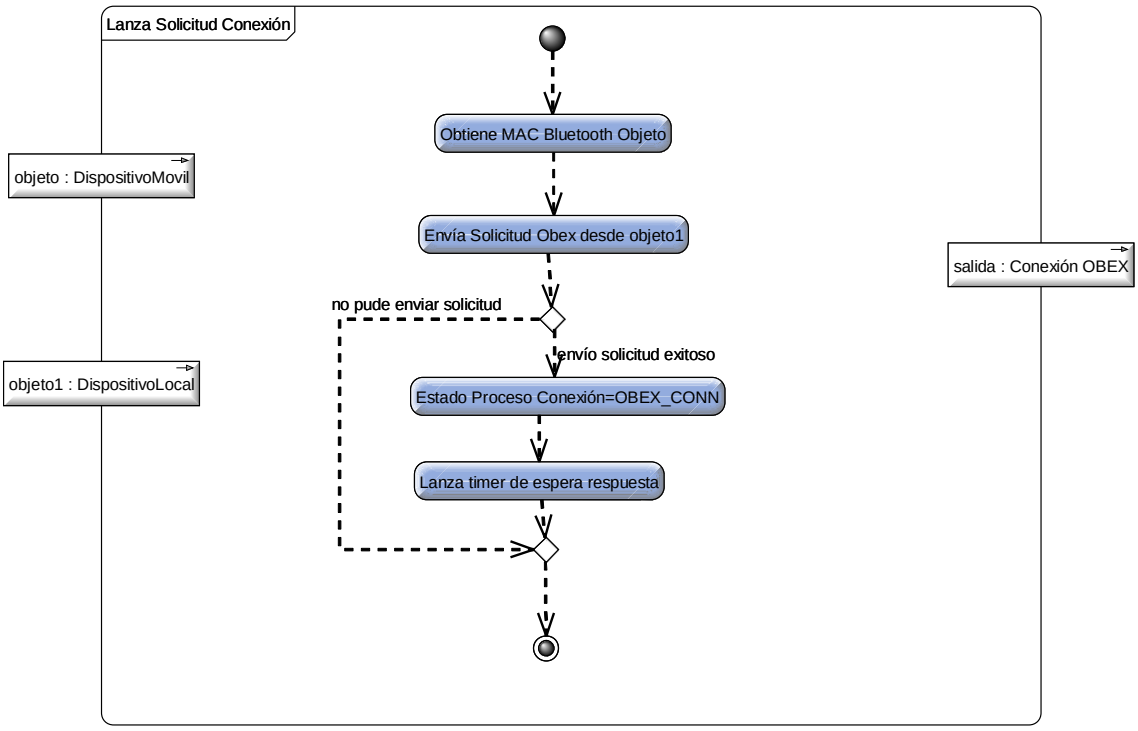

Revisa Contenido Dispositivo Móvil
Obtiene el contenido vigente
Obtiene la lista de contenidos asociados a móvil
móvil ya tiene contenido
Un dispositivo Bluetooth puede tener un máximo de 7 conexiones al mismo tiempo.
id : DispositivoMovil
no tengo conexiones disponibles
hay conexion disponible en dispostivo local
Incrementa contador conexiones activas dispositivo local
: Lanza Solicitud Conexión

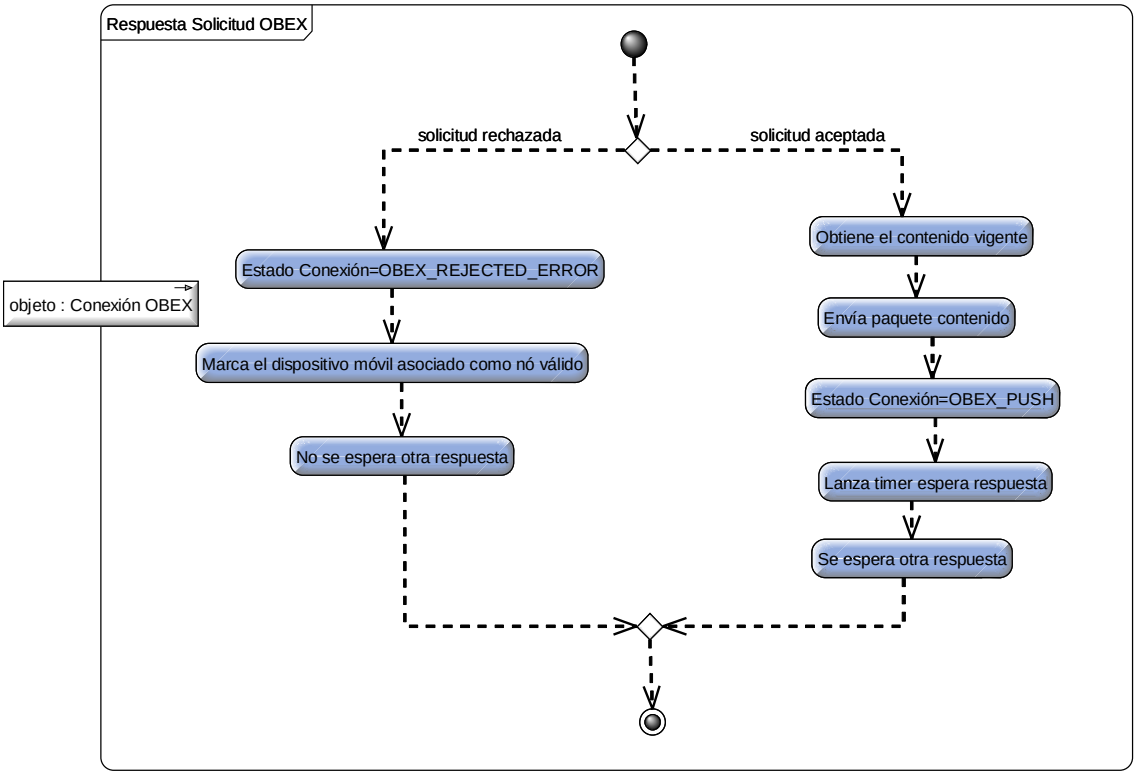

CargaDispositivosMóvilesAlmacenados
Abre archivo XML dispositivos móviles
archivo no contiene dispostivos móviles
archivo contiene dispositivos móviles
Lee dispositivo móvil archivo
Obtiene MAC Bluetooth móvil
móvil no tiene servicio OBEX
móvil tiene servicio OBEX
Obtiene canal RFCOMM servicio OBEX
Faltan móviles por leer
todos móviles leídos
Cierra archivo XML
ids : DispositivoMovil
{0+}

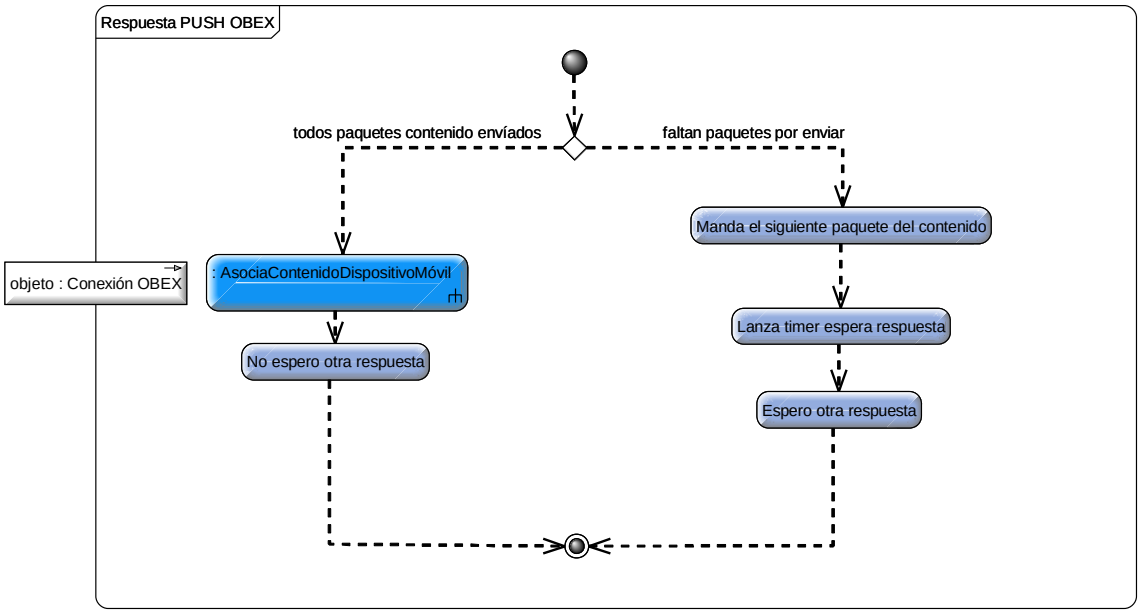

EsperaBluetooth
Lanza Timer espera 5 segundos
Revisa Vencimiento de Timer
Timer aún no vence
vence timer espera
Estado Bluetooth=BT_INIT

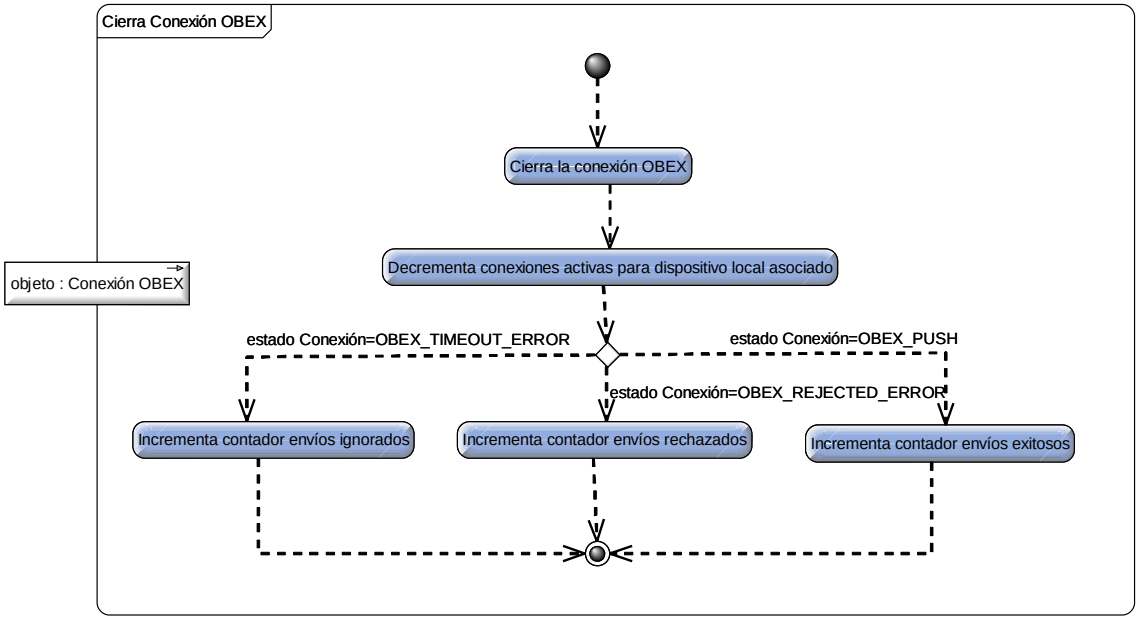
Caso de Uso Proceso Conexiones OBEX



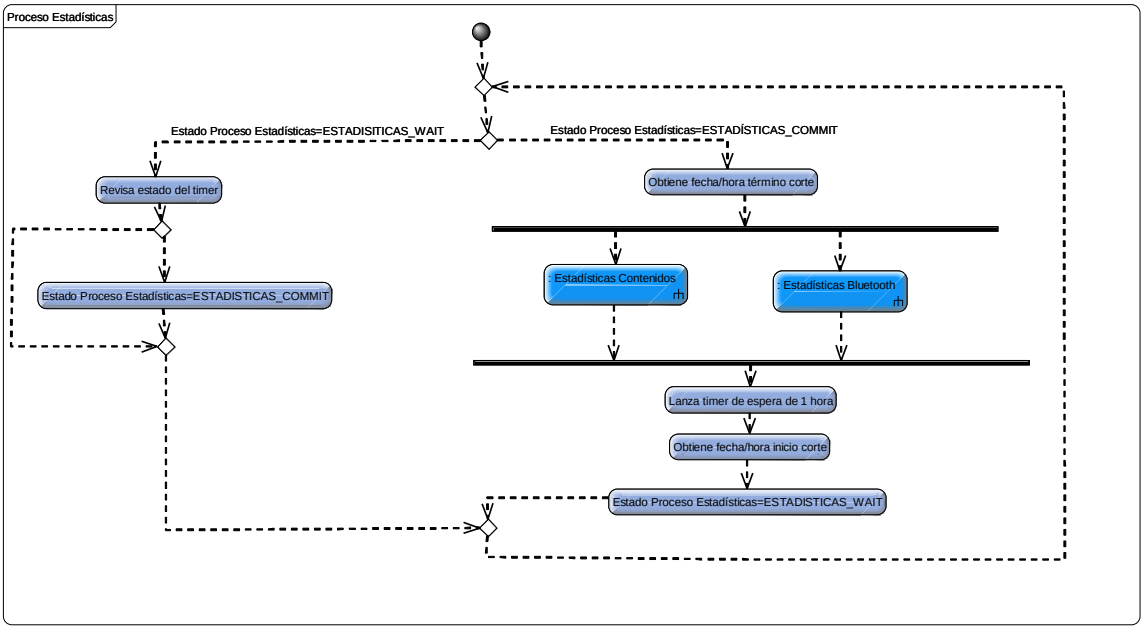

Lanza Solicitud Conexión
objeto : DispositivoMovil
objeto1 : DispositivoLocal
Obtiene MAC Bluetooth Objeto
Envía Solicitud Obex desde objeto1
no pude enviar solicitud
envío solicitud exitoso
Estado Proceso Conexión=OBEX_CONN
Lanza timer de espera respuesta
salida : Conexión OBEX

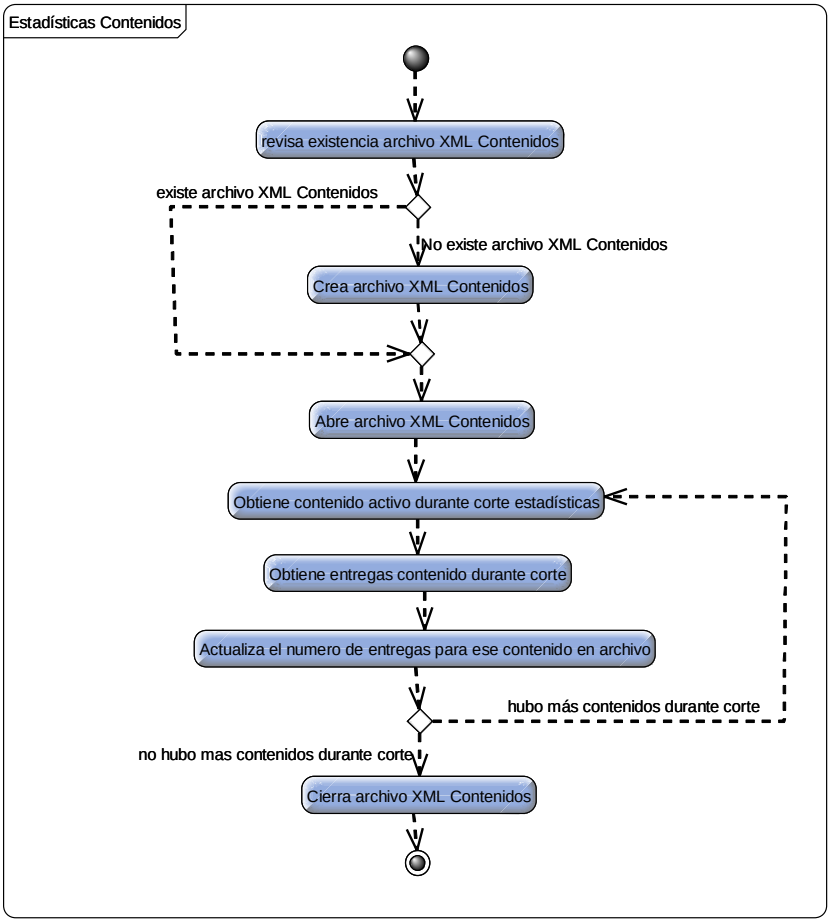

Respuesta Solicitud OBEX
objeto : Conexión OBEX
solicitud rechazada
solicitud aceptada
Estado Conexión=OBEX_REJECTED_ERROR
Marca el dispositivo móvil asociado como nó válido
No se espera otra respuesta
Obtiene el contenido vigente
Envía paquete contenido
Estado Conexión=OBEX_PUSH
Lanza timer espera respuesta
Se espera otra respuesta

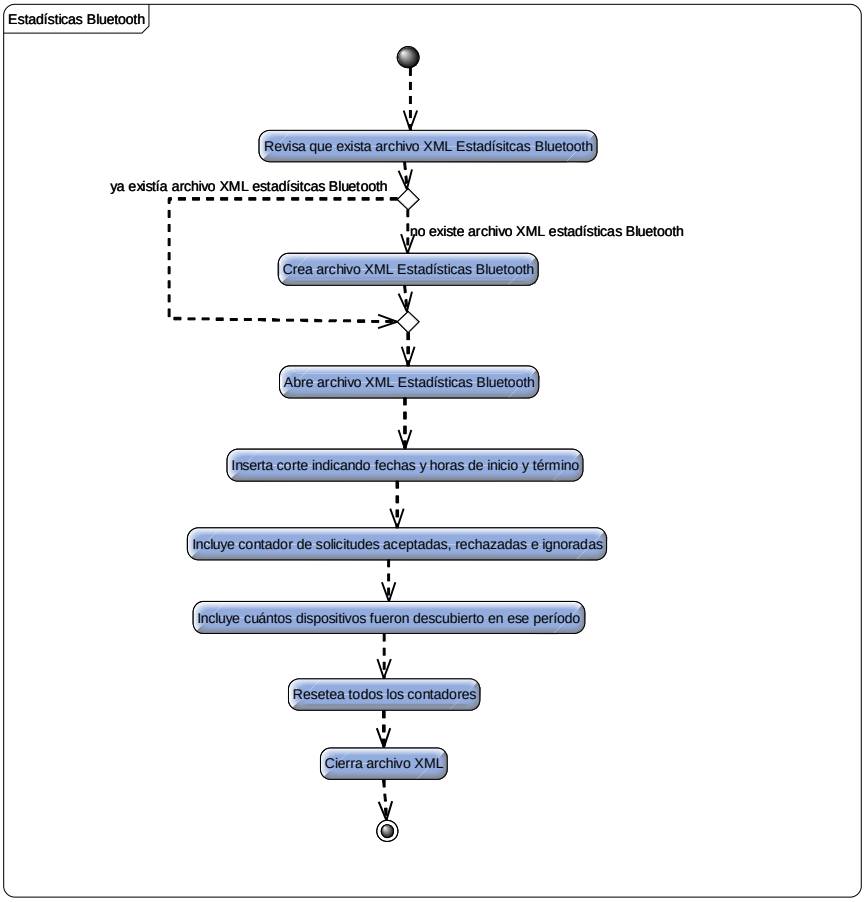

Respuesta PUSH OBEX
todos paquetes contenido envíados
faltan paquetes por enviar
objeto : Conexión OBEX
: AsociaContenidoDispositivoMóvil
rm
Manda el siguiente paquete del contenido
Lanza timer espera respuesta
No espero otra respuesta
Espero otra respuesta

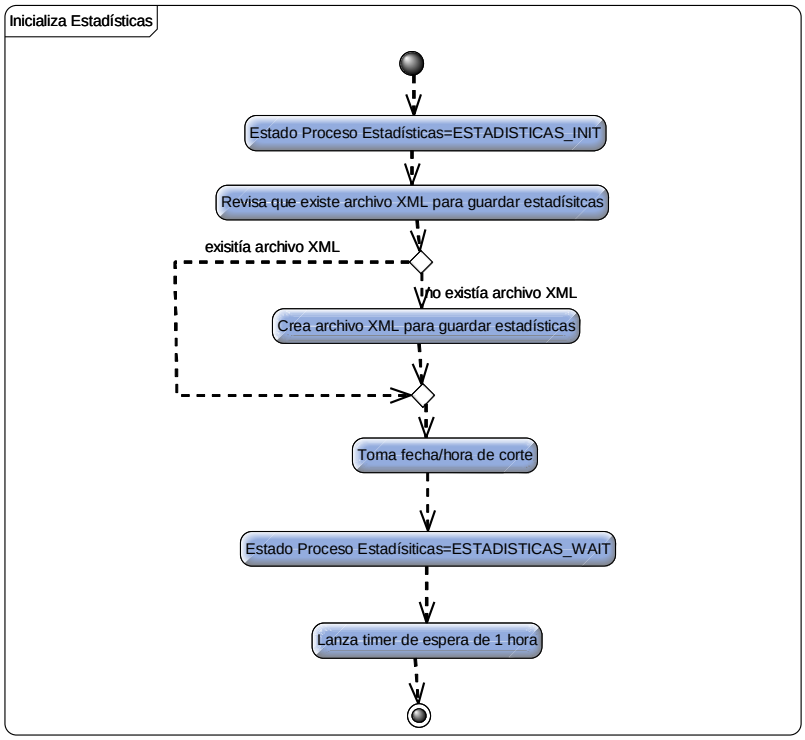

Cierra Conexión OBEX
Cierra la conexión OBEX
objeto : Conexión OBEX
Decrementa conexiones activas para dispositivo local asociado
estado Conexión=OBEX_TIMEOUT_ERROR
estado Conexión=OBEX_PUSH
estado Conexión=OBEX_REJECTED_ERROR
Incrementa contador envíos ignorados
Incrementa contador envíos rechazados
Incrementa contador envíos exitosos

Caso de Uso Proceso Estadísticas

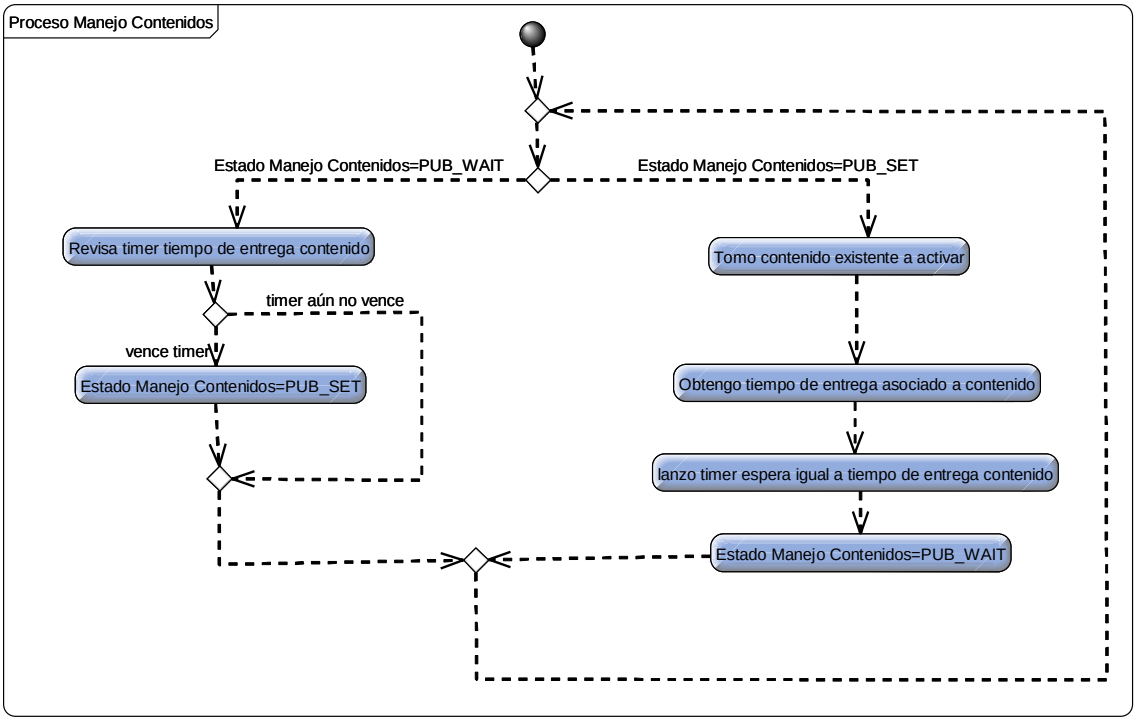


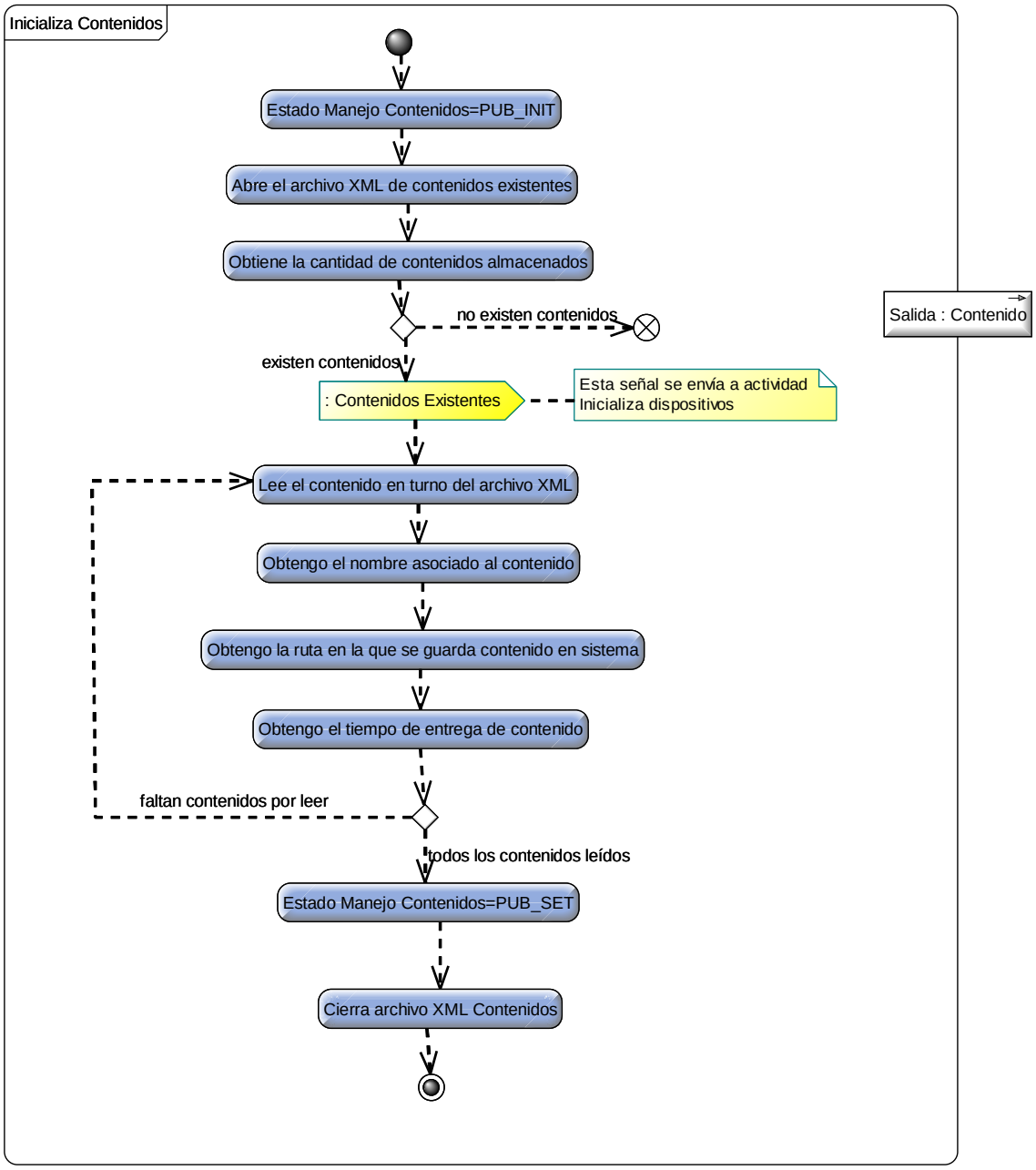


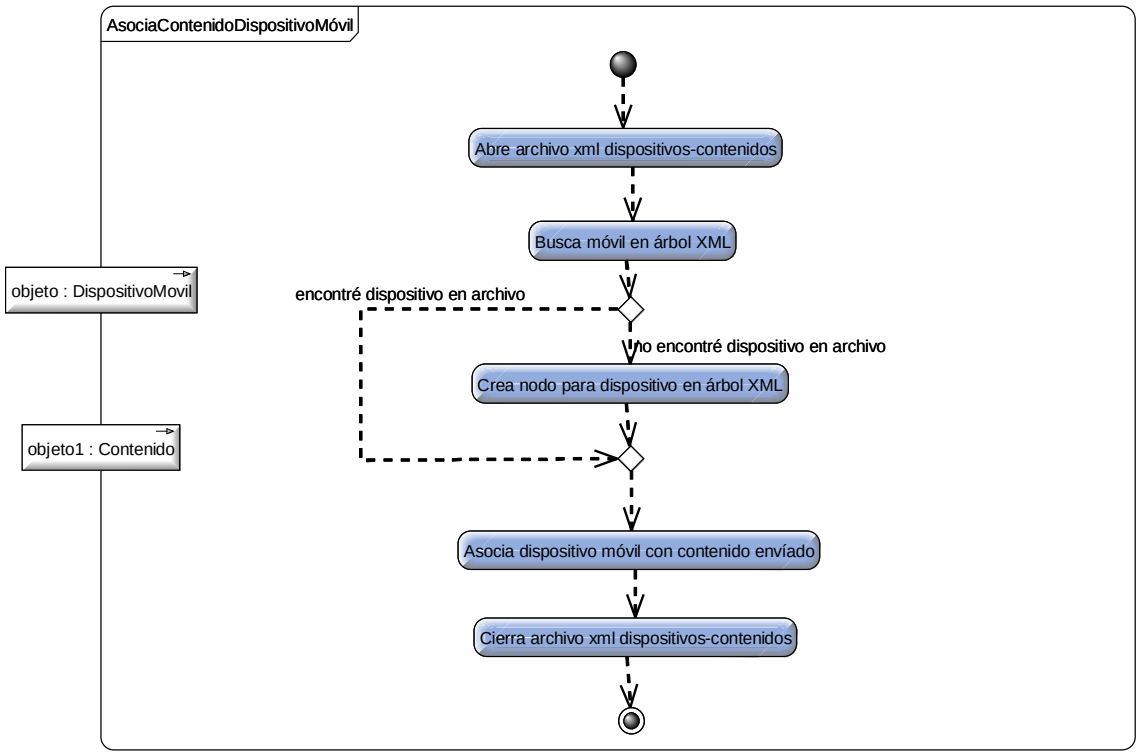

Estadísticas Bluetooth
Revisa que exista archivo XML Estadísitcas Bluetooth
ya existía archivo XML estadísitcas Bluetooth
no existe archivo XML estadísticas Bluetooth
Crea archivo XML Estadísticas Bluetooth
Abre archivo XML Estadísticas Bluetooth
Inserta corte indicando fechas y horas de inicio y término
Incluye contador de solicitudes aceptadas, rechazadas e ignoradas
Incluye cuántos dispositivos fueron descubierto en ese período
Resetea todos los contadores
Cierra archivo XML


Inicializa Estadísticas
Estado Proceso Estadísticas=ESTADISTICAS_INIT
Revisa que existe archivo XML para guardar estadísitcas
exisitía archivo XML
no existía archivo XML
Crea archivo XML para guardar estadísticas
Toma fecha/hora de corte
Estado Proceso Estadísiticas=ESTADISTICAS_WAIT
Lanza timer de espera de 1 hora

Caso de Uso Proceso Manejo Contenidos



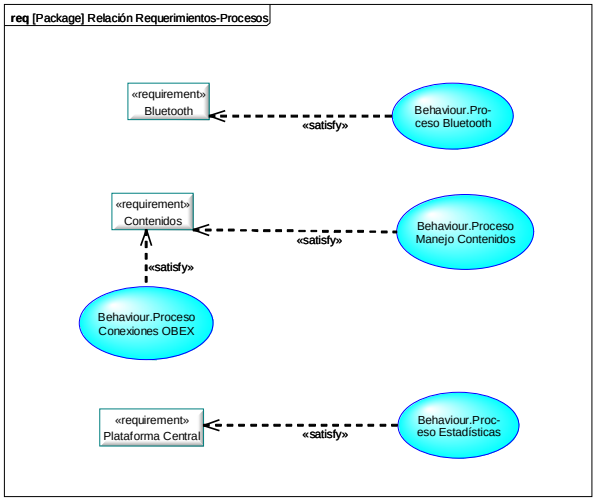

Inicializa Contenidos
Estado Manejo Contenidos=PUB_INIT
Abre el archivo XML de contenidos existentes
Obtiene la cantidad de contenidos almacenados
no existen contenidos
Salida : Contenido
existen contenidos
: Contenidos Existentes
Esta señal se envía a actividad Inicializa dispositivos
Lee el contenido en turno del archivo XML
Obtengo el nombre asociado al contenido
Obtengo la ruta en la que se guarda contenido en sistema
Obtengo el tiempo de entrega de contenido
faltan contenidos por leer
todos los contenidos leídos
Estado Manejo Contenidos=PUB_SET
Cierra archivo XML Contenidos


AsociaContenidoDispositivoMóvil
objeto : DispositivoMovil
objeto1 : Contenido
Abre archivo xml dispositivos-contenidos
Busca móvil en árbol XML
encontré dispositivo en archivo
no encontré dispositivo en archivo
Crea nodo para dispositivo en árbol XML
Asocia dispositivo móvil con contenido envíado
Cierra archivo xml dispositivos-contenidos

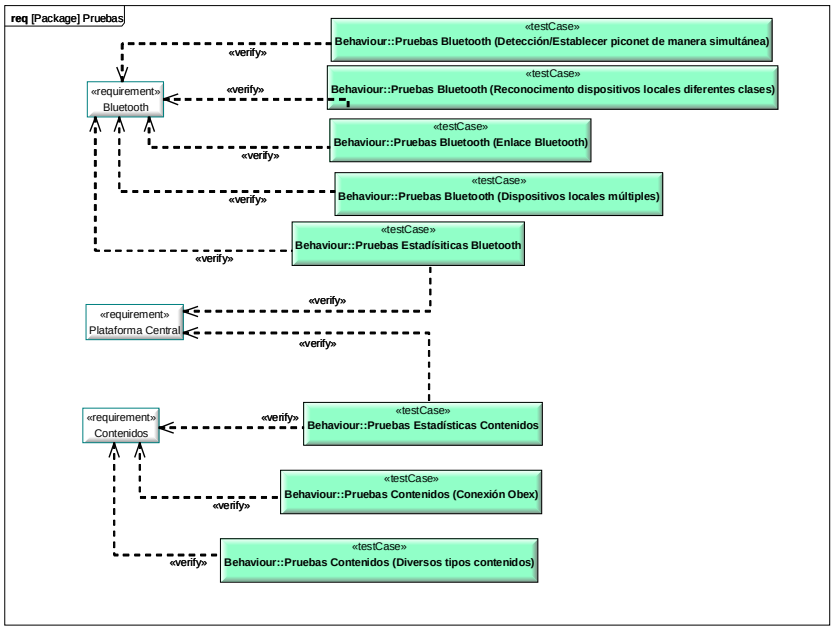
PRUEBAS Y CONCLUSIONES

CASOS DE PRUEBA

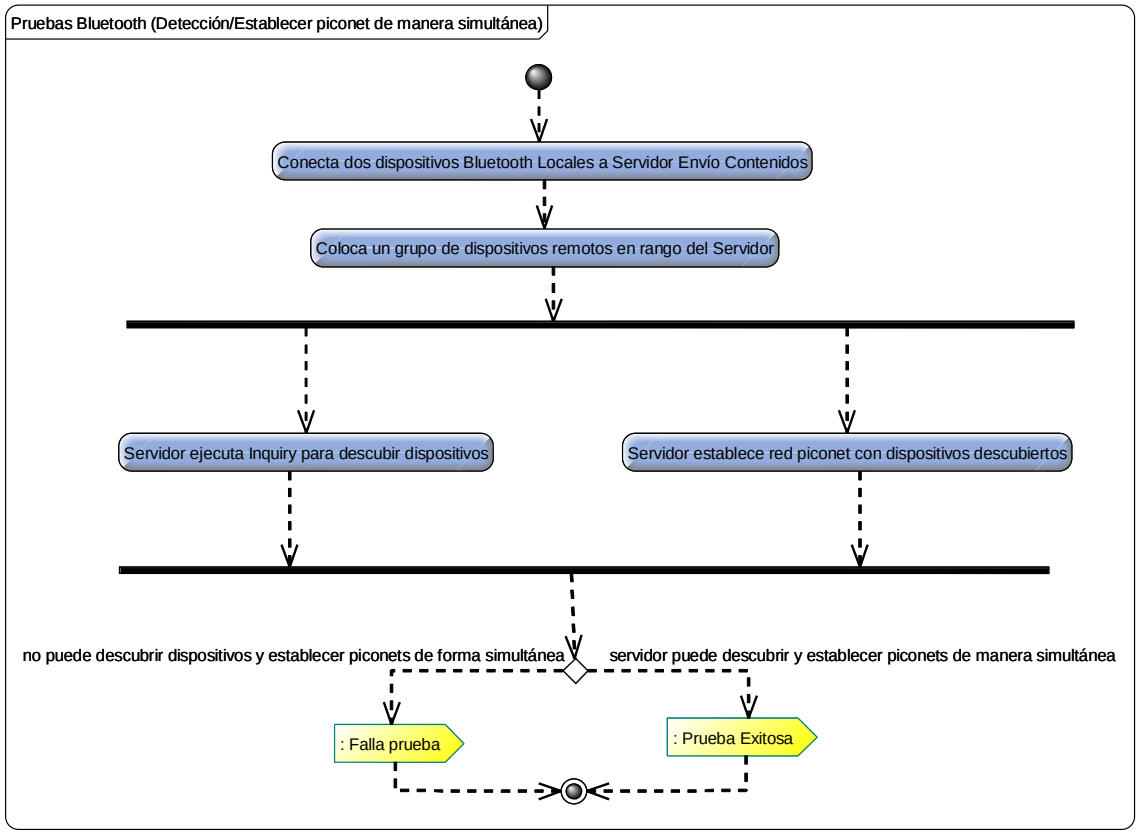
Retomando los requerimientos previamente especificados, el siguiente diagrama de requerimientos establece la relación existente entre los requerimientos y los casos de uso principales en el comportamiento del modelo.

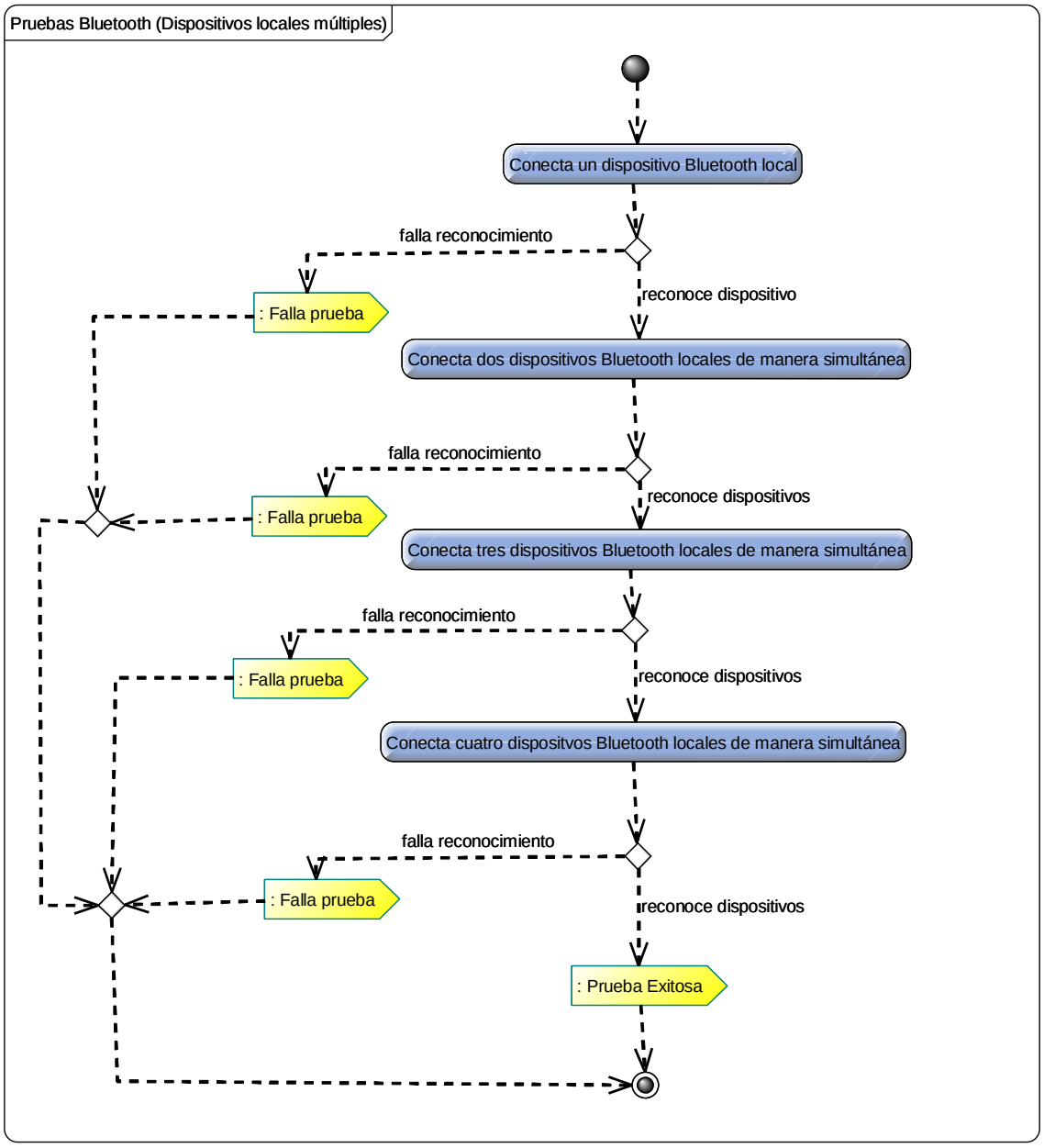


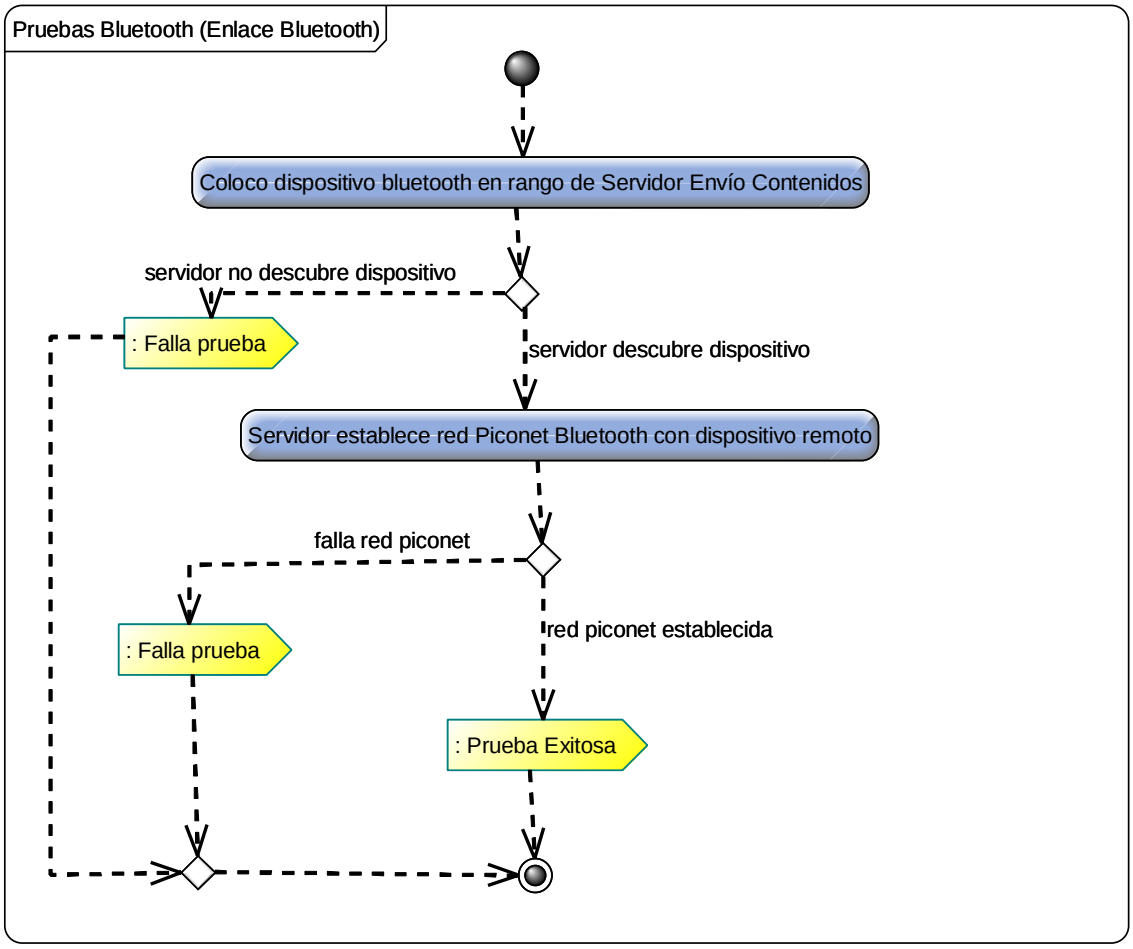
Como parte primordial del modelado del sistema, es necesario definir un conjunto de pruebas de manera que puedan comprobarse lo establecido por los requerimientos. El siguiente diagrama muestra el set de pruebas propuestos para validar la correcta operación del Servidor así como la relación existente entre los requerimientos y los casos de prueba propuestos.

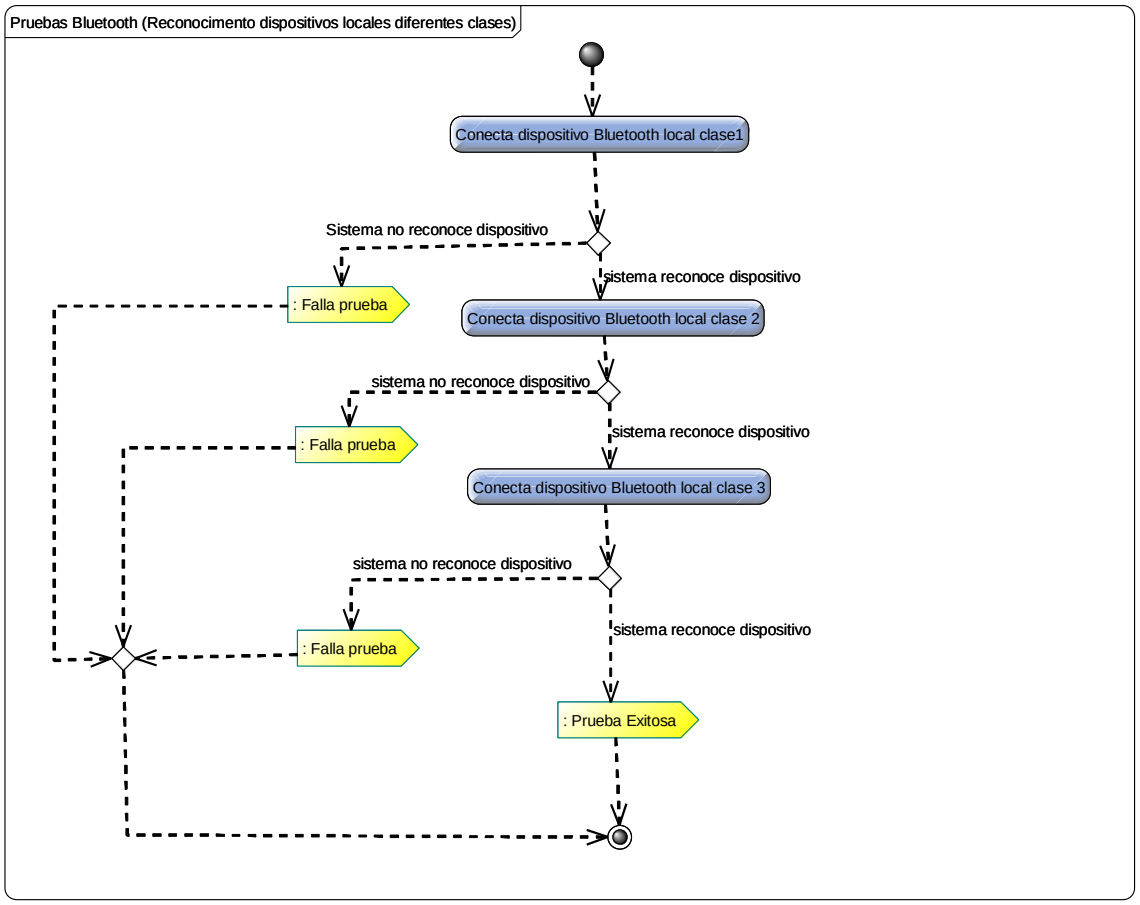


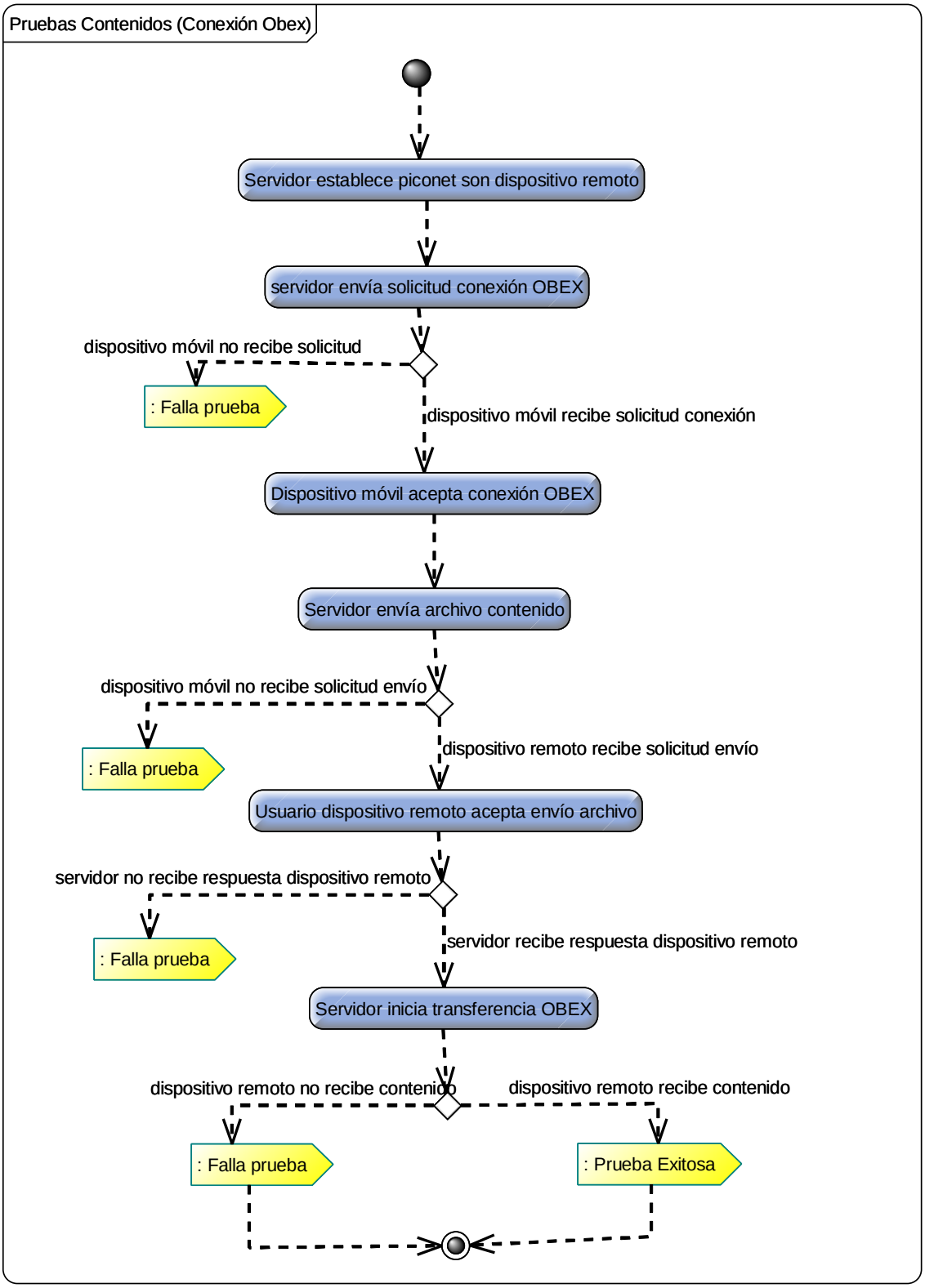
A continuación, haremos uso de Diagramas de Actividad para modelar el procedimiento a seguir en cada uno de los casos de pruebas propuesta así como las condiciones de falla o éxito de cada uno de los casos de prueba.

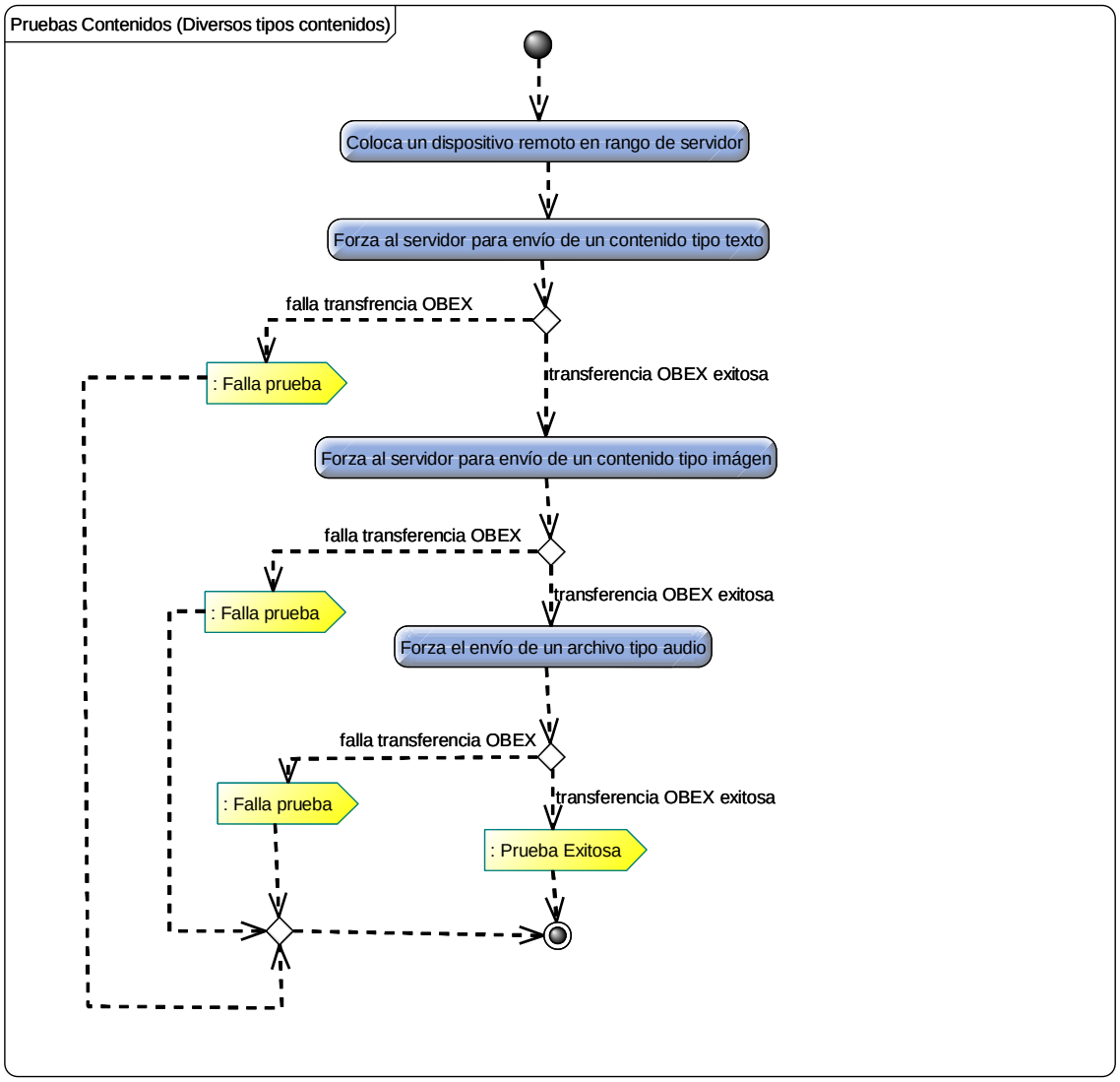


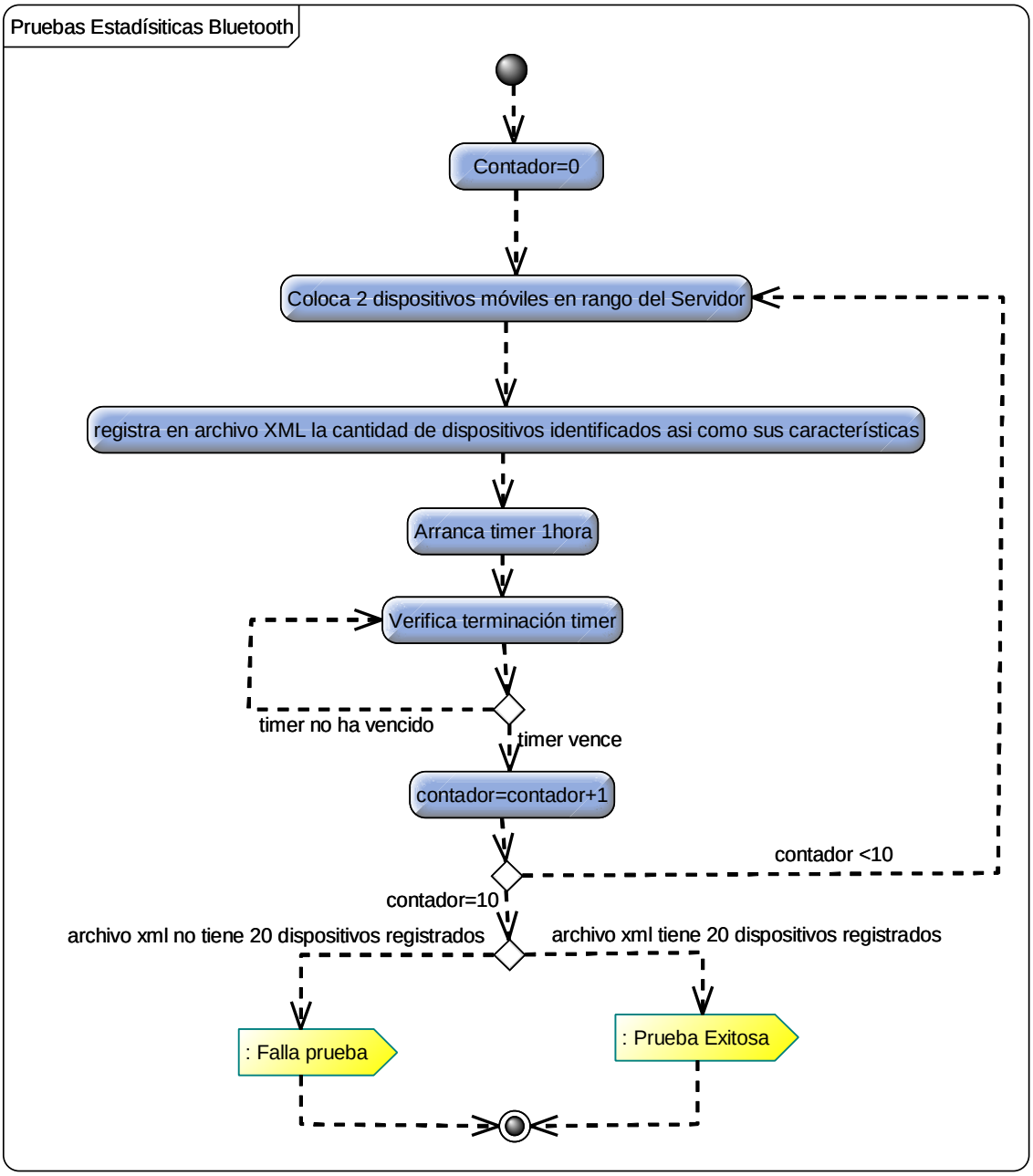

Pruebas Bluetooth (Dispositivos locales múltiples)
Conecta un dispositivo Bluetooth local
falla reconocimiento
: Falla prueba
reconoce dispositivo
Conecta dos dispositivos Bluetooth locales de manera simultánea
falla reconocimiento
: Falla prueba
reconoce dispositivos
Conecta tres dispositivos Bluetooth locales de manera simultánea
falla reconocimiento
: Falla prueba
reconoce dispositivos
Conecta cuatro dispositvos Bluetooth locales de manera simultánea
falla reconocimiento
: Falla prueba
reconoce dispositivos
: Prueba Exitosa


Pruebas Bluetooth (Enlace Bluetooth)
Coloco dispositivo bluetooth en rango de Servidor Envío Contenidos
servidor no descubre dispositivo
: Falla prueba
servidor descubre dispositivo
Servidor establece red Piconet Bluetooth con dispositivo remoto
falla red piconet
: Falla prueba
red piconet establecida
: Prueba Exitosa


Pruebas Bluetooth (Reconocimento dispositivos locales diferentes clases)
Conecta dispositivo Bluetooth local clase1
Sistema no reconoce dispositivo
sistema reconoce dispositivo
: Falla prueba
Conecta dispositivo Bluetooth local clase 2
sistema no reconoce dispositivo
sistema reconoce dispositivo
: Falla prueba
Conecta dispositivo Bluetooth local clase 3
sistema no reconoce dispositivo
sistema reconoce dispositivo
: Falla prueba
: Prueba Exitosa


Pruebas Contenidos (Conexión Obex)
Servidor establece piconet son dispositivo remoto
servidor envía solicitud conexión OBEX
dispositivo móvil no recibe solicitud
: Falla prueba
dispositivo móvil recibe solicitud conexión
Dispositivo móvil acepta conexión OBEX
Servidor envía archivo contenido
dispositivo móvil no recibe solicitud envío
: Falla prueba
dispositivo remoto recibe solicitud envío
Usuario dispositivo remoto acepta envío archivo
servidor no recibe respuesta dispositivo remoto
: Falla prueba
servidor recibe respuesta dispositivo remoto
Servidor inicia transferencia OBEX
dispositivo remoto no recibe contenido
dispositivo remoto recibe contenido
: Falla prueba
: Prueba Exitosa


Pruebas Contenidos (Diversos tipos contenidos)
Coloca un dispositivo remoto en rango de servidor
Forza al servidor para envío de un contenido tipo texto
falla transfrencia OBEX
: Falla prueba
transferencia OBEX exitosa
Forza al servidor para envío de un contenido tipo imágen
falla transferencia OBEX
: Falla prueba
transferencia OBEX exitosa
Forza el envío de un archivo tipo audio
falla transferencia OBEX
: Falla prueba
transferencia OBEX exitosa
: Prueba Exitosa


Pruebas Estadísiticas Bluetooth
Contador=0
Coloca 2 dispositivos móviles en rango del Servidor
registra en archivo XML la cantidad de dispositivos identificados asi como sus características
Arranca timer 1hora
Verifica terminación timer
timer no ha vencido
timer vence
contador=contador+1
contador <10
contador=10
archivo xml no tiene 20 dispositivos registrados
archivo xml tiene 20 dispositivos registrados
: Falla prueba
: Prueba Exitosa

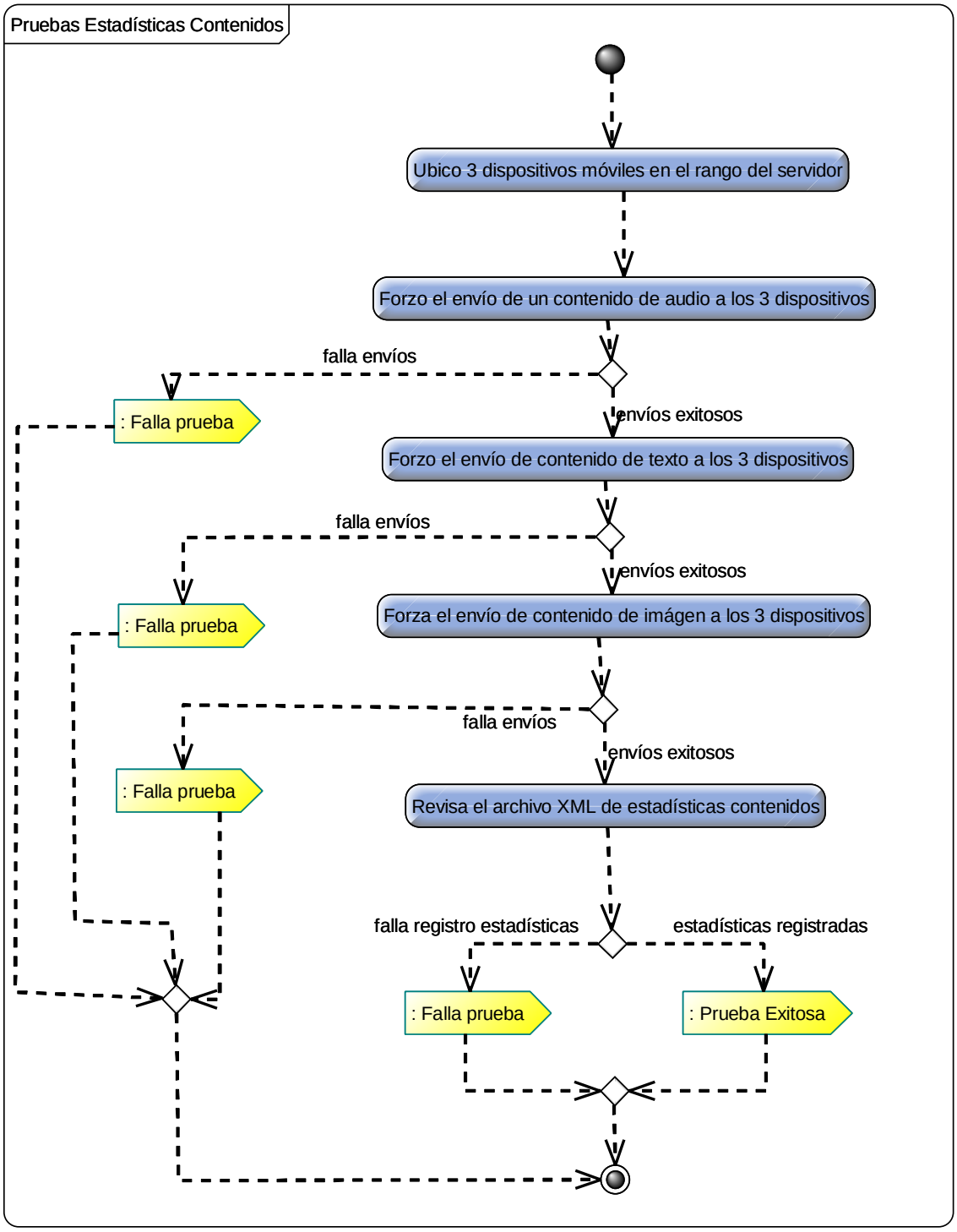

Pruebas Estadísticas Contenidos
Ubico 3 dispositivos móviles en el rango del servidor
Forzo el envío de un contenido de audio a los 3 dispositivos
falla envíos
: Falla prueba
envíos exitosos
Forzo el envío de contenido de texto a los 3 dispositivos
falla envíos
: Falla prueba
envíos exitosos
Forza el envío de contenido de imágen a los 3 dispositivos
falla envíos
: Falla prueba
envíos exitosos
Revisa el archivo XML de estadísticas contenidos
falla registro estadísticas
estadísticas registradas
: Falla prueba
: Prueba Exitosa

TABLA DE RESULTADOS

Las pruebas fueron realizadas haciendo uso de la plataforma de hardware/software de comunicaciones MSC-3G+ propiedad de Softel S.A. de C.V que se muestra en la siguiente figura:

Plataforma Comunicaciones MSC-3G+ propiedad de Softel S.A. de C.V.

La plataforma MSC-3G+ tiene las siguientes especificaciones:

Descripción del Sistema

Microprocesador	ARM9 Atmel
RAM	64 MB
Flash	256 MB
Interfaz Ethernet	5 Port 10/100 Base T
Interfaz Serial	RS-232, RS-485 (opcional)
Alimentación	Universal 80 to 250 Vac
Batería de Respaldo	2200 mAh, duración típica 4 horas
Dimensiones (mm)	277 X 197 X 44 (W X D X H)
Consumo	10 W max.
Sistema Operativo	Linux, distribución Debian

Interfaces y soportes de red

Ethernet 10/100
Interfaces Serial RS-232, RS-485
3.5G HSPA/UMTS Tribanda
2.5G Quad Band EDGE, GPRS
WiFi 802.11 g/b, Bluetooth (hasta 4 enlaces)
PPPoE for connection throught ADSL

El procedimiento a seguir al realizar las pruebas fue repetir cada uno de los casos de prueba propuestos 10 veces, la siguiente tabla muestra los resultados obtenidos:

Nombre Caso Prueba	Pruebas Exitosas	Pruebas Fallidas	Comentarios
Pruebas Bluetooth (Detección/Establecer piconet de manera simultánea)	10	0	
Pruebas Bluetooth (Dispositivos locales múltiples)	10	0	
Pruebas Bluetooth (Enlace Bluetooth)	10	0	
Pruebas Bluetooth (Reconocimento dispositivos locales diferentes clases)	10	0	
Pruebas Contenidos (Conexión Obex)	7	3	En algunos dispositivo móviles es obligatorio un proceso conocido como apareamiento Bluetoooth (*Bluetooth Pairing*), el cual consiste en el intercambio de la misma contraseña por parte de los dos dispositivos involucrados. La versión inicial del Servidor de Envío de Contenidos no incluye dicha funcionalidad, por lo que sólo funciona con dispositivos móviles a los cuales se les pueda configurar/omitir dicho apareamiento.
Pruebas Contenidos (Diversos tipos contenidos)	10	0	
Pruebas Estadísiticas Bluetooth	10	0	
Pruebas Estadísticas Contenidos	10	0	

CONCLUSIONES

Retomando lo planteado en la introducción, el objetivo de esta tesis fue desarrollar un servidor basado en el sistema operativo Linux que permitiera establecer comunicación entre un punto central y dispositivos móviles vía Bluetooth para el intercambio organizado de cualquier tipo de contenido. El modelado y diseño de dicho servidor debía seguir una metodología claramente acotado por lo definido por conceptos de ingeniería de sistemas y hacer uso de los lenguajes de modelado UML y SysML.

El modelo presentado logró resumir las etapas involucradas en el desarrollo de un sistema. Los requerimientos propuestos fueron claramente especificados y definidos. Partiendo de ellos, se modelaron las estructuras necesarias para controlar todo lo relativo a la gestión tanto de los enlaces Bluetooth como del manejo de los diversos contenidos ofrecidos por el sistema. Haciendo uso de diferentes perspectivas, se modeló el comportamiento necesario con el nivel de detalle suficiente para poder codificar procesos que cumplieran en su totalidad con lo esperado. Por último, se logró integrar como parte fundamental del modelo la definición de los casos de prueba necesarios para validar la correcta operación del sistema, así como la relación existente entre los requerimientos originales del sistema y los casos de prueba propuestos.

En lo que respecta a lograr enlaces Bluetooth haciendo uso de las librerías existentes en Linux (Bluez, OpenObex), la implementación se hizo sin ningún tipo de problemas. Las pruebas realizadas con módulos locales Bluetooth conectados al servidor con diferentes características (clase Bluetooth, hardware interno, controlador) demostraron que Bluez no presenta ningún tipo de problema al interactuar con las opciones de dispositivos Bluetooth existentes en el mercado. La interacción con dispositivos móviles remotos también funcionó de manera adecuada logrando que el servidor de envío de contenidos enviara archivos a computadores personales y teléfonos personales de diferentes marcas. Por lo tanto, la funcionalidad ofrecida por OpenObex fue comprobada en su totalidad. Los únicos problemas presentados durante el envío de contenidos se presentaron al tratar de interactuar con dispositivos que forzosamente necesitaban un intercambio de contraseñas (*pairing*) como parte del establecimiento de la conexión Bluetooth. A pesar de que Bluez y OpenObex ofrecen las librerías para su realización, la automatización de dicho intercambio de contraseñas no era parte de los requerimientos del servidor. Por lo tanto, una versión futura del Servidor de Envío de Contenidos debería ser capaz de resolver dicho problemas incrementando la lógica relacionada con la gestión del enlace Bluetooth.

Como conclusión general, la tesis cumplió con el objetivo principal de aplicar no sólo las técnicas propuestas por la ingeniería de sistemas sino lenguajes de modelado altamente utilizados en la industria como los son UML y SysML para resolver un problema en específico, como le es la comunicación Bluetooth entre dispositivos basada en Linux. Se comprobó cómo mediante el uso de un procedimiento claro y definido del ciclo de desarrollo de un sistema (definición requerimientos, modelado estructural y de comportamiento, ejecución y definición de pruebas), las posibilidades de llevar a buen término la solución de un problema, en cualquier rama de la ingeniería, se incrementan de manera significativa.

ANEXOS
CÓDIGO FUENTE SERVIDOR ENVÍO CONTENIDOS

Nota: el código incluido sólo es el referente a los procesos cuyo comportamiento fue modelado con anterioridad. En dicho código se hace referencia a funciones para el manejo de estructuras de datos como listas, FIFOs, etc. El código fuente de dichas librerías no será incluido como parte del anexo. El lenguaje de programación utilizado es C y el compilador utilizado fue GCC.

Archivos de cabecera

```
/*
 * SoftelObexServer.h
 *
 *
 *      Author: islas
 */

#ifndef SOFTELOBEXSERVER_H_
#define SOFTELOBEXSERVER_H_
#include<logCtrl.h>
//#include<Util.h>
#include "utilerias.h"
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<bTree.h>
#include<colaReg.h>
#include<lists.h>
#include<errno.h>
#include<string.h>
#include<utilerias.h>
#include<sys/time.h>
#include<SoftelBluetooth.h>
#include<SoftelObex.h>
#include<SoftelXml.h>
#include<SoftelPublicidad.h>
#include<SoftelEstadisticas.h>
#include<aux.h>
#include<ObexConn.h>

/*para el manejo de logs*/
OBJ_LOG *log_ptr;
/*arbol para el manejo de dispositivos*/
BTREE *Bluetooth_Btree;
/*para el manejo de la cola de elementos activos*/
COLA activosCola;
/*para el manejo de modificaciones a informaciÃ³n XML */
COLA xmlCola;
/*para habilitar el envÃ-o de publicidad*/
int tengo_publicidad;
/*para saber la ruta del directorio en el que estÃ¡ la aplicaciÃ³n*/
char    path[100];
/*lista para el manejo de publicidades*/
TYPE_LIST lista_publicidad;
typedef struct
{
  int cliente;
  int dispositivo;
  char nombre[100];
  char servidor[20];
  char hora_inicio[3];
  char hora_fin[3];
}CONFIGURACION;
/*para guardar la configuraciÃ³n del servidor OBEX*/
CONFIGURACION config;
typedef struct
{
  char nombre_BT[250];   //nombre asociado al dispositivo BT
  char direccion_BT[19];  //direccion BT en formato XX:XX:XX:XX:XX:XX
  bdaddr_t target;  //direcciÃ³n BT en formato 6 bytes
  char canal_OBEX;
  int proceso; //bandera para prender cuando estoy negociando el envio de un archivo a ese dispositivo
  unsigned int canales_obex[4];
  int canales_obex_encontrados;  //si es negativo indica que ya se le preguntÃ³ y no contesto si tiene Obex....decremento y no vuelvo a preguntar hasta llegar a -30
  char path[100];  //path del archivo que se le enviarÃ¡
  char nombre_archivo[50]; //nombre del archivo que se enviarÃ¡
  int rechaza;  //si esta en uno indica que el dispositivo no aceptÃ³ una transferencia, por lo tanto no se lo molestarÃ¡ de nuevo
  int time_out_count; //veces seguidas que el dispositivo no ha respondido a una peticiÃ³n de tranferencia...a la 5 seguida no se vuelve a molestar
  TYPE_LIST lista_publicidad;  //guarda la publicidad que se le ha mandado a ese dispositivo
}DISPOSITIVO_BT;

//estados de actividad para los bt locales
#define LOCAL_INACTIVE   0
#define LOCAL_DISCOVERY   1
#define LOCAL_PUSH      2
typedef struct
{
```

```
  int etiqueta;
  struct hci_dev_info di;
  int socket;
  bdaddr_t source; //direcciÃ³n BT local en formato 6 bytes
  int last_dev_id;
  char addr[19];
  int activo; //para saber si el dispositivo estÃ¡ idle o si esta descubriendo remotos o enviando
publicidad
  int total_push; //cada dispositivo sÃ³lo puede tener hasta 7 conexiones al mismo tiempo
}BT_LOCAL;

//para saber a quÃ© dispositivo bt le voy a enviar la info y quÃ© dispositivo local voy a
utilizar
typedef struct
{
  BT_LOCAL *bt_local_ptr;
  DISPOSITIVO_BT *bt_ptr;
}BT_REMOTO_LOCAL;
//para saber cuÃ¡ntos dispositivos BT tengo conectados localmente, esto determina las
capacidades de reparto de publicidad
int conectados;
typedef struct
{
  int tipo;
  char direccion_BT[19];  //direccion BT en formato XX:XX:XX:XX:XX:XX
  char nombre_archivo[50];  //nombre de la publicidad asociada en caso de ser necesario
}TRAN_XML;
typedef struct
{
  int id;      //identificador de la publicdad
  char nombre[50];  //nombre
  char archivo[20];  //nombre del archivo asociado
  int horas;    //horas que mantendrÃ© la publicidad como la activa antes de cambiar
}PUBLICIDAD_BT;
/*aqui se va a guardar la publicidad activa*/
PUBLICIDAD_BT *publicidad_actual;
unsigned long pointer, key;
// Datos para control de Thread
typedef struct {
    struct      timespec tw;
      int       loop_rxThread;// indica cuando terminar thread <rxThread>
      pthread_t      apThread;
      pthread_mutex_t in_mutex;    // permite acceso a <in> de manera excluyente
}OBEX_SERVER_THREAD;
/*para los thread de descubrimiento de dispositivos,manejo de conexiones,manejo de
xml,manejo de publicidad,estadisticas*/
OBEX_SERVER_THREAD threadBT;
OBEX_SERVER_THREAD threadObex;
OBEX_SERVER_THREAD threadXML;
OBEX_SERVER_THREAD threadPublicidad;
OBEX_SERVER_THREAD threadEstadisticas;
/*semÃ¡foro para acceso al Ã¡rbol de dispositivos*/
pthread_mutex_t btree_lock;
/*semÃ¡foro para acceso a la publicidad,tanto a la lista como a la publicidad actual*/
pthread_mutex_t pub_lock;
/*constantes para saber que tipo de evento xml es el que estoy recibiendo en la cola*/
#define SOFTEL_XML_DEVICE   1    //lo uso para agregar un dispositivo nuevo al Ã¡rbol
#define SOFTEL_XML_FILE     2    ///lo uso apra agregar una archivo de publicidad enviado
nuevo a un dispositivo
#define SOFTEL_XML_REJECT   3    //indica que el dispositivo ya nos rechazÃ³ una vez, no
volver a enviar
#endif /* SOFTELOBEXSERVER_H_ */
```

```
/*
 * SoftelBluetooth.h
 *
 *
 *      Author: islas
 */

#ifndef SOFTELBLUETOOTH_H_
#define SOFTELBLUETOOTH_H_

#include<sys/socket.h>
#include<sys/ioctl.h>
#include<bluetooth/bluetooth.h>
#include<bluetooth/hci.h>
#include<bluetooth/hci_lib.h>
#include<bluetooth/sdp.h>
#include<bluetooth/sdp_lib.h>
#include<lists.h>



typedef enum
{
   BT_IDLE=0,
   BT_INIT,
   BT_DISCOVERY,
   BT_WAIT,
   BT_HWERROR,
   BT_FIN
}BT_STATUS;


//estados de actividad para los bt locales
#define LOCAL_INACTIVE   0
#define LOCAL_DISCOVERY   1
#define LOCAL_PUSH      2



typedef struct
{
   BT_STATUS status;
   struct hci_dev_list_req *dl;
   struct hci_dev_req *dr;
   TYPE_LIST bluetooth_devices;
   int total_devices; //este es un consecutivo de cuantos he disp locales he econtrado desde arranque, no cuantos tengo actualmente..puede verse como historico
   int etiqueta; //consecutivo para saber cuales de los dispositivos de la lista estan activos y poder eliminar el resto
}BT_CONTROL;

BT_CONTROL bt_control;


void *SoftelBluetoothThread (void *a);


#endif /* SOFTELBLUETOOTH_H_ */
```

```
/*
 * SoftelObex.h
 *
 *
 *      Author: islas
 */

#ifndef SOFTELOBEX_H_
#define SOFTELOBEX_H_

#include <SoftelObexServer.h>
//mÃ¡ximo numero de conexiones
#define MAX_CONN 56

// Datos para control de Threads de conexiones
typedef struct {
     struct timespec tw;
       int      loop_rxThread;// indica cuando terminar thread <rxThread>
       pthread_t       apThread;
       pthread_mutex_t in_mutex;     // permite acceso a <in> de manera excluyente
       int activa;
       void *datos;
}OBEX_CONN_THREAD;

typedef struct
{
  OBEX_CONN_THREAD   threads[MAX_CONN];   //amarrada   a   un   mÃ¡ximo   de   56
conexiones al mismo tiempo (8 dispositivos BT locales)
  int en_proceso;
}OX_CONTROL;

OX_CONTROL ox_control;

void *SoftelObexThread (void *a);
int checa_disponibilidad(void);

#endif /* SOFTELOBEX_H_ */
```

```
/*
 * ObexConn.h
 *
 *
 *      Author: islas
 */

#ifndef OBEXCONN_H_
#define OBEXCONN_H_
#include <openobex/obex.h>
#define OBEX_PUSH           5
#define OBEX_TIMEOUT_ERROR   115
#define OBEX_REJECTED_ERROR 111
void *ObexConnThread(void *a);
#endif /* OBEXCONN_H_ */
```

```
/*
 * SoftelEstadisticas.h
 *
 *
 *      Author: islas
 */
#ifndef SOFTELESTADISTICAS_H_
#define SOFTELESTADISTICAS_H_
/*estructura para el manejo de estadisticas generales de envio y rechazo*/

typedef enum
{
  ESTADISTICAS_INIT=0,
  ESTADISTICAS_WAIT,
  ESTADISTICAS_COMMIT
}ESTADISTICAS_STATUS;

typedef struct
{
  ESTADISTICAS_STATUS status;
  char fecha_inicio[20];
  char fecha_termino[20];
  int envios;
  int rechazos;
  int timeout;
  int descubiertos;
}CONTROL_ESTADISTICAS;

CONTROL_ESTADISTICAS control_estadisticas;
void *SoftelEstadisticasThread(void *a);
#endif /* SOFTELESTADISTICAS_H_ */
/*
 * SoftelPublicidad.h
 *
 *  Created on: Jan 22, 2009
 *      Author: islas
 */

#ifndef SOFTELPUBLICIDAD_H_
#define SOFTELPUBLICIDAD_H_

typedef enum{
  PUB_INIT=0,
  PUB_SET,
  PUB_WAIT
}PUB_STATUS;

typedef struct
{
  PUB_STATUS status;
}PUB_CONTROL;

PUB_CONTROL pub_control;

void *SoftelPublicidadThread (void *a);
int SftPubRevisa(void *ptr);

#endif /* SOFTELPUBLICIDAD_H_ */
```

```
/*
 * SoftelXml.h
 *
 *
 *      Author: islas
 */

#include <libxml2/libxml/xmlreader.h>
#include <libxml2/libxml/encoding.h>
#include <libxml2/libxml/xmlwriter.h>
#include <libxml2/libxml/tree.h>
#include <libxml2/libxml/xmlmemory.h>
#include <libxml2/libxml/parser.h>



#ifndef SOFTELXML_H_
#define SOFTELXML_H_

#define MY_ENCODING "ISO-8859-1"

typedef enum
{
   XML_INIT=0,
   XML_CREATE_FILE,
   XML_PARSE_FILE,
   XML_IDLE
}XML_STATUS;

typedef struct
{
   XML_STATUS status;
   xmlTextReaderPtr reader;
}XML_CONTROL;

XML_CONTROL xml_control;

int SftXmlConfig(void);
int SftXmlAgregaCorte(void);
void *SoftelXmlThread (void *a);



#endif /* SOFTELXML_H_ */
```

Archivos Fuentes

```
/*
 * SoftelObexServer.c
 *
 *
*      Author: islas
 */


#include<SoftelObexServer.h>


int asignaValor(char *ptr1,char *ptr2)
{
  return 1;
}

void init(char *nombre_app)
{
  pthread_attr_t attr;
  char    nombre[100];
  char nombre_log[100];
  char ruta_log[50];
  memset(ruta_log,0x0,50);
  memset(nombre_log,0x0,100);
  /*inicializa el log*/
  preDirectorio(nombre_app, path, nombre, ruta_log);
  sprintf(nombre_log,"%s/ObexServer.log",ruta_log);
  log_ptr=iniArcLog (5,1000000,nombre_log,"Softel Obex Server V1.0 log file");
  /*termina de inicializar el log*/
  /*para tener todos los archivos en /media/data/bluetooth*/
  memset(path,0x0,sizeof(path));
  strcpy(path,"/media/data/bluetooth");
  /*Prepara atributos  de procesos*/
  pthread_attr_init (&attr);
  pthread_attr_setdetachstate          (&attr,          PTHREAD_CREATE_JOINABLE);          //
PTHREAD_CREATE_DETACHED,                    PTHREAD_CREATE_UNDETACHED,
PTHREAD_CREATE_JOINABLE
  bt_control.status=BT_IDLE;
  pub_control.status=PUB_INIT;
  control_estadisticas.status=ESTADISTICAS_INIT;
  Bluetooth_Btree=NULL;
  COLA_crea (&activosCola,NULL);
  COLA_crea (&xmlCola,NULL);
  /*inicializa el semÃ¡foro para control de acceso al Ã¡rbol y a publicidad*/
  pthread_mutex_init(&btree_lock, NULL);
  pthread_mutex_init(&pub_lock, NULL);
  /*no tengo publicidad hasta leer archivo*/
  tengo_publicidad=0;
  /*inicializa la lista de publicidades*/
  LIST_Create(&lista_publicidad);
  /*Crea procesos para interacciÃ³n Bluetooth,Obex,XML,publicidad. Incluye  estructuras de
control y  semÃ¡foros*/

threadXML.loop_rxThread=iniThread(&threadXML,&threadXML.in_mutex,&threadXML.ap
Thread,SoftelXmlThread,&attr);

threadPublicidad.loop_rxThread=iniThread(&threadPublicidad,&threadPublicidad.in_mutex,&
threadPublicidad.apThread,SoftelPublicidadThread,&attr);
      threadBT.loop_rxThread          =          iniThread(&threadBT,          &threadBT.in_mutex,
&threadBT.apThread,SoftelBluetoothThread, &attr);
      threadObex.loop_rxThread=iniThread(&threadObex,                &threadObex.in_mutex,
&threadObex.apThread,SoftelObexThread, &attr);
      threadEstadisticas.loop_rxThread=iniThread(&threadEstadisticas,
&threadEstadisticas.in_mutex, &threadEstadisticas.apThread,SoftelEstadisticasThread, &attr);
}

int main(int argc, char *argv[])
{
  struct timespec tw;
  iniThread_memory();
  init(argv[0]);
  while(!SftXmlConfig())
  {
    elog_printf(log_ptr,"BLUETOOTH: no pude leer configuraciÃ³n...espero 10 min");
    segSleep(&tw,600,0);
    nanosleep (&tw, NULL);
  }
  segSleep(&tw,10,0);
  while(1)
  {
     nanosleep (&tw, NULL);
  }
  return 1;
}
```

```
/*
 * SoftelBluetooth.c
 *
 *
 *      Author: islas
 */

#include<SoftelObexServer.h>

//para poder identificar los que si sean telÃ©fonos por medio del Class of Device
char *major_devices[] = { "Miscellaneous",
              "Computer",
              "Phone",
              "LAN Access",
              "Audio/Video",
              "Peripheral",
              "Imaging",
              "Uncategorized" };


static int bt_fill_channs(sdp_data_t *p, unsigned int *channs, int nchanns)
{
  int n, proto = 0;

  for (n = 0; p; p = p->next) {
    switch (p->dtd) {
    case SDP_UUID16:
    case SDP_UUID32:
    case SDP_UUID128:
      proto = sdp_uuid_to_proto(&p->val.uuid);
      break;
    case SDP_UINT8:
      if (proto == RFCOMM_UUID) {
        if (n < nchanns) {
          //elog_printf(log_ptr,"Found channel %u n %d\n",(unsigned int) p->val.uint8,n);
          channs[n++] = p->val.uint8;
        }
      }
      break;
    }
  }

  return n;
}

static void bt_find_class(void *value, void *priv)
{
  char uuid_str[MAX_LEN_UUID_STR];
  char svc_uuid_str[MAX_LEN_SERVICECLASS_UUID_STR];
  uuid_t *uuid = (uuid_t *) value;
  sdp_uuid2strn(uuid, uuid_str, MAX_LEN_UUID_STR);
  sdp_svclass_uuid2strn(uuid, svc_uuid_str, MAX_LEN_SERVICECLASS_UUID_STR);
  //elog_printf(log_ptr,"\tFound \"%s\" (0x%s)", svc_uuid_str, uuid_str);
  if (sdp_uuid_to_proto(uuid) == OBEX_OBJPUSH_SVCLASS_ID)
    *(int *) priv = 1;
}


static int bt_sdp_search(bdaddr_t const *iface, bdaddr_t const *bdaddr,
        uuid_t *uidsearch, unsigned int *channs, int nchanns)
{
  int n, s;
  uint32_t range = 0x0000ffff;
  sdp_list_t *attrid, *search, *seq, *next;
  sdp_session_t *sess;
  char str[32];

  sess = sdp_connect(BDADDR_ANY, bdaddr, SDP_RETRY_IF_BUSY);
  ba2str(bdaddr, str);
  if (!sess) {
    elog_printf(log_ptr, "SDP:Failed to connect to SDP server on %s, %s",
      str, strerror(errno));
    return -1;
  }
  attrid = sdp_list_append(0, &range);
  search = sdp_list_append(0, uidsearch);
  if (sdp_service_search_attr_req(sess, search, SDP_ATTR_REQ_RANGE, attrid, &seq)) {
    elog_printf(log_ptr, "SDP:Service search failed: %s", strerror(errno));
    sdp_close(sess);
    return -1;
  }
  sdp_list_free(attrid, 0);
  sdp_list_free(search, 0);
  for (n = 0; seq; seq = next) {
    sdp_record_t *rec = (sdp_record_t *) seq->data;
    sdp_list_t *proto = NULL, *desc;
    uuid_t suidsearch;
    sdp_list_t *classes = NULL;
    sdp_get_service_classes(rec, &classes);
    if (classes) {
      int cfound = 0;
      sdp_list_foreach(classes, bt_find_class, &cfound);
```

```
      if (cfound) {
        if (sdp_get_access_protos(rec, &proto) == 0) {
          for (; proto; proto = proto->next) {
            desc = (sdp_list_t *) proto->data;
            for (; desc; desc = desc->next) {
              s = bt_fill_channs((sdp_data_t *) desc->data,
                    channs + n, nchanns - n);
              if (s > 0)
                n += s;
            }
          }
        }
        if (sdp_get_group_id(rec, &suidsearch) != -1) {
          if (suidsearch.value.uuid16 != uidsearch->value.uuid16) {
            s = bt_sdp_search(iface, bdaddr, &suidsearch, channs + n,
                  nchanns - n);
            if (s > 0)
              n += s;
          }
        }
      }
      sdp_list_free(classes, free);
    }
    next = seq->next;
    free(seq);
    sdp_record_free(rec);
  }
  sdp_close(sess);
  return n;
}

int SFBT_OBEXpush_discovery(bdaddr_t *target,unsigned int *channs,BT_LOCAL
*bt_local_ptr)
{
  uuid_t uidsearch;
  int canales;
  char direccion_BT[19];
  int nchanns;
  nchanns=sizeof(channs) / sizeof(channs[0]);
  ba2str( target, direccion_BT);
  //elog_printf(log_ptr,"direcciÃ³n a la que le buscare canal obex %s",direccion_BT);
  sdp_uuid16_create(&uidsearch, OBEX_UUID);
  canales=bt_sdp_search(&bt_local_ptr->source, target, &uidsearch, channs, nchanns);
  return canales;
}

void SFBT_revisa_nombrelocal(int sock,int dev_id)
{
  char name[100];
  memset(name,0x0,100);
  if(strlen(config.nombre)>0)
  {
    hci_read_local_name(sock, sizeof(name), name, 1000);
    if(strstr(name,config.nombre)==NULL)
    {
      memset(name,0x0,100);
      sprintf(name,"%s-%d",config.nombre,dev_id);
      if(!hci_write_local_name(sock,name, 2000)<0)
        elog_printf(log_ptr,"BLUETOOTH:cambio nombre dispositivo a %s",name);
      else
        elog_printf(log_ptr,"BLUETOOTH:no pude cambiar el nombre a dispositivo");
    }
  }
}

int SFBT_ValidaHora(void)
{
  int exito=0;
  char fecha[20];
  //2009-01-27 16:27:16
  char *ptr=NULL;
  char hora[3];
  int actual;
  int inicio;
  int fin;
  memset(fecha,0x0,20);
  memset(hora,0x0,3);
  if(strlen(config.hora_inicio)!=2 || strlen(config.hora_fin)!=2)
  {
    exito=1;   //no tengo fechas de fin o inicio capturadas....hazlo sin importar horario
  }
  else
  {
    getlocaltime(fecha);
    ptr=strstr(fecha," ");
    if(ptr!=NULL)
    {
      memcpy(hora,ptr+1,2);
      actual=atoi(hora);
      inicio=atoi(config.hora_inicio);
      fin=atoi(config.hora_fin);
      if(actual>inicio&&actual<fin)
```

```
        exito=1;
      }
    }
    return exito;
}

void BtLocalLimpiaLista(int etiqueta)
{
    int total=0;
    unsigned long ptr,llave;
    int contador;
    BT_LOCAL *bt_local_ptr=NULL;
    total =(int)LIST_Count(&bt_control.bluetooth_devices);
    if(total>0)
    {
      for(contador=0;contador<total;contador++)
      {
        if (LIST_MoveNext(&bt_control.bluetooth_devices,1,(unsigned long *)&llave,&ptr)==
LIST_SUCCESS)
        {
          bt_local_ptr=(BT_LOCAL *)ptr;
          if(bt_local_ptr->etiqueta!=etiqueta)
          {
            //ya quitaron ese dongle...borralo
            LIST_Delete(&bt_control.bluetooth_devices,llave,&ptr);
            bt_local_ptr=(BT_LOCAL *)ptr;
            free(bt_local_ptr);
          }
        }
      }
    }
}

void BtLocalActualizaLista(BT_LOCAL bt_local)
{
    BT_LOCAL *bt_local_ptr=NULL;
    int total=0;
    int encuentro=0;
    int contador;
    int last_activo;
    int last_push;
    int dev_id_last;
    unsigned long ptr,llave;
    total =(int)LIST_Count(&bt_control.bluetooth_devices);
    if(total>0)
    {
      for(contador=0;contador<total;contador++)
      {
        if (LIST_MoveNext(&bt_control.bluetooth_devices,1,(unsigned long *)&llave,&ptr)==
LIST_SUCCESS)
        {
          bt_local_ptr=(BT_LOCAL *)ptr;
          if(!strcmp(bt_local_ptr->addr,bt_local.addr))
          {
            encuentro=1;
            dev_id_last=bt_local_ptr->di.dev_id;
            last_activo=bt_local_ptr->activo;
            last_push=bt_local_ptr->total_push;
            //actualizo los parÃ¡metros y la etiqueta de control
            memcpy(bt_local_ptr,&bt_local,sizeof(BT_LOCAL));
            bt_local_ptr->etiqueta=bt_control.etiqueta;
            bt_local_ptr->last_dev_id=dev_id_last;
            bt_local_ptr->activo=last_activo;
            bt_local_ptr->total_push=last_push;
            break;
          }
        }
      }
    }
    if(total==0||encuentro==0)
    {
      bt_control.total_devices++;
      bt_local_ptr=(BT_LOCAL *)malloc(sizeof(BT_LOCAL));
      memcpy(bt_local_ptr,&bt_local,sizeof(BT_LOCAL));
      pthread_mutex_lock (&threadBT.in_mutex);
      if(LIST_Insert(&bt_control.bluetooth_devices,                                    (unsigned
long)bt_control.total_devices,bt_local_ptr)!=LIST_SUCCESS)
      {
        elog_printf(log_ptr,"BLUETOOTH:no pude guardar dispositivo %s en
lista",bt_local_ptr->addr);
        free(bt_local_ptr);
      }
      else
      {
        bt_local_ptr->etiqueta=bt_control.etiqueta;
        bt_local_ptr->last_dev_id=-1;
      }
      pthread_mutex_unlock (&threadBT.in_mutex);
    }
}

BT_LOCAL *BtGetLocalDev()  //siempre DEBE regresar al menos un id vÃ¡lido
```

```
{
  int total;
  int contador;
  unsigned long ptr,llave;
  BT_LOCAL *bt_local_ptr=NULL;
  total =(int)LIST_Count(&bt_control.bluetooth_devices);
  for(contador=0;contador<total;contador++)
  {
    if (LIST_MoveNext(&bt_control.bluetooth_devices,1,(unsigned long *)&llave,&ptr)==
LIST_SUCCESS)
    {
      bt_local_ptr=(BT_LOCAL *)ptr;
      if(bt_local_ptr->activo==LOCAL_INACTIVE)
      {
        bt_local_ptr->activo=LOCAL_DISCOVERY;
        bt_local_ptr->total_push=0;
        bt_local_ptr->socket=hci_open_dev(bt_local_ptr->di.dev_id);
        elog_printf(log_ptr,"BLUETOOTH:voy a utilizar dispositivo local %s para
descubrir",bt_local_ptr->addr);
        break;
      }
    }
  }
  return bt_local_ptr;
}

void *SoftelBluetoothThread (void *a)
{
  OBEX_SERVER_THREAD *pthread;
  inquiry_info *ii = NULL;
  int max_rsp, num_rsp,len,flags,cuenta,encontrados;
  int ctl;
  char nombre_BT[250];
  char direccion_BT[19];
  DISPOSITIVO_BT bt;
  BT_LOCAL bt_local;
  BT_LOCAL *bt_local_ptr;
  int prueba;
  char archivo[150];
  TRAN_XML *tran;
  uint8_t cls[3];
  int contador;
  unsigned long ptr,llave;
  DISPOSITIVO_BT *bt_ptr=NULL;
  pthread=(OBEX_SERVER_THREAD *)a;
  segSleep (&pthread->tw, 1, 0);
  while (pthread->loop_rxThread==0)
  {
    nanosleep (&pthread->tw, NULL);
  }
  memset(&bt_control,0x0,sizeof(bt_control));
  elog_printf(log_ptr,"BLUETOOTH:arrancado!!!");
  LIST_Create(&bt_control.bluetooth_devices);
  bt_control.total_devices=0;
  bt_control.etiqueta=0;
  while(pthread->loop_rxThread)
  {
    switch(bt_control.status)
    {
    case BT_IDLE:  //nos quedamos aqui hasta que el thread xml nos diga que ya tenemos toda
la info cargada y podemos empezar a polear HCI_MAX_DEV
      break;
    case BT_INIT:
      if (!(bt_control.dl = malloc(HCI_MAX_DEV * sizeof(struct hci_dev_req) +
sizeof(uint16_t))))
      {
        elog_printf(log_ptr,"BLUETOOTH:no pude inicializar lista dispositivos!!");
        segSleep(&pthread->tw,30,0);
        bt_control.status=BT_HWERROR;
        break;
      }
      bt_control.dl->dev_num=HCI_MAX_DEV;
      bt_control.dr=bt_control.dl->dev_req;
      conectados=0;
      bt_control.etiqueta++;
      ctl= socket(AF_BLUETOOTH, SOCK_RAW, BTPROTO_HCI);
      if(ctl>0)
      {
        if (ioctl(ctl, HCIGETDEVLIST, (void *) bt_control.dl) < 0)
        {
          elog_printf(log_ptr,"BLUETOOTH:Can't get device list");
          segSleep(&pthread->tw,30,0);
          bt_control.status=BT_HWERROR;
        }
        else
        {
          elog_printf(log_ptr,"BLUETOOTH:tengo %d elementos en la lista de
Bluetooth",bt_control.dl->dev_num);
          for (contador = 0; contador< bt_control.dl->dev_num; contador++)
          {
            memset(&bt_local,0x0,sizeof(bt_local));
            bt_local.di.dev_id = (bt_control.dr+contador)->dev_id;
```

```
            hci_devba(bt_local.di.dev_id,&bt_local.source);
            bt_local.socket=hci_open_dev(bt_local.di.dev_id);
            if(bt_local.socket>=0)
            {
              hci_read_bd_addr(bt_local.socket, &bt_local.di.bdaddr, 1000);
              ba2str(&bt_local.di.bdaddr,bt_local.addr);
              if(strstr(bt_local.addr,"00:00:00:00:00:00")==NULL)
              {
                SFBT_revisa_nombrelocal(bt_local.socket,bt_local.di.dev_id);
                elog_printf(log_ptr,"BLUETOOTH:disp local encontrado, pos %d id %d
nombre %s dirBT %s"
                    ,contador,bt_local.di.dev_id,bt_local.di.name,
                    bt_local.addr);
                BtLocalActualizaLista(bt_local);
                conectados++;
              }
              else
              {
                elog_printf(log_ptr,"BLUETOOTH:dispostivo con id %d tiene dir
invÃ¡lida...manda reseteo",bt_local.di.dev_id);
                if (ioctl(ctl, HCIDEVRESET, bt_local.di.dev_id) < 0 )
                {
                  elog_printf(log_ptr, "BLUETOOTH:Reset fallÃ³ para hci%d:error %d:
%s",bt_local.di.dev_id,errno, strerror(errno));
                }
                else
                  elog_printf(log_ptr, "BLUETOOTH:Reset para hci%d",bt_local.di.dev_id);
                if (ioctl(ctl, HCIDEVDOWN,bt_local.di.dev_id) < 0)
                  elog_printf(log_ptr, "BLUETOOTH:Can't down device hci%d: %s
(%d)",bt_local.di.dev_id, strerror(errno), errno);
                else
                  elog_printf(log_ptr,"BLUETOOTH: cierro hci%d",bt_local.di.dev_id);
                if (ioctl(ctl, HCIDEVUP,bt_local.di.dev_id) < 0)
                {
                  if (errno != EALREADY)
                    elog_printf(log_ptr, "BLUETOOTH:Can't init device hci%d: %s
(%d)",bt_local.di.dev_id, strerror(errno), errno);
                }
                else
                  elog_printf(log_ptr,"BLUETOOTH:inicializo hci%d",bt_local.di.dev_id);
              }
              hci_close_dev(bt_local.socket);
            }
            else
            {
                elog_printf(log_ptr,"BLUETOOTH: no pude abrir socket a dispositivo local
%d!!!",bt_local.di.bdaddr);
              }
            }
            if(conectados>0)
            {
              elog_printf(log_ptr,"BLUETOOTH:en total tengo %d dispositivos
locales",conectados);
              segSleep(&pthread->tw,1,0);
              bt_control.status=BT_DISCOVERY;
            }
            else
            {
              elog_printf(log_ptr,"BLUETOOTH:No encontrÃ© ningÃºn dispositivo BT
local!!!");
              segSleep(&pthread->tw,30,0);
              bt_control.status=BT_HWERROR;
            }
            BtLocalLimpiaLista(bt_control.etiqueta);
          }
          free(bt_control.dl);
          close(ctl);
        }
        break;
      case BT_DISCOVERY:
        len  = 8;   //8 originalmente
        max_rsp = 255;  //255 originalmente
        flags = IREQ_CACHE_FLUSH;
        ii = (inquiry_info*)malloc(max_rsp * sizeof(inquiry_info));
        encontrados=0;
        if(conectados==1)
          pthread_mutex_lock (&pthread->in_mutex);
        bt_local_ptr=BtGetLocalDev();
        num_rsp = hci_inquiry(bt_local_ptr->di.dev_id, len, max_rsp, NULL, &ii, flags);
        if(num_rsp<0)
        {
          elog_printf(log_ptr,"BLUETOOTH:No pude hacer hci inquiry, mando a
BT_HWERROR");
          segSleep(&pthread->tw,1,0);
          bt_control.status=BT_HWERROR;
        }
        else
        {
          if(num_rsp>0)
          {
            if(bt_control.status!=BT_WAIT)
```

```
            bt_control.status=BT_WAIT;
          elog_printf(log_ptr,"BLUETOOTH:encontrÃ© %d dispositivos",num_rsp);
          for(len=0;len<num_rsp;len++)
          {
            memset(direccion_BT,0x0,19);
            ba2str(&(ii+len)->bdaddr,direccion_BT);
            /*revisamos que no tengamos el dispositivo ya insertado en el Ã¡rbol*/
            pthread_mutex_lock (&btree_lock);
            bt_ptr=buscaBTree (&Bluetooth_Btree,direccion_BT);
            pthread_mutex_unlock (&btree_lock);
            if(bt_ptr==NULL)
            {
              memcpy(&cls,&(ii+len)->dev_class,sizeof(cls));
              //primero tenemos que validar que el dispositivo que encontramos en realidad se
un telÃ©fono, si no ni guardes en lista
              if ((cls[1] & 0x1f) >= sizeof(major_devices) / sizeof(*major_devices))
              {
                elog_printf(log_ptr,"BLUEOOTH:Invalid Device Class!");
                continue;
              }
              elog_printf(log_ptr,"BLUETOOTH:  Device  class  :%s  addr:%s",
major_devices[cls[1] & 0x1f],
                  direccion_BT);
              if(strstr(major_devices[cls[1] & 0x1f],"Phone")==NULL)
                continue;
              bt_ptr=(DISPOSITIVO_BT *)malloc(sizeof(DISPOSITIVO_BT));
              memset(&bt,0x0,sizeof(bt));
              memcpy(&bt.target,&(ii+len)->bdaddr,sizeof(bt.target));
              strncpy(bt.direccion_BT,direccion_BT,strlen(direccion_BT));
              segSleep(&pthread->tw,1,0); //??????
              nanosleep (&pthread->tw, NULL);
            //                elog_printf(log_ptr,"BLUETOOTH:  dispositivo  encontrado  dir
%s",bt.direccion_BT);
              memcpy(bt_ptr,&bt,sizeof(DISPOSITIVO_BT));
              LIST_Create(&bt_ptr->lista_publicidad);
            }
            else
            {
              if(bt_ptr->canales_obex_encontrados<0) //ya preguntÃ© y no tiene OBEX,
decremente hasta llegar al timer para volver a preguntar
              {
                bt_ptr->canales_obex_encontrados--;
                if(bt_ptr->canales_obex_encontrados<-15)
                {
                  elog_printf(log_ptr,"BLUETOOTH:Voy a a volver a preguntar por Obex en
dispositivo %s",bt_ptr->direccion_BT);
                  bt_ptr->canales_obex_encontrados=0;
                }
              }

            }
          if(bt_ptr->canales_obex_encontrados==0) //no se si el dispositivo tiene obex o no,
pregunta
          {

            memset(nombre_BT,0x0,250);
            //if          (hci_read_remote_name(bt_control.sock,          &(ii+len)->bdaddr,
sizeof(nombre_BT),nombre_BT, 0) < 0)
            strcpy(nombre_BT, "[unknown]");
            strncpy(bt_ptr->nombre_BT,nombre_BT,strlen(nombre_BT));
            /*una vez encontrada, necesitamos saber si tiene canal OBEX disponible antes de
introducirlo en el Ã¡rbol,
             * para esto utilizamos SDP
             */
            max_rsp=SFBT_OBEXpush_discovery(&bt_ptr->target,bt_ptr-
>canales_obex,bt_local_ptr);
            if(max_rsp<=0)
            {
              elog_printf(log_ptr,"BLUETOOTH: el dispositivo no tiene canal OBEX
asociado");
              bt_ptr->canales_obex_encontrados=-1;
            }
            else
            {
              bt_ptr->canales_obex_encontrados=max_rsp;
              elog_printf(log_ptr,"BLUETOOTH: dispositivo  %s tiene %d canales OBEX
PUSH"
                  ,bt_ptr->direccion_BT,bt_ptr->canales_obex_encontrados);
              for(cuenta=1;cuenta<=bt_ptr->canales_obex_encontrados;cuenta++)
                elog_printf(log_ptr,"BLUETOOTH:canal  obex  push  %d:%d",cuenta,bt_ptr-
>canales_obex[cuenta-1]);
              control_estadisticas.descubiertos++;
            }
            pthread_mutex_lock (&btree_lock);
            inserta_BTree(&Bluetooth_Btree,bt_ptr->direccion_BT,bt_ptr);
            pthread_mutex_unlock (&btree_lock);
            if(bt_ptr->canales_obex_encontrados>0)
            {
              tran=(TRAN_XML *)malloc(sizeof(TRAN_XML));
              memset(tran,0x0,sizeof(TRAN_XML));
              tran->tipo=SOFTEL_XML_DEVICE;
```

```
                strncpy(tran->direccion_BT,bt_ptr->direccion_BT,strlen(bt_ptr->direccion_BT));
                COLA_inserta (&xmlCola,tran);
              }
            }
            else
            {
              elog_printf(log_ptr,"BLUETOOTH:ya tenia registrado a equipo %s canales obex %d",bt_ptr->direccion_BT,bt_ptr->canales_obex_encontrados);
            }
            if(bt_ptr->canales_obex_encontrados>0) //el dispositivo si tiene canales OBEX...avisa a thread OBEX de su presencia
            {
              /*si no tengo publicidad  no puedo enviar nada...pero si puedo descubrir telÃ©fonos para almacenar su info*/
              /*tambiÃ©n valida que la hora este dentro de la de operacion*/
              if((tengo_publicidad==2)&&SFBT_ValidaHora())
              {
                /*aqui tengo que implementar polÃ-ticas de envÃ-o para no ser muy invasivo y no repetir dispositivos que ya tengan promociones,
                 *
                 */
                if(!bt_ptr->rechaza&&!bt_ptr->proceso)
                {
                //     if(strstr(bt_ptr->direccion_BT,"00:21:FC:98:D0:E6")!=NULL||strstr(bt_ptr->direccion_BT,"00:1A:DC:98:FE:79")!=NULL/*
                //                                                                ||strstr(bt_ptr->direccion_BT,"00:1B:AF:F6:F2:5E")!=NULL||strstr(bt_ptr->direccion_BT,"08:00:28:A5:62:C2")!=NULL*/)
                //   {
                    /*revisamos que la publicidad actual no hay sido mandada a ese dispositivo ya*/
                    if(!SftPubRevisa(bt_ptr))
                    {
                      memset(archivo,0x0,sizeof(archivo));
                      pthread_mutex_lock (&pub_lock);
                      sprintf(archivo,"%s/publicidad/%s",path,publicidad_actual->archivo);
                      strcpy(bt_ptr->path,archivo);
                      strcpy(bt_ptr->nombre_archivo,publicidad_actual->archivo);
                      pthread_mutex_unlock (&pub_lock);
                      COLA_inserta (&activosCola,bt_ptr);
                      encontrados++;
                      if(conectados==1)
                      {
                        if (encontrados>7) //no tiene caso tener mas de 7 elementos activos en la cola
                          break;    //ya tenemos la cola llena...no es necesario seguir recorriendo a los dispositivos
                      }
                      else
                      {     //cuando tengamos mÃ¡s de un dispositivo necesitamos validar que todavÃ-a tenemos dispositivos con canales disponibles
                        if(!checa_disponibilidad())
                          break;
                      }
                    }
                    else
                    {
                      elog_printf(log_ptr,"BLUETOOTH: el dispositivo %s ya tiene publicidad %s",
                          bt_ptr->direccion_BT,publicidad_actual->nombre);
                    }

                //  }
                }
              }
            }
          }
        }
        else
        {
          bt_control.status=BT_WAIT;
        }
      }
        free(ii);
        hci_close_dev(bt_local_ptr->socket);
        bt_local_ptr->activo=LOCAL_INACTIVE;
        if(conectados==1)
          pthread_mutex_unlock (&pthread->in_mutex);
      break;
    case BT_WAIT:
      segSleep(&pthread->tw,5,0);
      bt_control.status=BT_INIT;
      break;
    case BT_HWERROR:
      segSleep(&pthread->tw,1,0);
      ctl= socket(AF_BLUETOOTH, SOCK_RAW, BTPROTO_HCI);
      if(ctl>0)
      {
        prueba =(int)LIST_Count(&bt_control.bluetooth_devices);
        for(contador=0;contador<prueba;contador++)
        {
```

```
            if            (LIST_MoveNext(&bt_control.bluetooth_devices,1,(unsigned            long
*)&llave,&ptr)== LIST_SUCCESS)
            {
              bt_local_ptr=(BT_LOCAL *)ptr;
              if(bt_local_ptr->last_dev_id!=-1)
              {
                if (ioctl(ctl, HCIDEVRESET, bt_local_ptr->last_dev_id) < 0 )
                {
                  elog_printf(log_ptr, "BLUETOOTH:Reset fallÃ³ para hci%d:error %d:
%s",bt_local_ptr->last_dev_id,errno, strerror(errno));
                }
                else
                  elog_printf(log_ptr,      "BLUETOOTH:Reset      para      hci%d",bt_local_ptr-
>last_dev_id);
                if (ioctl(ctl, HCIDEVDOWN,bt_local_ptr->last_dev_id) < 0)
                  elog_printf(log_ptr,      "BLUETOOTH:Can't      down      device      hci%d:      %s
(%d)",bt_local_ptr->last_dev_id, strerror(errno), errno);
                else
                  elog_printf(log_ptr,"BLUETOOTH: cierro hci%d",bt_local_ptr->last_dev_id);
                if (ioctl(ctl, HCIDEVUP,bt_local_ptr->last_dev_id) < 0)
                {
                  if (errno != EALREADY)
                    elog_printf(log_ptr,      "BLUETOOTH:Can't      init      device      hci%d:      %s
(%d)",bt_local_ptr->last_dev_id, strerror(errno), errno);
                }
                else
                  elog_printf(log_ptr,"BLUETOOTH:inicializo                  hci%d",bt_local_ptr-
>last_dev_id);
              }
            }
          }
          close(ctl);
        }
        else
          elog_printf(log_ptr,"BLUETOOTH:No pude abrir socket para reseteo!!!");
        bt_control.status=BT_INIT;
        break;
      case BT_FIN:
        break;
      }
      nanosleep (&pthread->tw, NULL);
    }
    elog_printf(log_ptr,"Termina SoftelBluetoothThread!!!");
    return NULL;
}
```

```
/*
 * SoftelObex.c
 *
 *
 *      Author: islas
 */

#include<SoftelObexServer.h>

void SFTOX_lanza_conn(DISPOSITIVO_BT *bt_ptr,BT_LOCAL *bt_local_ptr)
{
  pthread_attr_t attr;
  int contador;
  BT_REMOTO_LOCAL *bt_remloc_ptr;
  pthread_attr_init (&attr);
  pthread_attr_setdetachstate (&attr, PTHREAD_CREATE_JOINABLE);
  //busca la primera conexiÃ³n que no este en uso
  for(contador=0;contador<MAX_CONN;contador++)
  {
    if(ox_control.threads[contador].activa==0)
      break;
  }
  if(contador==MAX_CONN)
    elog_printf(log_ptr,"OBEX:no tengo conexiones disponibles,no debi de haber
entrado!!!!!");
  else
  {
    bt_remloc_ptr=(BT_REMOTO_LOCAL *)malloc(sizeof(BT_REMOTO_LOCAL));
    bt_remloc_ptr->bt_local_ptr=bt_local_ptr;
    bt_remloc_ptr->bt_ptr=bt_ptr;
    bt_remloc_ptr->bt_ptr->proceso=1;
    ox_control.threads[contador].datos=(void *)bt_remloc_ptr;
    ox_control.threads[contador].activa=1;
    ox_control.threads[contador].loop_rxThread = iniThread(&ox_control.threads[contador],
&ox_control.threads[contador].in_mutex,
        &ox_control.threads[contador].apThread,ObexConnThread, &attr);
    if(ox_control.threads[contador].loop_rxThread==1)
    {
      elog_printf(log_ptr,"OBEX:lanzÃ© el thread de conexiÃ³n %d",contador);
    }
    else
    {
      ox_control.threads[contador].activa=0;
    }
  }
}

int checa_disponibilidad(void)
{
  int exito=0;
  int total=0;
  int contador;
  unsigned long ptr,llave;
  BT_LOCAL *bt_local_ptr;
  total =(int)LIST_Count(&bt_control.bluetooth_devices);
  for(contador=0;contador<total;contador++)
  {
    if (LIST_MoveNext(&bt_control.bluetooth_devices,1,(unsigned long *)&llave,&ptr)==
LIST_SUCCESS)
    {
      bt_local_ptr=(BT_LOCAL *)ptr;
      if(bt_local_ptr->activo==LOCAL_INACTIVE)
      {
        exito=1;
        break;
      }
      else if(bt_local_ptr->activo==LOCAL_PUSH&&bt_local_ptr->total_push<7)
      {
        exito=1;
        break;
      }
    }

  }
  return exito;
}

int checa_liberar(void)
{
  int exito=0;
  int contador;
  for(contador=0;contador<MAX_CONN;contador++) //ahora revisamos si podemos liberar
semÃ¡foro
  {
    if(ox_control.threads[contador].activa==1)
    {
      exito=1; //tengo al menos una conexiÃ³n viva, no liberes el semÃ¡foro
      break;
    }
  }
  return exito;
}
```

```
BT_LOCAL *ObexGetLocalDev()  //siempre DEBE regresar al menos un id vÃ¡lido
{
  int total;
  int contador;
  unsigned long ptr,llave;
  BT_LOCAL *bt_local_ptr=NULL;
  total =(int)LIST_Count(&bt_control.bluetooth_devices);
  for(contador=0;contador<total;contador++)
  {
    if  (LIST_MoveNext(&bt_control.bluetooth_devices,1,(unsigned  long  *)&llave,&ptr)==
LIST_SUCCESS)
    {
      bt_local_ptr=(BT_LOCAL *)ptr;
      if(bt_local_ptr->activo==LOCAL_INACTIVE)
      {
        bt_local_ptr->activo=LOCAL_PUSH;
        bt_local_ptr->total_push=1;
        bt_local_ptr->socket=hci_open_dev(bt_local_ptr->di.dev_id);
        elog_printf(log_ptr,"BLUETOOTH:voy a utilizar dispositivo local %s para envio por
primera vez push %d",bt_local_ptr->addr,bt_local_ptr->total_push);
        break;
      }
      else if(bt_local_ptr->activo==LOCAL_PUSH)
      {
        if(bt_local_ptr->total_push<7)
        {
          bt_local_ptr->total_push++;
          elog_printf(log_ptr,"BLUETOOTH:incremente el contador de conexiones push a %d
para dispositivo %s",bt_local_ptr->total_push,bt_local_ptr->addr);
          break;
        }

      }
    }
  }
  return bt_local_ptr;
}

void *SoftelObexThread (void *a)
{
  OBEX_SERVER_THREAD *pthread;
  DISPOSITIVO_BT *bt_ptr;
  BT_LOCAL *bt_local_ptr=NULL;
  pthread=(OBEX_SERVER_THREAD *)a;
  segSleep (&pthread->tw, 1, 0);
  while (pthread->loop_rxThread==0)
  {
    nanosleep (&pthread->tw, NULL);
  }
  elog_printf(log_ptr,"OBEX:arrancado!!!");
  memset(&ox_control.threads,0x0,sizeof(ox_control.threads));
  while(pthread->loop_rxThread)
  {
    bt_ptr=NULL;
    if(checa_disponibilidad())
    {
      bt_ptr=(DISPOSITIVO_BT *)COLA_obtiene (&activosCola);
    }
    if(bt_ptr!=NULL)
    {
      //revisa si tienes conexiones activas, si no...prende el semÃ¡foro
      if(!ox_control.en_proceso)
      {
        pthread_mutex_lock (&threadBT.in_mutex);
        ox_control.en_proceso=1;
        elog_printf(log_ptr,"OBEX:Va a ser la primera conexiÃ³n activa...prendo semÃ¡foro");
      }
      bt_local_ptr=ObexGetLocalDev();
      elog_printf(log_ptr,"OBEX: Voy a enviar archivo a dispositivo  direccion %s",bt_ptr-
>direccion_BT);
      SFTOX_lanza_conn(bt_ptr,bt_local_ptr);
      segSleep (&pthread->tw, 0, 5000);
    }
    if(!checa_liberar()&&ox_control.en_proceso)
    {
      ox_control.en_proceso=0;
      pthread_mutex_unlock (&threadBT.in_mutex);
      elog_printf(log_ptr,"OBEX:Ya no tengo conexiones activas...apago semÃ¡foro");
      segSleep (&pthread->tw, 1, 0);
    }
    nanosleep (&pthread->tw, NULL);
  }
  return NULL;
}
```

```
/*
 * ObexConn.c
 *
 *
 *      Author: islas
 */
#include<SoftelObexServer.h>
#include<openobex/obex.h>
#include<openobex/obex_const.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <stdint.h>

#define EOBEX_OK 0
#define EOBEX_ABORT 1
#define EOBEX_HUP 2
#define EOBEX_PARSE 3

typedef struct{
int serverdone;
int clientdone;
char *get_name; /* Name of last get-request */
int rsp;/* error code */
int opcode;
char *arg; /* response storage place */
uint32_t con_id;/* connection id */
void *private;
}client_context_t;

int get_filesize(const char *filename)
{
  struct stat stats;
  stat(filename, &stats);
  return (int) stats.st_size;
}

uint8_t *easy_readfile(const char *filename, int *file_size)
{
  int actual;
  int fd;
  uint8_t *buf;

  fd = open(filename, O_RDONLY, 0);
  if (fd == -1) {
    return NULL;
  }
  *file_size = get_filesize(filename);
  elog_printf(log_ptr,"OBEX_CONN:name=%s, size=%d", filename, *file_size);
  if (!(buf = malloc(*file_size))) {
    return NULL;
  }

  actual = read(fd, buf, *file_size);
  close(fd);

  *file_size = actual;
  return buf;
}

void request_done(obex_t * handle, obex_object_t * object, int obex_cmd, int obex_rsp)
{
  client_context_t *gt = OBEX_GetUserData(handle);

  elog_printf(log_ptr,"Command (%02x) has now finished, rsp: %02x", obex_cmd, obex_rsp);

  switch (obex_cmd) {
  case OBEX_CMD_DISCONNECT:
    elog_printf(log_ptr,"Disconnect done!");
    OBEX_TransportDisconnect(handle);
    gt->clientdone = 1;
    break;
  case OBEX_CMD_CONNECT:
    elog_printf(log_ptr,"Connected!");
    /* connect_client(handle, object, obex_rsp); */
    gt->clientdone = 1;
    break;
  case OBEX_CMD_GET:
    elog_printf(log_ptr,"*** Warning, getclient commented out");
    /* get_client(handle, object, obex_rsp); */
    gt->clientdone = 1;
    break;
  case OBEX_CMD_PUT:
    /* put_client(handle, object, obex_rsp); */
    gt->clientdone = 1;
    break;
  case OBEX_CMD_SETPATH:
```

```
      elog_printf(log_ptr,"*** Warning, setpath_cleitn commented out");
      /* setpath_client(handle, object, obex_rsp); */
      gt->clientdone = 1;
      break;
    default:
      elog_printf(log_ptr,"Command (%02x) has now finished", obex_cmd);
      break;
    }
}

void obex_event(obex_t * handle, obex_object_t * object, int mode, int event,
      int obex_cmd, int obex_rsp)
{
    client_context_t *gt;
    gt = OBEX_GetUserData(handle);
//    elog_printf(log_ptr,"obex_event: client_context_t = %p", gt);
    switch (event) {
    case OBEX_EV_PROGRESS:
//      elog_printf(log_ptr,"Made some progress...");
      break;
    case OBEX_EV_ABORT:
      elog_printf(log_ptr,"Request aborted!");
      gt->rsp = -EOBEX_ABORT;
      break;
    case OBEX_EV_REQDONE:
      elog_printf(log_ptr,"ReqDone");
      request_done(handle, object, obex_cmd, obex_rsp);
      /* server_done(handle, object, obex_cmd, obex_rsp); */
      break;
    case OBEX_EV_REQHINT:
      /* Accept any command. Not rellay good, but this is a test-program :
       * ) */
      /* OBEX_ObjectSetRsp(object, OBEX_RSP_CONTINUE, OBEX_RSP_SUCCESS); */
      elog_printf(log_ptr,"SetResp");
      break;
    case OBEX_EV_REQ:
      /* server_request(handle, object, event, obex_cmd); */
      elog_printf(log_ptr,"Server request");
      break;
    case OBEX_EV_LINKERR:
      OBEX_TransportDisconnect(handle);
      elog_printf(log_ptr,"Link broken!");
      gt->rsp = -EOBEX_HUP;
      break;
    case OBEX_EV_PARSEERR:
      elog_printf(log_ptr,"Parse error!");
      gt->rsp = -EOBEX_PARSE;
      break;
    default:
      elog_printf(log_ptr,"Unknown event %02x!", event);
      break;
    }
}


int handle_response(obex_t * handle)
{
    int err = 0;
    client_context_t *gt = OBEX_GetUserData(handle);
    gt->clientdone = 0;
    while (!gt->clientdone) {
      if ((err = OBEX_HandleInput(handle, 1)) < 0) {
        elog_printf(log_ptr,"Error while doing OBEX_HandleInput()");
        break;
      }
      err = (gt->rsp == EOBEX_OK) ? 0 : gt->rsp;
    }
    return err;
}

int obex_disconnect(obex_t * handle)
{
    int err;
    obex_object_t *oo;
    client_context_t *gt = OBEX_GetUserData(handle);

    oo = OBEX_ObjectNew(handle, OBEX_CMD_DISCONNECT);
    err = OBEX_Request(handle, oo);
    if (err)
      return err;
    handle_response(handle);
    free(gt);
    return 0;
}

obex_t *__obex_connect(int devid, void *addr,int port, int *err)
{
    obex_t *handle;
    obex_object_t *oo;
    client_context_t *gt;
    gt = malloc(sizeof(client_context_t));
```

```
  if (gt == NULL)
    return NULL;
  memset(gt,0x0, sizeof(client_context_t));
  if (!(handle = OBEX_Init(OBEX_TRANS_BLUETOOTH, obex_event, 0)))
  {
    elog_printf(log_ptr,"OBEX_CONN: OBEX_Init failed: %s", strerror(errno));
    *err = -1;
    return NULL;
  }
  OBEX_SetUserData(handle, gt);
  elog_printf(log_ptr,"OBEX_CONN:voy a abrir la sesiÃ³n de transporte bt");
  *err = BtOBEX_TransportConnect(handle, BDADDR_ANY,addr, (uint8_t)port);
  if (*err<0)
  {
    elog_printf(log_ptr,"OBEX_CONN:Can not Connect to BT Transport Error=%d
%s",errno,strerror(errno));
    *err=errno;
    return NULL;
  }
  elog_printf(log_ptr,"transport connect regresa %d",*err);
  /* create new object */
  oo = OBEX_ObjectNew(handle, OBEX_CMD_CONNECT);
  *err = OBEX_Request(handle, oo);
  if (*err || gt->rsp) {
    *err = *err ? *err: gt->rsp;
    free(gt);
    elog_printf(log_ptr,"OBEX_CONN:__obex_connect: error=%d", *err);
    return NULL;
  }
  *err = handle_response(handle);
  if (*err || gt->rsp) {
    *err = *err ? *err: gt->rsp;
    obex_disconnect(handle);
    elog_printf(log_ptr,"OBEX_CONN:__obex_connect: error=%d", *err);
    return NULL;
  }
  elog_printf(log_ptr,"Connection established");
  return handle;
}


void *ObexConnThread(void *a)
{
  OBEX_CONN_THREAD *pthread;
  BT_REMOTO_LOCAL *bt_remloc_ptr;
  TRAN_XML *tran;
  char *name;
  int name_len,err;
  uint8_t *buf;
  int file_size;
  char *namebuf;
  obex_t *handle;
  obex_object_t *oo;
  obex_headerdata_t hv;
  client_context_t *gt;
  pthread=(OBEX_CONN_THREAD *)a;
  segSleep (&pthread->tw, 1, 0);
  while (pthread->loop_rxThread==0)
  {
    nanosleep (&pthread->tw, NULL);
  }
  bt_remloc_ptr=(BT_REMOTO_LOCAL *)pthread->datos;
  name=bt_remloc_ptr->bt_ptr->nombre_archivo;
  name_len = (strlen(name) + 1) << 1;
  if ((namebuf = malloc(name_len)))
  {
    OBEX_CharToUnicode((uint8_t *) namebuf, (uint8_t *) name, name_len);
  }
  buf = easy_readfile(bt_remloc_ptr->bt_ptr->path, &file_size);
  if (buf == NULL)
  {
      elog_printf(log_ptr,"OBEX_CONN:Can't find file %s", name);
      free(namebuf);
      pthread->activa=0;
      bt_remloc_ptr->bt_local_ptr->total_push--;
      bt_remloc_ptr->bt_ptr->proceso=0;
      if(bt_remloc_ptr->bt_local_ptr->total_push==0)
      {
        elog_printf(log_ptr,"OBEXCONN: ya no tengo mÃ¡s conexiones push para dispositivo
%s, cierro socket",bt_remloc_ptr->bt_local_ptr->addr);
        bt_remloc_ptr->bt_local_ptr->activo=LOCAL_INACTIVE;
        hci_close_dev(bt_remloc_ptr->bt_local_ptr->socket);
      }
      free(bt_remloc_ptr);
      return NULL;
  }
  handle = __obex_connect(bt_remloc_ptr->bt_local_ptr->di.dev_id,&bt_remloc_ptr->bt_ptr-
>target,bt_remloc_ptr->bt_ptr->canales_obex[0], &err);
  if (handle == NULL)
  {
    elog_printf(log_ptr,"OBEX_CONN:Unable to connect to the server %s error
%d",bt_remloc_ptr->bt_ptr->direccion_BT,err);
```

```
    if(err==OBEX_REJECTED_ERROR)
    {
      bt_remloc_ptr->bt_ptr->rechaza=1;
      elog_printf(log_ptr,"OBEX_CONN:bandera rechazo habilitada para %s",bt_remloc_ptr->bt_ptr->direccion_BT);
      pthread_mutex_lock(&threadEstadisticas.in_mutex);
      control_estadisticas.rechazos++;
      pthread_mutex_unlock(&threadEstadisticas.in_mutex);
    }
    else if(err==OBEX_TIMEOUT_ERROR)
    {
      bt_remloc_ptr->bt_ptr->time_out_count++;
      pthread_mutex_lock(&threadEstadisticas.in_mutex);
      control_estadisticas.timeout++;
      pthread_mutex_unlock(&threadEstadisticas.in_mutex);
      elog_printf(log_ptr,"OBEX_CONN:incremento contador de time outs a %d para %s",bt_remloc_ptr->bt_ptr->time_out_count,bt_remloc_ptr->bt_ptr->direccion_BT);
      if(bt_remloc_ptr->bt_ptr->time_out_count==5)
      {
        bt_remloc_ptr->bt_ptr->rechaza=1;
        elog_printf(log_ptr,"OBEX_CONN:cantidad mÃ¡xima de time outs alcanzada!! enciendo rechazo para %s",bt_remloc_ptr->bt_ptr->direccion_BT);
      }
    }
    if(bt_remloc_ptr->bt_ptr->rechaza==1)
    {
      tran=(TRAN_XML *)malloc(sizeof(TRAN_XML));
      memset(tran,0x0,sizeof(TRAN_XML));
      tran->tipo=SOFTEL_XML_REJECT;
      strncpy(tran->direccion_BT,bt_remloc_ptr->bt_ptr->direccion_BT,strlen(bt_remloc_ptr->bt_ptr->direccion_BT));
      COLA_inserta (&xmlCola,tran);
    }
    free(namebuf);
    bt_remloc_ptr->bt_local_ptr->total_push--;
    bt_remloc_ptr->bt_ptr->proceso=0;
    if(bt_remloc_ptr->bt_local_ptr->total_push==0)
    {
      elog_printf(log_ptr,"OBEXCONN: ya no tengo mÃ¡s conexiones push para dispositivo %s, cierro socket",bt_remloc_ptr->bt_local_ptr->addr);
      bt_remloc_ptr->bt_local_ptr->activo=LOCAL_INACTIVE;
      hci_close_dev(bt_remloc_ptr->bt_local_ptr->socket);
    }
    free(bt_remloc_ptr);
    pthread->activa=0;
    return NULL;
  }
  //limpiamos el contador de time outs si es el caso
  if(bt_remloc_ptr->bt_ptr->time_out_count>0)
    bt_remloc_ptr->bt_ptr->time_out_count=0;
  gt = OBEX_GetUserData(handle);
  elog_printf(log_ptr,"obex_push: client_context_t = %p", gt);
  gt->opcode = OBEX_PUSH;
  elog_printf(log_ptr,"Sending file: %s, path: %s, size: %d", name, bt_remloc_ptr->bt_ptr->path, file_size);
  oo = OBEX_ObjectNew(handle, OBEX_CMD_PUT);
  hv.bs = (uint8_t *) namebuf;
  OBEX_ObjectAddHeader(handle, oo, OBEX_HDR_NAME, hv, name_len, 0);
  hv.bq4 = file_size;
  OBEX_ObjectAddHeader(handle, oo, OBEX_HDR_LENGTH, hv, sizeof(uint32_t), 0);
  hv.bs = buf;
  OBEX_ObjectAddHeader(handle, oo, OBEX_HDR_BODY, hv, file_size, 0);
  err = OBEX_Request(handle, oo);
  if (err)
  {
    pthread->activa=0;
    free(namebuf);
    free(buf);
    bt_remloc_ptr->bt_local_ptr->total_push--;
    bt_remloc_ptr->bt_ptr->proceso=0;
    if(bt_remloc_ptr->bt_local_ptr->total_push==0)
    {
      elog_printf(log_ptr,"OBEXCONN: ya no tengo mÃ¡s conexiones push para dispositivo %s, cierro socket",bt_remloc_ptr->bt_local_ptr->addr);
      bt_remloc_ptr->bt_local_ptr->activo=LOCAL_INACTIVE;
      hci_close_dev(bt_remloc_ptr->bt_local_ptr->socket);
    }
    free(bt_remloc_ptr);
    return NULL;
  }
  err = handle_response(handle);  //?? falta validar bien que en realidad el archivo si fue entregado correctamente
  obex_disconnect(handle);
  pthread_mutex_lock(&threadEstadisticas.in_mutex);
  control_estadisticas.envios++;
  pthread_mutex_unlock(&threadEstadisticas.in_mutex);
  //agregramos la publicidad a la lista del dispositivo y a xml
  pthread_mutex_lock (&pub_lock);
  if(LIST_Insert(&bt_remloc_ptr->bt_ptr->lista_publicidad, (unsigned long)publicidad_actual->id,publicidad_actual)!=LIST_SUCCESS)
    elog_printf(log_ptr,"OBEX:No pude agregar publicidad %s a lista de dispositivo %s",publicidad_actual->nombre,bt_remloc_ptr->bt_ptr->direccion_BT);
```

```
  else
  {
    tran=(TRAN_XML *)malloc(sizeof(TRAN_XML));
    memset(tran,0x0,sizeof(TRAN_XML));
    tran->tipo=SOFTEL_XML_FILE;
    strncpy(tran->direccion_BT,bt_remloc_ptr->bt_ptr->direccion_BT,strlen(bt_remloc_ptr-
>bt_ptr->direccion_BT));
    COLA_inserta (&xmlCola,tran);
  }
  pthread_mutex_unlock (&pub_lock);
  free(buf);
  free(namebuf);
  bt_remloc_ptr->bt_local_ptr->total_push--;
  bt_remloc_ptr->bt_ptr->proceso=0;
  if(bt_remloc_ptr->bt_local_ptr->total_push==0)
  {
    elog_printf(log_ptr,"OBEXCONN: ya no tengo mÃ¡s conexiones push para dispositivo
%s, cierro socket",bt_remloc_ptr->bt_local_ptr->addr);
    bt_remloc_ptr->bt_local_ptr->activo=LOCAL_INACTIVE;
    hci_close_dev(bt_remloc_ptr->bt_local_ptr->socket);
  }
  else
    elog_printf(log_ptr,"OBEXCONN: dispositivo %s, tiene %d conexiones push
pendiente",bt_remloc_ptr->bt_local_ptr->addr,bt_remloc_ptr->bt_local_ptr->total_push);
  free(bt_remloc_ptr);
  pthread->activa=0;
  return NULL;
}
```

```
/*
 * SoftelEstadisticas.c
 *      Author: islas
 */
#include<SoftelObexServer.h>
int SftEstCambioHora(void)
{
  int exito=0;
  char fecha[20];
  char hora1[3];
  char hora2[3];
  char *ptr1;
  char *ptr2;
  memset(fecha,0x0,20);
  memset(hora1,0x0,3);
  memset(hora2,0x0,3);
  //2009-01-26 17:28:18
  getlocaltime(fecha);
  ptr1=strstr(control_estadisticas.fecha_inicio," ");
  ptr2=strstr(ptr1,":");
  memcpy(hora1,ptr1+1,strlen(ptr1+1)-strlen(ptr2));
  ptr1=strstr(fecha," ");
  ptr2=strstr(ptr1,":");
  memcpy(hora2,ptr1+1,strlen(ptr1+1)-strlen(ptr2));
  if(strcmp(hora1,hora2))
  {
    strcpy(control_estadisticas.fecha_termino,fecha);
    elog_printf(log_ptr,"EST: fecha tÃ©rmino %s",fecha);
    exito=1;
  }
  return exito;
}
void *SoftelEstadisticasThread(void *a)
{
  OBEX_SERVER_THREAD *pthread;
  pthread=(OBEX_SERVER_THREAD *)a;
  segSleep (&pthread->tw, 1, 0);
  while (pthread->loop_rxThread==0)
  {
    nanosleep (&pthread->tw, NULL);
  }
  elog_printf(log_ptr,"Estadisticas:arrancado!!!");
  while(pthread->loop_rxThread)
  {
    switch(control_estadisticas.status)
    {
      case ESTADISTICAS_INIT:
        pthread_mutex_lock(&pthread->in_mutex);
        memset(control_estadisticas.fecha_inicio,0x0,20);
        memset(control_estadisticas.fecha_termino,0x0,20);
        control_estadisticas.envios=0;
        control_estadisticas.rechazos=0;
        control_estadisticas.timeout=0;
        control_estadisticas.descubiertos=0;
        getlocaltime(control_estadisticas.fecha_inicio);
        control_estadisticas.status=ESTADISTICAS_WAIT;
        pthread_mutex_unlock(&pthread->in_mutex);
        segSleep (&pthread->tw, 10, 0);
        break;
      case ESTADISTICAS_WAIT:
        if(SftEstCambioHora())
        {
          control_estadisticas.status=ESTADISTICAS_COMMIT;
          elog_printf(log_ptr,"EST: Cambio hora detectado inicio %s fin %s...voy a escribir
corte",
              control_estadisticas.fecha_inicio,control_estadisticas.fecha_termino);
          segSleep (&pthread->tw, 1, 0);
        }
        break;
      case ESTADISTICAS_COMMIT:
        pthread_mutex_lock(&pthread->in_mutex);
        if(SftXmlAgregaCorte())
          elog_printf(log_ptr,"EST:corte agregado!!!");
        else
          elog_printf(log_ptr,"EST: no pude agregar corte!!!");
        memset(control_estadisticas.fecha_inicio,0x0,20);
        memset(control_estadisticas.fecha_termino,0x0,20);
        control_estadisticas.envios=0;
        control_estadisticas.rechazos=0;
        control_estadisticas.timeout=0;
        control_estadisticas.descubiertos=0;
        getlocaltime(control_estadisticas.fecha_inicio);
        control_estadisticas.status=ESTADISTICAS_WAIT;
        segSleep (&pthread->tw, 10, 0);
        pthread_mutex_unlock(&pthread->in_mutex);
        break;
    }
    nanosleep (&pthread->tw, NULL);
  }
  return NULL;
}
```

```
/*
 * SoftelPublicidad.c
 *
 *
 *      Author: islas
 */
#include<SoftelObexServer.h>
int SftPubRevisa(void *ptr)
{
  DISPOSITIVO_BT *bt_ptr;
  PUBLICIDAD_BT *pt;
  int exito=0;
  int total;
  int contador;
  bt_ptr=(DISPOSITIVO_BT *)ptr;
  total =(int)LIST_Count(&bt_ptr->lista_publicidad);
  if(!total)
    return exito;
  pthread_mutex_lock (&pub_lock);
  for(contador=0;contador<total;contador++)
  {
    if    (LIST_MoveNext(&bt_ptr->lista_publicidad,1,(unsigned    long    *)&key,&pointer)==
LIST_SUCCESS)
    {
      pt=(PUBLICIDAD_BT *)pointer;
      if(!strcmp(pt->archivo,publicidad_actual->archivo))
      {
        exito=1;
        break;
      }      }    }
  pthread_mutex_unlock (&pub_lock);
  return exito;
}
int SftPubAsigna(PUBLICIDAD_BT **pub_ptr)
{
  int exito=0;
  int total,contador;
  PUBLICIDAD_BT *pt;
  total =(int)LIST_Count(&lista_publicidad);
  if(total<1)
    return exito;
  else
  {
    exito=1;
    pthread_mutex_lock(&pub_lock);
    for(contador=0;contador<total;contador++)
    {
      if      (LIST_MoveNext(&lista_publicidad,1,(unsigned      long      *)&key,&pointer)==
LIST_SUCCESS)
      {
        pt=(PUBLICIDAD_BT *)pointer;
        if(*pub_ptr==NULL)  //es la primera vez, asigna la primera
        {
          *pub_ptr=(PUBLICIDAD_BT *)pointer;
          break;
        }
        else
        {
          if(strcmp(pt->archivo,(*pub_ptr)->archivo))
          {
            *pub_ptr=(PUBLICIDAD_BT *)pointer;
            elog_printf(log_ptr,"PUB:publicidad    asignada    id    %d    nombre    %s",(*pub_ptr)-
>id,(*pub_ptr)->nombre);
            exito=2;
            break;
          }
        }
      }
    }
    pthread_mutex_unlock(&pub_lock);
  }
  return exito;
}
void *SoftelPublicidadThread (void *a)
{
  OBEX_SERVER_THREAD *pthread;
  pthread=(OBEX_SERVER_THREAD *)a;
  time_t secs;
  segSleep (&pthread->tw, 1, 0);
  int valor;
  while (pthread->loop_rxThread==0)
  {
    nanosleep (&pthread->tw, NULL);
  }
  while(pthread->loop_rxThread)
  {
    switch(pub_control.status)
    {
    case PUB_INIT:
      if(tengo_publicidad==1)  //implica  que  xml  ya  leyÃ³  publicidad  y  me  llenÃ³  la
lista...manda asignar laprimera
      {
```

```
          publicidad_actual=NULL;
          pub_control.status=PUB_SET;
        }
        break;
      case PUB_SET:
        valor=SftPubAsigna(&publicidad_actual);
        if(valor==0)
        {
          elog_printf(log_ptr,"PUB:no pude asignar publicidad, lista vacÃ-a!!!!");
        }
        else
        {
          if(tengo_publicidad!=2) // ya tengo una publicidad asignada como la actual...avisa a
bluetooth
            tengo_publicidad=2;
          if(valor==2)
            elog_printf(log_ptr,"PUB:cambio publicidad!!!");
          else
            elog_printf(log_ptr,"PUB:no encontrÃ© otra publicidad...dejo la que tenÃ-a!!");
        }
        pub_control.status=PUB_WAIT;
        break;
      case PUB_WAIT:
        secs=publicidad_actual->horas;
        if(secs>24)
          secs=24;
        secs=secs*3600;
        segSleep (&pthread->tw,secs, 0);
        nanosleep (&pthread->tw, NULL);
        pub_control.status=PUB_SET;
        elog_printf(log_ptr,"PUB:voy a cambiar publicidad!!!");
        segSleep (&pthread->tw, 1, 0);
        break;
      }
      nanosleep (&pthread->tw, NULL);
    }
    return NULL;
}
```

```
/*
 * SoftelXml.c
 *
 *
 *      Author: islas
 */
#include<SoftelObexServer.h>
xmlChar *ConvertInput(const char *in, const char *encoding)
{
   xmlChar *out;
   int ret;
   int size;
   int out_size;
   int temp;
   xmlCharEncodingHandlerPtr handler;
   if (in == 0)
      return 0;
   handler = xmlFindCharEncodingHandler(encoding);
   if (!handler)
   {
      elog_printf(log_ptr,"XML:ConvertInput: no encoding handler found for '%s'",
            encoding ? encoding : "");
      return 0;
   }
   size = (int) strlen(in) + 1;
   out_size = size * 2 - 1;
   out = (unsigned char *) xmlMalloc((size_t) out_size);
   if (out != 0)
   {
      temp = size - 1;
      ret = handler->input(out, &out_size, (const xmlChar *) in, &temp);
      if ((ret < 0) || (temp - size + 1))
      {
         if (ret < 0)
         {
            elog_printf(log_ptr,"XML:ConvertInput: conversion wasn't successful.");
         }
         else
         {
            elog_printf(log_ptr,"XML:ConvertInput: conversion wasn't successful. converted:
%i octets.\n",temp);
         }
         xmlFree(out);
         out = 0;
      }
      else
      {
         out = (unsigned char *) xmlRealloc(out, out_size + 1);
         out[out_size] = 0;  /*null terminating out */
      }
   }
   else
   {
      elog_printf(log_ptr,"XML:ConvertInput: no mem");
   }
   return out;
}

xmlTextReaderPtr SftXmlCreateFile(void)
{
  int rc;
  xmlTextReaderPtr ptr;
  xmlTextWriterPtr writer;
  xmlChar *tmp;
  char archivo[150];
  /* Load XML document */
  memset(archivo,0x0,sizeof(archivo));
  sprintf(archivo,"%s/dispositivos.xml",path);
  writer = xmlNewTextWriterFilename("dispositivos.xml", 0);
  if (writer == NULL)
  {
        elog_printf(log_ptr,"XML:No pude crear xml writer");
       return NULL;
  }
  rc = xmlTextWriterStartDocument(writer, NULL, MY_ENCODING, NULL);
  if (rc < 0)
  {
    elog_printf(log_ptr,"XML: Error en xmlTextWriterStartDocument");
     return NULL;
  }
  rc = xmlTextWriterStartElement(writer, BAD_CAST "DISPOSITIVOS_BT");
  if (rc < 0)
  {
    elog_printf(log_ptr,"XML:testXmlwriterFilename: Error at xmlTextWriterStartElement");
    return NULL;
  }
  tmp = ConvertInput("Tabla para almacenar la informaciÃ³n de todos los dispositivos
encontrados",MY_ENCODING);
  rc = xmlTextWriterWriteComment(writer, tmp);
  if (rc < 0)
  {
```

```
    elog_printf(log_ptr,"XML:testXmlwriterFilename: Error at
xmlTextWriterWriteComment");
    return NULL;
  }
  if (tmp != NULL)
    xmlFree(tmp);
  rc = xmlTextWriterEndDocument(writer);
  if (rc < 0)
  {
    elog_printf(log_ptr,"XML:testXmlwriterFilename: Error at
xmlTextWriterEndDocument");
    return NULL;
  }
  xmlFreeTextWriter(writer);
  ptr=xmlReaderForFile(archivo, NULL, 0);
  return ptr;
}
void SftXmlDoFile(char *nombre,char *raiz,const char *comment)
{
  int rc;

  xmlTextWriterPtr writer;
  xmlChar *tmp;
  char archivo[150];
  /* Load XML document */
  memset(archivo,0x0,sizeof(archivo));
  sprintf(archivo,"%s/%s",path,nombre);
  writer = xmlNewTextWriterFilename(archivo, 0);
  if (writer == NULL)
  {
       elog_printf(log_ptr,"XML:No pude crear xml writer");
       return;
  }
  rc = xmlTextWriterStartDocument(writer, NULL, MY_ENCODING, NULL);
  if (rc < 0)
  {
    elog_printf(log_ptr,"XML: Error en xmlTextWriterStartDocument");
     return;
  }
  rc = xmlTextWriterStartElement(writer, BAD_CAST raiz);
  if (rc < 0)
  {
    elog_printf(log_ptr,"XML:testXmlwriterFilename: Error at xmlTextWriterStartElement");
    return;
  }
  tmp = ConvertInput(comment,MY_ENCODING);
  rc = xmlTextWriterWriteComment(writer, tmp);
  if (rc < 0)
  {
    elog_printf(log_ptr,"XML:testXmlwriterFilename: Error at
xmlTextWriterWriteComment");
    return;
  }
  if (tmp != NULL)
    xmlFree(tmp);
  rc = xmlTextWriterEndDocument(writer);
  if (rc < 0)
  {
    elog_printf(log_ptr,"XML:testXmlwriterFilename: Error at
xmlTextWriterEndDocument");
  }
  xmlFreeTextWriter(writer);
  return;
}
void SftXmlCheckFiles(void)
{
  xmlDocPtr doc=NULL;
  char archivo[150];
  char nombre[50];
  char raiz[50];
  /*primero revisa que exista el archivo de dispositivos*/
  memset(archivo,0x0,sizeof(archivo));
  sprintf(archivo,"%s/dispositivos.xml",path);
  doc = xmlParseFile(archivo);
  if(doc==NULL)
  {
    memset(nombre,0x0,50);
    strcpy(nombre,"dispositivos.xml");
    memset(raiz,0x0,50);
    strcpy(raiz,"DISPOSITIVOS_BT");
    SftXmlDoFile(nombre,raiz,"Tabla para almacenar la informaciÃ³n de todos los
dispositivos BT descubiertos");
  }
  if(doc!=NULL)
    xmlFreeDoc(doc);
  /*primero revisa que exista el archivo disp_pub.xml*/
  memset(archivo,0x0,sizeof(archivo));
  sprintf(archivo,"%s/disp_pub.xml",path);
  doc = xmlParseFile(archivo);
  if(doc==NULL)
  {
    memset(nombre,0x0,50);
```

```
    strcpy(nombre,"disp_pub.xml");
    memset(raiz,0x0,50);
    strcpy(raiz,"DISP_PUB");
    SftXmlDoFile(nombre,raiz,"Tabla para almacenar la informaciÃ³n de todos las publicidades enviadas a los dispositivos");
  }
  if(doc!=NULL)
    xmlFreeDoc(doc);
  /*ahora el archivo estadisticas_generales.xml*/
  memset(archivo,0x0,sizeof(archivo));
  sprintf(archivo,"%s/estadisticas_generales.xml",path);
  doc = xmlParseFile(archivo);
  if(doc==NULL)
  {
    memset(nombre,0x0,50);
    strcpy(nombre,"estadisticas_generales.xml");
    memset(raiz,0x0,50);
    strcpy(raiz,"ESTADISTICAS");
    SftXmlDoFile(nombre,raiz,"Ã□rbol xml para almacenar la informacion de las estadisticas generales de envios");
  }
  if(doc!=NULL)
    xmlFreeDoc(doc);
  /*ahora el archivo estadisticas_pub.xml*/
  memset(archivo,0x0,sizeof(archivo));
  sprintf(archivo,"%s/estadisticas_pub.xml",path);
  doc = xmlParseFile(archivo);
  if(doc==NULL)
  {
    memset(nombre,0x0,50);
    strcpy(nombre,"estadisticas_pub.xml");
    memset(raiz,0x0,50);
    strcpy(raiz,"ESTADISTICAS_PUB");
    SftXmlDoFile(nombre,raiz,"Ã□rbol xml para almacenar la informacion de las estadisticas de envÃ-o de las publicidades existentes");
  }
  if(doc!=NULL)
    xmlFreeDoc(doc);
}
void SftXmlAgregaPubDisp(void *pt)
{
  DISPOSITIVO_BT *bt_ptr;
  xmlDocPtr doc=NULL;
  xmlNodePtr raiz=NULL,disp=NULL,elemento=NULL,pub=NULL,file=NULL,fecha=NULL;
  char archivo[150];
  char string_fecha[20];
  int encontre_disp=0;
  bt_ptr=(DISPOSITIVO_BT *)pt;
  memset(archivo,0x0,sizeof(archivo));
  memset(string_fecha,0x0,sizeof(string_fecha));
  sprintf(archivo,"%s/disp_pub.xml",path);
  doc = xmlParseFile(archivo);
  if (doc == NULL)
  {
    elog_printf(log_ptr, "XML: No pude abrir archivo disp_pub para agregar pub");
    xmlFreeDoc(doc);
    return;
  }
  raiz = xmlDocGetRootElement(doc);
  if (raiz == NULL)
  {
    elog_printf(log_ptr,"XML:documento vacÃ-o para escritura de disp_pub!!!");
    xmlFreeDoc(doc);
    return;
  }
  if (xmlStrcmp(raiz->name, (xmlChar *) "DISP_PUB"))
  {
    elog_printf(log_ptr,"XML:document of the wrong type, root node != DISP_PUB");
    xmlFreeDoc(doc);
    return;
  }
  for(disp=raiz->children;disp!=NULL;disp=disp->next)
  {
    if(!xmlStrcmp(disp->name, (xmlChar *) "DISPOSITIVO"))
    {
      encontre_disp=0;
      for(elemento=disp->children;elemento!=NULL;elemento=elemento->next)
      {
        if(!xmlStrcmp(elemento->children->content,(xmlChar *)bt_ptr->direccion_BT))
        {
          encontre_disp=1;
          pub= xmlNewChild(disp, NULL,(xmlChar *)"PUBLICIDAD", NULL);
          file=xmlNewTextChild(pub,NULL,(xmlChar *)"ARCHIVO",(xmlChar *)publicidad_actual->archivo);
          getlocaltime(string_fecha);
          fecha=xmlNewTextChild(pub,NULL,(xmlChar *)"FECHA",(xmlChar *)string_fecha);
          break;
        }
      }
    }
```

```
    if(encontre_disp==1)
    {
      elog_printf(log_ptr,"XML: agregue archivo publicidad %s para dispositivo
%s",publicidad_actual->archivo,bt_ptr->direccion_BT);
      break;
    }
  }
  if(!encontre_disp)//no estaba el dispositivo, lo agrego con su publicidad
  {
    disp=elemento=NULL;
    disp= xmlNewChild(raiz, NULL,(xmlChar *)"DISPOSITIVO", NULL);
    elemento=xmlNewTextChild(disp,NULL,(xmlChar  *)"DIRECCION",(xmlChar  *)bt_ptr-
>direccion_BT);
    pub= xmlNewChild(disp, NULL,(xmlChar *)"PUBLICIDAD", NULL);
    file=xmlNewTextChild(pub,NULL,(xmlChar                *)"ARCHIVO",(xmlChar
*)publicidad_actual->archivo);
    getlocaltime(string_fecha);
    fecha=xmlNewTextChild(pub,NULL,(xmlChar *)"FECHA",(xmlChar *)string_fecha);
  }
  xmlSaveFormatFile (archivo, doc, 0);
  xmlFreeDoc(doc);
}

int SftXmlAgregaCorte(void)
{
  xmlDocPtr doc=NULL;
  xmlNodePtr
raiz=NULL,corte=NULL,intervalo=NULL,aceptados=NULL,rechazados=NULL,ignorados=N
ULL,descubiertos=NULL;
  char interval[43];
  char archivo[150];
  char numero[9];
  int exito=0;
  /* Load XML document */
  memset(archivo,0x0,150);
  memset(interval,0x0,43);
  sprintf(archivo,"%s/estadisticas_generales.xml",path);
  doc = xmlParseFile(archivo);
  if (doc == NULL)
  {
    elog_printf(log_ptr, "XML: No pude abrir archivo estadisticas generales");
    xmlFreeDoc(doc);
    return exito;
  }
  raiz = xmlDocGetRootElement(doc);
  if (raiz == NULL)
  {
    elog_printf(log_ptr,"XML:documento vacÃ-o para agregar corte!!!");
    xmlFreeDoc(doc);
    return exito;
  }
  if (xmlStrcmp(raiz->name, (xmlChar *) "ESTADISTICAS"))
  {
    elog_printf(log_ptr,"document of the wrong type, root node != ESTADISTICAS");
    xmlFreeDoc(doc);
    return exito;
  }
  corte = xmlNewChild(raiz, NULL,(xmlChar *)"CORTE", NULL);

sprintf(interval,"%s,%s",control_estadisticas.fecha_inicio,control_estadisticas.fecha_termino);
  intervalo=xmlNewTextChild(corte,NULL,(xmlChar *)"INTERVALO",(xmlChar *)interval);
  memset(numero,0x0,9);
  sprintf(numero,"%d",control_estadisticas.envios);
  aceptados=xmlNewTextChild(corte,NULL,(xmlChar                *)"ACEPTADOS",(xmlChar
*)numero);
  memset(numero,0x0,9);
  sprintf(numero,"%d",control_estadisticas.rechazos);
  rechazados=xmlNewTextChild(corte,NULL,(xmlChar             *)"RECHAZADOS",(xmlChar
*)numero);
  memset(numero,0x0,9);
  sprintf(numero,"%d",control_estadisticas.timeout);
  ignorados=xmlNewTextChild(corte,NULL,(xmlChar                *)"IGNORADOS",(xmlChar
*)numero);
  memset(numero,0x0,9);
  sprintf(numero,"%d",control_estadisticas.descubiertos);
  descubiertos=xmlNewTextChild(corte,NULL,(xmlChar        *)"DESCUBIERTOS",(xmlChar
*)numero);
  xmlSaveFormatFile (archivo, doc, 0);
  xmlFreeDoc(doc);
  exito=1;
  return exito;
}

void SftXmlAgregaEstPub(void *pbt)
{
  PUBLICIDAD_BT *pbt_ptr;
  xmlDocPtr doc=NULL;
  xmlNodePtr
raiz=NULL,disp=NULL,elemento=NULL,pub=NULL,file=NULL,envio=NULL;
  char archivo[150];
  int encontre_pub=0,exito=0;
  long valor=0;
```

```
  char valor_string[10];
  pbt_ptr=(PUBLICIDAD_BT *)pbt;
  memset(valor_string,0x0,sizeof(valor_string));
  sprintf(archivo,"%s/estadisticas_pub.xml",path);
  doc = xmlParseFile(archivo);
  if (doc == NULL)
  {
    elog_printf(log_ptr, "XML: No pude abrir archivo estadisticas_pub.xml para agregar estadisitica");
    xmlFreeDoc(doc);
    return;
  }
  raiz = xmlDocGetRootElement(doc);
  if (raiz == NULL)
  {
    elog_printf(log_ptr,"XML:documento vacÃ-o para escritura de estadistica publicidad!!!");
    xmlFreeDoc(doc);
    return;
  }
  if (xmlStrcmp(raiz->name, (xmlChar *) "ESTADISTICAS_PUB"))
  {
    elog_printf(log_ptr,"XML:document of the wrong type, root node != ESTADISITICAS_PUB");
    xmlFreeDoc(doc);
    return;
  }
  for(disp=raiz->children;disp!=NULL;disp=disp->next)
  {
    if(!xmlStrcmp(disp->name, (xmlChar *) "PUBLICIDAD"))
    {
      encontre_pub=0;
      for(elemento=disp->children;elemento!=NULL;elemento=elemento->next)
      {
        if(!xmlStrcmp(elemento->children->content,(xmlChar *)pbt_ptr->nombre))
        {
          encontre_pub=1;
        }
        else if(!xmlStrcmp(elemento->name,(xmlChar *)"ENVIOS"))
        {
          if(encontre_pub)
          {
            valor=atol((char *)xmlNodeGetContent(elemento));
            valor++;
            if(valor<999999999)
              sprintf(valor_string,"%ld",valor);
            xmlNodeSetContent(elemento->children,(xmlChar *)valor_string);
            exito=1;
            break;
          }
        }
      }
    }
    if(exito==1)
    {
      elog_printf(log_ptr,"XML: aumento estadÃ-sitca a %ld para publicidad %s",valor,pbt_ptr->nombre);
      break;
    }
  }
  if(!exito)//no estaba el dispositivo, lo agrego con su publicidad
  {
    disp=NULL;
    disp= xmlNewChild(raiz, NULL,(xmlChar *)"PUBLICIDAD", NULL);
    pub=xmlNewTextChild(disp,NULL,(xmlChar *)"NOMBRE",(xmlChar *)pbt_ptr->nombre);
    file=xmlNewTextChild(disp,NULL,(xmlChar *)"ARCHIVO",(xmlChar *)pbt_ptr->archivo);
    envio=xmlNewTextChild(disp,NULL,(xmlChar *)"ENVIOS",(xmlChar *)"1");
  }
  xmlSaveFormatFile (archivo, doc, 0);
  xmlFreeDoc(doc);
}

void SftXmlDispPub(DISPOSITIVO_BT *bt_ptr)
{
  xmlDocPtr doc=NULL;
  xmlNodePtr raiz=NULL,disp=NULL,elemento=NULL,elemento1=NULL;
  int encontre_disp;
  char archivo[150];
  char file[50];
  unsigned long pointer, key;
  int total_publicidades;
  int contador;
  PUBLICIDAD_BT *pt;
  /* Load XML document */
  memset(archivo,0x0,sizeof(archivo));
  sprintf(archivo,"%s/disp_pub.xml",path);
  doc = xmlParseFile(archivo);
  if (doc == NULL)
  {
    elog_printf(log_ptr, "XML: No pude abrir archivo disp_pub para parseo");
    xmlFreeDoc(doc);
```

```
    return;
  }
  raiz = xmlDocGetRootElement(doc);
  if (raiz == NULL)
  {
    elog_printf(log_ptr,"XML:documento vacÃ-o para lectura de disp_pub!!!");
    xmlFreeDoc(doc);
    return;
  }
  if (xmlStrcmp(raiz->name, (xmlChar *) "DISP_PUB"))
  {
    elog_printf(log_ptr,"XML:document of the wrong type, root node != DISP_PUB");
    xmlFreeDoc(doc);
    return;
  }
  for(disp=raiz->children;disp!=NULL;disp=disp->next)
  {
    if(!xmlStrcmp(disp->name, (xmlChar *) "DISPOSITIVO"))
    {
      encontre_disp=0;
      for(elemento=disp->children;elemento!=NULL;elemento=elemento->next)
      {
        if(!xmlStrcmp(elemento->children->content,(xmlChar *)bt_ptr->direccion_BT))
        {
          encontre_disp=1;
        }
        else if(!xmlStrcmp(elemento->name,(xmlChar *)"PUBLICIDAD"))
        {
          if(encontre_disp)
          {
            total_publicidades =(int)LIST_Count(&lista_publicidad);
            //solo voy a llenar la lista de publicidad del dispositivo con publicidades exisitentes
            if(total_publicidades>0)
            {
              memset(file,0x0,sizeof(file));
              for(elemento1=elemento->children;elemento1!=NULL;elemento1=elemento1-
>next)
              {
                if(!xmlStrcmp(elemento1->name,(xmlChar *)"ARCHIVO"))
                {
                  strncpy(file,(char   *)elemento1->children->content,strlen((char   *)elemento1-
>children->content));
                  pthread_mutex_lock(&pub_lock);
                  for(contador=0;contador<total_publicidades;contador++)
                  {
                    if              (LIST_MoveNext(&lista_publicidad,1,(unsigned              long
*)&key,&pointer)== LIST_SUCCESS)
                    {
                      pt=(PUBLICIDAD_BT *)pointer;
                      if(!strcmp(pt->archivo,file))
                      {
                        if(LIST_Insert(&bt_ptr->lista_publicidad,          (unsigned          long)pt-
>id,pt)==LIST_SUCCESS)
                        {
                          elog_printf(log_ptr,"AgreguÃ© publicidad id %d nombre %s archivo
%s a lista dispositivo %s",
                                      pt->id,pt->nombre,pt->archivo,bt_ptr->direccion_BT);
                        }
                      }
                    }
                  }
                  pthread_mutex_unlock(&pub_lock);
                }
              }
            }
          }
        }
      }
    }
  }

  int SftXmlParser(void)
  {
    xmlDocPtr doc=NULL;
    xmlNodePtr raiz=NULL,disp=NULL,elemento=NULL;
    int completo=0;
    DISPOSITIVO_BT bt;
    DISPOSITIVO_BT *bt_ptr;
    char archivo[150];
    /* Load XML document */
    memset(archivo,0x0,sizeof(archivo));
    sprintf(archivo,"%s/dispositivos.xml",path);
    doc = xmlParseFile(archivo);
    if (doc == NULL)
    {
      elog_printf(log_ptr, "XML: No pude abrir archivo para parseo");
      xmlFreeDoc(doc);
      return 0;
    }
    raiz = xmlDocGetRootElement(doc);
```

```
  if (raiz == NULL)
  {
    elog_printf(log_ptr,"XML:documento vacÃ-o para lectura de elementos!!!");
    xmlFreeDoc(doc);
    return 0;
  }
  if (xmlStrcmp(raiz->name, (xmlChar *) "DISPOSITIVOS_BT"))
  {
    elog_printf(log_ptr,"XML:document of the wrong type, root node != DISPOSITIVOS_BT");
    xmlFreeDoc(doc);
    return 0;
  }
  for(disp=raiz->children;disp!=NULL;disp=disp->next)
  {
    if(!xmlStrcmp(disp->name, (xmlChar *) "DISPOSITIVO"))
    {
      memset(&bt,0x0,sizeof(bt));
      completo=0;
      for(elemento=disp->children;elemento!=NULL;elemento=elemento->next)
      {
        if(!xmlStrcmp(elemento->name, (xmlChar *) "DIRECCION"))
        {
          strncpy(bt.direccion_BT,(char *)elemento->children->content,strlen((char *)elemento->children->content));
          completo=1;
        }
        else if(!xmlStrcmp(elemento->name, (xmlChar *) "CANAL_OBEX"))
        {
          bt.canales_obex[0]=atoi((char *)elemento->children->content);
          bt.canales_obex_encontrados=1;
          if(completo==1)
            completo=2;
        }
        else if(!xmlStrcmp(elemento->name, (xmlChar *) "RECHAZA"))
        {
          bt.rechaza=atoi((char *)elemento->children->content);
          if(completo==2)
            completo=3;
        }
        if(completo==3) //tengo la info del dispositivo...guarda en la cola
        {
          bt_ptr=(DISPOSITIVO_BT *)malloc(sizeof(DISPOSITIVO_BT));
          memcpy(bt_ptr,&bt,sizeof(DISPOSITIVO_BT));
          str2ba(bt_ptr->direccion_BT,&bt_ptr->target);
          LIST_Create(&bt_ptr->lista_publicidad);
          elog_printf(log_ptr,"XML:leo de archivo dispositivo %s canal obex %d rechaza %d"
              ,bt_ptr->direccion_BT,bt_ptr->canales_obex[0],bt_ptr->rechaza);
          SftXmlDispPub(bt_ptr);
          pthread_mutex_lock (&btree_lock);
          inserta_BTree(&Bluetooth_Btree,bt_ptr->direccion_BT,bt_ptr);
          pthread_mutex_unlock (&btree_lock);
        }
      }
    }
  }
  xmlFreeDoc(doc);
  return 1;
}

int SftXmlConfig(void)
{
  xmlDocPtr doc=NULL;
  xmlNodePtr raiz=NULL,elemento=NULL;
  char archivo[150];
  int len;
  /* Load XML document */
  memset(archivo,0x0,150);
  memset(&config,0x0,sizeof(config));
  sprintf(archivo,"%s/configuracion.xml",path);
  doc = xmlParseFile(archivo);
  if (doc == NULL)
  {
    elog_printf(log_ptr, "XML: No pude parsear archivo configuracion");
    xmlFreeDoc(doc);
    return 0;
  }
  raiz = xmlDocGetRootElement(doc);
  if (raiz == NULL)
  {
    elog_printf(log_ptr,"XML:documento vacÃ-o para leer configuracion!!!");
     xmlFreeDoc(doc);
     return 0;
  }
  if (xmlStrcmp(raiz->name, (xmlChar *) "CONFIGURACION"))
  {
    elog_printf(log_ptr,"document of the wrong type, root node != CONFIGURACION");
     xmlFreeDoc(doc);
     return 0;
  }
  for(elemento=raiz->children;elemento!=NULL;elemento=elemento->next)
  {
```

```
    if(!xmlStrcmp(elemento->name, (xmlChar *) "ID_CLIENTE"))
      config.cliente=atoi((char *)elemento->children->content);
    else if(!xmlStrcmp(elemento->name, (xmlChar *) "ID_DISP"))
      config.dispositivo=atoi((char *)elemento->children->content);
    else if(!xmlStrcmp(elemento->name, (xmlChar *) "SERVIDOR"))
      strncpy(config.servidor,(char *)elemento->children->content,strlen((char *)elemento-
>children->content));
    else if(!xmlStrcmp(elemento->name, (xmlChar *) "NOMBRE"))
    {
      len=strlen((char *)elemento->children->content);
      if(len>=100)
        len=99;
      strncpy(config.nombre,(char *)elemento->children->content,len);
    }
    else if(!xmlStrcmp(elemento->name, (xmlChar *) "HORARIO"))
    {
      if(strlen((char *)elemento->children->content)!=5)
        elog_printf(log_ptr,"XML:Error formato horario en configuracion, espero XX-XX");
      else
      {
        strncpy(config.hora_inicio,(char *)elemento->children->content,2);
        strncpy(config.hora_fin,(char *)&elemento->children->content[3],2);
      }
    }
  }
  elog_printf(log_ptr,"MAIN: ConfiguraciÃ³n cargada: cliente %d dispositivo %d nombre %s
servidor %s hora inicio %s, hora fin %s",

config.cliente,config.dispositivo,config.nombre,config.servidor,config.hora_inicio,config.hora
_fin);
  xmlFreeDoc(doc);
  return 1;
}
int SftXmlAgrega(DISPOSITIVO_BT *bt_ptr)
{
  xmlDocPtr doc=NULL;
  xmlNodePtr raiz=NULL,disp=NULL,dir=NULL,can=NULL,act=NULL;
  char canal[3];
  char archivo[150];
  /* Load XML document */
  memset(archivo,0x0,sizeof(archivo));
  sprintf(archivo,"%s/dispositivos.xml",path);
  doc = xmlParseFile(archivo);
  if (doc == NULL)
  {
    elog_printf(log_ptr, "XML: No pude parsear archivo");
    xmlFreeDoc(doc);
    return 0;
  }
  raiz = xmlDocGetRootElement(doc);
  if (raiz == NULL)
  {
    elog_printf(log_ptr,"XML:documento vacÃ-o para agregar elementos!!!");
    xmlFreeDoc(doc);
    return 0;
  }
  if (xmlStrcmp(raiz->name, (xmlChar *) "DISPOSITIVOS_BT"))
  {
    elog_printf(log_ptr,"document of the wrong type, root node != DISPOSITIVOS_BT");
    xmlFreeDoc(doc);
    return 0;
  }
  memset(canal,0x0,sizeof(canal));
  sprintf(canal,"%d",bt_ptr->canales_obex[0]);
  disp = xmlNewChild(raiz, NULL,(xmlChar *)"DISPOSITIVO", NULL);
  dir=xmlNewTextChild(disp,NULL,(xmlChar *)"DIRECCION",(xmlChar *)bt_ptr-
>direccion_BT);
  can=xmlNewTextChild(disp,NULL,(xmlChar *)"CANAL_OBEX",(xmlChar *)canal);
  act=xmlNewTextChild(disp,NULL,(xmlChar *)"RECHAZA",(xmlChar *)"0");
  xmlSaveFormatFile (archivo, doc, 0);
  xmlFreeDoc(doc);
  return 1;
}
int SftXmlAgregaRechzo(DISPOSITIVO_BT *bt_ptr)
{
  int exito=0;
  int encontre_disp=0;
  xmlDocPtr doc=NULL;
  xmlNodePtr raiz=NULL,disp=NULL,elemento=NULL;
  char archivo[150];
  /* Load XML document */
  memset(archivo,0x0,sizeof(archivo));
  sprintf(archivo,"%s/dispositivos.xml",path);
  doc = xmlParseFile(archivo);
  raiz = xmlDocGetRootElement(doc);
  if (raiz == NULL)
  {
    elog_printf(log_ptr,"XML:documento vacÃ-o para agregar rechazo elementos!!!");
    xmlFreeDoc(doc);
      return exito;
  }
  if (xmlStrcmp(raiz->name, (xmlChar *) "DISPOSITIVOS_BT"))
```

```
  {
    elog_printf(log_ptr,"document of the wrong type in reject, root node != DISPOSITIVOS_BT");
    xmlFreeDoc(doc);
    return exito;
  }
  for(disp=raiz->children;disp!=NULL;disp=disp->next)
  {
    if(!xmlStrcmp(disp->name, (xmlChar *) "DISPOSITIVO"))
    {
      encontre_disp=0;
      for(elemento=disp->children;elemento!=NULL;elemento=elemento->next)
      {
        if(!xmlStrcmp(elemento->children->content,(xmlChar *)bt_ptr->direccion_BT))
        {
          encontre_disp=1;
        }
        else if(!xmlStrcmp(elemento->name,(xmlChar *)"RECHAZA"))
        {
          if(encontre_disp)
          {
            xmlNodeSetContent(elemento->children,(xmlChar *)"1");
            exito=1;
            break;
          }
        }
      }
    }
    if(exito==1)
      break;
  }
  xmlSaveFormatFile (archivo, doc, 0);
  xmlFreeDoc(doc);
  return exito;
}
int SftXmlPubFile(void)
{
  int exito=0;
  xmlDocPtr doc=NULL;
  xmlNodePtr raiz=NULL,pub=NULL,elemento=NULL;
  int completo;
  PUBLICIDAD_BT pbt;
  PUBLICIDAD_BT *pbt_ptr;
  char archivo[150];
  /* Load XML document */
  memset(archivo,0x0,sizeof(archivo));
  sprintf(archivo,"%s/publicidad.xml",path);
  doc = xmlParseFile(archivo);
  if (doc == NULL)
  {
    elog_printf(log_ptr, "XML: No pude abrir archivo publicidad para parseo");
    xmlFreeDoc(doc);
    return exito;
  }
  raiz = xmlDocGetRootElement(doc);
  if (raiz == NULL)
  {
    elog_printf(log_ptr,"XML:documento vacÃ-o para lectura de elementos!!!");
    xmlFreeDoc(doc);
    return exito;
  }
  if (xmlStrcmp(raiz->name, (xmlChar *) "PUBLICIDADES_BT"))
  {
    elog_printf(log_ptr,"XML:document of the wrong type, root node != PUBLICIDADES");
    xmlFreeDoc(doc);
    return exito;
  }
  for(pub=raiz->children;pub!=NULL;pub=pub->next)
  {
    if(!xmlStrcmp(pub->name, (xmlChar *) "PUBLICIDAD"))
    {
      memset(&pbt,0x0,sizeof(pbt));
      completo=0;
      for(elemento=pub->children;elemento!=NULL;elemento=elemento->next)
      {
        if(!xmlStrcmp(elemento->name, (xmlChar *) "ID"))
        {
          pbt.id=atoi((char *)elemento->children->content);
          completo=1;
        }
        else if(!xmlStrcmp(elemento->name, (xmlChar *) "NOMBRE"))
        {
          strncpy(pbt.nombre,(char *)elemento->children->content,strlen((char *)elemento->children->content));
          if(completo==1)
            completo=2;
        }
        else if(!xmlStrcmp(elemento->name, (xmlChar *) "ARCHIVO"))
        {
          strncpy(pbt.archivo,(char *)elemento->children->content,strlen((char *)elemento->children->content));
          if(completo==2)
```

```
            completo=3;
        }
        else if(!xmlStrcmp(elemento->name, (xmlChar *) "HORAS"))
        {
          pbt.horas=atoi((char *)elemento->children->content);
          if(completo==3)
            completo=4;
        }
        if(completo==4)
        {
          pbt_ptr=(PUBLICIDAD_BT *)malloc(sizeof(PUBLICIDAD_BT));
          memcpy(pbt_ptr,&pbt,sizeof(PUBLICIDAD_BT));
          pthread_mutex_lock (&pub_lock);
          if(LIST_Insert(&lista_publicidad, (unsigned long)pbt_ptr->id,pbt_ptr)==LIST_SUCCESS)
          {
            elog_printf(log_ptr,"XML:publicidad encontrada id %d nombre %s archivo %s horas %d",
                pbt_ptr->id,pbt_ptr->nombre,pbt_ptr->archivo,pbt_ptr->horas);
            if(exito==0)
              exito=1;
          }
          else
          {
            elog_printf(log_ptr,"XML:no pude guardar info publicidad %s en lista",pbt_ptr->nombre);
            free(pbt_ptr);
          }
          pthread_mutex_unlock (&pub_lock);
        }

      }
    }
  }
  xmlFreeDoc(doc);
  return exito;
}
void SftXmlEjecuta(TRAN_XML *tran)
{
  DISPOSITIVO_BT *bt_ptr;
  int exito;
  elog_printf(log_ptr,"XML:recibo tran %d para direcciÃ³n %s",tran->tipo,tran->direccion_BT);
  pthread_mutex_lock (&btree_lock);
  bt_ptr=(DISPOSITIVO_BT *)buscaBTree (&Bluetooth_Btree,tran->direccion_BT);
  pthread_mutex_unlock (&btree_lock);
  if(bt_ptr==NULL)
  {
      elog_printf(log_ptr,"XML:recibo un apuntador nulo a dispositivo!!!");
  }
  else
  {
    switch(tran->tipo)
    {
    case SOFTEL_XML_DEVICE:
      exito=SftXmlAgrega(bt_ptr);
      if(exito)
        elog_printf(log_ptr,"XML:equipo %s agregado",bt_ptr->direccion_BT);
      else
        elog_printf(log_ptr,"XML:no pude agregar a equipo %s",bt_ptr->direccion_BT);
      break;
    case SOFTEL_XML_FILE:
      SftXmlAgregaPubDisp(bt_ptr);
      SftXmlAgregaEstPub(publicidad_actual);
      break;
    case SOFTEL_XML_REJECT:
      exito=SftXmlAgregaRechzo(bt_ptr);
      if(exito)
        elog_printf(log_ptr,"XML:rechazo de equipo %s agregado",bt_ptr->direccion_BT);
      else
        elog_printf(log_ptr,"XML:no pude agregar rechazo de equipo %s",bt_ptr->direccion_BT);
      break;
    }
  }
  free(tran);
}
void *SoftelXmlThread (void *a)
{
  OBEX_SERVER_THREAD *pthread;
  TRAN_XML *tran;
  char archivo[150];
  pthread=(OBEX_SERVER_THREAD *)a;
  segSleep (&pthread->tw, 1, 0);
  while (pthread->loop_rxThread==0)
  {
    nanosleep (&pthread->tw, NULL);
  }
  elog_printf(log_ptr,"XML:arrancado!!!");
  while(pthread->loop_rxThread)
  {
    switch(xml_control.status)
```

```
    {
    case XML_INIT:
      if(SftXmlPubFile())
      {
        elog_printf(log_ptr,"XML: encontre publicidades!!! habilito lectura de publicidades dispositivos");
        tengo_publicidad=1;
      }
      /*revisa que existan todos los archivos que necesitamos, y si no los crea*/
      SftXmlCheckFiles();
      /* Load XML document */
      memset(archivo,0x0,sizeof(archivo));
        sprintf(archivo,"%s/dispositivos.xml",path);
      xml_control.reader = xmlReaderForFile(archivo, NULL, 0);
      if(xml_control.reader==NULL)
        xml_control.status=XML_CREATE_FILE; //no tengo Ã¡rbol...hay que hacerlo
      else
      {
        xml_control.status=XML_PARSE_FILE;  //ya tengo el archivo, haz el parseo
        xmlFreeTextReader(xml_control.reader);
      }
      break;
    case XML_CREATE_FILE:
      xml_control.reader=SftXmlCreateFile();
      if(xml_control.reader!=NULL)
      {
        xml_control.status=XML_IDLE;
        bt_control.status=BT_INIT;   //no tenemos dispositivos aÃºn, asi es que avisale al thread BT que comience a buscar!!!
        elog_printf(log_ptr,"XML:info cargada...avisa Bluetooth!!");
        xmlFreeTextReader(xml_control.reader);
      }
      break;
    case XML_PARSE_FILE:
      if(SftXmlParser())
      {
        elog_printf(log_ptr,"XML:ya pude parsear la info..aviso Bluetooth!!");
        xml_control.status=XML_IDLE;
        bt_control.status=BT_INIT;
      }
      break;
    case XML_IDLE:
      tran=NULL;
      tran=(TRAN_XML *)COLA_obtiene (&xmlCola);
      if(tran!=NULL)
      {
        SftXmlEjecuta(tran);
      }
      break;
    }
    nanosleep (&pthread->tw, NULL);
  }
  return NULL;
}
```

BIBLIOGRAFÍA

-PATIL, Basavaraj, et al. “IP in Wireless Networks”. Prentice Hall, 2003.

-www.bluetooth.org

-BAKKER, Dee, et al. “Bluetooth End to End”. Wiley, 2002

-HUANG S., Albert, et al. “Bluetooth Essentials for Programmers”. Cambridge University Press, 2007.

-BOVET P., Daniel, et al. “Understanding the Linux Kernel” O´Reilley, 2000.

-FOWLER, Martin. “UML Distilled, Third Edition” . Addison-Wesley 2004

-HOLT, Jon, et a.l. “SysML for Systems Engineers”. The Institute of Engineering and Technology, 2008

-HOLT, Jon. “UML for Systems Engineers: watching the wheels”.The Institute of Engineering and Technology, 2007

-WEILKIENS, Tim. “Systems Engineering with UML/SysML: Modeling, Analysis, Design” . OMG Press, 2007.

-WEILKIENS, Tim, et al. “UML 2 Certification Guide Fundamental & Intermediate Exams” . Morgan Kaufmann, 2006.