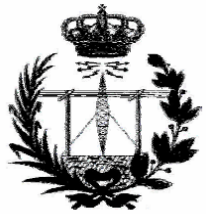
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

E. T. S. Ingeniería de Telecomunicaciones



Componentes software para gestión de dispositivos físicos.

Estudio comparativo de soluciones basadas en .NET y JavaBeans

Andrés Garro Cascales

Título del proyecto

Componentes software para gestión de dispositivos físicos.
Estudio comparativo de soluciones basadas en .NET y JavaBeans

Autor

Andrés Garro Cascales

Titulación

Ingeniero Técnico en Telecomunicación.
Especialidad Telemática

Directora

Cristina Vicente Chicote

Índice

CAPÍTULO I

Introducción

En la actualidad la programación es una herramienta imprescindible para el trabajo diario con computadoras, esto conlleva un trabajo más rápido y eficaz.

La programación ha avanzado mucho en muy poco tiempo y han surgido distintos lenguajes de programación que ofrecen formas distintas de generar un código útil. Estos lenguajes de programación son muy variados y se han ido reciclando con el tiempo para mejorar la legibilidad del código de cara al programador y aumentar su simplicidad, y para ofrecer nuevas prestaciones tanto al programador como a los usuarios.

Cualquier programador experimentado, hoy por hoy, ha podido comprobar que en función del código que se desee generar es más rentable utilizar uno u otro lenguaje de programación debido a los requisitos del programa que se va a realizar, sería sin duda un gran logro reunir todas las cualidades positivas de varios lenguajes de programación para un uso práctico más logrado.

El problema que se plantea en este proyecto surge de cara a la formación de un nuevo programador. Si se revisa cualquier manual de programación independientemente del lenguaje de programación, se puede comprobar como todos los ejemplos y todas las explicaciones varían muy poco de un lenguaje a otro, esto lleva a una nueva perspectiva, debido a por qué la enseñanza de todos los lenguajes de programación es tan parecida cuando unos distan bastante de otros.

Bien, si se requiere encontrar dos formas de "programar" muy distintas solamente se tiene que contrastar el lenguaje de programación con el lenguaje ensamblador, en estos dos casos se observa que la implementación de funciones en el lenguaje de programación sobre un sistema operativo dista muchísimo de la utilización de rutinas y acumuladores en el lenguaje ensamblador sobre un procesador.

Hoy en día, se requiere el uso de la ingeniería para asociar la informática y las telecomunicaciones a efectos prácticos, como por ejemplo a la modernización de las tecnologías en el trabajo actual, todo esto es abarcado por la Domótica.

Cuando se ha de realizar un programa y se ha de llevar a efectos prácticos (a una plataforma hardware), es necesaria la utilización del lenguaje ensamblador, pero para la utilización de cierto hardware existen lenguajes de programación que ofrecen plataformas para operar sobre ese hardware directamente, facilitando la velocidad, legibilidad y prestaciones del código generado.

He aquí la propuesta de este proyecto, por qué son tan desconocidas las técnicas de tratamiento hardware utilizando de plataforma un lenguaje de programación, cuando son evidentemente muy útiles y requeridas en el mundo actual.

1.1 Objetivos

El objetivo principal del proyecto es la realización de un estudio sobre los lenguajes de programación orientados a componentes físicos y partir de ese estudio desarrollar los siguientes puntos establecidos:

- Primero se seleccionará un elemento hardware de un computador, este elemento será sobre el cual se realizará el código necesario para los siguientes apartados. El elemento seleccionado es el puerto RS-232, también conocido por Puerto Serie o puerto COM. La elección de este puerto se debe a que es un puerto de comunicaciones bastante antiguo por lo que no resultará difícil encontrar gran cantidad de información sobre el manejo del mismo, en cambio la elección de otro puerto más novedoso como el puerto Universal Serial Bus *USB* si puede dar este tipo de problema, y porque a pesar de ser antiguo todavía no es un puerto descatalogado en la mayoría de computadores nuevos.

- Selección de el/los tipos de componente/s (dependiendo del lenguaje de programación que se vaya a utilizar). En este caso se puede escoger entre gran variedad de lenguajes de programación. Evidentemente se descartaron para la realización de este software los lenguajes de programación más obsoletos, dejando los más novedosos y más útiles con opciones para la realización de los componentes. Finalmente se decidió realizar los componentes en los lenguajes de programación Java y .NET, para ello se utilizarán las aplicaciones: BeanBox la cual nos ofrece la realización de un componente JavaBeans, Microsoft Visual Studio .NET. Se han elegido estos dos tipos de componentes por diferentes motivos.

  JavaBeans: La elección de un componente JavaBeans se realizó debido a que Java ofrece un motor de bases de datos muy potente, una avanzada programación orientada a objetos y sobre todo la portabilidad (la portabilidad de Java permite llevar un código compilado de un sistema operativo a otro). Además Java es uno de los lenguajes más utilizados en esta última época y con más auge.

  .NET: La elección de un componente .NET se debe a varias causas. La primera es que a diferencia de Java .NET no permite portar el código de un sistema operativo a otro, la herramienta Microsoft Visual Studio .NET es una herramienta que pertenece al grupo Microsoft y únicamente funciona con sistemas operativos de este grupo. Independientemente de esto .NET

funciona con el sistema operativo Windows XP (el más utilizado del mundo) y es una herramienta muy novedosa, aunque la gran posibilidad que ofrece esta herramienta es la posibilidad de crear componentes en un lenguaje de programación y pasarlo a otros. Por ejemplo, Microsoft Visual Studio. NET posee varios lenguajes de programación (ASP, C++, C#, J#, Visual Basic, …), y permite que un componente .NET creado en uno de estos lenguajes pueda ser utilizado por cualquier otro.

- Estudio previo para la realización de el/los componente/s que se van a realizar. Este estudio se realizará tomando como base el temario de la carrera para realizar el componente JavaBeans y la información obtenida del libro .NET in Samples de Jan Seda para realizar el componente .NET, además de la información necesaria obtenida por Internet.

- Diseño de los componentes realizados. Generación del código necesario para controlar el puerto serie en Java y en Microsoft Visual Studio .NET, por medio de los componentes JavaBeans y .NET.

- Comparativas entre los distintos componentes diseñados y realizados. Diferencias entre ambos códigos, ventajas e inconvenientes de los mismos, dificultades para la realización de los códigos, manejabilidad, etc .

1.2 Organización de la Memoria

El resto de esta memoria se organiza en los siguientes capítulos:

- ✓ **Capítulo I:** Introducción, Objetivos y Fases del proyecto.
- ✓ **Capítulo II:** Introducción a la Ingeniería Software.
- ✓ **Capítulo III:** Arquitecturas software basadas en componentes.
- ✓ **Capítulo IV:** Diseño de los componentes JavaBean y .NET para la gestión de dispositivos físicos a través de los puertos de un ordenador.
- ✓ **Capítulo VII:** Conclusiones finales.
- ✓ **Anexo A:** Código del componente JavaBean (“PuertoSerie.java”).
- ✓ **Anexo B:** Código del componente .NET (“RS232.cs”).
- ✓ **Anexo C1:** Código de la aplicación que ejecuta el componente .NET (“Form.jsl”).
- ✓ **Anexo C2:** Código de la aplicación que ejecuta el componente .NET (“Error.jsl”).
- ✓ **Bibliografía**

CAPITULO II

Ingeniería del Software

2.1 Comienzos de la Programación

La historia de la programación ha ido estrechamente ligada a la historia de la computación, esto se debe a que en un principio la computación o el inicio de esta se realizaba por máquinas mecánicas o manuales, a las que no se le podía dar aún el nombre de computadoras.

Dentro de la historia de la computación no se puede llamar computadora a un artefacto que carecía de programas o software, pero algunos artefactos se remontan a los inicios del almacenamiento de datos a través de un mecanismo.

Continuando con la historia de la computación se puede hablar ahora de la Pascalina, inventada por Blaise Pascal en el siglo XVII en Francia. A través de este artefacto los datos se representaban a través de las posiciones de los engranajes, los datos eran introducidos a mano estableciendo dichas posiciones finales de las ruedas, algo parecido a leer los números de un cuentakilómetros.

En la historia de la computación se puede llamar primera computadora a la maquina creada por Charles Babbage, profesor matemático de la Universidad de Cambridge en el siglo XIX. La idea fue concebida para resolver el problema de las tablas matemáticas. El gobierno Británico subvencionó el proyecto de una máquina de diferencias, un dispositivo mecánico para efectuar sumas repetidas.

En Francia un fabricante de tejidos ponía su granito de arena a la historia de la computación, creando un telar que reproducía patrones de tejidos, leyendo información codificada en patrones hechos de agujeros perforados en papel duro.

Charles Babbage intentó crear una máquina analítica basándose en la idea de las tarjetas perforadas del telar francés, intentando crear una máquina capaz de calcular con precisión de 20 dígitos.

La historia de la computación daba pasitos muy pequeños en aquella época por lo que se retrasó aún más el comienzo de la programación. No fue hasta 1944 cuando en la Universidad de Harvard se construyó una máquina basada en dispositivos electromecánicos llamados relevadores, aún no se puede hablar de computadora de acuerdo a unos patrones establecidos actualmente y de acuerdo a la programación de hoy en día.

Llegó el momento crucial en la historia de la computación en 1947, en la Universidad de Pensilvania, donde fue construida la primera computadora electrónica que ocupaba el sótano entero de la Universidad. La historia de la computación comenzó aquí su increíble recorrido. Este proyecto era respaldado por el Ministerio de Defensa de USA.

Dos años después se uniría al proyecto el matemático húngaro John Von Newman, quien es considerado en la historia de la computación el padre de las computadoras por su valiosa aportación al desarrollo de estas increíbles máquinas.

Fue fundamental en la historia de la computación el avance incluido por Von Newman, permitiendo que coexistan datos con instrucciones en la memoria, pudiendo así ser programados estos en un lenguaje.

Avanzada la mitad del siglo XX, la ingeniería de las computadoras avanzaba cada vez con mayor velocidad, reduciendo estas en tamaño y mejorando cada vez más sus funciones y capacidad de memoria y procesamiento.

La historia de la computación nos lleva hasta la década de los 60 donde las computadoras se programaban con cintas perforadas y otras por medio de cableado en un tablero. Un equipo de expertos: analistas, diseñadores, programadores y operadores resolvían los problemas y cálculos solicitados por la administración. En estas cintas perforadas se establecían las primeras operaciones programadas, además en ellas ya se introducen los primeros patrones y normas de programación, es el momento en el que las personas comienzan a interactuar con las computadoras por medio de un “lenguaje de programación” aún sin definir por completo.

Por último en la historia de la computación aparecen las primeras computadoras personales, que necesitan ser programadas y sus programas ser guardados en grabadoras de astte, luego se avanzó hasta poder guardar los datos en unidades de disco flexibles.

Debido a la complejidad que de la cual se componen los programas y las computadoras y a que esta avanzaba en el tiempo, se fue abriendo paso a los lenguajes de programación y a la importancia que tienen los mismos en la historia de la computadoras, de la programación y como no de la importancia en la actualidad. Poco a poco se fueron estableciendo unas normas de programación de acuerdo a los sistemas inteligibles de cada época, en ese preciso instante comienza uno de los valores más importantes de la programación, se está hablando de una reutilización del software.

2.2 Desarrollo de la Programación

Programar consiste en desarrollar código (programas) capaz de procesar información.

Programación como término se utiliza para designar la creación de programas a pequeña escala, el desarrollo de sistemas complejos se denomina ingeniería de software.

Una computadora es totalmente inútil si no dispone de un programa capaz de procesar información.

Para que se realice dicho procesamiento de información habrá sido necesario construir un ordenador (hardware), pensar y crear un programa (software) y ejecutar dicho programa o aplicación en el computador. La última de estas fases es la que realiza el usuario, las anteriores son realizadas por técnicos que construyen el hardware y por programadores que desarrollan el software.

Programación e Ingeniería de Software son complementarias entre sí. Para el desarrollo de grandes sistemas informáticos se divide el trabajo en tareas que diversos programadores desarrollarán. Al terminar se unen las piezas como en un puzzle para completar el sistema en si. Así programación también se aplica para el desarrollo de grandes sistemas en las ingenierías de software.

La programación tiene como objetivo el tratamiento de la información correctamente, con lo que se espera que un programa dé el resultado correcto y no uno erróneo. Así que cada aplicación debe funcionar según lo esperado en términos de programación.

Otro objetivo fundamental de la programación es que sean de códigos claros y legibles, con lo que si un programador inicia un programa y no lo termina, otro programador sea capaz de entender la codificación y poder terminarlo. Normalmente en programación existen ciertas normas no escritas de como han de nombrarse los componentes, objetos o controles de cada sistema, así como sus variables que deben ser relativas al termino al cual se van a vincular.

Por último la programación pretende que sus programas sean útiles y eficientes. De multitud de maneras la programación nos dará el mismo resultado de un programa, un buen programador llegará al mismo resultado con un mínimo de código y de la forma más clara y lógica posible.

De los anteriormente nombrados objetivos de la programación el más importante es el de la corrección, ya que un código claro y legible facilita el mantenimiento de la aplicación o sistema.

2.3 Ingeniería Software

La importancia de la programación y del software en general es evidente, gracias a este se eliminan grandes costes de producción, además de las ventajas que incorpora como su portabilidad, maleabidlidad, tolerancia a errores, etc…, todo esto como se puede ver son propiedades del software, que para ser correctamente controladas y aprovechadas se han de regir por unos principios. Antes de generar cualquier código hemos de tener en cuenta todos los principios de la ingeniería software, ya que nos van a delimitar problemas futuros mediante soluciones actuales.

Ahora se presentan algunos principios generales de importancia, que son centrales para desarrollar software de forma exitosa, y que tratan tanto del proceso de ingeniería de software como del producto final. El proceso adecuado ayudará a desarrollar el producto deseado, pero también el producto deseado afectará la elección del proceso a utilizar. Un problema tradicional de la ingeniería de software es poner el énfasis en el proceso o en el producto excluyendo al otro, sin embargo, ambos son importantes.

Estos principios son suficientemente generales para ser aplicados a lo largo del proceso de construcción y gestión del software, sin embargo no son suficientes para guiar el desarrollo ya que describen propiedades deseables de los procesos y productos de software; para aplicarlos es necesario contar con métodos apropiados y técnicas específicas. Los métodos son guías generales que gobiernan la ejecución de alguna actividad, presentan enfoques rigurosos, sistemáticos y disciplinados, por otro lado, las técnicas son más mecánicas y se refieren a aspectos más "técnicos" que los métodos y tienen aplicación restringida. Una metodología es un conjunto de métodos y técnicas cuyo propósito es promover cierto enfoque para la resolución de un problema mediante ese conjunto seleccionado. Las herramientas son desarrolladas para apoyar la aplicación de técnicas, métodos y metodologías. Los principios son la base de todos los métodos, técnicas, metodologías y herramientas.

2.3.1 Rigor y Formalidad

En cualquier proceso creativo existe la tendencia a seguir la inspiración del momento de forma no estructurada, sin ser precisos; el desarrollo de software es de por sí una actividad creativa. Por otro lado, el rigor es un complemento necesario de la creatividad en todas las actividades de la ingeniería; únicamente a través de un enfoque riguroso podrán producirse productos más confiables, controlando sus costos e incrementando el grado de confianza en los mismos. El rigor no tiene por qué restringir la creatividad, por el contrario, puede potenciar la creatividad aumentando la confianza del ingeniero en los resultados de la misma, una vez que estos son analizados a la luz de evaluaciones rigurosas. Paradójicamente el rigor es una cualidad intuitiva que no puede ser definida en forma rigurosa, pero sí pueden alcanzarse varios niveles de rigurosidad siendo el más alto la formalidad.

La formalidad es un requerimiento más fuerte que el rigor: requiere que el proceso de software sea guiado y evaluado por leyes matemáticas. Obviamente formalidad implica rigor pero no a la inversa: se puede ser riguroso incluso informalmente. En todos los campos de la ingeniería el proceso de diseño sigue una secuencia de pasos bien definidos, establecidos en forma precisa y posiblemente probados, siguiendo en cada paso algún método o aplicando alguna técnica. Estos métodos y técnicas estarán basados en alguna combinación de resultados teóricos derivados de un modelado formal de la realidad, ajustes empíricos que tienen en cuenta fenómenos no presentes en el modelo, y métodos prácticos de evaluación que dependen de la experiencia pasada ("rules of thumb").

Un ingeniero debe saber cómo y cuándo ser formal si es requerido, entendiendo el nivel de rigor y formalidad que debe ser alcanzado dependiendo de la dificultad conceptual de la tarea y su criticidad, lo que puede variar para diferentes partes del mismo sistema. Por ejemplo, partes críticas pueden requerir una descripción formal de las funciones esperadas y un enfoque formal para su evaluación mientras que partes estándares o bien entendidas requerirán enfoques más simples. Esto se aplica también en el caso de la ingeniería de software, por ejemplo en el caso de la especificación del software la cual puede establecerse de forma rigurosa utilizando lenguaje natural o también puede darse formalmente mediante

una descripción formal en un lenguaje de sentencias lógicas. La ventaja de la formalidad sobre el rigor es que puede ser la base para la mecanización del proceso, por ejemplo si se quiere utilizar la descripción formal para crear el programa si éste no existe, o para mostrar que el programa se corresponde con las especificaciones establecidas si tanto el programa como las especificaciones existen.

Tradicionalmente es en la fase de codificación donde se utiliza un enfoque formal ya que los programas son objetos formales: son escritos en un lenguaje cuya sintaxis y semántica están completamente definidas. Los programas son descripciones formales que son manipuladas automáticamente por los compiladores que chequean su corrección y las transforman en una forma equivalente en otro lenguaje (ensamblador o lenguaje de máquina), todo lo cual es posible gracias a la utilización de la formalidad en la programación.

La aplicación del principio de rigor y formalidad tienen influencia beneficiosa en la obtención de cualidades del software como la confiabilidad, la facilidad de verificación y mantenimiento, la reutilización, la portabilidad, la comprensibilidad, o la interoperabilidad. Por ejemplo, una documentación del software rigurosa o incluso formal puede mejorar todas estas cualidades sobre una documentación informal que puede ser ambigua, inconsistente e incompleta.

El principio de rigor y formalidad también se aplica al proceso de software; la documentación rigurosa del proceso ayuda a que éste sea reutilizado en proyectos similares y también ayuda a mantener un producto existente permitiendo que las modificaciones se realicen partiendo del nivel intermedio apropiado, en lugar de hacerlo solamente sobre el código final. Si el proceso de software está especificado en forma rigurosa, los gerentes podrán controlar su adecuación y evaluar su oportunidad para mejorar la productividad.

2.3.2 Separación de intereses

Este principio permite enfrentarse a los distintos aspectos individuales de un problema de forma que pueda concentrarse en cada uno por separado. En el desarrollo de un producto de software deben tomarse muchas decisiones como las funciones que serán ofrecidas, la confiabilidad esperada, eficiencia de tiempo y espacio, relaciones con el ambiente como recursos de software o hardware especial, interfaces de usuario, entre otras. Otras decisiones tienen que ver con el proceso de desarrollo como el ambiente de desarrollo, la organización y estructura del equipo, la agenda, los procedimientos de control, las estrategias de diseño, los mecanismos de recuperación frente a errores, entre otras. Y otras más que tienen que ver con temas económicos y financieros. Muchas de estas decisiones pueden no estar relacionadas entre sí por lo que obviamente podrán ser tratadas en forma separada, pero muchas otras estarán fuertemente relacionadas y será prácticamente imposible tener en cuenta todos los temas al mismo tiempo o por parte de las mismas personas. La única forma de enfrentar la complejidad de un proyecto es separar los distintos intereses.

La primera forma en la que se pueden separar los distintos intereses es según el tiempo, lo que permite planificar las distintas actividades y eliminar el trabajo extra que implica cambiar de una a otra en forma no restringida. Esta separación según el tiempo es la motivación que hay tras el ciclo de vida del software; un modelo racional de la secuencia de actividades que deberían seguirse en la producción de software.

Otra forma de separación de intereses es en términos de las cualidades que deberían tratarse por separado, por ejemplo podrían enfrentarse separadamente la eficiencia y

corrección de un programa, primero diseñándolo cuidadosa y estructuradamente para garantizar su corrección a priori y luego reestructurarlo para mejorar su eficiencia.

Otro tipo importante de separación de intereses permite que distintas visiones del software sean analizadas en forma separada, por ejemplo al analizar los requerimientos de una aplicación podría ser de ayuda concentrarse por un lado en los datos que fluyen de una actividad a otra y por otro lado en el flujo de control que gobierna la sincronización de dichas actividades. Ambas ayudan a entender el sistema y ninguna de las dos provee una visión completa del mismo.

Otra forma más de aplicación de este principio es enfrentar partes del mismo sistema en forma separada, esto es en términos de tamaño. Este es un concepto fundamental que debe dominarse para enfrentar la complejidad de la producción de software, y es tan importante que se trata como un punto aparte bajo el principio de modularidad.

Si bien podrían perderse algunas optimizaciones potenciales al no tener en cuenta el problema en su conjunto, la complejidad global puede resolverse mucho mejor concentrándose en los distintos aspectos por separado, incluso si no fuera posible descomponer el problema en los distintos aspectos en forma inmediata, es posible tomar inicialmente algunas decisiones de diseño generales y luego aplicar el principio de separación de intereses en forma efectiva.

Como observación final, la separación de intereses podría resultar en la separación de responsabilidades al enfrentarse a los distintos aspectos a tener en cuenta, por lo tanto es la base para dividir el trabajo en un problema complejo en asignaciones de trabajo específicas posiblemente a personas distintas con distintas habilidades.

2.3.3 Modularidad

Un sistema complejo puede dividirse en piezas más simples llamadas módulos, un sistema compuesto de módulos es llamado modular. El principal beneficio de la modularidad es que permite la aplicación del principio de separación de intereses en dos fases: al enfrentar los detalles de cada módulo por separado ignorando detalles de los otros módulos, y al enfrentar las características globales de todos los módulos y sus relaciones para integrarlos en un único sistema coherente. Si estas fases son ejecutadas en ese orden se dice que el sistema es diseñado de abajo hacia arriba (*bottom up*), en el orden inverso se dice que el sistema es diseñado de arriba hacia abajo (*top down*).

El principio de modularidad tiene tres (3) objetivos principales: capacidad de descomponer un sistema complejo, capacidad de componerlo a partir de módulos existentes y comprensión del sistema en piezas (o pedazos).

La posibilidad de descomponer un sistema se basa en dividir en sub-problemas de forma *top down* el problema original y luego aplicar el principio a cada sub-problema en forma recursiva. Este procedimiento refleja el bien conocido principio de Divide y Vencerá.

La posibilidad de componer un sistema está basada en obtener el sistema final de forma *bottom up* a partir de componentes elementales. Idealmente en la producción de software de quererlo así se podría ensamblar nuevas aplicaciones tomando módulos de una biblioteca y combinándolos para formar el producto requerido; estos módulos deberían ser diseñados con el objetivo expreso de ser reutilizables.

La capacidad de comprender cada parte de un sistema de forma separada ayuda a la modificabilidad del sistema. Debido a la naturaleza evolutiva del software muchas veces se debe volver hacia atrás al trabajo previo y modificarlo. Si el sistema solo puede ser comprendido como un todo las modificaciones serán difíciles de aplicar y el resultado será poco confiable. Cuando se hace necesario reparar el sistema, si este es modular se restringirá y facilitará la búsqueda de la fuente de error en componentes separados.

Para alcanzar estos objetivos los módulos en los que se divida el sistema deben tener alta cohesión y bajo acoplamiento. Un módulo tiene alta cohesión si todos sus elementos están fuertemente relacionados y son agrupados por una razón lógica, esto es todos cooperan para alcanzar un objetivo común que es la función del módulo. La cohesión es una propiedad interna de cada módulo, por el contrario el acoplamiento caracteriza las relaciones de un módulo con otros. El acoplamiento mide la interdependencia de dos módulos, por ejemplo si el módulo A hace una llamada a una rutina provista por el módulo B o accede a una variable declarada por el módulo B. Si dos módulos dependen fuertemente uno del otro tienen un alto acoplamiento lo que los vuelve difíciles de analizar, comprender, modificar, testar o reutilizar en forma separada. Idealmente se quiere que los módulos de un sistema tengan bajo acoplamiento.

Una estructura modular con alta cohesión y bajo acoplamiento permite ver los módulos como cajas negras cuando se describe la estructura global del sistema y luego encarar cada módulo por separado cuando se analiza o describe la funcionalidad del módulo.

2.3.4 Abstracción

La abstracción es un proceso mediante el cual se identifican los aspectos relevantes de un problema ignorando los detalles; es un caso especial del principio de separación de intereses en el cual se separan los aspectos importantes de los detalles de menor importancia. Lo que se abstrae y lo que se considera dependerá del propósito de la abstracción, por lo que podrán hacerse distintas abstracciones de la misma realidad cada una de las cuales proveerá una visión de la realidad que sirve para un propósito específico.

Por ejemplo, cuando los requerimientos de una nueva aplicación son analizados y especificados se construye un modelo de la aplicación propuesta, el cual podrá ser expresado en varias formas dependiendo del grado requerido de rigor y formalidad. Sin importar cual sea el lenguaje elegido para expresar los requerimientos, lo que se provee es un modelo que abstrae los detalles que se decidió que podían ser ignorados en forma segura. Los lenguajes de programación también son abstracciones construidas sobre el hardware que proveen constructores útiles y poderosos para escribir programas ignorando detalles como el número de bits que se utilizan para representar números o los mecanismos de direccionamiento, lo que permite concentrarse en el problema a resolver en lugar de la forma de instruir a la máquina para hacerlo.

El principio de abstracción es un principio importante que se aplica tanto a los productos de software como a los procesos. En este último caso, por ejemplo, al realizar la estimación de costos para una nueva aplicación una forma posible es identificar algunos factores claves del nuevo sistema y extrapolar los valores a partir de perfiles de costo de sistemas previos similares. Los factores claves utilizados para realizar el análisis son abstracciones del sistema.

2.3.5 Anticipación al cambio

El software sufre cambios constantemente, como se vio al tratar la mantenibilidad del software estos cambios pueden surgir por la necesidad de eliminar errores que no fueron detectados antes de liberar la aplicación, o por la necesidad de apoyar la evolución de la aplicación debido a nuevos requerimientos o cambios en los requerimientos existentes.

La habilidad del software para evolucionar no viene sola sino que requiere un esfuerzo especial para anticipar cómo y cuándo pueden ocurrir estos cambios. Cuando se identifican posibles cambios futuros, se debe tener cuidado de proceder de forma que estos sean fáciles de aplicar, es importante aislar los posibles cambios en porciones específicas del software de tal forma que estén restringidos a esas partes.

La anticipación al cambio es posiblemente el principio que más distingue al software de otros tipos de producción industrial. Muchas veces una aplicación de software es desarrollada mientras sus requerimientos aún no están completamente comprendidos, al ser liberado y obtener retroalimentación del usuario debe evolucionar con nuevos requerimientos o cambios a los requerimientos ya existentes los cuales pueden tener distintos orígenes, por ejemplo debido a cambios en el ambiente de la organización. Por lo tanto este principio puede ser utilizado para lograr la evolucionabilidad del software y también la reusabilidad de componentes, viendo la reusabilidad como evolucionabilidad de granularidad más fina, a nivel de componentes.

La aplicación de este principio requiere que se disponga de herramientas apropiadas para gestionar las varias versiones y revisiones del software de forma controlada. Debe ser posible almacenar y recuperar documentación, fuentes, ejecutables, etc. de una base de datos que actúe como repositorio central de componentes reusables, y el acceso a la misma debe estar controlado. Un sistema de software debe mantenerse consistente, incluso cuando se aplican cambios a algunos de sus componentes. La disciplina que estudia esta clase de problemas es la Gestión de Configuración y se verá posteriormente.

La anticipación al cambio también se aplica al proceso de desarrollo de software, por ejemplo, en la gestión del proyecto los gerentes deberían anticipar los efectos de una reducción de personal, estimar los costos y diseñar la estructura de la organización que apoyará la evolución del software, y decidir cuando vale la pena invertir tiempo y esfuerzo en la producción de componentes reutilizables tanto como parte de un proyecto de desarrollo de software o como un esfuerzo de desarrollo paralelo.

2.3.6 Generalidad

El principio de generalidad establece que al tener que resolver un problema se debe buscar un problema más general que posiblemente esté oculto tras el problema original, puesto que puede suceder que el problema general no sea mucho más complejo (a veces puede ser incluso más simple) que el original y posiblemente la solución al problema general tenga potencial de reuso, o exista en el mercado como producto off-the-shelf, o se diseñe un módulo que puede ser invocado por más de un punto en la aplicación en lugar de tener varias soluciones especializadas.

Por otro lado, una solución general posiblemente sea más costosa en términos de rapidez de ejecución, requerimientos de memoria o tiempo de desarrollo, que una solución especializada al problema original, por lo que debe evaluarse la generalidad respecto al

costo y la eficiencia al momento de decidir qué vale más la pena, una solución general o una especializada.

La generalidad es un principio fundamental si se tiene como objetivo el desarrollo de herramientas generales o paquetes para el mercado, ya que para ser exitosas deberán cubrir las necesidades de distintas personas. Estos productos de propósito general, comúnmente denominados productos OTS (*off-the-shelf)* como, por ejemplo, los procesadores de texto, representan una tendencia general en el software; para cada área específica de aplicación existen paquetes generales que proveen soluciones estándares a problemas comunes. Esta tendencia es idéntica a lo que ocurrió en otras áreas de la industria como por ejemplo, los automóviles que en los inicios de la tecnología automotriz era posible hacer autos de acuerdo a los requerimientos específicos de un cliente, pero a medida que el área se fue industrializando solo podían encargarse a partir de un catálogo y actualmente no es posible pedir un diseño de auto personal a menos que se esté dispuesto a pagar una enorme cantidad de dinero.

2.3.7 Incrementalidad

La incrementalidad caracteriza un proceso que se desarrolla en forma de pasos, en incrementos, alcanzando el objetivo deseado mediante aproximaciones sucesivas al mismo, donde cada aproximación es alcanzada a través de un incremento de la previa.

Una forma de aplicar el principio de incrementalidad consiste en identificar subconjuntos tempranos de una aplicación que sean útiles de forma que se obtenga retroalimentación (feedback) temprana del cliente. Esto permite que la aplicación evolucione de forma controlada en los casos en que los requerimientos iniciales no son estables o no están completamente entendidos. La motivación de este principio es que muchas veces no es posible obtener todos los requerimientos antes de comenzar el desarrollo de una aplicación sino que éstos van emergiendo a partir de la experimentación con la aplicación o partes de ésta. Por lo tanto, cuanto antes se pueda contar con feedback del usuario sobre la utilidad de la aplicación, más fácil será incorporar los cambios requeridos al producto. Este principio está ligado al principio de anticipación al cambio y es otro de los principios en los que se basa la evolucionabilidad.

La incrementalidad se aplica a muchas de las cualidades del software vistas previamente. Se puede por ejemplo, comenzar con un núcleo de la aplicación que sea útil e ir agregando funcionalidades, también se puede agregar “performance” en forma incremental si por ejemplo, la versión inicial enfatizaba las interfaces de usuario y la confiabilidad, luego sucesivas liberaciones irán mejorando la eficiencia en tiempo y espacio.

Cuando se construye una aplicación en forma incremental, los pasos intermedios pueden ser prototipos del producto final, esto es solamente una aproximación al mismo. Obviamente un ciclo de vida basado en prototipos es bastante distinto al tradicional modelo en cascada, y está basado en un modelo de desarrollo más flexible e iterativo. Estas diferencias tendrán efectos no solo en los aspectos técnicos sino también en los organizativos y de gestión.

Como se mencionaba en el principio de anticipación al cambio, el desarrollo de software de forma evolutiva requiere tener especial cuidado en la gestión de documentación, programas, datos de testeo, etc... que son desarrollados para las varias versiones del software. Cada incremento significativo debe ser registrado, la documentación debe poder ser fácilmente recuperada, los cambios deben aplicarse en forma ordenada, etc.

Si todo lo anterior no se realiza con cuidado, un intento de desarrollo evolutivo podría rápidamente transformarse en un desarrollo de software indisciplinado, perdiéndose todas las ventajas del desarrollo incremental.

2.4. Lenguajes de Programación

A la hora de programar hay que utilizar un lenguaje de programación. Los lenguajes de programación son un conjunto de patrones o de medidas sintácticas y semánticas que van a permitir a un programador diseñar una aplicación para un usuario en función de uno o varios sistemas operativos.

Los lenguajes de programación son herramientas que nos permiten crear programas y software. Una computadora funciona bajo control de un programa el cual debe estar almacenado en la unidad de memoria; tales como el disco duro.
Los lenguajes de programación de una computadora en particular se conocen como código de máquinas o lenguaje de máquinas y son los que permiten al programador por medio de un código legible para él crear aplicaciones que interactúen con la computadora con lenguajes de bajo nivel (bytecode).

Estos lenguajes codificados en una computadora específica no podrán ser ejecutados en otra computadora diferente. Para que estos programas funcionen para diferentes computadoras hay que realizar una versión para cada una de ellas, lo que implica el aumento del costo de desarrollo. Por otra parte, los lenguajes de programación en código de máquina son verdaderamente difíciles de entender para una persona, ya que están compuestos de códigos numéricos sin sentido nemotécnico.

Los lenguajes de programación facilitan la tarea de programación, ya que disponen de formas adecuadas que permiten ser leídas y escritas por personas, a su vez resultan independientes del modelo de computador a utilizar. Los lenguajes de programación representan en forma simbólica y en manera de un texto los códigos que podrán ser leídos por una persona. Los lenguajes de programación son independientes de las computadoras a utilizar.

Existen estrategias que permiten ejecutar en una computadora un programa realizado en un lenguaje de programación simbólico. Los procesadores del lenguaje son los programas que permiten el tratamiento de la información en forma de texto, representada en los lenguajes de programación simbólicos.

Hay lenguajes de programación que utilizan compilador. La ejecución de un programa con compilador requiere de dos etapas:

1) Traducir el programa simbólico a código máquina.

2) Ejecución y procesamiento de los datos.

Otros lenguajes de programación utilizan un programa intérprete o traductor, el cual analiza directamente la descripción simbólica del programa fuente y realiza las instrucciones dadas. El intérprete en los lenguajes de programación simula una máquina virtual, donde el lenguaje de máquina es similar al lenguaje fuente. La ventaja del proceso interprete es que no necesita de dos fases para ejecutar el programa, sin embargo su inconveniente es que la velocidad de ejecución es más lenta ya que debe analizar e interpretar las instrucciones contenidas en el programa fuente.

Algunos lenguajes de programación:

- Pascal
- Visual Basic
- Delphi
- Java
- C/C++

Actualmente existen múltiples herramientas (editores y compiladores), muchas de ellas gráficas, para los distintos lenguajes de programación. Herramientas como JBuilder, Delphi, o Visual C++, facilitan enormemente la programación de aplicaciones en distintos lenguajes. En estos programas se ofrece mediante una representación gráfica (icono) el código de las utilidades visuales más comunes.

Cuando se selecciona una utilidad visual y se añade en una aplicación que se está creando se puede comprobar como el programa en sí mismo ha introducido el código necesario para que esa utilidad visual aparezca en la aplicación. Además también se comprueba que se pone al servicio del programador (generalmente en forma de paleta) un listado con todas las propiedades de la utilidad visual y la posibilidad de modificar las mismas.

Como cabe de esperar según lo especificado anteriormente, en el momento en que una propiedad de una utilidad visual (objeto) es modificada, también se puede comprobar como se ha agregado automáticamente el texto necesario en nuestra página de código para hacerlo funcionar correctamente.

Como ya quedó indicado en la ingeniería software, este es el caso más claro que tenemos de una correcta reutilización del software.

El software que se ofrece por estos programas es cien por cien reutilizable para cualquier caso, esto es debido a que al no pertenecer a una aplicación en concreto se establecen en él únicamente los patrones generales para crearlo y la posibilidad de modificar sus propiedades de acuerdo a las necesidades de la aplicación que se esté generando.

Evidentemente cada una de estas utilidades visuales crea un código genérico reutilizable y sin errores de sintaxis, además es de lectura comprensible por el programador y el código que creará se basará en un lenguaje de programación o en otro de acuerdo con el programa usado. Por ejemplo, si se utiliza el programa JBuilder se generará un código en Java, si se utiliza el programa Visual C++ se generará un código en lenguaje C, si se utiliza Delphi se generará un código en lenguaje Delphi, etc.

A estas utilidades visuales que generan código, que lo hacen de acuerdo a lo exigido por la ingeniería software, que facilitan el trabajo al programador, se les da el nombre de componente.

CAPÍTULO III

Arquitecturas basadas en componentes

3.1 Componentes Software

En las primeras épocas de la computación las computadoras operaban independientemente una de otra sin tener comunicación entre ellas. Las aplicaciones de software eran comúnmente desarrolladas para un propósito específico. La acción de compartir los datos entre sistemas era mínima y se hacía de una manera muy fácil, se transportaban los medios de almacenamiento (tarjetas, cintas, discos, etc.) de un lado a otro. El próximo paso fue conectar las computadoras a través de una red usando protocolos propietarios, luego los protocolos fueron estandarizados. Más tarde llegó la era de los sistemas abiertos y la integración de sistemas por los cuales un cliente podía elegir varios componentes de hardware de diferentes vendedores e integrarlos para crear una configuración necesaria con costos razonables.

Aparecen entonces nuevas técnicas de desarrollo software que se fueron sucediendo unas a otras, desde la programación estructurada y modular hasta la programación orientada a objetos, siempre buscando reducir costos y aumentar la capacidad de reuso. Si bien la programación orientada a objetos fomentó el reuso y permitió reducir costos, no se ha logrado aún el objetivo último: comprar componentes de varios proveedores e integrarlos para formar aplicaciones que a su vez se integren para formar sistemas completos.

Para lograr la integración total de componentes realizados por terceras partes es necesario llegar a un acuerdo común en el que se establezcan los mecanismos necesarios para que esa integración se haga efectiva. Será necesario especificar de manera independiente al lenguaje de programación en el que se desarrolló el componente cuáles

son sus puntos de acceso (funciones), luego será necesario establecer los mecanismos de comunicación entre componentes, que podrían estar ejecutándose en una máquina remota.

En este sentido, y buscando satisfacer esa necesidad de mecanismos estándar e interfaces abiertas, son tres los esfuerzos que más han sobresalido. Por un lado, Microsoft ha introducido en el mercado sus tecnologías COM, DCOM y COM+. Otro participante es Sun Microsystems, que ha presentado Java Beans. El tercero es el Object Management Group, un consorcio integrado por varias industrias importantes, que ha desarrollado CORBA (Common Request Broker Architecture).

3.2 COM / DCOM

Microsoft ha distribuido COM (DCOM) extiende COM (Component Object Model) para soportar comunicación entre objetos en ordenadores distintos, en una LAN, WAN, o incluso en Internet. Con DCOM una aplicación puede ser distribuida en lugares que dan más sentido al cliente y a la aplicación.

Como DCOM es una evolución lógica de COM, se pueden utilizar los componentes creados en aplicaciones basadas en COM, y trasladarlas a entornos distribuidos. DCOM maneja detalles muy bajos de protocolos de red, por lo que uno se puede centrar en la realidad de los negocios: proporcionar soluciones a clientes.

Actualmente DCOM viene con los sistemas operativos Windows 2000, Windows NT, Windows 98, Windows XP y también está disponible una versión para Windows 95 en la página de Microsoft. También hay una implementación de DCOM para Apple Macintosh y se está trabajando en implementaciones para plataformas UNIX como Solaris.

3.2.1 La arquitectura DCOM

DCOM es una extensión de COM, y ésta define como los componentes y sus clientes interactúan entre sí. Esta interacción es definida de tal manera que el cliente y el componente se pueden conectar sin la necesidad de un sistema intermedio. El cliente llama a los métodos del componente sin tener que preocuparse de niveles más complejos

En los actuales sistemas operativos, los procesos están separados unos de otros. Un cliente que necesita comunicarse con un componente en otro proceso no puede llamarlo directamente, y tendrá que utilizar alguna forma de comunicación entre procesos que proporcione el sistema operativo. COM proporciona este tipo de comunicación de una forma transparente: intercepta las llamadas del cliente y las reenvía al componente que está en otro proceso.

Cuando el cliente y el componente residen en distintas máquinas, DCOM simplemente reemplaza la comunicación entre procesos locales por un protocolo de red. Ni el cliente, ni el componente se enteran de que la unión que los conecta es ahora un poco más grande.

Las librerías de COM proporcionan servicios orientados a objetos a los clientes y a los componentes, y utilizan RPC y un proveedor de seguridad para generar paquetes de red estándar que entiendan el protocolo estándar de DCOM.

3.2.2 Los Componentes y su reutilización

Muchas aplicaciones distribuidas no están desarrolladas.

Al existir infraestructuras de hardware, software, componentes, al igual que herramientas, se necesita poder integrarlas y nivelarlas para reducir el desarrollo y el tiempo de trabajo y el coste. DCOM toma ventaja de forma directa y transparente de los componentes COM y herramientas ya existentes. Un gran mercado de todos los componentes disponibles haría posible reducir el tiempo de desarrollo integrando soluciones estandarizadas en las aplicaciones de usuario. Muchos desarrolladores están familiarizados con COM y pueden aplicar fácilmente sus conocimientos a las aplicaciones distribuidas basadas en DCOM.

Cualquier componente que sea desarrollado como una parte de una aplicación distribuida es un candidato para ser reutilizado. Organizando los procesos de desarrollo alrededor del paradigma de los componentes permite continuar aumentando el nivel de funcionalidad en las nuevas aplicaciones y reducir el tiempo de desarrollo.

Diseñando para COM y DCOM se asegura que los componentes creados serán útiles ahora y en el futuro.

3.2.3 Independencia de la localización

Cuando se comienza a implementar una aplicación distribuida en una red real, aparecen distintos conflictos en el diseño:

- Los componentes que interactúan más a menudo deberían estar localizados más cerca.
- Algunos componentes sólo pueden ser ejecutados en máquinas específicas o en lugares específicos.
- Los componentes más pequeños aumentan la flexibilidad, pero también aumentan el tráfico de red.
- Los componentes más grandes reducen el tráfico de red, pero también reducen la flexibilidad.

Con DCOM, estos temas críticos de diseño pueden ser tratados de forma bastante sencilla, ya que estos detalles no se especifican en el código fuente. DCOM olvida completamente la localización de los componentes, ya esté en el mismo proceso que el cliente o en una máquina en cualquier lugar del mundo. En cualquier caso, la forma en la que el cliente se conecta a un componente y llama a los métodos de éste es idéntica. No es únicamente que DCOM no necesite cambios en el código fuente, sino que además no necesita que el programa sea recompilado. Una simple reconfiguración cambia la forma en la que los componentes se conectan entre sí.

La independencia de localización en DCOM simplifica enormemente las tareas de los componentes de aplicaciones distribuidas para alcanzar un nivel de funcionamiento óptimo. Si se supone, por ejemplo, que cierto componente debe ser localizado en una máquina específica y en un lugar determinado. Si la aplicación tiene numerosos componentes pequeños, se puede reducir la carga de la red situándolos en la misma LAN, en la misma

máquina o incluso en el mismo proceso. Si la aplicación está compuesta por un pequeño número de grandes componentes, la carga de red es menor y no es un problema, por tanto se pueden poner en las máquinas más rápidas disponibles independientemente de donde estén situadas.

Con la independencia de localización de DCOM, la aplicación puede combinar componentes relacionados en máquinas "cercanas" entre si, en una sola máquina o incluso en el mismo proceso. Incluso si un gran número de pequeños componentes implementan la funcionalidad de un gran módulo lógico, podrán interactuar eficientemente entre ellos.

3.2.4 Independencia del lenguaje de programación

Una cuestión importante durante el diseño e implementación de una aplicación distribuida es la elección del lenguaje o herramienta de programación. La elección es generalmente un término medio entre el coste de desarrollo, la experiencia disponible y la funcionalidad. Como una extensión de COM, DCOM es completamente independiente del lenguaje. Virtualmente cualquier lenguaje puede ser utilizado para crear componentes COM, y estos componentes pueden ser utilizados por muchos más lenguajes y herramientas. Java, Microsoft Visual C++, Microsoft Visual Basic, Delphi, PowerBuilder, y Micro Focus COBOL interactúan perfectamente con DCOM.

Con la independencia de lenguaje de DCOM, los desarrolladores de aplicaciones pueden elegir las herramientas y lenguajes con los que estén más familiarizados. La independencia del lenguaje permite crear componentes en lenguajes de nivel superior como Microsoft Visual Basic, y después re-implementarlos en distintos lenguajes como C++ o Java, que permiten tomar ventaja de características avanzadas como multi-hilo.

3.2.5 Independencia del protocolo

Muchas aplicaciones distribuidas tienen que ser integradas en la infraestructura de una red existente. Necesitar un protocolo específico de red, obligará a mejorar todos los clientes, lo que es inaceptable en muchas situaciones. Los desarrolladores de aplicaciones tienen que tener cuidado de mantener la aplicación lo más independiente posible de la infraestructura de la red.

DCOM proporciona esta transparencia: DCOM puede utilizar cualquier protocolo de transporte, como TCP/IP, UDP, IPX/SPX y NetBIOS. DCOM proporciona un marco de seguridad a todos estos protocolos.

Los desarrolladores pueden simplemente utilizar las características proporcionadas por DCOM y asegurar que sus aplicaciones son completamente independientes del protocolo.

3.3 CORBA

CORBA es un Middleware o marco de trabajo estándar y abierto de objetos distribuidos que permite a los componentes en la red interoperar en un ambiente común sin importar el lenguaje de desarrollo, sistema operacional, tipo de red, etc. En esta arquitectura, los métodos de un objeto remoto pueden ser invocados "transparentemente" en un ambiente distribuido y heterogéneo a través de un ORB (Object Request Broker). Además del objetivo básico de ejecutar simplemente métodos en objetos remotos, CORBA

adiciona un conjunto de servicios que amplían las potencialidades de éstos objetos y conforman una infraestructura sólida para el desarrollo de aplicaciones críticas de negocio.

CORBA es la respuesta del "Grupo de Gestión de Objetos" (Object Management Group – OMG) a la necesidad de interoperabilidad ante la gran proliferación de productos hardware y software, es decir permitir a una aplicación comunicarse con otra sin importar el tipo de red, protocolo, sistema operacional o lenguaje de desarrollo.

CORBA automatiza muchas tareas comunes y "pesadas" de programación de redes tales como registro, localización y activación de objetos; manejo de errores y excepciones; codificación y decodificación de parámetros, y protocolo de transmisión.

En un ambiente CORBA, cada Implementación de Objeto, define bien su Interfaz a través una especificación normalizada conocida como IDL (Interface Definition Language) a través de la cual en forma Estática (en el momento de compilación) o en forma Dinámica (en el momento de ejecución) un Cliente que requiera el servicio de una Implementación de Objeto, puede ser ejecutada. Las invocaciones a métodos remotos son enviadas por los clientes llamando objetos locales llamados "Stubs" (generados por un compilador de IDL - Estático), el cual intercepta dichas invocaciones y continúa el proceso de llevar y retornar automáticamente dicha invocación. La Implementación del objeto, no tiene que conocer el mecanismo por el cual un Cliente le ha invocado un servicio.

Cuando el Cliente y una Implementación de Objeto están distribuidos por una red, ellos usan el protocolo GIOP/IIOP suministrado por la arquitectura para lograr la comunicación.

La forma de cómo una Implementación de Objeto (desarrollada por un programador de aplicaciones) se conecta a un ORB, es a través de un Adaptador de Objetos. Este adaptador recibe las peticiones por la red e invoca los servicios a la implementación correspondiente.

Actualmente CORBA ya ha resuelto los problemas fundamentales de interoperabilidad y comunicación entre objetos [Omg95a] [Omg99a] [Vin98] y se han definido y especificado un conjunto de servicios comunes requeridos para la construcción de las aplicaciones [Omg95b] [Omg95c] [Omg98c], pero donde hay gran actividad es en la especificación de objetos comunes por dominio de aplicación o conocidas en CORBA como Interfaces de Dominio. Allí se trabajan en áreas como Telecomunicaciones, Medicina, Finanzas, Manufactura, etc. [Omg98a] [Omg98b] [Omg99b] [Omg99c].

CORBA está fundamentado en dos modelos: Un modelo de Objetos, el cual agrega todas las características de Orientación por Objetos como Tipos de Datos, Abstracción, Polimorfismo y Herencia y un modelo de Referencia o Arquitectura conocida como OMA (Object Management Architecture).

3.3.1 Servicios Middleware

Para resolver los problemas inherentes a sistemas heterogéneos y distribuidos, que dificultan la implementación de verdaderas aplicaciones empresariales, los proveedores de software están ofreciendo interfaces de programación y protocolos estándares. Estos servicios se denominan usualmente servicios middleware, porque se encuentran en una capa intermedia, por encima del sistema operativo y del software de red y por debajo de las

aplicaciones de los usuarios finales. En esta sección se describen las características principales de los servicios middleware.

Un servicio middleware es un servicio de propósito general que se ubica entre plataformas y aplicaciones. Por plataformas se entiende el conjunto de servicios de bajo nivel ofrecidos por la arquitectura de un procesador y el conjunto de API´s de un sistema operativo. Como ejemplos de plataformas se pueden citar: Intel x86 y Win-32, SunSPARCStation y Solaris, IBM RS/6000 y AIX, entre otros.

Un servicio middleware está definido por las API´s y el conjunto de protocolos que ofrece. Pueden existir varias implementaciones que satisfagan las especificaciones de protocolos e interfaces. Los componentes middleware se distinguen de aplicaciones finales y de servicios de plataformas específicas por cuatro importantes propiedades:

- Son independientes de las aplicaciones y de las industrias para las que éstas se desarrollan.
- Se pueden ejecutar en múltiples plataformas.
- Se encuentran distribuidos.
- Soportan interfaces y protocolos estándar.

Debido al importante papel que juegan una interfaz estándar en la portabilidad de aplicaciones y un protocolo estándar en la interoperabilidad entre aplicaciones, varios esfuerzos se han realizado para establecer un estándar que pueda ser reconocido por los mayores participantes en la industria del software. Algunos de ellos han alcanzado instituciones como ANSI e ISO; otros han sido propuestos por consorcios de industrias como ser la Open Software Foundation y el Object Management Group y otros han sido impulsados por industrias con una gran cuota de mercado. Este último es el caso de Microsoft con su Windows Open Services Architecture.

CORBA es el estándar propuesto por el OMG. El OMG fue fundado en 1989 y es el más grande consorcio de industrias de la actualidad, con más de 700 compañías que son miembros del grupo. Opera como una organización no comercial sin fines de lucro, cuyo objetivo es lograr establecer todos los estándares necesarios para lograr interoperabilidad en todos los niveles de un mercado de objetos.

Originalmente los esfuerzos de la OMG se centraron en resolver un problema fundamental: cómo lograr que sistemas distribuidos orientados a objetos implementados en diferentes lenguajes y ejecutándose en diferentes plataformas interactúen entre ellos. Más allá de los problemas planteados por la computación distribuida, problemas más simples como la falta de comunicación entre dos sistemas generados por compiladores de C++ distintos que corren en la misma plataforma frenaron los esfuerzos de integración no bien comenzados. Para opacar aún más el escenario, distintos lenguajes de programación ofrecen modelos de objetos distintos. Los primeros años de la OMG estuvieron dedicados a resolver los principales problemas de cableado. Como resultado se obtuvo la primera versión del Common Object Request Broker, publicado en 1991. Hoy en día, el último estándar aprobado de CORBA está por la versión 2.3, y la versión 3.0 está a punto de ser lanzada.

Desde sus principios, el objetivo de CORBA fue permitir la interconexión abierta de distintos lenguajes, implementaciones y plataformas. De esta forma, CORBA cumple con las cuatro propiedades enumeradas como deseables de los servicios middleware. Para lograr estos objetivos, la OMG decidió no establecer estándares binarios (como es el caso de

COM); todo está estandarizado para permitir implementaciones diferentes y permitir que aquellos proveedores que desarrollan CORBA pueden ofrecer valor agregado. La contrapartida es la imposibilidad de interactuar de manera eficiente a nivel binario. Todo producto que sea compatible con CORBA debe utilizar los costosos protocolos de alto nivel.

CORBA está constituido esencialmente de tres partes: un conjunto de interfaces de invocación, el ORB (object request broker) y un conjunto de adaptadores de objetos (objects adapters). CORBA va más allá de simples servicios middleware, provee una infraestructura (framework) para construir aplicaciones orientadas a objetos. Las interfaces definen los servicios que prestan los objetos, el ORB se encarga de la localización e invocación de los métodos sobre los objetos y el object adapter es quien liga la implementación del objeto con el ORB.

Para que las interfaces de invocación y los adaptadores de objetos funcionen correctamente, se deben cumplir dos requisitos importantes. En primer lugar, las interfaces de los objetos deben describirse en un lenguaje común. En segundo lugar, todos los lenguajes en los que se quieran implementar los objetos deben proveer un mapeo entre los elementos propios del lenguaje de programación y el lenguaje común. La primera condición permite generalizar los mecanismos de pasaje de parámetros (marshaling y unmarshaling). La segunda permite relacionar llamadas de un lenguaje en particular o a un lenguaje en particular con el lenguaje de especificación común. Este lenguaje común fue una parte esencial de CORBA desde sus orígenes y es conocido como el OMG IDL: Interface Definition Language. Existen mapeos del OMG IDL a C, C++, Java y Smalltalk,

3.3.2 Arquitectura de CORBA

CORBA define una arquitectura para los objetos distribuidos. El paradigma básico de CORBA es el de un pedido servicios de un objeto distribuido. Todo definido por el OMG que está en términos de este paradigma básico.

Los servicios que un objeto proporciona son dados por su interfaz. Las interfaces se definen en la lengua de la definición de interfaz de OMG (IDL). Los objetos distribuidos son identificados por las referencias del objeto, que son mecanografiadas por los interfaces de IDL.

3.3.3 CORBA como estándar para los objetos distribuidos

Una de las metas de la especificación de CORBA es que los clientes y las puestas en práctica del objeto son portables. La especificación de CORBA define el interfaz de un programador del uso (API) para los clientes de un objeto distribuido así como un API para la puesta en práctica de un objeto de CORBA. Esto significa que el código escrito para un producto de CORBA del vendedor podría, con un mínimo de esfuerzo, ser reescrito para trabajar con el producto de un diverso vendedor. Sin embargo, la realidad de los productos de CORBA en el mercado es hoy que los clientes de CORBA son portables pero las puestas en práctica del objeto necesitan alguna reanudación virar hacia el lado de babor a partir de un producto de CORBA a otro.

CORBA 2.0 agregó interoperabilidad como meta en las especificaciones. El detalle de CORBA 2.0 es que en él definen el protocolo de la red, llamado IIOP (el Internet Enterrar-orbe protocolo), permite que un cliente usando un productos de CORBA de cualquier vendedor para comunicarse hace trampas en los objetos usando un producto de CORBA de

cualquier otro vendedor. Él trabaja de IIOP a través de Internet, o más exacto a través de cualquier puesta en práctica de TCP/IP.

La interoperabilidad es más importante en un sistema distribuido que la portabilidad. IIOP se usa en otros sistemas que no intentan proporcionar el API de CORBA ni siquiera. En particular, IIOP se usa como el protocolo de transporte para una versión de Java RMI (para que llame "RMI encima de IIOP"). Desde que EJB se define correctamente en lo que se refiere a RMI y puede usar IIOP también. Hay varios servidores de la aplicación disponible en el uso del mercado IIOP pero no exponen el API de CORBA entero. Esto se debe a que todos ellos usan IIOP, programas escritos entre sí al interoperar de estos API diferentes y con programas escritos al API de CORBA.

3.3.4 CORBA y el desarrollo basado en componentes

Como se mencionó anteriormente, el desarrollo basado en componentes que se puedan comprar y poner en marcha sin mayores dificultades (*Plug & Play*) es una meta largamente perseguida ya que facilitaría enormemente la reutilización del software. En esta sección se hace una pequeña introducción al desarrollo basado en componentes y el papel en el que se compromete a CORBA en este ámbito.

Muy frecuentemente el término componente se utiliza sin precisar su significado, dando por sentado que todos lo conocen. Pero no siempre es así, y al utilizar el término componente puede suceder que no siempre se esté hablando de lo mismo. Es deseable entonces comenzar con una definición de componente.

Un componente ha sido definido en la *European Conference on Object Oriented Programming* (ECOOP) de 1996 como una "una unidad de composición con interfaces contractuales especificadas y dependencias de contexto explícitas". Un componente de software puede ser desarrollado independientemente y utilizado por terceras partes para integrarlo mediante composición a sus sistemas. Los componentes son para crear software utilizando composición, por eso es esencial que sean independientes y que se presenten en formato binario, permitiendo así distintos vendedores e integración robusta.

Para que un componente pueda ser integrado por terceras partes en sus sistemas, éste deber ser suficientemente auto-contenido y debe proveer una especificación de lo que requiere y provee. En otras palabras, los componentes deben encapsular su implementación e interactuar con otros componentes a través de interfaces bien definidas.

Un componente no es un objeto. A diferencia de los objetos, los componentes no tienen estado. Esto quiere decir que un componente no puede distinguirse de una copia de sí mismo. Un componente puede tomar la forma de un archivo ejecutable o una biblioteca dinámica que usualmente cobra vida a través de objetos, pero no es este un requisito indispensable. De hecho, los primeros componentes conocidos (aunque en su momento no se los haya definido así) fueron las bibliotecas de procedimientos. Sin embargo, los objetos, con sus características de encapsulación y polimorfismo, facilitan la construcción e integración de componentes.

Y al hablar de objetos vale la pena distinguir aquí los objetos de las clases. Una clase es una definición de propiedades y funcionalidades que han de ser provistas por los objetos. A partir de una clase es posible instanciar objetos. Los componentes pueden contener una o más clases y serán los clientes de los componentes quienes soliciten la creación de las instancias de estas clases.

Como escenario de ejemplo para mostrar cómo sería la construcción de software basado en componentes, se podría pensar en una librería. Una librería necesita un sistema que le permita llevar el stock de sus libros, contabilizar sus ventas y gestionar los pedidos a sus proveedores. Quien estuviera a cargo de este proyecto de desarrollo podría adquirir un componente de facturación que provee las siguientes clases (entre otras):

El fabricante del componente de Facturación no sabe de antemano en que ambientes se va a utilizar su componente. Por lo tanto, decide no ofrecer una implementación de la clase producto. Sólo provee su interfaz para que quién utilice el componente sea quien finalmente la implemente de acuerdo a sus necesidades. Sí en cambio ofrece la implementación completa de la clase venta. La especificación de la clase venta podría indicar que:

- *Número de Venta* es un número entero que se asigna automáticamente al crearse una nueva instancia de un objeto Venta.
- *Productos* es un conjunto de productos (existirá en otro lado la definición de conjunto con sus operaciones de inserción y eliminación). Al agregar un producto a la venta se debe especificar su cantidad. Esta cantidad será deducida de la cantidad en stock de ese producto.
- Etc.

El programador del sistema para la librería decide crear una clase Libro que implementa la interfaz de producto (un objeto Libro se comportará como si fuera un objeto producto cuando sea necesario), y así podrá crear ventas de libros.

Si bien este es un ejemplo simple, es un escenario al que se desearía poder llegar con el desarrollo basado en componentes.

Esto es bastante complejo, ya que el programador que debe integrar el componente a su aplicación, surgen algunas incógnitas que debería poder resolver. El fabricante del componente solamente ha entregado un archivo ejecutable que se debe iniciar en la máquina en la que se ejecuta la aplicación y la interfaz definida en un lenguaje de definición de interfaces (IDL) estándar. El programador desea ahora:

- **Mapear la interfaz***:* ¿Cómo hace el programador para mapear las clases que el componente provee al lenguaje en que realizará su implementación final? Seguramente el lenguaje de programación elegido provee mecanismos para definir clases, pero es necesario que la definición que se haga de la clase en ese lenguaje corresponda a la definición que dio el fabricante del componente.
- **Crear objetos**: ¿Cómo hace el programador para crear una instancia del objeto Venta? Es necesario que exista un mecanismo para indicar al componente que cree una instancia del objeto Venta. Una vez creada la instancia ¿Cómo se logra acceder a sus propiedades o métodos?
- **Transparencia**: El componente sería de poca utilidad si su utilización no fuera transparente. Si para cada llamada al componente el programador debiera utilizar un servicio del sistema operativo de llamada entre procesos (IPC), o peor aun si el componente es remoto, un servicio de llamada remota a procedimientos (RPC), está claro que dejaría el componente de lado pues es más trabajo utilizarlo que hacer un programa desde cero.

Estas son sólo algunas de las cuestiones que el programador debería poder resolver para poder utilizar el componente. En el caso de que el programador llegara a comprar otro

componente, es seguro que desea que los mecanismos de utilización sean uniformes para no tener que resolverlas nuevamente. Los servicios middleware que provee CORBA buscan y resuelven estos problemas.

3.3.5 CORBA Services

OMA esta construida sobre un fundamento y arquitectura CORBA que desarrolla la visión de la OMG de componentes de software *plug-and-play*. Los *CORBA Services* especifican servicios básicos que casi todos los objetos necesitan.

Para cada componente de OMA, la OMG provee una especificación formal descrita en OMG IDL (como invocar cada operación dentro de un objeto) y su semántica en lenguaje inglés (que hace cada operación y las reglas de comportamiento). Un proveedor no tiene que suministrar obligatoriamente ningún servicio adicional al ORB, pero si este lo ofrece, deberá hacerlo de acuerdo a la especificación que para este servicio tiene la OMG.

Los *CORBA Services* proveen servicios a nivel de aplicación fundamentales para las aplicaciones orientadas por objetos y componentes en entornos distribuidos. La OMG ha definido alrededor de 13 servicios de objetos. Los cuales son:

Nombres	Propiedades
Ciclo de vida	Relaciones
Eventos	Consulta
Transacciones	Persistencia
Seguridad	Concurrencia
Tiempo	Externalización
Licencia	

De éstos 13 se destacan los siguientes servicios clave:

- Acceso a referencias de objetos a través de la red, soportada por el servicio de Nombres y de Trader.
- Notificación de eventos importantes o cambios de estado en un objeto, soportado por el servicio de Eventos y de Notificación.
- Soporte para semántica transaccional (two-phase commit y rollback) soportado por el servicio de Transacciones.
- Soporte para seguridad, soportada por el servicio de Seguridad.

- **Nombres:** Permite a componentes descubrir otros componentes en un ambiente distribuido, básicamente es un servicio de localización que asocia identificadores a manejadores que proveen una forma de conectar el componente deseado en un sistema distribuido. Este asocia nombres - organizados jerárquicamente - a objetos.

- **Ciclo de vida:** Básicamente es un servicio de configuración, define servicios y convenciones para crear, borrar, copiar y mover componentes en un sistema distribuido.

- **Eventos:** Implementa un modelo de comunicación desacoplado, basado en el paradigma publicación/suscripción. En este modelo, uno o más publicadores pueden enviar mensajes relacionados con un tópico específico, mientras que un grupo de suscriptores recibe los mensajes asincrónicamente. Con estos mecanismos los publicadores pueden generar evento sin tener que conocer la identificación de sus consumidores y viceversa. Hay dos acercamientos, modelo Push y modelo Pull, en el modelo Push los publicadores toman la iniciativa de iniciar la comunicación, en el modelo Pull, los suscriptores requieren eventos de los publicadores.

- **Transacciones:** Gestiona interacciones entre objetos distribuidos estableciendo puntos de Commit y delimitación de transacciones. Este servicio soporta varios modelos y permite interoperabilidad entre diferentes arquitecturas de red y modelos de programación.

- **Seguridad:** Controla la identificación de los elementos del sistema distribuido (usuarios, objetos, componentes, etc.) que permite verificar que un cliente esta autorizado a acceder los servicios de una Impementación remota. Adicionalmente permite la comunicación segura sobre enlaces de comunicación inseguros, ofreciendo confidencialidad e integridad de la información transmitida.

- **Tiempo:** Suministra información del tiempo y permite la definición de llamadas periódicas.

- **Licencia:** Controla la utilización de objetos específicos, inhibiendo uso inadecuado de derechos y propiedad intelectual.

- **Propiedades:** Provee la habilidad de dinámicamente asociar propiedades a los objetos los cuales pueden ser consultados o modificados por otros elementos.

- **Relaciones:** Permite la definición del papel ejecutado por objetos CORBA en una relación.

- **Consulta:** Permite a los usuarios y objetos invocar operaciones de consulta sobre colecciones de objetos.

- **Persistencia:** Provee mecanismos para retener y mantener el estado de objetos persistentes.

- **Concurrencia:** permite la coordinación de múltiples accesos a recursos compartidos a través de la provisión de varios modelos de locks y permite resolución flexible de conflictos.

- **Externalización:** define convenciones y protocolos para representar el estado de información relacionado a un objeto en la forma de una secuencia de datos, permitiendo a esta información ser almacenada o transferida a otras localizaciones.

3.3.6 CORBA Facilities Horizontales

Estas facilidades CORBA tanto horizontales como verticales, son diseñadas para completar la arquitectura entre los Servicios básicos de CORBA y las aplicaciones especificas de industria. Con una arquitectura completa, las compañías compartirán datos a nivel de aplicación y funcionalidades para la integración de diferentes sistemas de

información. Las facilidades representan un punto medio entre una aplicación particular y los servicios y ORB.

La OMG ha definido como Facilidades horizontales las siguientes:

- Interfaz de usuario
- Administración de información
- Administración de sistemas
- Administración de tareas

Hoy en día estas especificaciones han sido adheridas a ORBOS (ORB y Servicios) y ya no esta como un grupo aparte. Específicamente a nivel de facilidades verticales la OMG ha definido las siguientes:

- Especificación para la administración de sistemas XCMF.
- Facilidad para colar de impresión.

3.3.7 CORBA Facilities Verticales o de Dominio

La potencialidad que representa el lenguaje IDL para la especificación de un objeto o componente ha permitido trabajar en la normalización de intereses comunes para un sector de mercado particular y que una vez se llegue a un acuerdo en cuanto a estas especificaciones, sería estándar dentro de este mercado.

Para esto se ha creado el Domain Technology Committee (DTC) y esta a su vez está organizada en una serie de Domain Task Forces (DTF) los cuales escriben documentos de requerimientos (RFI y RFP) para nuevas especificaciones y evalúan y recomiendan especificaciones candidatas. Basados en las recomendaciones de los DTF, el DTC conduce un proceso formal de votación para asegurar que cumple todos los requerimientos del sector y no sólo de quien haya propuesto. Posteriormente estas recomendaciones requieren ser enviadas a la Junta de Directores de la OMG para hacerla una especificación oficial. Actualmente hay 8 DTF:

- Objetos de negocio
- Finanzas y seguros
- Comercio Electrónico
- Manufactura
- Salud o Medicina
- Telecomunicaciones
- Transportes
- Investigación de ciencias de la vida

También bajo la OMG pero que no tienen DTF se encuentran dos Grupos de Interés Especial:

- Utilities (principalmente energía eléctrica)
- Estadística

Seis especificaciones ya han sido adoptadas oficialmente como estándares de dominio vertical, ellos son:

- Facilidad Currency del DTF de Finanzas

- Un conjunto de Habilitadores para la administración de datos de productos, del DTF de manufactura.
- Servicio de Identificación de Personas (PIDS) de CORBAMed
- Servicio de Consulta Lexicon de CORBAMed
- Control y Administración de flujos de Audio/Vídeo, del DTF de telecomunicaciones
- Servicio de Notificación, del DTF de Telecomunicaciones.

3.3.8 Nuevas especificaciones de CORBA

Después que de fuese completada y normalizada la versión 2.0 en 1996, la OMG continuo trabajando en la incorporación de aspectos importantes que deben ser tenidos en cuenta en un sistema distribuido y ha ampliado su modelo de objetos para incorporar aspectos como: múltiples interfaces por objeto, paso de objetos por valor, modelo de componentes, soporte para tiempo real y tolerancia a fallos entre otros [Omg00a]. CORBA 3.0 se refiere a una serie de nuevas especificaciones que unidas dan una nueva dimensión a CORBA. Estas nuevas especificaciones están divididas en tres categorías:

- Integración con Internet.
- Control de Calidad de Servicio
- Arquitectura de componentes CORBA

Por otro lado, han aumentado las especificaciones de mercados verticales, o lo que en CORBA se conoce como Dominios Verticales y mediante la utilización de IDL, existen muchos mercados estandarizando en CORBA (Finanzas, Seguros, Comercio electrónico, Medicina, Manufactura, Telecomunicaciones, Transportes, Investigación y Objetos de negocio).

3.3.9 Integración con Internet

Esta integración esta siendo desarrollada por la especificación "firewall". La especificación CORBA 3 de firewalls define firewalls a nivel de transporte, de aplicación y conexiones bidireccionales GIOP útiles para callbacks y notificación de eventos.

Los firewalls de transporte trabajan a nivel de TCP, para lo que la IANA ha reservado los puertos bien conocidos 683 para IIOP y 684 para IIOP sobre SSL. También hay una especificación de CORBA sobre SOCKS.

En CORBA es frecuente que un objeto invoque un método (callback) o notifique de algún suceso a su cliente, por esto el objeto puede comportarse como cliente, por esto generalmente se requiere abrir una nueva conexión TCP en sentido inverso el cual puede ser detenido por un firewall. Bajo esta especificación, una conexión IIOP se le permite llevar invocaciones en el sentido inverso bajo ciertas condiciones que no comprometan la seguridad de la conexión.

3.3.10 Modelo de componentes de CORBA (CORBABeans)

CORBA Components extiende el modelo de objeto de la OMG para incluir un número de características importantes en un sistema distribuido. La noción de componente puede no corresponder al modelo uno a uno ni de un objeto CORBA, esta centrado más en la relación de un Componente con un conjunto de interfaces.

Los componentes tienen identificadores de instancias, así como propiedades que son externamente accedidas y que soportan mecanismos de notificación (servicio de eventos) y validación cuando alguna propiedad cambia.

Los componentes de CORBA no solo serán trasladados a los lenguajes ya soportados, sino también a otros modelos de componentes como los Java Beans. Igualmente se esta trabajando en una especificación para un lenguaje de scripting que facilite el nivel de usuario la utilización de componentes.

3.4 Common Gateway Interface (CGI)

Common Gateway Interface es una interfaz al servidor Web que permite extender la funcionalidad de éste. Con CGI se puede interactuar con los usuarios que acceden a un sitio en particular. En un nivel teórico, los CGI permiten extender las capacidades del servidor para interpretar las entradas obtenidas del browser (navegador) y regresar la información apropiada de acuerdo a la entrada del usuario. En un nivel práctico, CGI es una interfaz que facilita la escritura de programas para que se comuniquen fácilmente con el servidor.

Usualmente, si se desea extender las capacidades del servidor Web, se tendría que modificar el servidor manualmente. Esta no es una solución deseable puesto que requiere un entendimiento de bajo nivel de programación de red sobre el Internet y el protocolo World Wide Web. También requiere editar y recompilar el código fuente del servidor o escribir un servidor dedicado para cada tarea. Por ejemplo, si se quiere extender el servidor para que actúe como una compuerta Web a e-mail que tomará la entrada del usuario desde el browser y lo reenviará a otro usuario, se tendría que insertar código en el servidor que interpretara la entrada desde el browser, reenviar la entrada al otro usuario y regresar una respuesta al browser sobre una conexión de red.

Primeramente, ésta tarea requiere hacer accesible el código del servidor, algo que no siempre es posible. En segundo lugar, es difícil y requiere excesivo conocimiento técnico. Tercero, funciona solamente para un servidor específico. Si se quiere mover el servidor Web a una plataforma diferente, se tendrá que reiniciar o al menos desperdiciar mucho tiempo al trasladar el código a esa plataforma.

CGI proporciona una solución portable y simple a estos problemas. El protocolo CGI define una forma estándar para que los programas se comuniquen con el servidor Web. Sin mucho conocimiento especial, se puede escribir un programa en cualquier lenguaje de computación que interactúe con el servidor Web. Este programa trabajará con todos los servidores Web que entiendan al protocolo CGI.

La comunicación CGI está manejada sobre la entrada y salida estándar, lo que significa que si se conoce como imprimir en pantalla y leer datos utilizando el lenguaje de programación, se puede escribir una aplicación sobre el servidor Web. Programar las aplicaciones CGI es casi equivalente a programar cualquier otra aplicación. Por ejemplo, si se desea programar un programa "Hola México", se utilizan las funciones de imprimir en pantalla del lenguaje y el formato definido para programas CGI para imprimir en pantalla el mensaje apropiado.

Debido a que CGI es una "interfaz común", no está restringida a ningún lenguaje de computación en particular. Una pregunta importante es ¿qué lenguaje de programación se utiliza para programar CGIs? Se puede utilizar cualquier lenguaje que realice lo siguiente:

- Imprimir a la salida estándar.
- Leer desde la entrada estándar.
- Leer desde variables de ambiente.

La mayoría de los lenguajes de programación y muchos lenguajes desempeñan estas tres actividades y se pueden utilizan cualquiera de ellas.

Los lenguajes caen bajo una de las siguientes clases: compilar o interpretar. Un lenguaje compilador tal como C o C++ tiende a ser más pequeño y más rápido, mientras que el lenguaje intérprete tal como Perl o Rexx requieren cargar, algunas veces, un intérprete grande antes de comenzar. Adicionalmente, se puede distribuir el código binario (código compilado en lenguaje máquina) sin el código fuente, si el lenguaje utilizado es del tipo compilador. La distribución de scripts interpretados normalmente significa distribuir el código fuente.

Antes de escoger el lenguaje, se deben considerar las prioridades. Se necesita balancear la ganancia de velocidad y eficiencia de un lenguaje de programación contra la facilidad de programación de otro.

Existen alternativas importantes para aplicaciones CGI. Ahora, muchos servidores incluyen una programación API que hace más fácil programar directamente extensiones al servidor en contra parte a separar aplicaciones CGI. Los servidores APIs tienden a ser más eficientes que los programas CGI. Algunas aplicaciones se manejan por nuevas tecnologías desarrolladas en la parte del cliente (en lugar del servidor) tales como Java.

3.5 Java

Java es una arquitectura neutral, orientada a objetos, portable y un lenguaje de programación de alto desempeño que proporciona un ambiente de ejecución dinámica, distribuida, robusta, segura y multihilos. La principal ventaja de Java para computación distribuida radica en la capacidad de descargar el ambiente. En términos de una arquitectura de objeto distribuido totalmente nueva, Java proporciona las siguientes opciones: Java Remote Method Invocation (RMI), Java IDL y la empresa JavaBean. La especificación RMI es un API que nos permite crear objetos escritos puramente en lenguaje de programación Java, cuyos métodos se invocan de una Máquina Virtual Java diferente (JVM Java Virtual Machine). La tecnología Java IDL para objetos distribuidos facilita que los objetos interactúen a pesar de estar escritos en lenguaje de programación Java u otro lenguaje tal como C, C++, COBOL, entre otros.

3.5.1 Java RMI

Básicamente RMI proporciona la capacidad para llamadas a métodos sobre objetos remotos, los cuales convierten al componente de transporte del objeto en arquitectura de objeto distribuido. También proporciona mecanismos para el registro y persistencia del objeto. Ofrece servicios distribuidos tales como Java IDL que proporciona una forma de conectar, transparentemente, a los clientes Java a los servidores de red utilizando la industria estándar: Lenguaje de Definición de Interfaces (IDL Interface Definition Language). Las aplicaciones RMI están, a menudo, compuestas de dos programas separados: un servidor y un cliente. Una aplicación común del servidor crea algunos objetos remotos, realiza referencias para accederlos y se encuentra en espera de que los clientes invoquen los métodos sobre éstos objetos remotos. Una aplicación del cliente tiene una

referencia remota a uno o más objetos remotos en el servidor y entonces invoca al método sobre ellos. RMI proporciona los mecanismos a través de los cuales el servidor y el cliente se comunican e intercambian información. Las aplicaciones utilizan uno o dos mecanismos para obtener referencias a objetos remotos. Una aplicación registra sus objetos remotos con la facilidad de denominación del RMI, o bien la aplicación pasa y regresa la referencia a los objetos remotos como parte de su operación normal. Los detalles de comunicación entre objetos remotos está a cargo del RMI; para el programador, la comunicación remota se asemeja a la invocación del método Java. Dado que RMI permite que un solicitante pase objetos a objetos remotos, RMI proporciona los mecanismos necesarios para cargar un código de objeto, así como también de transmitir sus datos. RMI soporta su propio protocolo de transporte denominado Protocolo de Mensajes Remotos Java (JRMP Java Remote Messaging Protocol) para definir el conjunto de formatos de mensajes que permiten que los datos pasen a través de una red de computadoras a otra.

La siguiente ilustración representa una aplicación distribuida RMI que utiliza el registro para obtener una referencia a un objeto remoto. El servidor llama al registro para asociar (o ligar) un nombre con un objeto remoto. El cliente busca el objeto remoto por su nombre en el registro del servidor y entonces invoca un método sobre él. La ilustración también muestra que el sistema RMI utiliza un servidor Web para cargar los códigos en bytes de las clases, del servidor al cliente y del cliente al servidor, para los objetos cuando se les necesita.

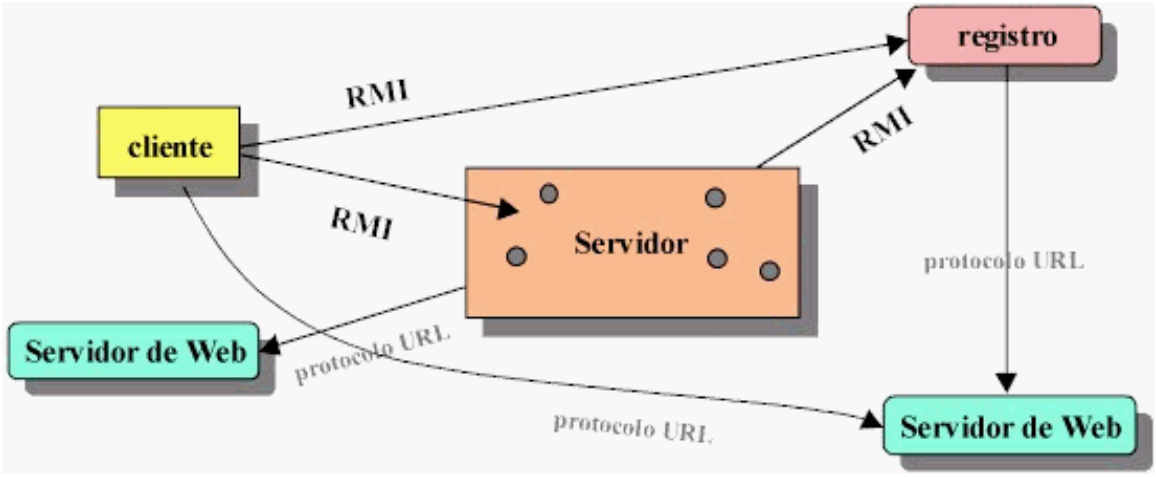


Ilustración 1

RMI pasa objetos por su tipo verdadero permitiendo que nuevos tipos sean introducidos en una máquina virtual remota por consiguiente extendiendo el comportamiento de una aplicación de manera dinámica. RMI trata un objeto remoto de manera diferente de un objeto local cuando el objeto se le pasa de una máquina virtual a otra. En lugar de hacer una copia de la implementación del objeto en la máquina virtual receptora. El solicitante invoca un método en el stub local que tiene como responsabilidad cargar la llamada al método en el objeto remoto. Un stub para objetos remotos implementa el mismo conjunto de interfaces remotas que el mismo objeto remoto implementa. Esto permite que un stub se acople a cualquiera de las interfaces que el objeto remoto implemente. Sin embargo, esto también significa que solamente aquellos métodos, definidos en una interfaz remota, estén disponibles para su solicitud en la máquina virtual receptora.

3.5.2 Java IDL

Java IDL esta basado en CORBA (Common Object Request Brokerage Architecture). Una característica clave de CORBA es un lenguaje-neutral IDL (Interface Definition Language). Cada lenguaje que soporta CORBA tiene su propio traductor IDL y como su nombre lo dice, Java IDL soporta la traducción para Java. Para soportar la interacción entre objetos en programas separados, Java IDL proporciona un ORB (Object Request Broker). El ORB es una librería clase que facilita la comunicación de bajo nivel entre las aplicaciones Java IDL y otras aplicaciones CORBA.

3.5.3 ¿RMI o IDL?

- Java RMI es una solución 100% Java para objetos remotos que proporciona todas las ventajas de las capacidades de Java "una vez escrito, ejecutarlo en cualquier parte". Los servidores y clientes desarrollados con Java RMI se muestran en cualquier lugar en la red sobre cualquier plataforma que soporta el ambiente de ejecución de Java. Java IDL, en contraste, está basado en una industria estándar para solicitar, de manera remota, a objetos escritos en cualquier lenguaje de programación. Como resultado, Java IDL proporciona un medio para conectar aplicaciones "transferidas" que aún cubren las necesidades de comercio pero que fueron escritos en otros lenguajes.
- Actualmente Java RMI y Java IDL emplean protocolos diferentes para la comunicación entre objetos sobre diferentes plataformas. Java IDL utiliza el protocolo estándar de CORBA IIOP (Internet Inter-Orb Protocol). Al igual que IDL, IIOP facilita la residencia de objetos en diversas plataformas y escritos en diversos lenguajes para interactuar de manera estándar. Actualmente Java RMI utiliza el protocolo JRMP (Java Remote Messaging Protocol), un protocolo desarrollado específicamente para los objetos remotos de Java.
- En Java IDL, un cliente interactúa con un objeto remoto por referencia. Esto es, el cliente nunca tiene una copia actual del objeto servidor en su propio ambiente de ejecución. En su lugar, el cliente utiliza stubs en la ejecución local para manipular el objeto servidor residiendo sobre la plataforma remota. En contraste, RMI facilita que un cliente interactúe con un objeto remoto por referencia, o por su valor descargándolo y manipulándolo en el ambiente de ejecución local. Esto se debe a que todos los objetos en RMI son objetos Java. RMI utiliza las capacidades de serialización del objeto, que proporciona el lenguaje Java, para transportar los objetos desde el servidor al cliente. Java IDL, dado que interactúa con objetos escritos en cualquier lenguaje, no toma ventaja de "una vez escrito, ejecutarlo en cualquier parte", característica del lenguaje de programación Java.

3.6 Comparación de Arquitecturas

Neutralidad de la Arquitectura vs. Transparencia en la Comunicación: CORBA proporciona transparencia en la comunicación, transparencia local/remota e independencia de la plataforma. Java RMI proporciona una arquitectura neutral, transparencia en la comunicación pero no proporciona transparencia local/remota. DCOM proporciona soporte para solamente plataformas Windows. CGI, por su parte, proporciona transparencia en la comunicación, transparencia local/remota, independencia de la plataforma y una arquitectura neutral.

Abstracción: CORBA proporciona una solución abstracta de cómputo distribuido, que se puede implementar a través de plataformas, sistemas operativos y lenguajes. Mientras que DCOM, CGI y los Componentes Distribuidos Java especifican detalles de implementación.

Independencia del Lenguaje: Los estándares de CORBA, CGI y DCOM se utilizan en distintos lenguajes mientras que Java RMI se utilizan solamente con lenguaje Java. Sin embargo, Java IDL se utiliza en diversos lenguajes.

Java RMI también es un ORB en el sentido genérico, esto es nativo al lenguaje Java.

Complejidad y Madurez: CORBA es más maduro que las otras tres tecnologías y están disponibles diversas implementaciones de éste estándar.

CORBA utiliza IDL debido a lo cual, el desarrollo de aplicaciones es más complejo que DCOM, CGI y los Componentes Java.

Tecnología: A pesar que Java y CGI's proporciona medios para cómputo distribuido es aún una tecnología de programación mientras que CORBA es una tecnología de integración. DCOM es también una tecnología de integración pero solamente bajo plataformas Windows.

Servicios Distribuidos: CORBA proporciona un conjunto más completo de servicios distribuidos que Java y DCOM. Aunque Java esta creciendo con API's para proporcionar éstos servicios.

Paso por Valor/Referencia: CORBA actualmente no soporta el paso de objetos por valor mientras que DCOM, Java RMI y CGI's (en desarrollo a través de URN) si los soporta.

Simplicidad en el desarrollo de aplicaciones: Las aplicaciones distribuidas se producen fácilmente empleando DCOM y Java puesto que existen herramientas disponibles para el desarrollo de la aplicación. CORBA carece de esta facilidad y así es difícil construir aplicaciones CORBA, lo mismo sucede con CGI's.

Herencia: DCOM proporciona soporte para objetos que tienen interfaces múltiples pero no soporta herencia en la interfaz. CORBA soporta herencia en la interfaz pero no soporta objetos que tienen interfaces múltiples.

Recolección de Basura (Garbage Collection GC): DCOM y los objetos Java tienen recolección de basura, lo cual no tiene CORBA ni CGI's.

Otros: CORBA realiza la tarea de registro de objetos, la generación de referencia a objeto y la inicialización del skeleton de manera implícita. En DCOM estas tareas son manejadas por el programador explícitamente o bien, se realizan dinámicamente durante el tiempo de ejecución.

El protocolo DCOM esta fuertemente atado a RPC, CORBA no lo está.

Las tecnologías desarrolladas en la parte cliente tales como Java, no son probables que reemplacen a los CGI porque existen ciertas aplicaciones que las del lado del servidor están mejor adaptados para realizarse.

Muchas de las limitaciones de CGI son limitaciones de HTML o HTTP. Dado que los estándares del Web, en general, que involucran estas limitaciones no afectan las capacidades de CGI.

Arquitectura	Independencia del Lenguaje	Tipo del Lenguaje	Tecnología	Paso por Valor/Referencia	Independencia de Plataforma
CORBA	*si*	*integración*	*por referencia*	*no*	*si*
DCOM	*si*	*integración (Windows)*	*por valor*	*si*	*no, sólo para Windows*
JAVA	*sólo con lenguaje Java*	*programación*	*por valor*	*si*	*si*
CGI	*si*	*programación*	*ambas*	*no*	*si*

Al programar con los programas de ayuda que ofrecen estas tecnologías se pueden todo tipo de componentes, siempre a nivel de lenguaje de programación.

Si escogemos algún programa, en concreto por ejemplo JBuilder, Delphi o Visual C++, queda visible que ofrecen un gran número de componentes muy útiles para añadir a una aplicación creada por un programador, como Frames, Canvas, Images, Scroll Bars, etc. e incluso algunos permiten añadir animaciones o sonidos.

Estos componentes como bien se conoce por lo anteriormente mencionado cumplen unos rigurosos controles antes de salir al mercado como un producto ya finalizado.

En estos controles se establece que el componente ha de cumplir una función concreta basándose en unos patrones estrictos de generalidad. Mediante estos patrones al programador se le ofrece la posibilidad de modificar las características del componente para amoldarlo a la aplicación que éste esté realizando.

Estos componentes plantean una seria duda, como se puede comprobar todos ellos están orientados a aplicaciones gráficas, y algunos en casos muy determinados y especiales a aplicaciones sonoras, pero sería posible crear un componente orientado a una aplicación más bien física, es decir, existen componentes que están orientados a partes del hardware y no únicamente del software.

Esto es en verdad una duda razonable, ya que cualquier persona que indague un poco en este tema se dará cuenta de que se ofrecen posibilidad de controlar el hardware en lenguajes de más bajo nivel, pero no se presentan componentes para controlar hardware en lenguajes de más alto nivel, aún a sabiendas de que continuamente se utilizan aplicaciones que sí que controlan aspectos físicos de un computador.

Pero la pregunta es, esas aplicaciones han sido creadas desde cero o se han introducido en las mismas componentes de ayuda para controlar ese hardware.

CAPÍTULO IV

Diseño de Componentes

4.1 Selección y Preparación previa al Diseño

Tal y como se ha especificado en los objetivos, redactados en el apartado 1.1, lo primero para la consecución de los mismos debe ser marcar sobre que elemento hardware se van a realizar los componentes.

Lo mejor a tener en cuenta es que el elemento hardware a escoger esté lo suficientemente extendido, en primer lugar porque resultará más fácil encontrar las librerías con las clases necesarias para controlar el elemento hardware en lenguajes de programación con programación orientada a objetos y en segundo lugar porque así se conseguirá que el proyecto sea lo más útil, importante e interesante posible.

Así pues, se escogerá un elemento que esté en la mayoría de las computadoras, también es conveniente escoger un elemento fijo en una computadora, es decir, que actúe como periférico.

Por último, se ha de tener en cuenta que ha de ser un elemento estandarizado, ya que si se escoge un elemento no estandarizado se tendrá que realizar el componente basándose en la marca del fabricante sobre el cual se haya realizado. Además, al ceñirse sobre lo que se quiere conseguir con este proyecto, lo más lógico es escoger un elemento lo más sencillo posible.

Después de evaluar todo lo expuesto se decide que el elemento sobre el que se vaya a realizar la realización del componente sea el puerto serie del ordenador.

Ahora se tiene que escoger con que lenguajes se van a realizar los componentes sobre el puerto serie.

A lo largo de la memoria del proyecto se han visto numerosos lenguajes de programación y varios tipos de componentes que se pueden realizar, pero lo más acorde no sería realizar el componente en todos los lenguajes, si no realizar un número de componentes óptimo de acuerdo a que se satisfaga a la mayor parte de la demanda de la cuota de mercado del componente.

De acuerdo con esto se decide que en el proyecto se van a realizar dos componentes, uno será un componente JavaBeans que controla el puerto serie y el otro será un componente .Net que controla el puerto serie.

Por qué estos y no otros, por qué escoger JavaBeans y .Net. Sencillamente por lo explicado en el párrafo anterior, con estos dos componentes se cubren la mayoría de sistemas del mercado.

Se ha escogido el componente .Net porque este es el desarrollado como estándar por la empresa Microsoft y es sin duda el más extendido de cuantos se conocen, además ofrece la posibilidad de interoperar con CORBA, lo cual da mayor robustez y compatibilidad al componente creado.

Por otra parte también se ha escogido JavaBeans, el primer lugar porque al igual que el componente .Net ambos ofrecen la posibilidad de interoperar con CORBA, y en segundo lugar porque también es uno de los más extendidos, esto se debe a la portabilidad que ofrece Java con respecto a otros lenguajes de programación, la portabilidad de Java se basa en que el código es perfectamente portable sin ningún tipo de modificación para cualquier Sistema Operativo, variando en los Sistemas Operativos la máquina virtual de Java que ofrece Sun, de acuerdo con esto todos los códigos de aplicaciones correctamente hechas en Java son portables a cualquier Sistema Operativo para el cual Sun ofrezca una máquina virtual de Java. Además, se ha escogido este tipo de lenguaje porque cada vez hay más programadores adeptos a él.

4.2 Componente Bean

Para la realización del componente JavaBean se han utilizado dos programas, Kawa para la generación de código y BeanBox para la utilización del componente.

Ambos requieren para su correcto funcionamiento al menos la versión de JDK 1.2, aunque se han utilizado la JDK 1.4 (Java Development Kit) y la JRE 1.4 (Java Runtime Envieroment), todo lo mencionado es software de libre distribución y se puede encontrar en la página de Sun www.sun.com.

En los paquetes anteriormente mencionados se descargan las clases más características de un lenguaje de programación, pero no todas.

Para poder controlar el puerto serie mediante el lenguaje de programación java Sun en su página ofrece el paquete comm.jar, este aparece comprimido en un archivo con el nombre javacomm20-win32.zip, este proyecto se ha realizado en Windows XP, por lo que se procede a explicar como cargar el paquete en este Sistema Operativo.

Dentro del archivo javacomm20-win32.zip se pueden encontrar los archivos comm.jar y javax.comm.properties, los cuales hay que introducir en el directorio \lib que está contenido

dentro del directorio donde se instaló el JRE. También se encuentra el archivo win32com.dll, el cual se ha de copiar a los subdirectorios \system y \system32 del directorio donde se instaló Windows XP, por último hay que añadir una variable de entorno CLASSPATH que haga referencia al paquete comm.jar, esto último se realiza introduciendo en el archivo autoexec.nt de windows la siguiente línea de comandos:

```
SET CLASSPATH=.;"Unidad":\"Directorio del JRE"\lib\comm.jar
```

Ya estaría instalado el paquete comm.jar para controlar el puerto serie, una vez se haya reiniciado el computador se estaría en posición de utilizarlo.

Además se ha de añadir que dentro del archivo javacomm20-win32.zip aparecen algunos ejemplos, la licencia del paquete y los docs de ayuda del paquete, estos últimos son de gran ayuda a la hora de programar, en ellos se indicará el nombre de métodos, así como los tipos de variables que requieren, lo que devuelven y que realizan, el nombre de clase y subclases, y el nombre y tipo de variables internas.

Una vez se haya realizado toda la instalación lo primero que se debe hacer es aprender como funciona el puerto serie. En un principio se puede comprobar como en el paquete que se ha instalado no sólo aparece el puerto serie, si no que también aparece el controlador Java del puerto paralelo.

El paquete ofrece dos clases CommPort y CommPortIdentifier.

La clase CommPortIdentifier interactúa con CommPort. Esta clase sirve para enumerar los puertos y guardarlos en variables del tipo CommPort según un identificador.

La CommPort es la que permite realizar todo tipo de cambios sobre las características de un puerto seleccionado de acuerdo con el identificador obtenido por la clase CommPortIdentifier, además de esta extienden dos más que son SerialPort y ParallelPort, las cuales van a clasificar los puertos según se esté tratando con un puerto serie o con un puerto paralelo.

Para facilitar la explicación véase el diagrama UML de la figura 2.

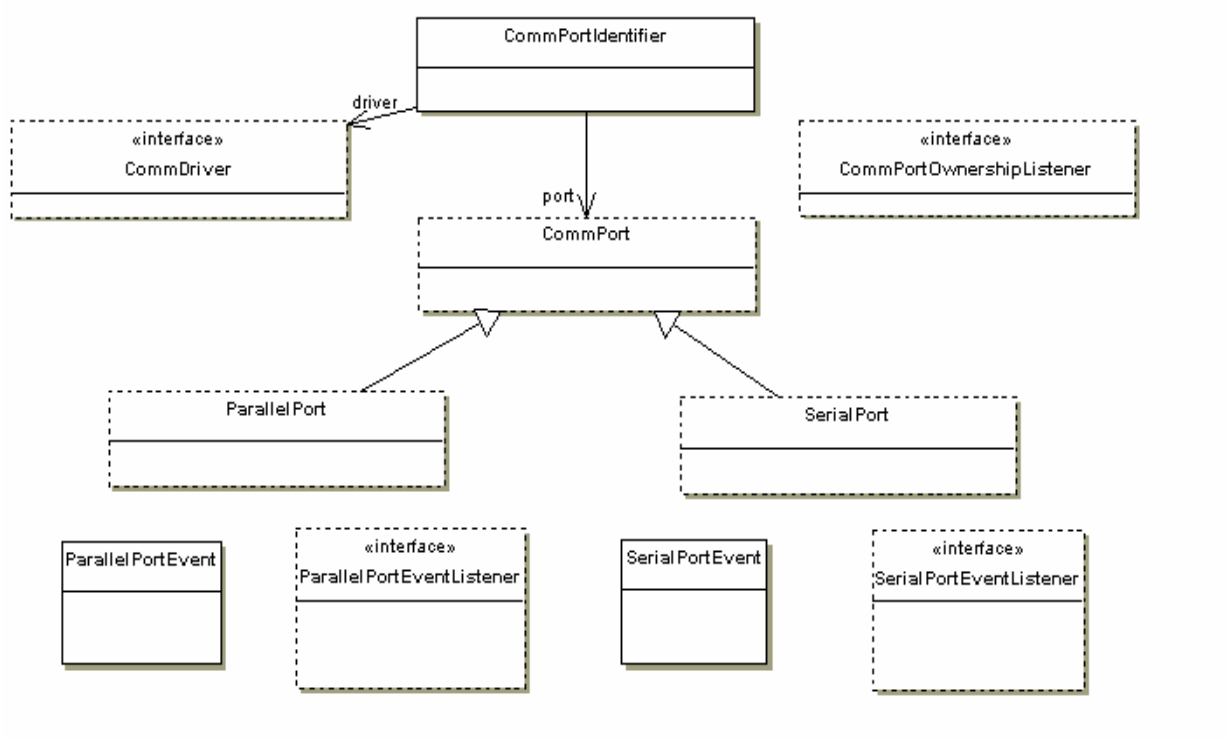


figura 2

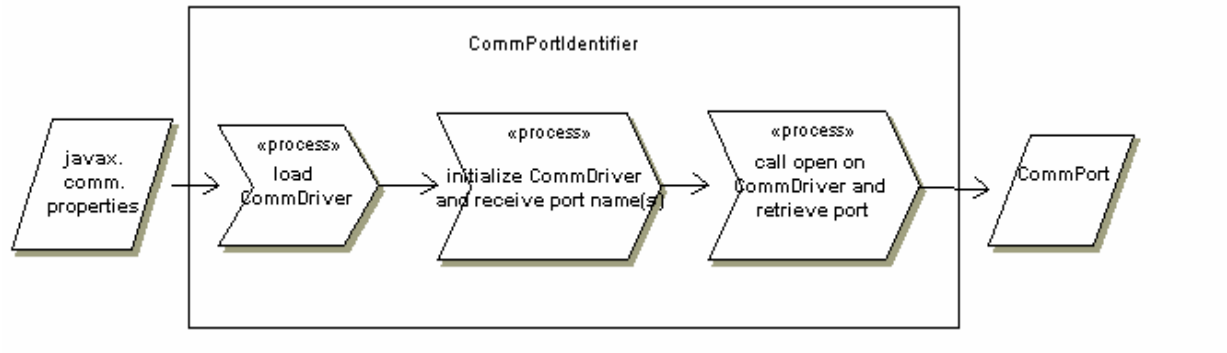


figura 3

En figura 3 además se puede ver una pequeña correlación que explica el funcionamiento del fiche javax.comm.properties. Aquí se comprueba que este fichero es el que ejecuta el driver mediante CommPortIdentifier, primero lo lee, después lo inicializa y se reciben los nombres de los puerto, y después del procesan y se abren pasando en último lugar a CommPort.

Con lo explicado anteriormente ya se conoce el cómo generar un código Java que controle el puerto serie del ordenador, ahora sólo faltaría convertir este en un componente.

Si acaso, además sería conveniente y como no también interesante conocer más en profundidad cómo enumerar los puertos serie y cómo a partir de su enumeración se comienzan a utilizar como objetos del tipo CommPort.

En la sección de código indicada a continuación como figura 4 se puede ver como una variable llamada ports del tipo Enumeration, mediante el método nextElement() va recogiendo los distintos identificadores de puerto especificados por una variable llamado port del tipo CommPortIdentifier, y esta mediante el método getPortType() y el switch()

creado va añadiendo los puertos serie a la variable vector puertosSerie y desechando los puertos paralelos.

```
private Enumeration ports;
 private Vector puertosSerie;
 private CommPortIdentifier port;

 puertosSerie = new Vector();
        ports = CommPortIdentifier.getPortIdentifiers();

        while (ports.hasMoreElements())
        {
          port = (CommPortIdentifier)ports.nextElement();

          switch (port.getPortType())
          {
            case CommPortIdentifier.PORT_PARALLEL:
              break;
            case CommPortIdentifier.PORT_SERIAL:
                    puertosSerie.add(port);
              break;
            default:  // Caso por defecto, no debería ocurrir nunca
              break;
          }  //Fin del switch

        }       //Fin del while
```

Figura 4

Según lo indicado en la página de Sun para crear un JavaBean simplemente, hay que seguir 3 pasos.

- El JavaBean ha de implementar Serializable.
- Se ha de crear un archivo llamado manifest.tmp, en el cual se indique en su interior el nombre del archivo de la clase o clases del JavaBean.
- Se ha de comprimir todo en un archivo de extensión .jar.

Según la documentación de la página de Sun, el componente cogerá por defecto los métodos generados en la clase como las propiedades a modificar.

Después de realizar estos pasos se puede comprobar que el componente JavaBean no funciona. Esto es fácilmente explicable, por definición y por incompatibilidad, ningún JavaBean puede ser de un clase abstract, y el paquete comm.jar todas las clases que contiene son abstract, es más todos los paquetes orientados a elementos físicos de un ordenador recopilados durante la realización de este proyecto son de clase de tipo abstract.

Esto supone un inconveniente considerable, y daría una explicación razonable a por qué no se realizan este tipo de componentes.

Con todo esto se obtiene una pequeña conclusión, y es que no es posible realizar un componente estrictamente orientado a un elemento físico con JavaBeans.

Aún así queda una solución posible, es evidente, pasa por crear un componente JavaBean gráfico y que este opere como una aplicación de tal forma de que se base únicamente en la variación de las características del puerto serie.

En realidad es esto lo que se ha realizado. Se ha creado un componente JavaBean, es decir, una clase a la que se ha llamado PuertoSerie.class que extiende de Container y dentro de la misma clase se ha generado una aplicación que controla el puerto serie.

```
public class PuertoSerie extends Container implements Serializable, ActionListener
```

Figura 5

Se ha utilizado la declaración de la clase según la figura 5.

Para ver el resto de la clase, véase Anexo 1.

Una vez generado todo el código se creará el archivo manifest.tmp, y en su interior se escribirán las siguientes líneas.

```
Name: PuertoSerie.class
Java-Bean: True
```

Después de crear el archivo se han de comprimir ambos en un archivo de extensión .jar, para ello bastará con que escribamos en la consola del cmd de Windows Xp la siguiente línea.

```
jar cfm PuertoSerie.jar manifest.tmp PuertoSerie.class
```

Esto habrá generado un archivo de nombre PuertoSerie.jar, y este es el componente JavaBean que se ha generado para controlar el puerto serie.

Por último, se cargará el componente en el BeanBox, para ello esta herramienta se ejecuta mediante un archivo por lotes llamado run.bat, cuando este se edita, se puede comprobar que redefine de nuevo la variable de entorno CLASSPATH durante su ejecución, por lo que para que el componente funcione correctamente se tendrá que añadir en este archivo la misma línea de comando que se añadió en el autoexec.nt para que Windows Xp reconociese el paquete comm.jar.

Después se añadirá nuestro componente al directorio jars, para que este se cargue desde el comienzo del arranque del programa.

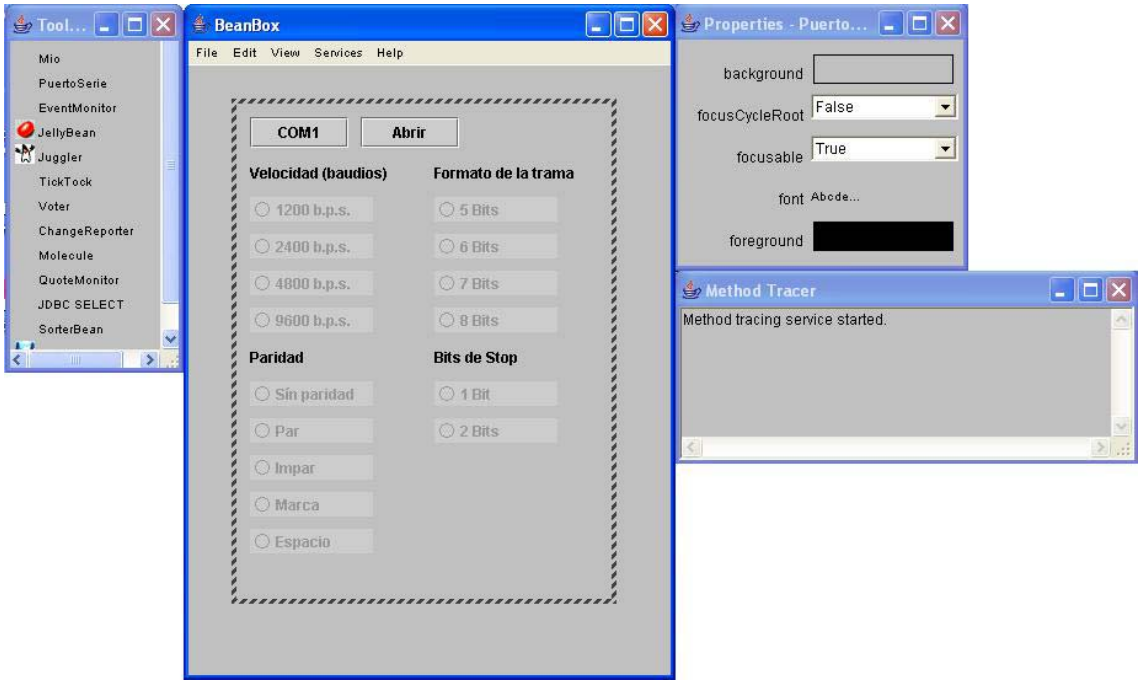


figura 6

Como se puede comprobar en la figura 6, aparece en el programa BeanBox una vez que ha cargado el componente JavaBean que se ha creado, pero a la derecha aparecen las propiedades y las características del Container y no del puerto serie. Tal y como se había especificado se ha creado un componente JavaBean de un Container y que comprende a su vez una aplicación que controla el puerto serie.

4.3 Componente .NET

Para la creación del componente .NET se ha requerido la utilización del programa Microsoft Visual Studio .NET.

Esta herramienta permite crear código en varios lenguajes de programación, ya sean J# (una aproximación de Java y compatible con este), C# (una aproximación de C++ y compatible con este), Visual Basic, ASP, ASP.NET y HTML. Además por sí solo permite crear clases, aplicaciones tanto en consola como para Microsoft Windows y bibliotecas de clases.

Esta herramienta requiere el uso del JDK (Java Development KIT) y JRE (Java Runtime Enviroment) al igual que la requiere JavaBeans, pero en este caso el programa contiene a ambas, eso sí creadas por Microsof en puesto de Sun.

Para la realización del componente se ha utilizado como apoyo una librería de clases denominada “SerialPort.dll”. Esta librería fue creada en el año 2003 por SAIC (Science Applications International Corporation) y su autor fue Timothy J. Krell.

Se han utilizado directamente las clases de esta librería y se han añadido a la nueva creada en este proyecto. También se podía haber utilizado esta librería directamente, pero a la hora de utilizar la creada en el proyecto y depender de esta inicialmente serían necesarias ambas, por lo que de esta forma se crea una librería independiente de cualquier otra y que se vale por sí sola para controlar el puerto RS-232.

El estudio de este componente se va a vasar en dos partes, la primera es la generación de la librería de clases en lenguaje C#, y la segunda es la generación de una aplicación en J# que utilice la librería de clases creada, esta segunda parte se realizará para crear una aplicación que demuestre el correcto funcionamiento de la librería .NET dentro del Microsoft Visual Studio .NET con dos lenguajes de programación distintos.

La nueva librería generada "PuertoSerie.dll", fue creada como un nuevo proyecto, una vez hecho esto Microsoft Visual Studio .NET ofrece la opción de escoger el lenguaje de programación deseado (se escogió C#) y después se ha de seleccionar el tipo de proyecto (librería de clases), esta librería se compone de una única clase denominada "RS232.cs", esta clase utiliza un objeto "SerialPort" y otro "WithEvents" que se encuentran en la librería "SerialPort.dll" utilizada de apoyo. Además para el correcto funcionamiento de todo se han tenido que añadir 12 clases más de la librería de apoyo.

La estructura de la librería creada, es una clase principal creada para el proyecto "RS232.cs", esta clase realiza llamadas a las clases "SerialPort.cs" y "WithEvents.cs".

La clase "SerialPort.cs" requiere un objeto "WithEvents" y realiza llamadas a las clases "SerialComm.cs" y "SerialConfig.cs", estas a su vez realizarían llamadas a todas las demás. Por lo que la estructura de clases quedaría representada según la figura 7.

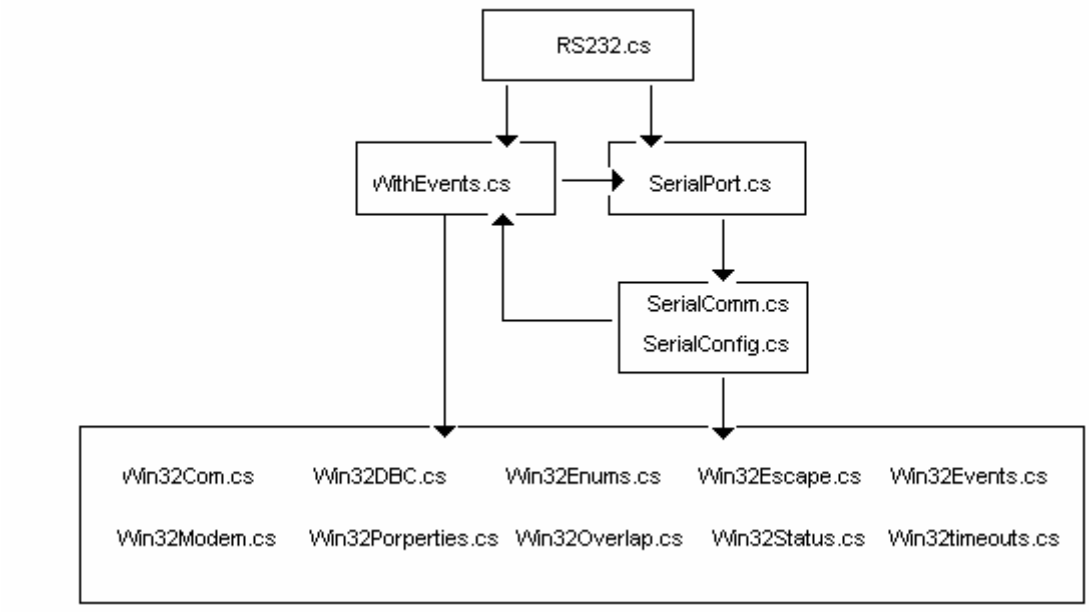


figura 7

Una vez creado el código y compilado sin errores la herramienta Microsoft Visual Studio .NET generará automáticamente la librería de clases con el nombre especificado "PuertoSerie.dll".

A partir de esta librería de clases se creará un nuevo proyecto, el tipo de lenguaje seleccionado será J#, y el tipo de proyecto será aplicación Windows.

Este proyecto constará de dos clases, una con el frame inicial ("Form.jsl") en el que se cargarán objetos necesarios para modificar las propiedades del puerto serie y con un hilo

lector (Thread) que cada 50 milisegundos comprobará automáticamente si se están recibiendo datos por el puerto serie, y otra clase frame ("Error.jsl") en la que se indicarán los errores según el tipo de excepción que se haya registrado.

Para la realización de las clases "Form.jsl" y "Error.jsl", previamente se ha registrado el componente .NET creado "PuertoSerie.dll" como una referencia de nuestro nuevo proyecto, de tal forma que nos permita utilizar las clases y los métodos indicados en este. Esto se realizó utilizando en el main menú de Microsoft Visual Studio .NET la opción Proyecto → Agregar referencia, aquí se pulsa el botón examinar y se busca la librería .dll generada anteriormente.

Después de generar esta aplicación se comprueba el correcto funcionamiento de la librería de clases "PuertoSerie.dll" sobre los puertos del tipo RS-232 en un equipo cualquiera. A raíz de aquí se puede observar de forma gráfica el comportamiento de los puertos gracias a la aplicación "COM.exe" generada para ello (es la que contiene las clases "Form.jsl" y "Error.jsl").

La aplicación generada queda representada en la figura 8, expuesta a continuación.

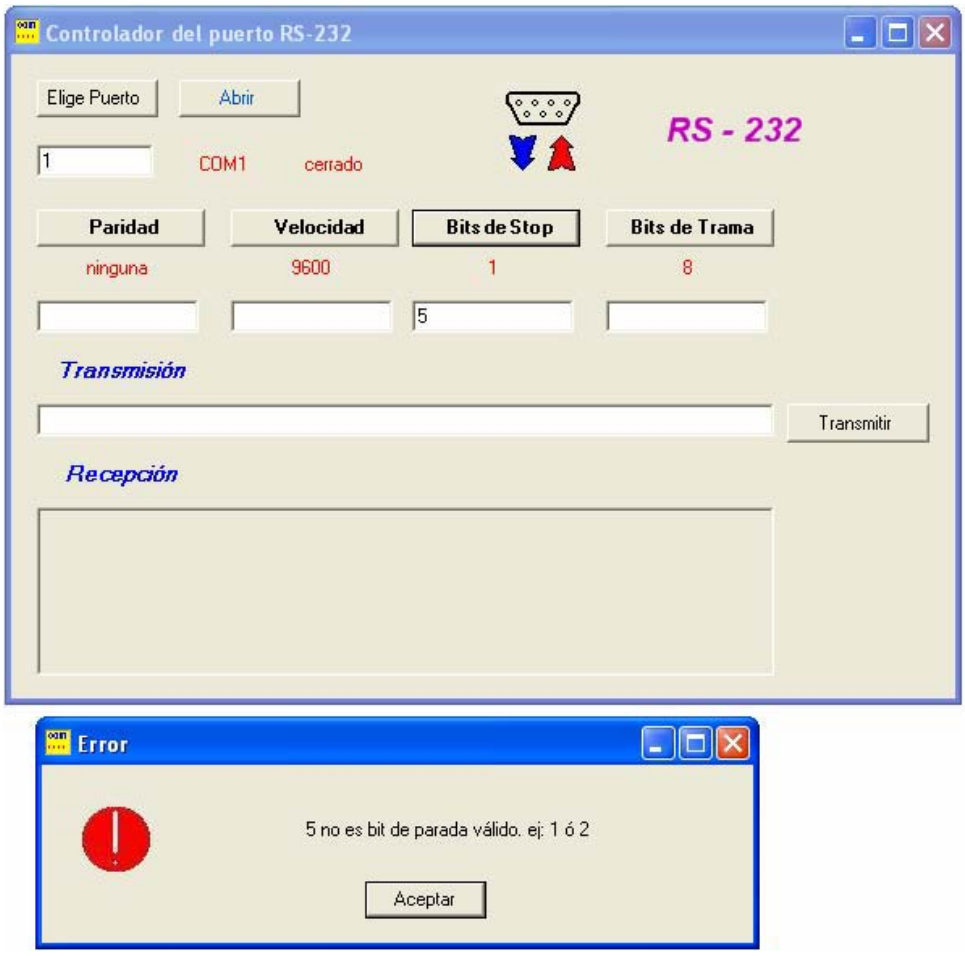


figura 8

Una última referencia a tener en cuenta es las opciones que se han activado en la librería PuertoSerie.dll son las necesarias para que los datos que se reciban sean caracteres ASCII.

CAPÍTULO V

Conclusión Final

Durante la realización de este proyecto se ha podido observar la evolución de los lenguajes de programación y los distintos tipos existentes más conocidos, a través de ellos se decidió generar dos componentes con los lenguajes más novedosos, más utilizados y los que ofrecían mejores prestaciones de cara a usuario y programador.

Con el componente JavaBean se ofrecen gran cantidad de prestaciones, como un lenguaje fácil y legible a la hora de usar por el programador, además de una interesante cualidad, la programación orientada a objetos. El componente JavaBean ofrece además al usuario la importante cualidad de ser portable por distintos sistemas operativos utilizando el mismo código compilado, ya que Java ofrece esta posibilidad variando sólo la JVM (Java Virtual Machina), compuesta por JDK (Java Development Kit) y JRE (Java Runtime Enviroment), siendo estas herramientas de libre distribución.

Se ha podido comprobar como el componente JavaBean es relativamente sencillo de generar pero debido a que es una clase abstracta para generarlo a hecho falta realizar una interfaz gráfica, sabiendo que existe una librería que ya ha sido utilizada para crear este proyecto, la generación de esa librería se aproximaría más a la consecución de los objetivos expuestos en este proyecto pero JavaBean no permite generar ese tipo de librería.

En el caso del componente .NET, según se ha podido comprobar es más complicado el uso de otras librerías. A diferencia de JavaBeans en Microsof Visual Studio .NET se tiene acceso a las clases principales cuando se agrega una referencia de una librería a un proyecto, esto hace que la generación de código y la legibilidad del código existente sean más complicadas de entender, pero por el contrario un componente .NET ofrece la posibilidad de añadir directamente las clases a la nueva librería que se está generando, mientras que JavaBeans no. De esta forma la librería generada en .NET es totalmente independiente de la librería utilizada para su creación, ya que la contiene en si mísma, y se basta por sí sola para realizar el servicio para el cual se hizo, mientras que JavaBeans sigue dependiendo de las librerías "*.dll" a partir de las cuales se creó. Por último, indicar que

Microsoft Visual Studio .NET ofrece una interfaz gráfica lo que hace más atractivo y fácil la utilización de las librerías recién creadas.

Además .NET ofrece la posibilidad de interactuar entre varios lenguajes operativos y es una de las herramientas más usadas lo que hace más atractivo su funcionamiento. De esta forma las ventajas de uno y otro quedan expuestas en la siguiente tabla.

	JavaBean	**.NET**
Portabilidad	*	
Fácil uso y legibilidad de código	*	
Independencia de las librerías de apoyo		*
Mayor control sobre las librerías de apoyo		*
Programación orientada a objetos	*	*
Lenguaje novedoso de última generación	*	*
Facilidad de uso de las nuevas librerías		*

En cuanto al problema planteado al principio del proyecto sobre la escasez de código referido al hardware, según el estudio realizado se llega a la conclusión de que a la hora de generar librerías totalmente independientes de una librería inicial, que es lo que verdaderamente le puede interesar a un programado, la realización de las mismas no es tan sencilla como se podía comprobar aparentemente con el componente JavaBean. También es de mencionar que a diferencia de los componentes no físicos de un computador, en los cuales los lenguajes de programación se parecen bastante, en los componentes físicos no existe ningún tipo de estándar, lo que implica que exista menor variedad de código y que estos códigos disten mucho unos de otros.

Anexo A : Componente PuertoSerie (JavaBean)

```
import java.awt.*;
import java.awt.event.*;
import javax.comm.*;
import java.util.*;
import javax.swing.*;
import java.io.Serializable;

public class PuertoSerie extends Container implements Serializable, ActionListener {

// ---------------------------------------------------------------------------------------------------------------
// Variables de instancia (para la identificación de los puertos del ordenador)
// ---------------------------------------------------------------------------------------------------------------

        private Enumeration ports;
        private Vector puertosSerie;
        private CommPortIdentifier port;
        private int n_puerto=0;
        private int timeout=2000;
        private SerialPort s=null;

        // variables para el Container

        private JButton eligePuerto;
        private JButton abreCierraPuerto;

        private JRadioButton v1200, v2400, v4800, v9600;
        private ButtonGroup velocidad;
        private JLabel baudio;

        private JRadioButton sp, par, impar, marca, espacio;
        private ButtonGroup paridad;
        private JLabel parity;

        private JRadioButton d5, d6, d7, d8;
        private ButtonGroup formato;
        private JLabel trama;

        private JRadioButton b1, b2;
        private ButtonGroup bitsstop;
        private JLabel bstop;

// ---------------------------------------------------------------------------------------------------------------
// Constructor, se generarán los identificadores para separar los puertos y todo lo relativo al frame
// ---------------------------------------------------------------------------------------------------------------

        public PuertoSerie ( ) {

                //      Se guardan los puertos serie en un vector de nombre puertosSerie y los guarda
                //      por la variable port del tipo CommPortIdentifier

                puertosSerie = new Vector ( );
                ports = CommPortIdentifier.getPortIdentifiers ( );

                while ( ports.hasMoreElements ( ) ) {
                    port = ( CommPortIdentifier ) ports.nextElement ( );
                    switch ( port.getPortType ( ) ) {
                            case CommPortIdentifier.PORT_PARALLEL:  break;
                            case CommPortIdentifier.PORT_SERIAL: puertosSerie.add(port); break;
                            default: break;   // Caso por defecto, no debería ocurrir nunca
                    } //Fin del switch
                } //Fin del while
```

```
//          Caso en el que no existan puertos Serie en este equipo

if ( puertosSerie.size ( ) == 0 ) {
    System.out.println("No se han reconocido Puertos Serie en este equipo");
}
else  {
    setLayout(null);

    //        Elige Puerto y Abre y Cierra

    eligePuerto = new JButton(((CommPortIdentifier)puertosSerie.get(0)).getName());
    eligePuerto.setBounds(10,10,80,25);
    eligePuerto.addActionListener(this);
    add(eligePuerto);

    abreCierraPuerto = new JButton("Abrir");
    abreCierraPuerto.setBounds(100,10,80,25);
    abreCierraPuerto.addActionListener(this);
    add(abreCierraPuerto);

    //        Elige velocidad

    baudio = new JLabel("Velocidad (baudios)");
    baudio.setBounds(10,45,150,20);
    add(baudio);
    v1200 = new JRadioButton("1200 b.p.s.");
    v1200.setBounds(10,75,100,20);
    v1200.addActionListener(this);
    v1200.setEnabled(false);
    v1200.setForeground(Color.gray);
    add(v1200);
    v2400 = new JRadioButton("2400 b.p.s.");
    v2400.setBounds(10,105,100,20);
    v2400.addActionListener(this);
    add(v2400);
    v2400.setEnabled(false);
    v2400.setForeground(Color.gray);
    v4800 = new JRadioButton("4800 b.p.s.");
    v4800.setBounds(10,135,100,20);
    v4800.addActionListener(this);
    add(v4800);
    v4800.setEnabled(false);
    v4800.setForeground(Color.gray);
    v9600 = new JRadioButton("9600 b.p.s.");
    v9600.setBounds(10,165,100,20);
    v9600.addActionListener(this);
    add(v9600);
    v9600.setEnabled(false);
    v9600.setForeground(Color.gray);
    velocidad = new ButtonGroup();
    velocidad.add(v1200);
    velocidad.add(v2400);
    velocidad.add(v4800);
    velocidad.add(v9600);

    //        Elige Paridad

    parity = new JLabel("Paridad");
    parity.setBounds(10,195,150,20);
    add(parity);
    sp = new JRadioButton("Sín paridad");
    sp.setBounds(10,225,100,20);
```

```
sp.addActionListener(this);
sp.setEnabled(false);
sp.setForeground(Color.gray);
add(sp);
par = new JRadioButton("Par");
par.setBounds(10,255,100,20);
par.addActionListener(this);
par.setEnabled(false);
par.setForeground(Color.gray);
add(par);
impar = new JRadioButton("Impar");
impar.setBounds(10,285,100,20);
impar.addActionListener(this);
impar.setEnabled(false);
impar.setForeground(Color.gray);
add(impar);
marca = new JRadioButton("Marca");
marca.setBounds(10,315,100,20);
marca.addActionListener(this);
marca.setEnabled(false);
marca.setForeground(Color.gray);
add(marca);
espacio = new JRadioButton("Espacio");
espacio.setBounds(10,345,100,20);
espacio.addActionListener(this);
espacio.setEnabled(false);
espacio.setForeground(Color.gray);
add(espacio);
paridad = new ButtonGroup();
paridad.add(sp);
paridad.add(par);
paridad.add(impar);
paridad.add(marca);
paridad.add(espacio);

//        Elige Formato de la trama

trama = new JLabel("Formato de la trama");
trama.setBounds(160,45,200,20);
add(trama);
d5 = new JRadioButton("5 Bits");
d5.setBounds(160,75,100,20);
d5.addActionListener(this);
d5.setEnabled(false);
d5.setForeground(Color.gray);
add(d5);
d6 = new JRadioButton("6 Bits");
d6.setBounds(160,105,100,20);
d6.addActionListener(this);
add(d6);
d6.setEnabled(false);
d6.setForeground(Color.gray);
d7 = new JRadioButton("7 Bits");
d7.setBounds(160,135,100,20);
d7.addActionListener(this);
add(d7);
d7.setEnabled(false);
d7.setForeground(Color.gray);
d8 = new JRadioButton("8 Bits");
d8.setBounds(160,165,100,20);
d8.addActionListener(this);
add(d8);
d8.setEnabled(false);
d8.setForeground(Color.gray);
```

```
                formato = new ButtonGroup();
                formato.add(d5);
                formato.add(d6);
                formato.add(d7);
                formato.add(d8);

                //      Elige Bits de parada

                bstop = new JLabel("Bits de Stop");
                bstop.setBounds(160,195,200,20);
                add(bstop);
                b1 = new JRadioButton("1 Bit");
                b1.setBounds(160,225,100,20);
                b1.addActionListener(this);
                b1.setEnabled(false);
                b1.setForeground(Color.gray);
                add(b1);
                b2 = new JRadioButton("2 Bits");
                b2.setBounds(160,255,100,20);
                b2.addActionListener(this);
                add(b2);
                b2.setEnabled(false);
                b2.setForeground(Color.gray);
                bitsstop = new ButtonGroup();
                bitsstop.add(b1);
                bitsstop.add(b2);

                setVisible(true);

        }  //Fin del else

} //Fin del constructor

// ---------------------------------------------------------------------------------------------------------------
// Escucha de los botones
// ---------------------------------------------------------------------------------------------------------------
public void actionPerformed (ActionEvent ae) {

  // Elige el puerto
            if (ae.getSource() == eligePuerto)  {

            // Aquí elige el siguiente puerto del vector y si no hay más vuelve al primero

                    if (n_puerto == puertosSerie.size ( ) - 1)    n_puerto=0;
                    else    n_puerto++;

            // Actualiza las opciones del Frame

                    if (s == null) {      //Si el puerto está abierto lo cierra
                            System.out.println ("El puerto "+ eligePuerto.getText( ) + " no abierto");
                    }
                    else {
                            cerrarPuerto();
                            abreCierraPuerto.setText("Abrir");
                    }
                    eligePuerto.setText(((CommPortIdentifier)puertosSerie.get(n_puerto)).getName());

            }
```

```
            //          Abre o cierra el puerto, dependiendo del estado en que se encuentre, y cambia
            //          la etiqueta del Botón

            if (ae.getSource() == abreCierraPuerto) {
                    if ("Abrir" == abreCierraPuerto.getText ( ) )  {
                            abrirPuerto();
                            abreCierraPuerto.setText("Cerrar");
                    }
                    else {
                            cerrarPuerto();
                            abreCierraPuerto.setText("Abrir");
                    }
            }

            //  Establece la velocidad indicada

            if (ae.getSource() == v1200)        establecerVelocidad(1200);
            if (ae.getSource() == v2400)        establecerVelocidad(2400);
            if (ae.getSource() == v4800         establecerVelocidad(4800);
            if (ae.getSource() == v9600)        establecerVelocidad(9600);

            //  Establece la Paridad Adecuada

            if (ae.getSource() == sp)           establecerBitsDeParidad(1);
            if (ae.getSource() == impar)        establecerBitsDeParidad(2);
            if (ae.getSource() == par)          establecerBitsDeParidad(3);
            if (ae.getSource() == marca)        establecerBitsDeParidad(4);
            if (ae.getSource() == espacio       establecerBitsDeParidad(5);

            // Establece el Formato de trama

            if (ae.getSource() == d5)           establecerBitsDeFormato(5);
            if (ae.getSource() == d6)           establecerBitsDeFormato(6);
            if (ae.getSource() == d7)           establecerBitsDeFormato(7);
            if (ae.getSource() == d8)           establecerBitsDeFormato(8);

            // Establece los Bits de Stop

            if (ae.getSource() == b1)           establecerBitsDeStop(1);
            if (ae.getSource() == b2)           establecerBitsDeStop(2);

    }   //Fin del ActionPerformed

// ---------------------------------------------------------------------------------------------------------------
// Abre un puerto según su identificador y lo asigna a un puerto serie
// ---------------------------------------------------------------------------------------------------------------
public void abrirPuerto ( )  {
    try {
            s=(SerialPort)((CommPortIdentifier)puertosSerie.get(n_puerto)).open(((CommPortIdentifier)
              puertosSerie.get(n_puerto)).getName(),timeout);

            // Activación de botones de velocidad

              v1200.setEnabled(true);
              v1200.setForeground(Color.black);
              v2400.setEnabled(true);
              v2400.setForeground(Color.black);
              v4800.setEnabled(true);
              v4800.setForeground(Color.black);
              v9600.setEnabled(true);
              v9600.setForeground(Color.black);
```

```
            // Activación de botones de paridad

               sp.setEnabled(true);
               sp.setForeground(Color.black);
               par.setEnabled(true);
               par.setForeground(Color.black);
               impar.setEnabled(true);
               impar.setForeground(Color.black);
               marca.setEnabled(true);
               marca.setForeground(Color.black);
               espacio.setEnabled(true);
               espacio.setForeground(Color.black);

            // Activación de botones de formato

               d5.setEnabled(true);
               d5.setForeground(Color.black);
               d6.setEnabled(true);
               d6.setForeground(Color.black);
               d7.setEnabled(true);
               d7.setForeground(Color.black);
               d8.setEnabled(true);
               d8.setForeground(Color.black);

            // Activación de botones de Bits de Stop

               b1.setEnabled(true);
               b1.setForeground(Color.black);
               b2.setEnabled(true);
               b2.setForeground(Color.black);
   }
   catch(PortInUseException piue)  {
            System.out.println("Excepción: " + piue.getMessage());
   }
}

// ------------------------------------------------------------------------------------------------------------------
//  Cierra un puerto ya abierto y asignado
// ------------------------------------------------------------------------------------------------------------------
public void cerrarPuerto ( )  {

   //  Desactivación de botones de velocidad
   v1200.setEnabled(false);
   v1200.setForeground(Color.gray);
   v2400.setEnabled(false);
   v2400.setForeground(Color.gray);
   v4800.setEnabled(false);
   v4800.setForeground(Color.gray);
   v9600.setEnabled(false);
   v9600.setForeground(Color.gray);

   // Desactivación de botones de paridad
   sp.setEnabled(false);
   sp.setForeground(Color.gray);
   par.setEnabled(false);
   par.setForeground(Color.gray);
   impar.setEnabled(false);
   impar.setForeground(Color.gray);
   marca.setEnabled(false);
   marca.setForeground(Color.gray);
   espacio.setEnabled(false);
   espacio.setForeground(Color.gray);
```

```
    // Desactivación de botones de formato
    d5.setEnabled(false);
    d5.setForeground(Color.gray);
    d6.setEnabled(false);
    d6.setForeground(Color.gray);
    d7.setEnabled(false);
    d7.setForeground(Color.gray);
    d8.setEnabled(false);
    d8.setForeground(Color.gray);

    // Desactivación de botones de Bits de Stop

    b1.setEnabled(false);
    b1.setForeground(Color.gray);
    b2.setEnabled(false);
    b2.setForeground(Color.gray);

    s.close();
    s=null;
  }

// ------------------------------------------------------------------------------------------------------------------
//  Otros métodos
// ------------------------------------------------------------------------------------------------------------------

    public int obtenerVelocidad ( )         { return s.getBaudRate(); }

    public int obtenerBitsDeParidad ( )   { return s.getParity(); }

    public int obtenerBitsDeFormato ( )  {return s.getDataBits();}

    public int obtenerBitsDeStop ( )        { return s.getStopBits(); }

    public void establecerVelocidad ( int velocidad )  {
            try {
                    s.setSerialPortParams (      velocidad,
                                                 obtenerBitsDeFormato ( ),
                                                 obtenerBitsDeStop ( ),
                                                 obtenerBitsDeParidad ( ));

            }  catch ( UnsupportedCommOperationException ucoe ) {
                    System.out.println ("Excepción: " + ucoe.getMessage());
            }
    }

    public void establecerBitsDeParidad ( int bitsDeParidad )  {
            try  {
                    s.setSerialPortParams (      obtenerVelocidad ( ),
                                                 obtenerBitsDeFormato ( ),
                                                 obtenerBitsDeStop ( ),
                                                 bitsDeParidad );

            } catch ( UnsupportedCommOperationException ucoe ) {
                    System.out.println  ("Excepción: " + ucoe.getMessage());
            }
    }
```

```
        public void establecerBitsDeFormato ( int bitsDeFormato )  {
                try {
                        s.setSerialPortParams  (      obtenerVelocidad ( ),
                                                      bitsDeFormato,
                                                      obtenerBitsDeStop ( ),
                                                      obtenerBitsDeParidad ( ));

                } catch ( UnsupportedCommOperationException ucoe ) {
                        System.out.println  ("Excepción: " + ucoe.getMessage());
                }
        }


        public void establecerBitsDeStop ( int bitsDeStop ) {
                try  {
                        s.setSerialPortParams  (      obtenerVelocidad ( ),
                                                      obtenerBitsDeFormato ( ),
                                                      bitsDeStop,
                                                      obtenerBitsDeParidad ( ));

                } catch ( UnsupportedCommOperationException ucoe ) {
                        System.out.println ( "Excepción: " + ucoe.getMessage ( ) );
                }
        }

} // class PuertoSerie
```

Anexo B: Componente PuertoSerie (.NET)

```
using System;
using System.IO;
namespace PuertoSerie
{
	/// <summary>
	/// </summary>

	public class RS232
	{
		// Declaración de variables necesarias
		#region variables
			private int puerto;
			private SerialPort ps;
			private WithEvents ve;
		#endregion

		#region Constructor
			public RS232( ) {
				// ve es una variable requerida por la librería usada a la cual
				// en esta nueva librería no se le va a dar uso.
				ve = new WithEvents();
				ps = new SerialPort(ve);
				this.configuracionInicial();
			}
		#endregion


		#region Métodos
			// Devuelve la cantidad de puertos RS-232 en el equipo
			public int numeroDePuertos ( ) {
				String s;
				int puerto=0;
				for ( int i=1; i<=32; i++ ) {
					s="COM"+i.ToString ( );
					if ( ps.Available(s)==true)  puerto++;
					else i=33;
				}
				return puerto;
			}

			// Devuelve verdadero si l puerto está abierto y falso en caso contrario
			public bool estaAbierto ( )  {  return ps.IsOpen; }

			// Abre el puerto que se le indica
			public void abrir ( ) { ps.Open(this.puerto); }

			// Cierra el puerto el ultimo puerto abierto
			public void cerrar ( ) { ps.Close(); }

			// Selecciona un puerto para trabajar
			public void eligePuerto ( int i ) { puerto = i; }

			// Indica el puerto con el que se está trabajando
			public int puertoSeleccionado ( ) { return puerto; }
```

```
// Elige la velocidad en baudios puede ser 110, 300, 600, 1200, 9600,
// 14400, 19200, 38400, 56000, 57600, 115200, 128000 ó 256000
public void eligeVelocidad ( int velocidad ) {
    switch (velocidad)  {
            case 110:       ps.Cnfg.BaudRate = LineSpeed.Baud_110;
                            break;
            case 300:       ps.Cnfg.BaudRate = LineSpeed.Baud_300;
                            break;
            case 600:       ps.Cnfg.BaudRate = LineSpeed.Baud_600;
                            break;
            case 1200:      ps.Cnfg.BaudRate = LineSpeed.Baud_1200;
                            break;
            case 2400:      ps.Cnfg.BaudRate = LineSpeed.Baud_2400;
                            break;
            case 4800:      ps.Cnfg.BaudRate = LineSpeed.Baud_4800;
                            break;
            case 9600:      ps.Cnfg.BaudRate = LineSpeed.Baud_9600;
                            break;
            case 14400:     ps.Cnfg.BaudRate = LineSpeed.Baud_14400;
                            break;
            case 19200:     ps.Cnfg.BaudRate = LineSpeed.Baud_19200;
                            break;
            case 38400:     ps.Cnfg.BaudRate = LineSpeed.Baud_38400;
                            break;
            case 56000:     ps.Cnfg.BaudRate = LineSpeed.Baud_56000;
                            break;
            case 57600:     ps.Cnfg.BaudRate = LineSpeed.Baud_57600;
                            break;
            case 115200:    ps.Cnfg.BaudRate = LineSpeed.Baud_115200;
                            break;
            case 128000:    ps.Cnfg.BaudRate = LineSpeed.Baud_128000;
                            break;
            case 256000:    ps.Cnfg.BaudRate = LineSpeed.Baud_256000;
                            break;
            default:        System.Console.WriteLine ("El Fomarto de velocidad
                            introducido no es el correcto. Ha de ser en baudios uno
                            de los siguientes: 110, 300");
                            break;
    }
}

// Elige el tipo de paridad, los valores posibles a introducir son:
// ninguna, impar, par, marca o espacio

public void eligeParidad(String paridad) {
    switch ( paridad ) {
            case "ninguna":     ps.Cnfg.Parity = Parity.None;
                                break;
            case "impar":       ps.Cnfg.Parity = Parity.Odd;
                                break;
            case "par":         ps.Cnfg.Parity = Parity.Even;
                                break;
            case "marca":       ps.Cnfg.Parity = Parity.Mark;
                                break;
            case "espacio":     ps.Cnfg.Parity = Parity.Space;
                                break;
            default:            System.Console.WriteLine("La paridad establecidad no
                                es correcta, las opciones son: ninguna, impar, par,
                                marca o espacio");
                                break;
    }
}
```

```
                    // Elige el tamaño en bits de la trama, los posibles valores son: 5, 6, 7 u 8
                    public void eligeTamañoDeLaTrama ( int tamaño ) {
                        switch ( tamaño ) {
                            case 5:   ps.Cnfg.DataBits = ByteSize.Five;
                                      break;
                            case 6:   ps.Cnfg.DataBits = ByteSize.Six;
                                      break;
                            case 7:   ps.Cnfg.DataBits = ByteSize.Seven;
                                      break;
                            case 8:   ps.Cnfg.DataBits = ByteSize.Eight;
                                      break;
                            default:  System.Console.WriteLine("El tamaño del formato de trama
                                      especificado no es válido, los valores correctos son: 5, 6, 7 u 8");
                                      break;
                        }
                    }

                    // Elige el numero de Bits de Stop, 1 ó 2
                    public void eligeBitsDeParada ( int parada ) {
                        switch ( parade ) {
                            case 1:   ps.Cnfg.StopBits = StopBits.One;
                                      break;
                            case 2:   ps.Cnfg.StopBits = StopBits.Two;
                                      break;
                            default:  System.Console.WriteLine("Los valores para los Bits de parada
                                      son pueden ser 1ó 2");
                                      break;
                        }
                    }

                    // Configuración estándar de salida, puerto: COM1, velocidad: 9600 baudios,
                    // paridad: ninguna, tamaño de la trama: 8 bits, Bits de parada: 1
                    public void configuracionInicial ( ) {
                            ps.Cnfg.Initialize(1);
                            this.eligePuerto(1);
                            if( this.estaAbierto() == true)
                                this.cerrar();
                            this.eligeVelocidad(9600);
                            this.eligeParidad("ninguna");
                            this.eligeTamañoDeLaTrama(8);
                            this.eligeBitsDeParada(1);
                    }

                    public void enviar ( String s )  {
                            ps.Send(s);
                    }

                    public String recibir ( ) {
                            String s;
                            uint nBytes;
                            byte[] b;
                            nBytes = ps.Recv(out b);
                            if(nBytes>0) {     s = System.Text.Encoding.ASCII.GetString(b);
                                               return s;
                            } else {           s = "";
                                               return s;
                            }
                    }

                    public bool activo ( String s ) { return ps.Available(s);  }

                #endregion
            }
        }
```

Anexo C1: Aplicación que ejecuta componente .NET

```
package COM;

import System.Drawing.*;
import System.Collections.*;
import System.ComponentModel.*;
import System.Windows.Forms.*;
import System.Data.*;
import PuertoSerie.*;
import java.lang.*;

/**
 * Descripción de resumen para Form1.
 */

public class Form1 extends System.Windows.Forms.Form implements Runnable
{
        /**
         * Variable del diseñador necesaria.
         */

        private System.ComponentModel.Container components = null;
        private System.Windows.Forms.TextBox puerto;
        private System.Windows.Forms.Button eligepuerto;
        private System.Windows.Forms.Button abrir;
        private System.Windows.Forms.Label paridad;
        private System.Windows.Forms.Label velocidad;
        private System.Windows.Forms.Label bs;
        private System.Windows.Forms.Label trama;
        private System.Windows.Forms.Label port;
        private System.Windows.Forms.Label estado;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.TextBox textBox2;
        private System.Windows.Forms.TextBox textBox3;
        private System.Windows.Forms.TextBox textBox4;
        private System.Windows.Forms.Button par;
        private System.Windows.Forms.Button vlc;
        private System.Windows.Forms.Button btstp;
        private System.Windows.Forms.Button btrama;
        private System.Windows.Forms.PictureBox pictureBox1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox textBox5;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Button tx;
        private System.Windows.Forms.TextBox textBox6;

        // Variables

        private RS232 ps;
        private Thread hiloLector;
```

```
public Form1()
{
	//
	// Necesario para mantener compatibilidad con el Diseñador de Windows Forms
	//

	InitializeComponent();

	ps = new RS232();
	hiloLector = new Thread(this);

	// Si no hay puertos finalizar la aplicación

	if ( ps.numeroDePuertos ( ) == 0 )	{
		Error nuevo;
		nuevo = new Error("No existen puertos RS-232 en este equipo");
		this.Dispose();
		this.Close();
	}
}


/**
 * Limpiar los recursos que se estén utilizando.
 */

protected void Dispose(boolean disposing)  {
	if (disposing) {
		if (components != null) {
			components.Dispose();
		}
	}
	super.Dispose(disposing);
}

#region Código generado por el Diseñador de Windows Forms

/**
 * Método necesario para mantener compatibilidad con el Diseñador; no modifique
 * el contenido del método con el editor de código.
 */
private void InitializeComponent ( ) {
	System.Resources.ResourceManager resources =
		new System.Resources.ResourceManager ( Form1.class.ToType ( ) );
	this.eligepuerto = new System.Windows.Forms.Button();
	this.puerto = new System.Windows.Forms.TextBox();
	this.abrir = new System.Windows.Forms.Button();
	this.paridad = new System.Windows.Forms.Label();
	this.velocidad = new System.Windows.Forms.Label();
	this.bs = new System.Windows.Forms.Label();
	this.trama = new System.Windows.Forms.Label();
	this.port = new System.Windows.Forms.Label();
	this.estado = new System.Windows.Forms.Label();
	this.textBox1 = new System.Windows.Forms.TextBox();
	this.textBox2 = new System.Windows.Forms.TextBox();
	this.textBox3 = new System.Windows.Forms.TextBox();
	this.textBox4 = new System.Windows.Forms.TextBox();
	this.par = new System.Windows.Forms.Button();
	this.vlc = new System.Windows.Forms.Button();
```

```
this.btstp = new System.Windows.Forms.Button();
this.btrama = new System.Windows.Forms.Button();
this.pictureBox1 = new System.Windows.Forms.PictureBox();
this.label1 = new System.Windows.Forms.Label();
this.textBox5 = new System.Windows.Forms.TextBox();
this.label2 = new System.Windows.Forms.Label();
this.label3 = new System.Windows.Forms.Label();
this.tx = new System.Windows.Forms.Button();
this.textBox6 = new System.Windows.Forms.TextBox();
this.SuspendLayout();
//
// eligepuerto
//
this.eligepuerto.set_Location(new System.Drawing.Point(16, 16));
this.eligepuerto.set_Name("eligepuerto");
this.eligepuerto.set_TabIndex(0);
this.eligepuerto.set_Text("Elige Puerto");
this.eligepuerto.add_Click( new System.EventHandler(this.button1_Click) );
//
// puerto
//
this.puerto.set_Location(new System.Drawing.Point(16, 56));
this.puerto.set_Name("puerto");
this.puerto.set_Size(new System.Drawing.Size(72, 20));
this.puerto.set_TabIndex(1);
this.puerto.set_Text("1");
//
// abrir
//
this.abrir.set_ForeColor(System.Drawing.SystemColors.get_ActiveCaption());
this.abrir.set_Location(new System.Drawing.Point(104, 16));
this.abrir.set_Name("abrir");
this.abrir.set_TabIndex(2);
this.abrir.set_Text("Abrir");
this.abrir.add_Click( new System.EventHandler(this.abrir_Click) );
//
// paridad
//
this.paridad.set_ForeColor(System.Drawing.Color.get_Red());
this.paridad.set_Location(new System.Drawing.Point(16, 120));
this.paridad.set_Name("paridad");
this.paridad.set_TabIndex(7);
this.paridad.set_Text("ninguna");
this.paridad.set_TextAlign(System.Drawing.ContentAlignment.MiddleCenter);
//
// velocidad
//
this.velocidad.set_ForeColor(System.Drawing.Color.get_Red());
this.velocidad.set_Location(new System.Drawing.Point(136, 120));
this.velocidad.set_Name("velocidad");
this.velocidad.set_TabIndex(8);
this.velocidad.set_Text("9600");
this.velocidad.set_TextAlign(System.Drawing.ContentAlignment.MiddleCenter);
```

```
//
// bs
//
this.bs.set_ForeColor(System.Drawing.Color.get_Red());
this.bs.set_Location(new System.Drawing.Point(248, 120));
this.bs.set_Name("bs");
this.bs.set_TabIndex(9);
this.bs.set_Text("1");
this.bs.set_TextAlign(System.Drawing.ContentAlignment.MiddleCenter);
//
// trama
//
this.trama.set_ForeColor(System.Drawing.Color.get_Red());
this.trama.set_Location(new System.Drawing.Point(368, 120));
this.trama.set_Name("trama");
this.trama.set_TabIndex(10);
this.trama.set_Text("8");
this.trama.set_TextAlign(System.Drawing.ContentAlignment.MiddleCenter);
//
// port
//
this.port.set_ForeColor(System.Drawing.Color.get_Red());
this.port.set_Location(new System.Drawing.Point(104, 56));
this.port.set_Name("port");
this.port.set_Size(new System.Drawing.Size(56, 23));
this.port.set_TabIndex(11);
this.port.set_Text("COM1");
this.port.set_TextAlign(System.Drawing.ContentAlignment.MiddleCenter);
//
// estado
//
this.estado.set_ForeColor(System.Drawing.Color.get_Red());
this.estado.set_Location(new System.Drawing.Point(168, 56));
this.estado.set_Name("estado");
this.estado.set_Size(new System.Drawing.Size(64, 23));
this.estado.set_TabIndex(12);
this.estado.set_Text("cerrado");
this.estado.set_TextAlign(System.Drawing.ContentAlignment.MiddleCenter);
//
// textBox1
//
this.textBox1.set_Location(new System.Drawing.Point(16, 152));
this.textBox1.set_Name("textBox1");
this.textBox1.set_TabIndex(13);
this.textBox1.set_Text("");
//
// textBox2
//
this.textBox2.set_Location(new System.Drawing.Point(136, 152));
this.textBox2.set_Name("textBox2");
this.textBox2.set_TabIndex(14);
this.textBox2.set_Text("");
```

```
//
// textBox3
//
this.textBox3.set_Location(new System.Drawing.Point(248, 152));
this.textBox3.set_Name("textBox3");
this.textBox3.set_TabIndex(15);
this.textBox3.set_Text("");
//
// textBox4
//
this.textBox4.set_Location(new System.Drawing.Point(368, 152));
this.textBox4.set_Name("textBox4");
this.textBox4.set_TabIndex(16);
this.textBox4.set_Text("");
//
// par
//
this.par.set_Font ( new System.Drawing.Font ( "Microsoft Sans Serif", 8.25F,
                        System.Drawing.FontStyle.Bold,
                        System.Drawing.GraphicsUnit.Point,
                        ((ubyte)(System.Byte)(((ubyte)0)))));
this.par.set_Location(new System.Drawing.Point(16, 96));
this.par.set_Name("par");
this.par.set_Size(new System.Drawing.Size(104, 23));
this.par.set_TabIndex(17);
this.par.set_Text("Paridad");
this.par.add_Click( new System.EventHandler(this.par_Click) );
//
// vlc
//
this.vlc.set_Font ( new System.Drawing.Font ( "Microsoft Sans Serif", 8.25F,
                        System.Drawing.FontStyle.Bold,
                        System.Drawing.GraphicsUnit.Point,
                        ((ubyte)(System.Byte)(((ubyte)0)))));
this.vlc.set_Location(new System.Drawing.Point(136, 96));
this.vlc.set_Name("vlc");
this.vlc.set_Size(new System.Drawing.Size(104, 23));
this.vlc.set_TabIndex(18);
this.vlc.set_Text("Velocidad");
this.vlc.add_Click( new System.EventHandler(this.vlc_Click) );
//
// btstp
//
this.btstp.set_Font ( new System.Drawing.Font ( "Microsoft Sans Serif", 8.25F,
                        System.Drawing.FontStyle.Bold,
                        System.Drawing.GraphicsUnit.Point,
                        ((ubyte)(System.Byte)(((ubyte)0)))));
this.btstp.set_Location(new System.Drawing.Point(248, 96));
this.btstp.set_Name("btstp");
this.btstp.set_Size(new System.Drawing.Size(104, 23));
this.btstp.set_TabIndex(19);
this.btstp.set_Text("Bits de Stop");
this.btstp.add_Click( new System.EventHandler(this.btstp_Click) );
```

```
//
// btrama
//
this.btrama.set_Font ( new System.Drawing.Font ( "Microsoft Sans Serif", 8.25F,
                        System.Drawing.FontStyle.Bold,
                        System.Drawing.GraphicsUnit.Point,
                        ((ubyte)(System.Byte)(((ubyte)0)))));
this.btrama.set_Location(new System.Drawing.Point(368, 96));
this.btrama.set_Name("btrama");
this.btrama.set_Size(new System.Drawing.Size(104, 23));
this.btrama.set_TabIndex(20);
this.btrama.set_Text("Bits de Trama");
this.btrama.add_Click( new System.EventHandler(this.btrama_Click) );
//
// pictureBox1
//
this.pictureBox1.set_Image ( ( ( System.Drawing.Image)
                              ( resources.GetObject("pictureBox1.Image"))));
this.pictureBox1.set_Location(new System.Drawing.Point(288, 8));
this.pictureBox1.set_Name("pictureBox1");
this.pictureBox1.set_Size(new System.Drawing.Size(80, 72));
this.pictureBox1.set_TabIndex(21);
this.pictureBox1.set_TabStop(false);
//
// label1
//
this.label1.set_Font ( new System.Drawing.Font ( "Arial", 15.75F,
                        ((System.Drawing.FontStyle)
                        ((System.Drawing.FontStyle.Bold | System.Drawing.FontStyle.Italic))),
                        System.Drawing.GraphicsUnit.Point,
                        ((ubyte)(System.Byte)(((ubyte)0)))));
this.label1.set_ForeColor (
                        System.Drawing.Color.FromArgb(((ubyte)(System.Byte)(((ubyte)192))),
                        ((ubyte)(System.Byte)(((ubyte)0))),
                       ((ubyte)(System.Byte)(((ubyte)192)))));
this.label1.set_Location(new System.Drawing.Point(376, 32));
this.label1.set_Name("label1");
this.label1.set_Size(new System.Drawing.Size(144, 32));
this.label1.set_TabIndex(22);
this.label1.set_Text("RS - 232");
this.label1.set_TextAlign(System.Drawing.ContentAlignment.MiddleCenter);
//
// textBox5
//
this.textBox5.set_Location(new System.Drawing.Point(16, 216));
this.textBox5.set_Name("textBox5");
this.textBox5.set_Size(new System.Drawing.Size(456, 20));
this.textBox5.set_TabIndex(23);
this.textBox5.set_Text("");
//
// label2
//
this.label2.set_Font ( new System.Drawing.Font("Microsoft Sans Serif", 9.75F,
                        ((System.Drawing.FontStyle)
                        ((System.Drawing.FontStyle.Bold | System.Drawing.FontStyle.Italic))),
                        System.Drawing.GraphicsUnit.Point,
                        ((ubyte)(System.Byte)(((ubyte)0)))));
this.label2.set_ForeColor(System.Drawing.Color.get_Blue());
this.label2.set_Location(new System.Drawing.Point(16, 184));
this.label2.set_Name("label2");
this.label2.set_TabIndex(24);
```

```
this.label2.set_Text("Transmisión");
this.label2.set_TextAlign(System.Drawing.ContentAlignment.MiddleCenter);
//
// label3
//
this.label3.set_Font ( new System.Drawing.Font ( "Microsoft Sans Serif", 9.75F,
                        ((System.Drawing.FontStyle)
                        ((System.Drawing.FontStyle.Bold | System.Drawing.FontStyle.Italic))),
                        System.Drawing.GraphicsUnit.Point,
                        ((ubyte)(System.Byte)(((ubyte)0)))));
this.label3.set_ForeColor(System.Drawing.Color.get_Blue());
this.label3.set_Location(new System.Drawing.Point(16, 248));
this.label3.set_Name("label3");
this.label3.set_TabIndex(25);
this.label3.set_Text("Recepción");
this.label3.set_TextAlign(System.Drawing.ContentAlignment.MiddleCenter);
//
// tx
//
this.tx.set_Location(new System.Drawing.Point(480, 216));
this.tx.set_Name("tx");
this.tx.set_Size(new System.Drawing.Size(88, 24));
this.tx.set_TabIndex(26);
this.tx.set_Text("Transmitir");
this.tx.add_Click( new System.EventHandler(this.tx_Click) );
//
// textBox6
//
this.textBox6.set_Enabled(false);
this.textBox6.set_Location(new System.Drawing.Point(16, 280));
this.textBox6.set_Multiline(true);
this.textBox6.set_Name("textBox6");
this.textBox6.set_Size(new System.Drawing.Size(456, 104));
this.textBox6.set_TabIndex(27);
this.textBox6.set_Text("");
//
// Form1
//
this.set_AutoScaleBaseSize(new System.Drawing.Size(5, 13));
this.set_ClientSize(new System.Drawing.Size(584, 398));
this.get_Controls().Add(this.textBox6);
this.get_Controls().Add(this.tx);
this.get_Controls().Add(this.label3);
this.get_Controls().Add(this.label2);
this.get_Controls().Add(this.textBox5);
this.get_Controls().Add(this.label1);
this.get_Controls().Add(this.pictureBox1);
this.get_Controls().Add(this.btrama);
this.get_Controls().Add(this.btstp);
this.get_Controls().Add(this.vlc);
this.get_Controls().Add(this.par);
this.get_Controls().Add(this.textBox4);
this.get_Controls().Add(this.textBox3);
this.get_Controls().Add(this.textBox2);
this.get_Controls().Add(this.textBox1);
this.get_Controls().Add(this.estado);
this.get_Controls().Add(this.port);
this.get_Controls().Add(this.trama);
this.get_Controls().Add(this.bs);
this.get_Controls().Add(this.velocidad);
this.get_Controls().Add(this.paridad);
this.get_Controls().Add(this.abrir);
this.get_Controls().Add(this.puerto);
this.get_Controls().Add(this.eligepuerto);
```

```
        this.set_Icon(((System.Drawing.Icon)(resources.GetObject("$this.Icon"))));
        this.set_Name("Form1");
        this.set_Text("Controlador del puerto RS-232");
        this.ResumeLayout(false);

}

#endregion

/**
 * Punto de entrada principal de la aplicación.
 */

/** @attribute System.STAThread() */

public static void main(String[] args)
{
        Application.Run(new Form1());
}

// Botón de selección de puerto
private void button1_Click (Object sender, System.EventArgs e)
{
        String s;
        s=puerto.get_Text();
        try {
                if(ps.activo("COM"+s)==false) {
                        Error nuevo;
                        nuevo = new Error("El puerto indicado no es correcto");
                } else {
                        ps.eligePuerto(java.lang.Integer.parseInt(s));
                        port.set_Text("COM"+s);
                        if(ps.estaAbierto()==true)  {
                                ps.cerrar();
                        }
                        ps.configuracionInicial();
                        abrir.set_Text("abrir");
                        paridad.set_Text("ninguna");
                        velocidad.set_Text("9600");
                        bs.set_Text("1");
                        trama.set_Text("8");
                        estado.set_Text("cerrado");
                }
        } catch(Exception ex) {
                Error nuevo;
                nuevo = new Error("El puerto indicado no es correcto");
        }
}

// Botón de paridad
private void par_Click (Object sender, System.EventArgs e) {
        String s;
        s=this.textBox1.get_Text();
        try {
                if ( java.lang.String.Equals(s,"ninguna")==true)  {
                        ps.eligeParidad(s);    paridad.set_Text(s);
                } else {
                        if(java.lang.String.Equals(s,"par")==true)   {
                                ps.eligeParidad(s);    paridad.set_Text(s);
                        } else  {
                                if(java.lang.String.Equals(s,"impar")==true)  {
                                        ps.eligeParidad(s);  paridad.set_Text(s);
                                }
```

```
                                else {
                                        if ( java.lang.String.Equals(s,"marca")==true ) {
                                                ps.eligeParidad(s);
                                                paridad.set_Text(s);
                                        } else  {
                                                if ( java.lang.String.Equals(s,"espacio")==true) {
                                                        ps.eligeParidad(s);
                                                        paridad.set_Text(s);
                                                } else  {
                                                        Error nuevo;
                                                        nuevo = new Error ( s+" no es paridad válida. ej:
                                                                par, impar, marca, espacio, ninguna");
                                                }
                                        }
                                }
                        }
                }
                catch ( Exception ex ) {
                        Error nuevo;
                        nuevo = new Error(s+" no es paridad válida. ej: par, impar, marca, espacio, ninguna");
                }
        }

        // Botón de velocidad
        private void vlc_Click (Object sender, System.EventArgs e) {
                int i;
                String s;
                s=this.textBox2.get_Text();
                try {
                        i=java.lang.Integer.parseInt(s);
                        if ( i==1200 ) { ps.eligeVelocidad(i);  velocidad.set_Text(s);
                        } else { if ( i==2400 )   { ps.eligeVelocidad(i); velocidad.set_Text(s);
                        } else { if ( i==4800 )   { ps.eligeVelocidad(i);  velocidad.set_Text(s);
                        } else { if ( i==9600 )   { ps.eligeVelocidad(i); velocidad.set_Text(s);
                        } else {if ( i==56000 )  { ps.eligeVelocidad(i); velocidad.set_Text(s);
                        } else {if ( i==128000 ){ ps.eligeVelocidad(i);  velocidad.set_Text(s);
                        } else {if ( i==256000 ){
                        } else {   Error nuevo;
                                nuevo = new Error( s+" no es velocidad válida. ej: 1200, 2400, 4800, 9600,
                                                56000, 128000, 256000");
                        }}}}}}}

                } catch ( Exception ex ) {
                        Error nuevo;
                        nuevo = new Error ( s+" no es velocidad válida. ej: 1200, 2400, 4800, 9600, 56000,
                                        128000, 256000");
                }
        }

        // Botón de Bits de Stop
        private void btstp_Click (Object sender, System.EventArgs e) {
                int i;
                String s;
                s=this.textBox3.get_Text();
                try  {
                        i=java.lang.Integer.parseInt(s);
                        switch ( i ) {
                                case 1:         ps.eligeBitsDeParada(i);        bs.set_Text(s);     break;
                                case 2:         ps.eligeBitsDeParada(i);        bs.set_Text(s);     break;
                                default:        Error nuevo;
                                                nuevo = new Error(s+" no es bit de parada válido. ej: 1 ó 2");
                                                break;
                        }
```

```
        } catch(Exception ex) {
                Error nuevo;
                nuevo = new Error(s+" no es bit de parada válido. ej: 1 ó 2");
                bs.set_Text("1");
        }
}

// Botón de Bits de trama
private void btrama_Click (Object sender, System.EventArgs e) {
        int i;
        String s;
        s=this.textBox4.get_Text();
        try  {
                i=java.lang.Integer.parseInt(s);
                switch ( i ) {
                        case 5:         ps.eligeTamañoDeLaTrama(i);  trama.set_Text(s);  break;
                        case 6:         ps.eligeTamañoDeLaTrama(i);  trama.set_Text(s);  break;
                        case 7:         ps.eligeTamañoDeLaTrama(i);  trama.set_Text(s);  break;
                        case 8:         ps.eligeTamañoDeLaTrama(i);  trama.set_Text(s);  break;
                        default:        Error nuevo;
                                        nuevo = new Error(s+" no es tamaño trama válida. ej: 5, 6, 7 u 8");
                                        break;
                }
        } catch ( Exception ex ) {
                Error nuevo;
                nuevo = new Error(s+" no es tamaño trama válida. ej: 5, 6, 7 u 8");
        }

}

// Botón de abrir y cerrar puerto
private void abrir_Click (Object sender, System.EventArgs e) {
        if(ps.estaAbierto()==true) {
                ps.cerrar();
                ps.configuracionInicial();
                abrir.set_Text("abrir");
                paridad.set_Text("ninguna");
                velocidad.set_Text("9600");
                bs.set_Text("1");
                trama.set_Text("8");
                estado.set_Text("cerrado");
        } else {
                ps.abrir();
                abrir.set_Text("cerrar");
                estado.set_Text("abierto");
        }

}

// Botón de transmitir
private void tx_Click (Object sender, System.EventArgs e) {
        String s;
        s=textBox5.get_Text();
        if(java.lang.String.Equals(s,"")==true) {
                Error nuevo;
                nuevo = new Error("El campo para enviar está vacío");
        } else {
                if(ps.estaAbierto()==false) {
                        Error nuevo;  nuevo =  new Error("El puerto está cerrado");
                } else {
                        ps.enviar(s);   textBox5.set_Text("");
                }
        }
}
```

```
        // Lectura con Thread
        public void run ( ) {
                String s;
                s = ps.recibir ( );
                if ( ps.estaAbierto()==true ) {
                        if(java.lang.String.Equals(s,"")==true) {
                        }
                        else {
                                s=java.lang.String.Concat(textBox6.get_Text(),s);
                                textBox6.set_Text(s);
                        }
                }
                else {
                }
                try {
                        Thread.sleep(50);
                }
                catch(InterruptedException ie) {
                }
        }
} // class Form1
```

Anexo C2: Aplicación de control de errores .NET

```
package COM;

import System.Drawing.*;
import System.Collections.*;
import System.ComponentModel.*;
import System.Windows.Forms.*;

/**
 * Descripción de resumen para Error.
 */
public class Error extends System.Windows.Forms.Form {
        private System.Windows.Forms.PictureBox pictureBox1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Label label1;

       /**
        * Variable del diseñador necesaria.
        */

        private System.ComponentModel.Container components = null;

        public Error (String s) {
              //
              // Necesario para mantener compatibilidad con el Diseñador de Windows Forms
              //
              InitializeComponent();
              label1.set_Text(s);
        }

       /**
        * Limpiar los recursos que se estén utilizando.
        */
        protected void Dispose(boolean disposing) {
                if (disposing) {
                        if (components != null) {
                                components.Dispose();
                        }
                }
                super.Dispose(disposing);
        }

        #region Código generado por el Diseñador de Windows Forms

       /**
        * Método necesario para mantener compatibilidad con el Diseñador; no modifique
        * el contenido del método con el editor de código.
        */

        private void InitializeComponent ( ) {
                System.Resources.ResourceManager resources =
                        new System.Resources.ResourceManager(Error.class.ToType());
                this.pictureBox1 = new System.Windows.Forms.PictureBox();
                this.button1 = new System.Windows.Forms.Button();
                this.label1 = new System.Windows.Forms.Label();
                this.SuspendLayout();
```

```
            //
            // pictureBox1
            //
                this.pictureBox1.set_Image (((System.Drawing.Image)
                                ( resources.GetObject ( "pictureBox1.Image"))));
                this.pictureBox1.set_Location(new System.Drawing.Point(16, 16));
                this.pictureBox1.set_Name("pictureBox1");
                this.pictureBox1.set_Size(new System.Drawing.Size(56, 64));
                this.pictureBox1.set_TabIndex(0);
                this.pictureBox1.set_TabStop(false);
            //
            // button1
            //
                this.button1.set_Location(new System.Drawing.Point(200, 72));
                this.button1.set_Name("button1");
                this.button1.set_TabIndex(1);
                this.button1.set_Text("Aceptar");
                this.button1.add_Click( new System.EventHandler(this.button1_Click) );
            //
            // label1
            //
                this.label1.set_Location(new System.Drawing.Point(88, 16));
                this.label1.set_Name("label1");
                this.label1.set_Size(new System.Drawing.Size(336, 48));
                this.label1.set_TabIndex(2);
                this.label1.set_TextAlign(System.Drawing.ContentAlignment.MiddleCenter);
            //
            // Error
            //
                this.set_AutoScaleBaseSize(new System.Drawing.Size(5, 13));
                this.set_ClientSize(new System.Drawing.Size(440, 110));
                this.get_Controls().Add(this.label1);
                this.get_Controls().Add(this.button1);
                this.get_Controls().Add(this.pictureBox1);
                this.set_Icon(((System.Drawing.Icon)(resources.GetObject("$this.Icon"))));
                this.set_Name("Error");
                this.set_Text("Error");
                this.ResumeLayout(false);
                this.set_Visible(true);

        }

        #endregion

        private void button1_Click (Object sender, System.EventArgs e)
        {
                this.Dispose();
                this.Close();
        }

} // class Error
```

Bibliografía

- D. Birngruber, W. Kurschl, J. Sametinger, "*Comparison of JavaBeans and ActiveX – A Case Study*", Smalltalk und Java in Industrie und Ausbildung (STJA 99), Erfurt (Germany), September 28-30, 1999.
- N. Medvidovic, N. R. Mehta, "*JavaBeans and Software Architecture*", The Internet Encyclopedia, Hossein Bidgoli (ed.), John Wiley & Sons, 2003.
- Componente SerialPort, información disponible en:

  http://www.gotdotnet.com/Community/UserSamples/Details.aspx?SampleGuid=b06e30f9-1301-4cc6-ac14-dfe325097c69
- Software de comunicación para puertos serie/paralelo/USB, información disponible en: http://www.bocasystems.com/bidirectional.html
- Programación en C y Java e historia de la computación, información disponible en: http://www.lenguajes-de-programacion.com
- Introducción, Propiedades, Creación y Uso de JavaBeans, información disponible en:

  http://www.sc.ehu.es/sbweb/fisica/cursoJava/applets/javaBeans/intro.htm
- Principios de la Ingeniería Software, información disponible en: http://siul02.si.ehu.es/~alfredo/iso/
- Componentes y comparación entre arquitecturas COM/DCOM, CORBA, CGI y JAVA, información disponible en:

  http://www.monografias.com/trabajos16/componentes/componentes.shtml
- Uso de Microsoft Visual Studio .NET, información disponible en: http://msdn.microsoft.com/library/spa/default.asp?url=/library/SPA/cptutorials/html/writing_simple__net_components.asp