Universidad de Valladolid

E. T. S. DE INGENIERÍA INFORMÁTICA

Ingeniería Técnica en Informática de Gestión

Línea de Productos de Marcadores Deportivos: Desarrollo de Paquetes Opcionales

---

Alumnos: **Mario Muñoz Sanz**
Daniel Becerril Rodríguez

Tutor: Miguel Ángel Laguna Serrano

AGRADECIMIENTOS

DANIEL:
Muchas son las personas que han colaborado para que este proyecto saliera adelante y mis ánimos no decayeran en ningún momento.

Gracias a mi compañero Mario, quien sin su ayuda esto no habría salido adelante.

Gracias a mis padres, quienes siempre han entendido la dificultad que conllevaba este proyecto, y que siempre han estado a mi lado depositando toda su confianza en mí. También a mis hermanos (Javi y Ana) y a mi cuñada (María Jesús), quienes me han ofrecido en todo momento su ayuda.

Gracias a Laura, a quien tengo la sensación de haber dedicado menos tiempo del necesario, y quien ha entendido las difíciles decisiones que he tenido que tomar durante la realización del proyecto.

Gracias a Aurelio, quien en los momentos de mayor dificultad del proyecto y de crisis personal ha sido y es uno de los apoyos más importantes de mi vida.

Gracias a mis amigos, en especial a Susana por apoyarme en los momentos difíciles derivados de este proyecto cuando no creía que pudiera sacarlo adelante. También a todos mis amigos de Palencia: Fernando, Jesús, Jorge, Goyo, Óscar, Carmen… y un largo etcétera, por darme ánimos y comprender la situación que he vivido a raíz de la realización del proyecto.

Muchas gracias a todos, porque sin vuestra ayuda este proyecto no habría sido posible

MARIO:
Gracias a mi compañero Dani, que ha sido un compañero ejemplar para mí y ha sabido comprender mi falta de tiempo.

Gracias a mis padres, que han estado siempre interesados en mi progreso y me han dado la ilusión por seguir adelante y acabar este proyecto. También doy las gracias a mis hermanos (Sara y Javi) por tener paciencia conmigo y echarme una mano cuando el tiempo me faltaba.

Gracias a Estrella, por haber sido el mejor apoyo, comprender mis momentos de estrés y sobre todo, por estar a mi lado en todo momento. Nunca terminare de agradecerle todo lo que me ha dado desde que la conocí.

Por último, dar las gracias a mis amigos, por haberse interesado siempre por mi trabajo y proporcionarme esos momentos de diversión tan necesarios en un tiempo tan estresante como este

También queremos agradecer a nuestro tutor Miguel Ángel toda la ayuda que nos ha prestado para la realización del proyecto. No queremos olvidarnos de Arturo y Sara, que forman parte esencial en este proyecto, al haberlo comenzado los 4 juntos.

ÍNDICES

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

PARTE 1

INTRODUCCIÓN

Capítulo 1

PRESENTACIÓN DEL PROYECTO

1.1. Descripción y ámbito del proyecto

Este proyecto ha sido propuesto por el Grupo de Investigación en Reutilización y Orientación al Objeto (GIRO) y está orientado al desarrollo de una línea de productos software en el dominio de los eventos deportivos.

Las Líneas de Productos Software (LPS) permiten la reutilización sistemática en los casos en los que se tienen familias de productos, es decir, aplicaciones similares pero diferenciadas por algunas características. El objetivo es sacar el máximo partido de los elementos comunes, y gestionar de una manera eficaz las variaciones.

Una Línea de Productos Software es un grupo de productos que comparten un conjunto de características comunes, y que individualmente ofrecen características propias que los diferencian del resto de productos de su misma familia. Cada uno de ellos está orientado a satisfacer las necesidades específicas de un segmento de mercado particular. De este modo se promueve la industrialización del desarrollo software.

Nuestro proyecto en concreto, consiste en el análisis, documentación y desarrollo de una línea de productos software en el dominio mencionado.

Uno de los principales problemas que plantea el desarrollo de líneas de productos es la representación y gestión tanto de la variabilidad como de las partes comunes de dicha línea de productos. La forma habitual de definir ambos aspectos durante la fase de requisitos es mediante modelos de características o “features”.

Otro de los problemas que plantean las líneas de producto es cómo plasmar dicha variabilidad en las fases posteriores y distinguir entre la variabilidad propia de la línea de productos y de los productos individuales. El proyecto realizado, se basa en una propuesta del grupo GIRO que propone la utilización de mecanismos UML. La propuesta consiste en la utilización del mecanismo de combinación de paquetes (“package merge”) para representar la variabilidad de la línea de productos, restringiendo los mecanismos clásicos de modelado (como sería el caso de la relación <<extiende>>) para expresar las variantes válidas en tiempo de ejecución de cada aplicación concreta.

Es decir, el propósito principal es aprovechar el modelo de desarrollo convencional aplicado al desarrollo de una línea de productos.

Este proyecto fin de carrera se ha realizado conjuntamente con otro coincidiendo en el tiempo. Nuestro proyecto trata de ampliar las funcionalidades de un marcador deportivo básico (diseñado conjuntamente por los autores de ambos proyectos, e implementado por los autores del otro proyecto: Arturo Nozal y Sara Calle). Nuestro Marcador Deportivo podrá estar distribuido en varios dispositivos móviles que se conectarán entre sí para formar un sistema cooperativo. Así profundizamos en el concepto de Líneas de Productos, creando nuevos productos a partir de otros ya implementados.

1.2. Alcance del proyecto

Este proyecto está orientado a facilitar y mejorar el seguimiento del transcurso de encuentros deportivos de todo tipo. Para ello, tiene como objetivo la creación de marcadores adaptables a las necesidades de los distintos deportes. Para ello, se utilizara el concepto de línea de producto software.

Su utilización está principalmente destinada a deportes escolares en los que el seguimiento de la evolución los encuentros, por parte de los asistentes, es difícil debido a la falta de medios, como por ejemplo marcadores electrónicos.

Además está destinado a todos aquellos que deseen obtener información sobre el estudio y desarrollo de líneas de producto software y el aprovechamiento para ello de herramientas ya existentes como es UML 2 así como el mecanismo de combinación de paquetes o "package merge".

Propone a empresas y particulares un paso más en la reutilización del software, pudiendo tener la capacidad de crear eficientemente varias variantes de un mismo producto aprovechando las partes comunes a todos, y particulares de cada variante.

1.3. Sobre esta documentación

El presente documento está estructurado en seis grandes bloques: Introducción, Desarrollo de la Solución, Manual de Usuario, Conclusiones, Referencias y Apéndices.

Este capítulo, titulado "*Presentación del proyecto*" junto con los siguientes "*Líneas de Producto Software" (*en el que se trata en profundidad el concepto de las LPS) y "*Introducción a los dispositivos móviles*" (que hace un breve repaso a los dispositivos móviles que hay en la actualidad), forman la Introducción. En dichos capítulos se da una idea general de lo que es el proyecto realizado y de los conceptos teóricos en los que se basa.

El siguiente bloque versa acerca del Desarrollo de la Línea de Productos. Está formado por los capítulos "*Análisis del sistema*", "*Diseño del Sistema*", "*Implementación de la solución*" y "*Pruebas*". En este bloque se explica con detalle todos los pasos que se han dado para la creación

de la línea de productos, desde los inicios con el diseño de la línea, hasta las pruebas finales, pasando por las etapas de diseño e implementación.

Posteriormente viene el bloque del Manual de Usuario. Este bloque pretende ser una guía de uso de nuestra aplicación. Se explica en él con todo detalle las funcionalidades de la aplicación, así como los pasos necesarios para llevar a cabo su instalación en el dispositivo móvil.

Tras éste está el bloque de Conclusiones. En él se pretende hacer un análisis a posteriori de todas las dificultades con las que nos hemos encontrado en la elaboración del proyecto, así como los objetivos que se han alcanzado de aquellos propuestos en la introducción. También se hace un repaso de todos los conocimientos adquiridos gracias a la realización del proyecto, y se exponen futuras líneas de trabajo que pueden surgir tras la realización de nuestra línea de productos.

A continuación nos encontramos con el apartado Referencias. En él mostramos todas las fuentes que hemos utilizado y consultado para realizar el proyecto. En este apartado se encuentran tanto las fuentes escritas (como son los libros), como las fuentes web.

Por último hemos añadido un bloque llamado Apéndices. La razón que nos ha llevado a añadirlo es que aquellos usuarios finales del proyecto tengan un conocimiento mayor de las tecnologías que hemos usado (razón por la cual se ha añadido información de las mismas en los apéndices A y B). El apéndice C es el contenido del CD ROM que se adjunta con esta memoria.

1.4. Objetivos del proyecto

- Objetivos específicos del proyecto:
  - Conocer y comprender el concepto de Línea de Productos Software.
  - Desarrollar una Línea de Productos Software de Marcadores Deportivos aplicados a dispositivos móviles.
  - Desarrollar dicha línea de productos con Visual Studio .NET para poder crear con facilidad nuevos marcadores deportivos seleccionando las características necesarias en cada momento.
  - Ampliar la funcionalidad de un marcador deportivo básico creado en otro Proyecto Fin de Carrera, utilizando el concepto de combinación de paquetes.
  - Integrar las nuevas funcionalidades con el núcleo de la Línea de Producto ya creado.
  - Generar varios prototipos de marcadores deportivos a partir de los paquetes desarrollados.

- Objetivos generales:
  - Conocer y utilizar un entorno de trabajo específico, en nuestro caso el desarrollo de aplicaciones para dispositivos móviles con Visual Studio .NET, centrándonos en el lenguaje de programación C#.
  - Poner en práctica nuestras habilidades para el trabajo en equipo, y mejorarlas en la medida de lo posible.

- Usar todos los conocimientos adquiridos a lo largo de nuestra carrera universitaria para la elaboración del Proyecto Fin de Carrera.

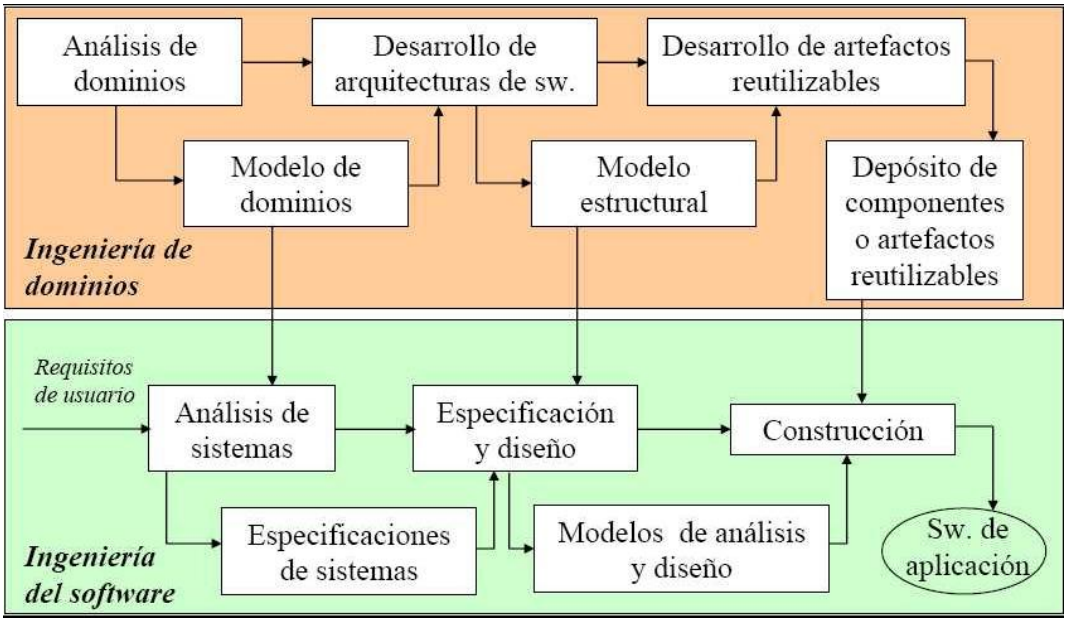
Capítulo 2
LÍNEAS DE PRODUCTO SOFTWARE

2.1.La reutilización del software

El término reutilización vio la luz en una conferencia de la OTAN de 1968 sobre Ingeniería del Software. Fue propuesto por M.D. McIlroy como una de las principales y mejores soluciones para aumentar la productividad de los desarrolladores de software, así como para aumentar la fiabilidad de los productos software construidos. La idea básica respondía al principio de aprovechar esfuerzos previos para completar un nuevo desarrollo.

Actualmente, el concepto de reutilización ha evolucionado hacia la idea de que todo el conocimiento y productos derivados de la producción de software son susceptibles de ser reutilizados en la construcción de nuevos sistemas. Este enfoque se basa en la selección e integración de elementos del software que hayan sido intencionadamente diseñados, desarrollados y documentados para servir como materia prima para nuevos productos software. De esta forma aparece el concepto de *asset* o de *componente software reutilizable*, planteado por Karlsson en 1995.

En resumidas cuentas, se podría definir reutilización como "cualquier procedimiento que conduce o ayuda a producir un sistema mediante el nuevo uso de algún elemento procedente de un esfuerzo de desarrollo anterior", según la definición de P. Freeman de 1987.



2.1: Proceso de reutilización del software

Aún hoy en día, la reutilización continúa siendo uno de los campos de mayor investigación dentro de la Ingeniería del Software. Esto se debe a que a pesar de sus avances y de ser la solución más realista para los problemas de productividad en el desarrollo, muchos autores afirman que no se han conseguido avances significativos para la utilización sistemática de la reutilización en el proceso de construcción de software.

Según otros autores, la reutilización externa, es decir, la que se lleva a cabo a la hora de reutilizar componentes software de terceros, ha sido un éxito. Algún ejemplo de este tipo sería la reutilización de sistemas operativos o de servidores de bases de datos.

Sin embargo, la reutilización interna desarrollada por las propias organizaciones no ha experimentado grandes avances, por lo que es un objetivo a conseguir. Para aumentar este tipo de reutilización, se han hecho intentos con las bibliotecas, la orientación a objetos, los componentes y últimamente con las Líneas de Producto Software.

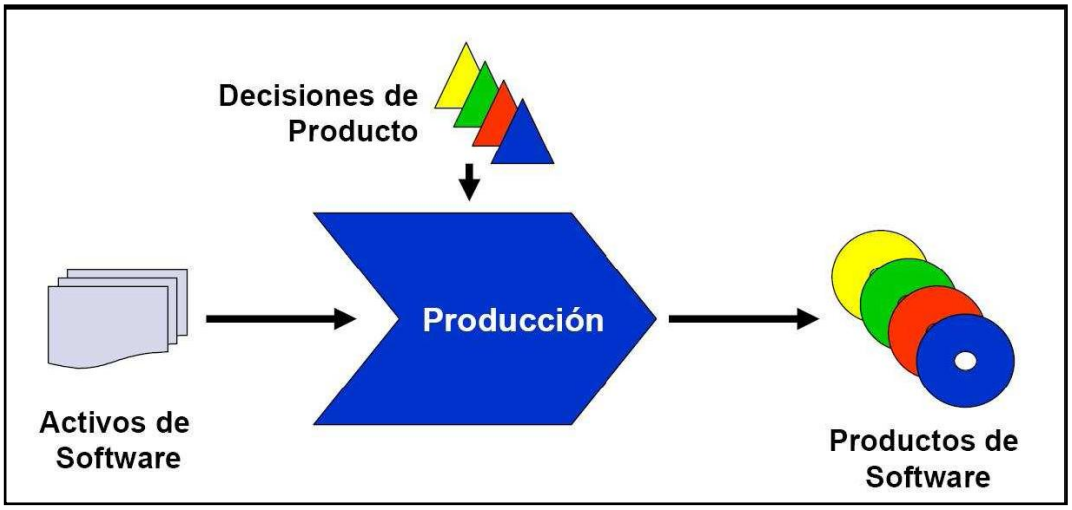
2.2. Descripción General

2.2.1. Introducción

La idea básica de lo que es una Línea de Producto Software está inspirada en los procesos de producción industrializada de los productos físicos, tales como la producción de vehículos o hardware. Consiste en el ensamblaje de partes de software previamente elaboradas.

"Una Línea de Producto Software es un conjunto de sistemas de software que comparten un conjunto común y gestionado de aspectos que satisfacen las necesidades específicas de un segmento de mercado o misión y que son desarrollados a partir de un conjunto común de activos fundamentales de software de una manera preescrita" (Clements y Northrop; Software Products Lines: Practices and Patterns, 2001).

Las Líneas de Producto Software pueden ser descritas en términos de cuatro conceptos básicos, que están representados en la siguiente figura:



2.2: Conceptos básicos de una Línea de Productos software

- Activos software reutilizables (assets): colección de componentes software, tales como requisitos, casos de uso, arquitectura o documentación, que pueden ser configurados y combinados de diferentes maneras para dar lugar a los distintos productos de una línea.
- Decisiones de Producto y Modelo de Decisión: cada uno de los productos pertenecientes a una línea se diferencia de los demás gracias a la elección de las características variables y opcionales que hay en el modelo de decisión. Es decir, quedan definidos por las decisiones de producto.
- Mecanismo y procesos de producción: los medios que permiten la configuración de productos a partir de activos software reutilizables. Durante este proceso, las decisiones de

producto determinan cuáles serán los componentes que van a utilizarse y cómo configurar los puntos de variabilidad entre ellos.

- Productos de Software: colección de todos los productos que pueden crearse para la línea de productos. El alcance de la línea de productos está determinado por el conjunto de productos software que puede ser producido a partir de los activos reutilizables y del modelo de decisión.

2.2.2. Beneficios

El beneficio principal de utilizar la tecnología de LPS es que la entrega de productos se hace de una manera más rápida y económica porque se reducen los costes de ingeniería y con una calidad mucho mayor ya que se reducen las tasas de errores.

Según Charles Krueger existen una serie de argumentos a favor de las Líneas de Producto, que se podrían dividir en beneficios tácticos y estratégicos.

Los beneficios tácticos serán los siguientes:

- Reducción en el tiempo medio de creación y entrega de nuevos productos.
- Reducción en el número medio de defectos por producto.
- Reducción en el esfuerzo medio requerido para desarrollar y mantener los productos.
- Reducción en el coste medio de producción.
- Incremento en el número total de productos que pueden ser desarrollados y mantenidos.
- Por otro lado los beneficios estratégicos de negocios serían:
- Reducción en el tiempo de entrega de nuevos productos.
- Mejoras en el valor competitivo del producto.
- Márgenes mayores de ganancias.
- Mejor calidad de los productos.
- Mejoras en la reputación de la empresa.
- Mayor escalabilidad del modelo de negocios en términos de productos y mercados.
- Mayor agilidad para expandir el negocio a nuevos mercados.
- Reducción de riesgos en la entrega de productos.

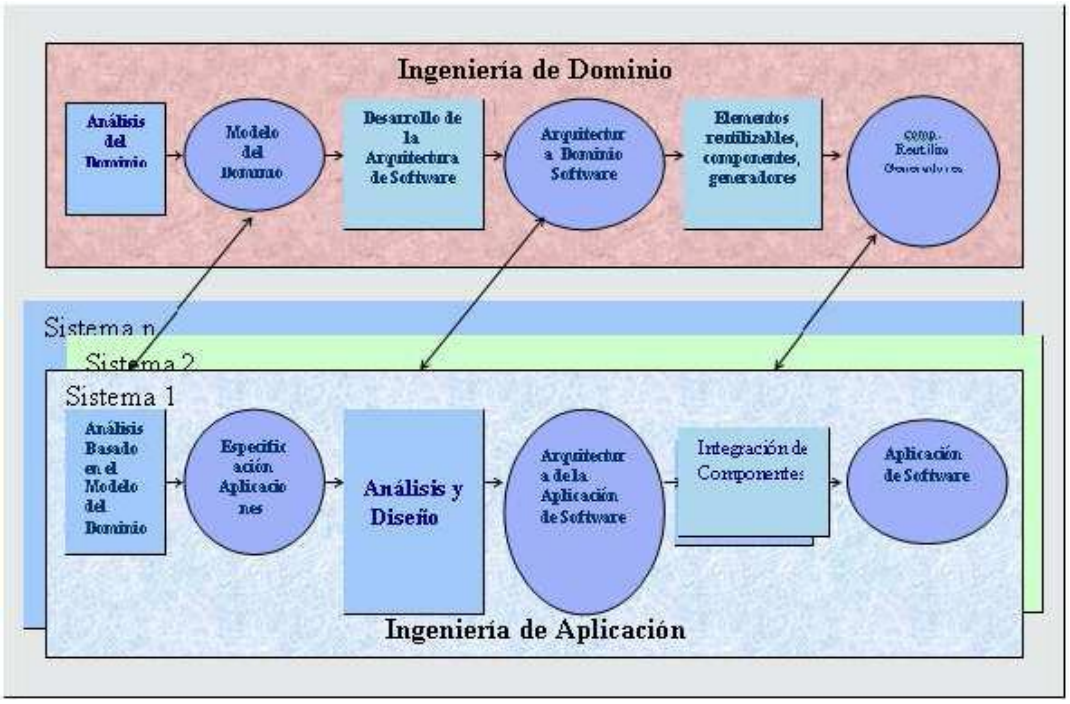
2.2.3. Desarrollo

El desarrollo de las Líneas de Producto Software introduce un cambio en la forma de crear software, ya que este trabajo se convierte en un proceso cada vez más industrializado. Lo que se pretende es reutilizar las partes que tienen en común todos los productos y desarrollar de forma eficiente y sistemática nuevos miembros de la familia simplemente con añadirle nuevas funcionalidades.

Las dos actividades clave para poder desarrollar una Línea de Producto son la Ingeniería de Dominio y la Ingeniería de Aplicación. Para exponer la relación existente entre ellas, lo mejor es definirlas y relacionarlas entre sí, y al mismo tiempo, con el concepto de reutilización.

La Ingeniería de Dominio y la Ingeniería de Aplicación son complementarias, interactúan en procesos paralelos. El modelo de procesos basado en componentes incorpora explícitamente la reutilización del software en el proceso de desarrollo de aplicaciones. Este modelo considera la reutilización desde las perspectivas de las dos ingenierías:

- Desarrollo de software para la reutilización: el propósito es producir componentes de software reutilizables. A este proceso se le denomina Ingeniería de Dominio.
- Desarrollo de software con reutilización: su propósito es desarrollar software reutilizando componentes existentes. Este proceso se llama Ingeniería de Aplicación.

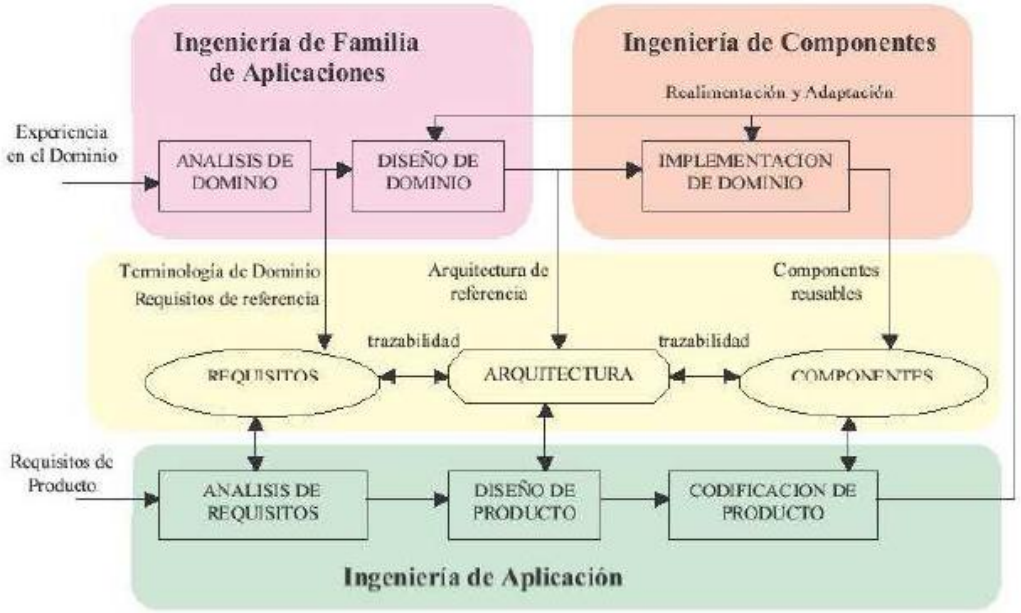


2.3: Ingenierías de dominio y de aplicación

La **Ingeniería de Dominio** se dirige a la creación sistemática de modelos de dominios, arquitecturas, componentes y artefactos de software reutilizables en el desarrollo de cualquier nuevo producto de una LPS.

La **Ingeniería de Aplicación** se basa en la reutilización de componentes existentes y en el conocimiento del dominio. Los productos de la LPS son construidos mediante el ensamblaje de

activos de software.



2.4: Desarrollo de una Línea de Producto Software

2.2.4. Trazabilidad de requisitos

La trazabilidad de requisitos es un reconocido factor de calidad en los procesos de desarrollo de software. Es la habilidad para seguir la vida de un requisito en ambos sentidos, hacia sus orígenes o hacia su implementación, a través de las explicaciones generadas en el proceso de desarrollo. Esta propiedad es clave para la consistencia entre los diferentes modelos y etapas del desarrollo, y en el mantenimiento de las Líneas de Producto Software.

2.3. Modelo de Características para tratar la variabilidad

2.3.1. Definición

La variabilidad se define, según Svanhberg, como la habilidad de cambio o de personalización de un sistema. Para la definición de requisitos de una línea de productos, hay que prestar especial atención al análisis de la parte común y la parte variable, estableciendo las dependencias que existen entre ellas.

Para ello, los métodos más utilizados son los basados en *features*, como FODA (*Feature Oriented Domain Analysis*). Esta es una metodología desarrollada por el SEI (Software Engineering Institute) para la aplicación del análisis de dominio, definiendo las etapas del método y los resultados obtenidos en cada una de ellas. El proceso se basa en identificar las características que los usuarios esperan comúnmente en las aplicaciones dentro de un dominio dado. El método FODA soporta el descubrimiento, análisis y la documentación de los aspectos comunes y las diferencias de un dominio.

Este método se basa en la utilización de árboles jerárquicos para organizar las capacidades de la línea de productos. Estas capacidades pueden ser obligatorias (comunes para todos los productos), opcionales o alternativas. Aquellas que son comunes van colocadas en los nodos raíz del árbol, mientras que las variantes representan las ramas. De este modo, en un único modelo se encuentran representadas todas las variaciones posibles de la línea de productos.

2.3.2. Problemas del modelo de características

Actualmente, los modelos de características presentan dos problemas:

1. Sólo se centran en aspectos funcionales de la línea, es decir, no han sido diseñados para representar información extra-funcional o de calidad.

2. El razonamiento automático propuesto sobre este tipo de modelos es muy limitado y en ningún caso tratan aspectos extra–funcionales.

En el enfoque de desarrollo de línea de productos, cada aplicación concreta se deriva del framework que implementa la arquitectura de la línea de productos. En este proceso, se deben seleccionar aquellas variantes que resultan apropiadas para los requisitos funcionales y no funcionales expresados por los usuarios. Esta actividad es esencialmente una selección de características efectuada por el ingeniero de aplicación, generando un sub-modelo de características, que a su vez (gracias a las relaciones de trazabilidad) generará por derivación toda o la mayor parte del código de la aplicación.

2.4. Trazabilidad

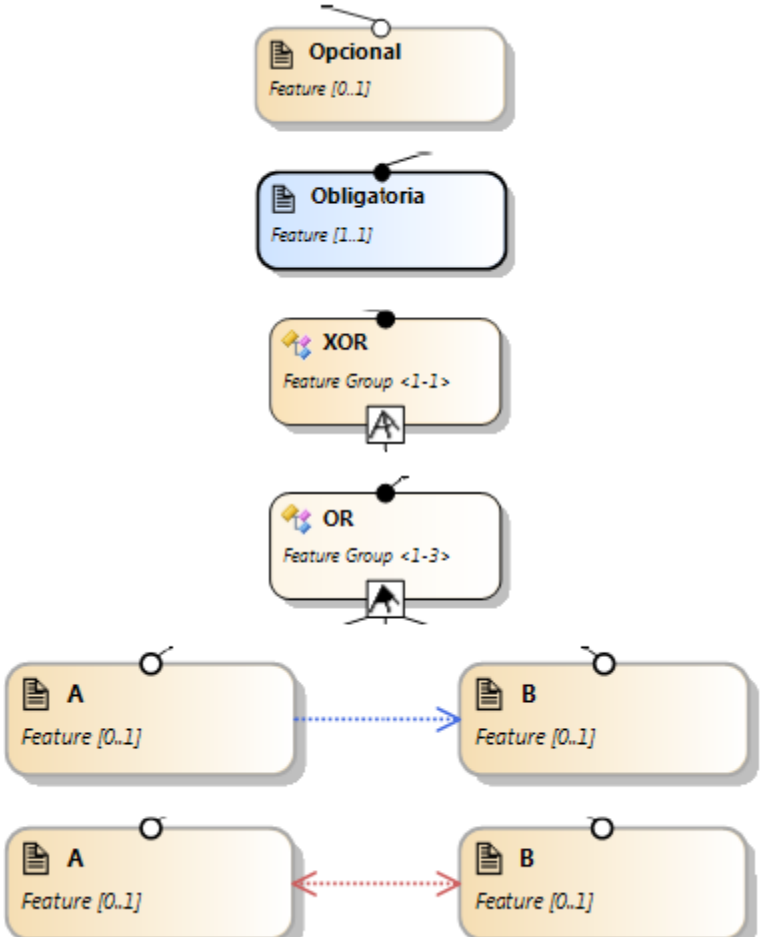
2.4.1. Problemas

La clave reside en la trazabilidad desde las metas/características hasta el código pasando por los modelos de diseño. Sin embargo, esta trazabilidad no es fácil de gestionar por varias razones.

En primer lugar, una característica opcional puede originar varios elementos en un modelo de diseño. Es decir, tenemos que asignar la relación de trazabilidad entre elementos de los dos niveles con una multiplicidad uno a varios, y lo mismo en sentido contrario, lo que complica rápidamente el modelo global haciendo que sea muy poco escalable.

El segundo problema tiene que ver con el hecho de que los mecanismos básicos de modelado de la variabilidad (la especialización en los diagramas de clases o la relación <<extend>> de los casos de uso) se utilizan en muchas ocasiones para expresar dos niveles de variabilidad distinta: el diseño de la arquitectura de la línea de productos (que se corresponde con requisitos opcionales) y el diseño de una aplicación concreta, que sigue teniendo variaciones en tiempo de ejecución.

El problema está en que el mismo mecanismo sirve para mostrar variaciones en tiempo de ejecución y en tiempo de configuración.



2.5 Notación en el modelo de características

Sin embargo, en el trabajo pionero de Jacobson, no hay diferencia fundamental entre la representación de la variabilidad en ambos niveles: por ejemplo la relación <<extend>> de los casos de uso es uno de los pilares con los que construye su método.

En resumen, este enfoque tiene como problema principal la falta de separación entre la variabilidad global de la línea de productos y la variabilidad residual de cada aplicación concreta.

2.4.2. Soluciones aportadas para gestionar la variabilidad

Las soluciones más recientes propuestas para mostrar la variabilidad de una línea de productos con UML pasan por modificar o anotar los modelos, tanto estructurales como

funcionales o incluso dinámicos. De este modo se resuelve el segundo problema, pero raramente se ataca el primero (múltiples dependencias entre elementos UML y características). La propuesta más reciente de Czarnecki y Antkiewicz, sí permite que cada característica opcional se refleje en una o varias partes de un diagrama, pero de nuevo es necesario introducir elementos auxiliares en el meta-modelo de UML.

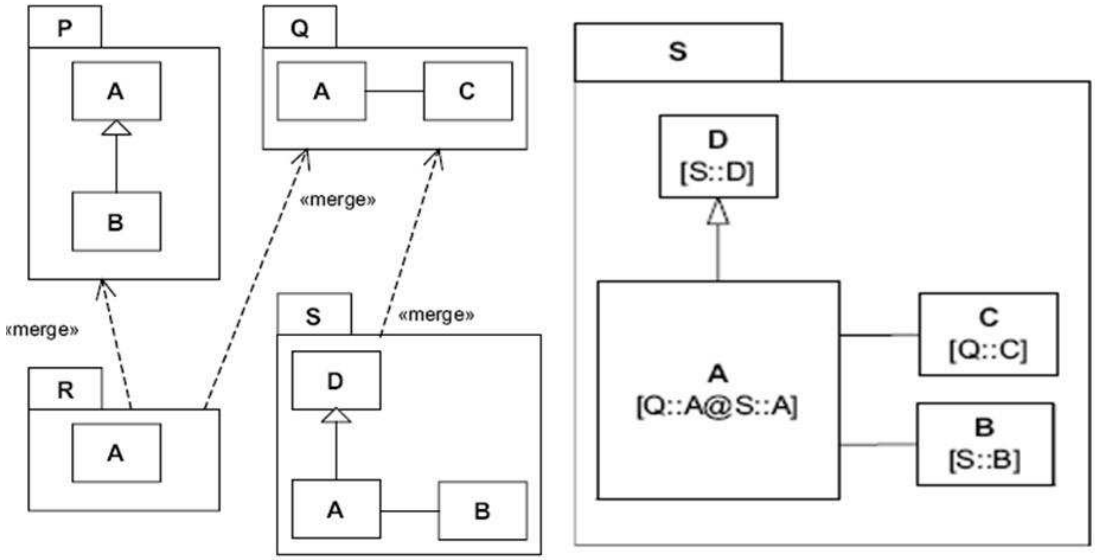
Como resultado del estudio de los puntos fuertes y débiles de estas propuestas, podemos establecer un conjunto mínimo de requisitos que debe cumplir una técnica útil de representación, y gestión de la variabilidad en el nivel de diseño de una línea de productos software:

1. Localizar en un solo punto del modelo de diseño todas las variaciones que origina cada característica opcional, de forma que se mantenga una correspondencia uno a uno y se facilite la gestión de la trazabilidad.
2. Separar la variabilidad originada en el nivel de la línea de productos de la variabilidad intrínseca en el nivel de las aplicaciones concretas, eliminando ambigüedades.
3. Mantener inalterado el meta-modelo de UML para eliminar la barrera de entrada a este paradigma para cualquier desarrollador además de permitir el uso de herramientas CASE convencionales.
4. Conectar con los modelos de implementación para acercarnos al ideal de desarrollo sin costuras, propugnado por el paradigma de orientación a objetos pero desechado muchas veces por irrealizable

2.5. Concepto de Paquetes. “Package Merge”, solución en el diseño

Para la correcta gestión de la variabilidad, el grupo GIRO aboga por expresar la variabilidad en los modelos UML utilizando el concepto de combinación de paquetes (o “package merge”), presente en el metamodelo de infraestructura de UML 2 y utilizado de forma exhaustiva en la definición misma de UML 2. El mecanismo “package merge” consiste fundamentalmente en añadir detalles de forma incremental (ver Figura 2, extraída del documento oficial de UML 2).

Según la especificación UML 2, se define como una relación entre dos paquetes que indica que los contenidos de ambos se combinan. Es similar a la generalización y se utiliza cuando elementos en distintos paquetes tienen el mismo nombre y representan el mismo concepto, comenzando por una base común.



2.6: Ejemplo del mecanismo "package merge"

Dicho concepto se extiende incrementalmente en cada paquete añadido. Seleccionando los paquetes deseados es posible obtener una definición a la medida de entre todas las posibles. Aunque nos interesan sobre todo los diagramas de clases, el mecanismo se puede extender a cualquier diagrama de UML, en particular los de casos de uso y los de secuencia.

Evidentemente, las reglas que establece la especificación UML2 son muy estrictas para evitar inconsistencias. Por ejemplo, no puede haber ciclos, las multiplicidades resultantes son las menos restrictivas de entre las posibles, o las operaciones deben conformar en número, orden y tipo de parámetros.

Los conceptos que maneja UML 2 son fundamentalmente:

- **Paquete combinable** (primer operando), **paquete receptor** (segundo operando) y **paquete resultado**, que conceptualmente es el resultado de la combinación y que en el modelo coincide con el paquete receptor (interpretando que se observa después de ejecutada la operación).
- Elemento combinable (perteneciente al paquete combinable), elemento receptor (perteneciente al paquete receptor y que si presenta coincidencia con un elemento combinable se fusiona con él) y elemento resultado, que conceptualmente es el resultado de la combinación de dos elementos. Aquellos elementos que no coincidan en los dos paquetes que se combinan se incorporan sin modificaciones al paquete resultado.

La filosofía general es que el elemento resultante tiene que ser al menos tan capaz como el original antes de la combinación. Este mecanismo nos permite establecer una trazabilidad clara entre los modelos de características y los artefactos UML.

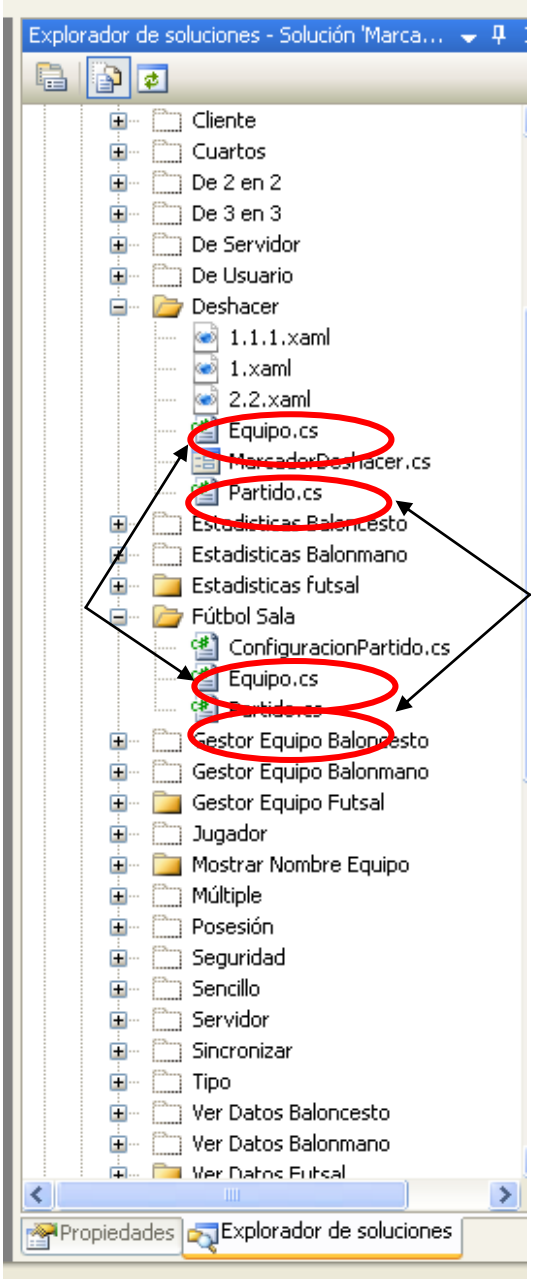
2.6. Clases parciales, solución en la implementación.

Para extender la trazabilidad hasta los modelos de implementación utilizamos el concepto de clases parciales. La utilización de "mixins" o clases parciales representa una alternativa a la herencia múltiple y una manera de manejar la variabilidad relacionada con aspectos. La intención es mantener la correspondencia uno a uno no sólo entre características y paquetes de diseño sino también con la estructura del código.

En este modelo de desarrollo del grupo GIRO se utiliza el mecanismo de clases parciales de C# para implementar la línea de productos dentro de la estructura de soluciones y paquetes de la plataforma MS Visual Studio 2008. Si el framework que implementa la arquitectura de la línea de productos está organizado en paquetes de clases parciales (un paquete base y tantos paquetes auxiliares como variaciones existen), para derivar una aplicación concreta basta con importar o referenciar los paquetes que se correspondan directamente con la configuración elegida en el modelo de características.

Para diseñar este diagrama de características se ha utilizado una herramienta diseñada por el grupo GIRO. Dicha herramienta se llama Feature Modelling Tool (FMT). Es una herramienta que permite modelar gráficamente y configurar modelos de características (features) dentro del entorno de desarrollo de Visual Studio. Se añade como plugin al IDE Visual Studio, y se puede comenzar a utilizar. En la figura de la página anterior (figura 2.6), podemos ver el significado de cada uno de sus elementos.

En la Figura 2-7 se aprecia una parte de la línea de producto, implementada utilizando paquetes de C#. Estos paquetes opcionales pueden añadirse o no al proyecto que representa una aplicación concreta de la línea de producto desde el explorador de soluciones.



2.7: Clases parciales en distintos paquetes

Capítulo 3

INTRODUCCIÓN A LOS DISPOSITIVOS MÓVILES

3.1. Aspectos generales de un dispositivo móvil

Los dispositivos móviles son aquellos dispositivos suficientemente pequeños para ser transportados y usados durante su transporte. Normalmente se sincronizan con un sistema de sobremesa para actualizar aplicaciones y datos.

Las características básicas de este tipo de dispositivos son:

- Son aparatos pequeños.
- Tienen capacidad de procesamiento.
- Pueden conectarse a la red.
- Tienen memoria limitada.
- Están diseñados específicamente para una función, pero que pueden llevar a cabo otras más generales.
- Normalmente se asocian al uso individual de una persona, tanto en posesión como en operación, el cual puede adaptarlos a su gusto.
- La mayoría de estos aparatos pueden ser transportados en el bolsillo del propietario. Otros están integrados dentro de otros mayores, controlando su funcionalidad.

Algunas de las características que hacen que estos dispositivos sean diferentes de los ordenadores de sobremesa son los siguientes:

- Funcionalidad limitada.
- No necesariamente extensible y actualizable.
- Menos complicado en su manejo.
- Fácil de aprender su operación.
- No se requieren usuarios expertos.

Algunos de estos dispositivos son los siguientes:

- Paginadores.
- Comunicadores de bolsillo.
- Teléfonos con pantalla para Internet (Internet Screen Phones):
- Sistemas de navegación de automóviles.
- Sistemas de entretenimiento.

- Sistemas de revisión e Internet (WebTV).
- Teléfonos móviles.
- Organizadores y asistentes personales digitales (Personal Digital Assistant o PDA)

3.2. Personal Digital Assistant (PDA)

Una PDA (Personal Digital Assistant o Ayudante personal digital) es un ordenador de mano originalmente diseñado como agenda electrónica. Hoy en día se puede usar como un ordenador doméstico (ver películas, crear documentos, navegar por Internet).

Una PDA puede funcionar como teléfono móvil, fax, explorador de Internet, organizador personal, GPS, etc.

La mayoría de PDAs empezaron a usarse con una especie de bolígrafo en lugar de teclado, por lo que incorporaban reconocimiento de escritura a mano. Hoy en día los PDAs pueden tener teclado y/o reconocimiento de escritura. Algunos PDAs pueden incluso reaccionar a la voz, mediante tecnologías de reconocimiento de voz.

Una PDA es similar a un PC. Tienen una placa base, un procesador, memoria RAM, memoria permanente (ROM). También tienen sistema operativo en el cual se ejecutan los programas que deseemos (juegos, agendas, hojas de cálculo, navegadores Web…). La forma de comunicarse con el PDA es la propia pantalla, que en algunos modelos es táctil. El lápiz que incorpora el aparato se utiliza a modo de ratón. Para escribir podemos utilizar un teclado virtual que se muestra en la pantalla o bien hacer uso de la característica de reconocimiento de escritura.

3.2.1. Elementos básicos de una PDA.

Los elementos principales que forman una PDA son los siguientes:

- **Procesador:** Los ordenadores de bolsillo suelen incluir procesadores de arquitectura diferente a los que encontramos en nuestros ordenadores personales, ya que han de tener un consumo muy reducido y adecuarse a las características físicas de los PDA. Por ejemplo la PDA AcerN30, utilizada para la realización del proyecto, tiene un procesador Samsung 2410 de 266 Mhz, a diferencia de los PC actuales que pueden tener hasta 3,5Ghz.

- **Software:** La funcionalidad de los ordenadores de bolsillo sólo está limitada por la de las aplicaciones que en ellos se instalen. Lo habitual es que los ordenadores de esta clase incorporen de serie las aplicaciones más comunes: agenda de contactos, calendario, notas, gestor de correo electrónico, etc. Más adelante, podremos añadir cualquier otro programa que adquiramos o descarguemos y que resuelva nuestras necesidades específicas.

- **Sistema operativo**: Los dos sistemas operativos más extendidos del mercado son Palm OS, de Palm Inc., y Windows Mobile, de Microsoft. Existen muchos otros pero la mejor opción sería elegir con un dispositivo que funcione con uno de estos dos, pues son los que tienen a su disposición un mayor número de actualizaciones y programas.
    - Palm OS**:** este sistema operativo se encuentra en los ordenadores comercializados por Palm y Handspring, entre otros. Es muy sencillo de utilizar.
    - Windows Mobile**:** la alternativa de Microsoft cuenta con una mayor difusión, y la variedad de aparatos basados en ella es mucho más amplia.

- **Conectividad**: El ordenador de bolsillo intercambia datos con el exterior a través de los puertos de comunicación. Existen dos tipos: por cable e inalámbrico. Entre los inalámbricos, existen tres vías de conectividad, infrarrojos, Bluetooth y Wifi. Los infrarrojos presentan más limitaciones que los otros dos, como pueden ser la falta de rango y la necesidad de mantener una línea visual entre los dispositivos.

  Actualmente, un mismo dispositivo puede incorporar dos o más sistemas simultáneamente.

- **Memoria**: La memoria de estos ordenadores es relativamente reducida. Los modelos más recientes están equipados con hasta 32 GB, lo que aporta una gran comodidad a la hora de cargar varias aplicaciones, sobre todo las de multimedia (ficheros MP3 o Internet).

  Por otro lado, la mayoría de los dispositivos disponen de un espacio para que se pueda añadir tarjetas de memoria, aumentando así la capacidad del dispositivo.

- **Ranuras de expansión**: para los usuarios que demandan mayores capacidades en movilidad es importante que la PDA disponga de ranura de expansión para los diferentes modelos de tarjetas. Con ellas se solucionan las limitaciones de los dispositivos, en cuanto a tamaño.

- **Teclado o lápiz**: La mayoría de PDAs no tienen un verdadero teclado. Para introducir texto existen dos métodos: el primero, un pequeño teclado que aparece en la pantalla, cuyas letras se seleccionan con el lápiz incorporado. El segundo, un sistema de reconocimiento de letras, cifras y caracteres especiales (signos de puntuación, operaciones aritméticas, etc.). Además, la mayoría de estos ordenadores soporta la conexión de un pequeño teclado externo, que nos facilitará la entrada de datos.

- **Pantalla:** El tamaño de la pantalla es bastante reducido. La mayor parte de los modelos sin teclado presentan una definición comprendida entre 160x160 y 320x240 píxeles, mientras que los modelos con teclado son más ricos en resolución, que puede llegar hasta los 640x240 píxeles. La más habitual es la ya comentada resolución de 320x240 píxeles.

- **Batería:** Un ordenador de bolsillo puede funcionar gracias a su batería interna o a las pilas, que se pueden cambiar por baterías recargables. En algunas PDA, una pila botón

asegura la conservación de los datos en la memoria RAM cuando la alimentación principal se agota.

- **Accesorios integrados:**

  - Altavoces, micrófono y toma de auriculares**:** no todos los modelos los llevan, pero la tendencia es que los vayan incorporando. Hace algunos años el micrófono se utilizaba para grabar datos que luego eran recuperados gracias al altavoz. Con la llegada del MP3, apareció la toma de auriculares.
  - Base de conexión**:** algunos ordenadores de bolsillo se entregan con el llamado Cradle, un dispositivo unido a la PDA, por lo que no es necesario enchufarlo cada vez que hay que hacer una sincronización o un intercambio de datos.
  - Lector biométrico**:** Algún modelo puede tener un sistema de seguridad para evitar el acceso a los datos contenidos en el ordenador por personas "no autorizadas". Básicamente es un pequeño lector de huellas dactilares que actúa de igual forma que la tradicional protección por contraseña.

3.2.2. Términos para referirse a los dispositivos móviles

Término	Descripción
PDA (Personal Digital Assistant) *Asistente Personal* *Agenda Electrónica*	Nombre genérico para cualquier dispositivo portátil de reducido tamaño.
Ordenador de mano	Lo mismo que PDA con la diferencia que aquí se suponen capacidades avanzadas de programación. Quedan excluidas las agendas sin capacidad de programación, como las de gama baja.
Handheld *HPD*	PDA con teclado tipo QWRTY incorporado, de más peso y con más prestaciones que un PPC
Palmhedl *Palm-size PC* *PPC* *Pocketpc*	PDA sin teclado. PocketPc es el nombre de uno de los sistemas operativos, pero actualmente se usa también como sinónimo de ese tipo de dispositivos.
Pocketpc	PDA con S.O. windows mobile Actualmente se usa este nombre asimismo con nombre genérico para este tipo de dispositivos.

3.1: Términos para referirse a los dispositivos móviles

3.3. Tipos de Personal Digital Assistant o P.D.A.

Hay muchos tipos diferentes, pero básicamente están divididos en 4 grandes familias.

3.3.1. Newton

3.2 Macintosh Newton

Uno de los primeros PDA fue el modelo Newton Message Pad (NMP) de Apple Computers, que apareció en el mercado en agosto de 1993. La Newton traía integrado un procesador ARM 610 de 20 MHz, tenía una capacidad máxima en RAM de 64 Kbytes, una resolución de 336 x 240 píxeles y con dimensiones de 19x11.4 centímetros. La Newton era muy grande, cara y complicada de operar, y su programa de reconocimiento de escritura tenía muchas limitaciones.

3.3.2. Palm

Una Palm es un dispositivo móvil, que contiene diferentes utilidades. Algunas de ellas vienen directamente de fábrica, como pueden ser, agenda, calculadora, libreta de direcciones y un anotador de ideas y otras pueden ser bajadas desde Internet.

Esta minicomputadora cuenta con diferentes elementos:

- Tamaño reducido
- Pantalla monocromática
- Mucha autonomía, semanas de uso continuado sin recargarlas.
- Microprocesador *:* capaz de resolver cálculos complejos
- Memoria: encargada de almacenar los datos.
- Lápiz: se utiliza para introducir los datos. Para meter letras se usa la parte inferior de la pantalla, y se escribía directamente las letras a mano alzada usando el método de escritura Graffiti.
- Dispositivo infrarrojo*:* sirve para intercambiar información con otras Palms (sin ningún tipo de cables) o con otros dispositivos de este tipo, como impresoras, cámaras, etc.
- Cradle: conecta la Palm a cualquier PC, permite bajar los correos electrónicos contenidos en la computadora. Hay Palms que vienen preparadas para conectarse a Internet sin necesidad de módem, en las cuales la conexión se realiza mediante una antena (Palm VII).

Además de brindarnos una vía de conexión con nuestras PC, los Cradles hacen una copia de seguridad de todo lo que tengas almacenado en la Palm, dentro de la PC.

- Sistema operativo PALM OS, que hace un uso muy eficiente de las limitadas prestaciones y que va por la versión 6. Ha habido una gran evolución entre la versión 5 y la 6 para poder hacer un uso más eficiente de los nuevos procesadores.
- Versiones compatibles entre sí.

3.3: Palm V

3.4: Palm tungsten

3.3.3. Epoc-Symbian

Este sistema operativo, el *EPOC*, fue inventado y desarrollado por la empresa inglesa *Psion*, y son típicos sus *handheld* con un auténtico teclado, pero con una pantalla monocromática que le permitía una gran autonomía. Las prestaciones eran mayores que las *PALM*, con muchísima mejor pantalla, y con un tamaño y peso muy reducido, y con mucha más autonomía que las *WINDOWS CE*. Hace poco hicieron una alianza entre unas cuantas empresas, como *Psion*, *Ericsson*, *Nokia*, y *Psion* dejó de fabricar sus propios *handhelds*, centrándose en crear el sistema operativo Symbian, tomando como base el S.O. *EPOC*.

Las características principales son:

- Necesita poco espacio.
- Usa escasos recursos de memoria de forma dinámica.
- Administra de forma eficiente la energía.
- Soporta en tiempo real los protocolos de comunicación y telefonía.
- Más tolerante a fallos que los PCs.
- Es actualizable.

3.5 Psion Revo

3.6 Psion 5MX

3.3.4. Windows Mobile

Windows Mobile es una versión del sistema operativo Windows diseñado para pequeños dispositivos, como en el caso de los Handhelds. La interfaz gráfica del Windows Mobile es muy similar a la del Windows XP.

El Windows Mobile tiene sus propias APIs para desarrollo, y necesita sus propios drivers para el hardware con el cual va a interactuar. Windows Mobile no es un sinónimo de Windows XP en forma pequeña, incrustada o modular.

Las principales características son:

- Tecnologías sin cable (wifi, bluetooth, IrDA, 802.1x y EAP, IPv6, OBEX, MediaSense, RTC/SIP) que permiten al dispositivo conectarse a infraestructuras ya existentes
- Amplia gama de arquitecturas (ARM, MIPS, SHx, x86)
- Tiempo real
- Soporte para lenguajes de Internet y XML
- Tecnología de emulación
- Sencillo entorno de desarrollo del Kernel y de aplicaciones

PARTE 2

DESARROLLO DE LA LÍNEA DE PRODUCTOS

Capítulo 4

ANÁLISIS DE LA LÍNEA DE PRODUCTOS

El análisis consiste en la comprensión y modelado de la aplicación y del dominio en el cual funciona. La entrada inicial de la fase de análisis es una descripción del problema que hay que resolver y proporciona una visión general conceptual del sistema propuesto. La salida del análisis es un modelo formal que captura los aspectos esenciales del sistema.

En este capítulo nos vamos a centrar en los objetivos principales de la aplicación y de los límites de la misma. Trataremos de identificar las necesidades reales del usuario final de la aplicación mediante los requisitos. La identificación de los requisitos es una parte fundamental debido a que si no son correctos puede conllevar a problemas en la propia utilización de la aplicación.

4.1 Entrevistas

Los requisitos que debe cumplir la aplicación en sí, vienen dados por una serie de entrevistas que hemos llevado a cabo con nuestro tutor de proyecto. Los requisitos han ido siendo identificados a partir de dichas entrevistas y con ello se han establecido prioridades de los mismos en concordancia con las necesidades de los usuarios finales y las funcionalidades y restricciones de la aplicación a diseñar.

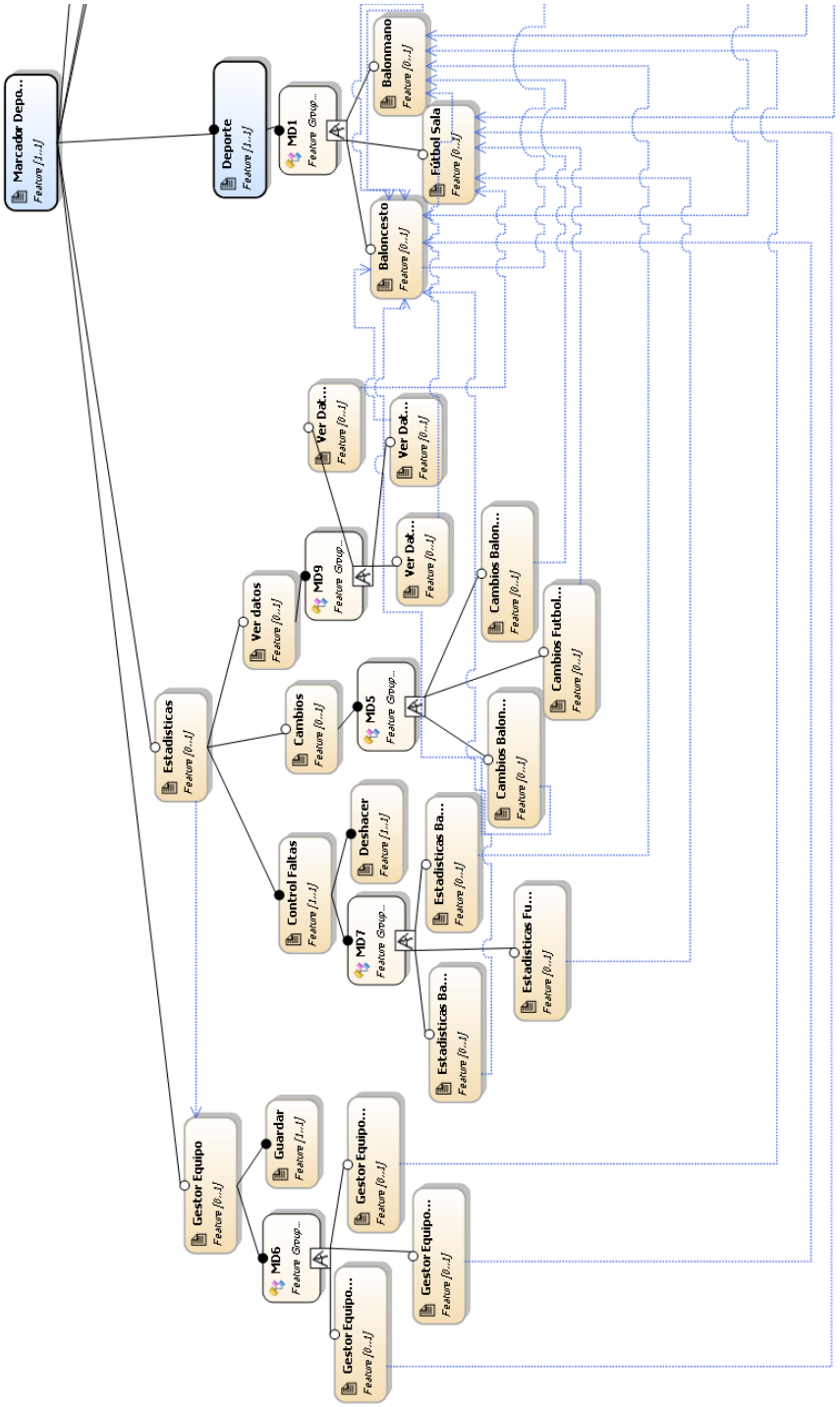
4.2 Modelo de características de una línea de marcadores deportivos

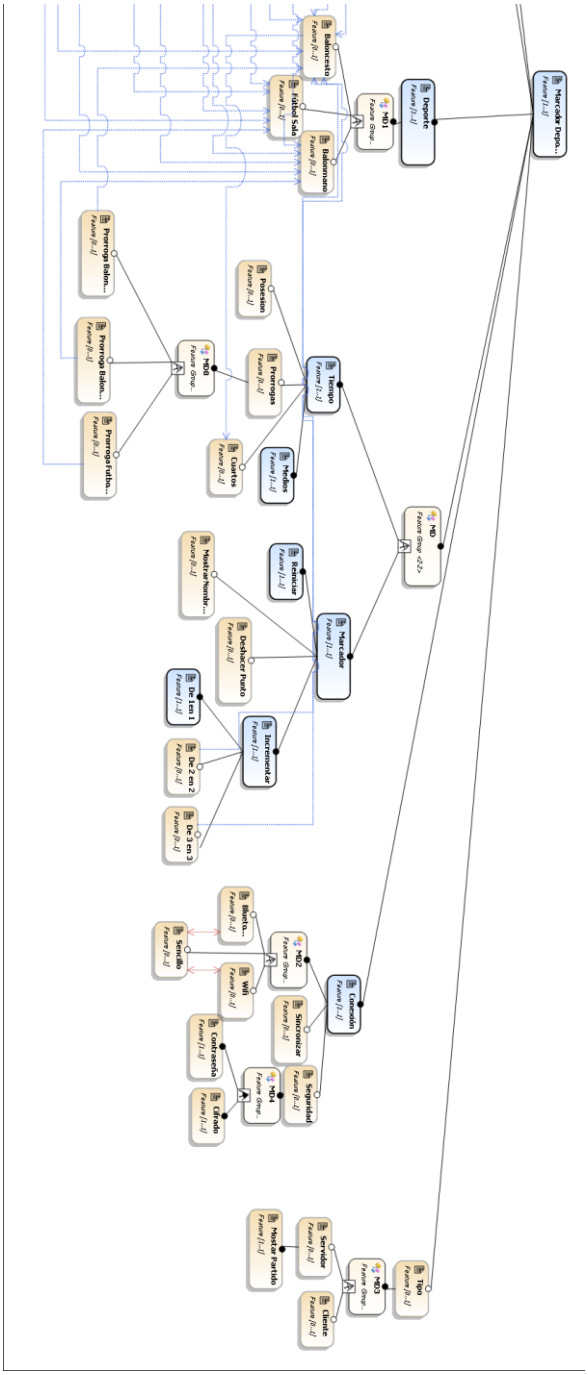
La diferencia entre posibles productos de la línea de productos está en las "características". Una característica (feature) puede definirse como algo relevante para el usuario del proyecto que represente un aspecto de un producto software.

Los requisitos de la línea de productos son la suma de los requisitos de todos los miembros que componen la línea. Así pues, una característica es un requisito que es proporcionado por uno o más miembros de la línea de productos. En particular, las características se usan para diferenciar los miembros de dicha línea y así determinar y definir la funcionalidad común y variable de una línea de productos software.

En el modelo de características a tratar se detallan todas las características de los marcadores deportivos, diferenciando las que son obligatorias o esenciales para un marcador preliminar. A partir de ahí se le añaden más características que permiten ampliar a nuestro gusto el marcador deportivo en sí. Es decir, obligatorias como llevar el conteo de puntos, del tiempo y de las partes. Luego las opcionales, como por ejemplo que se puedan ir sumando puntos de 2 en 2 o de 3 en 3, quien anota las canastas, el número de faltas cometidas, etc.

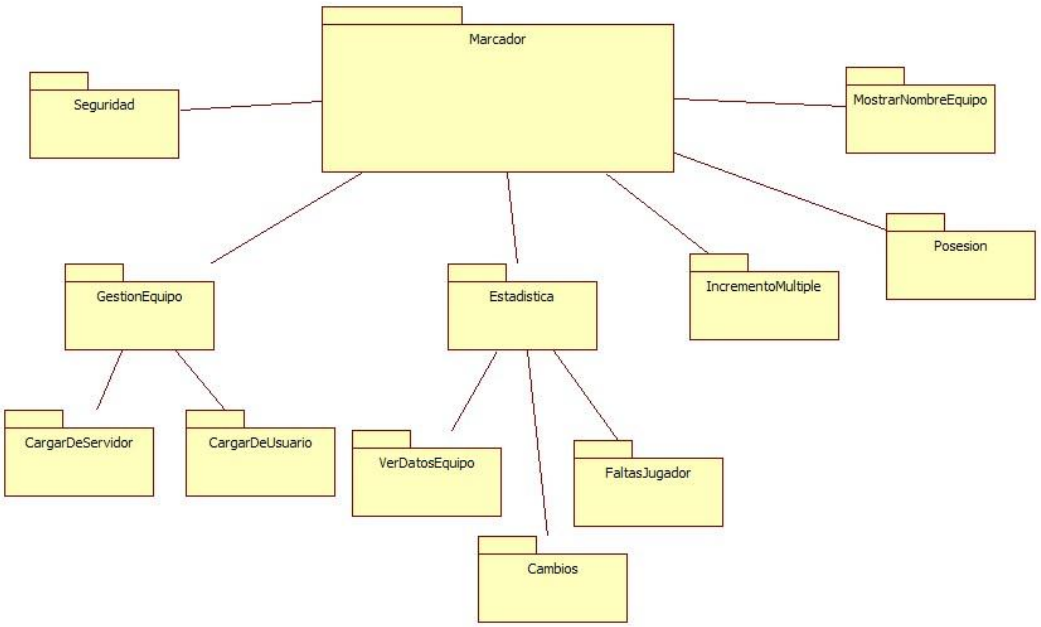
La siguiente figura muestra el modelo de características de la línea de marcadores deportivos, donde podemos ver representado tanto los paquetes obligatorios como los opcionales (los símbolos del diagrama de características están explicados en la figura 2.5.


Marcador Depo...
Feature [1...1]
Deporte
Feature [1...1]
MD1
Feature Group...
Balonmano
Feature [0...1]
Fútbol Sala
Feature [0...1]
Baloncesto
Feature [0...1]
Ver Dat...
Feature [0...1]
Ver Dat...
Feature [0...1]
MD9
Feature Group...
Ver Dat...
Feature [0...1]
Cambios Balon...
Feature [0...1]
Cambios Futbol...
Feature [0...1]
Ver datos
Feature [0...1]
Estadísticas
Feature [0...1]
Cambios
Feature [0...1]
MD5
Feature Group...
Cambios Balon...
Feature [0...1]
Deshacer
Feature [1...1]
Control Faltas
Feature [1...1]
Estadísticas Ba...
Feature [0...1]
MD7
Feature Group...
Estadísticas Fu...
Feature [0...1]
Estadísticas Ba...
Feature [0...1]
Guardar
Feature [1...1]
Gestor Equipo...
Feature [0...1]
Gestor Equipo
Feature [0...1]
MD6
Feature Group...
Gestor Equipo...
Feature [0...1]
Gestor Equipo...
Feature [0...1]



4.1: Diagrama de características

Este diagrama de características dibujado en la figura 4.1 puede traducirse en un diagrama de paquetes, siendo los paquetes las partes opcionales de la línea de productos. Así pues, nuestro sistema puede dividirse en los siguientes paquetes:



4.2 Diagrama de paquetes

4.3 Catálogo de requisitos

Un requisito es una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado. También se aplica a las condiciones que debe cumplir o poseer un sistema para satisfacer un contrato, una norma o una especificación. Por tanto, el conjunto de requisitos determinan lo que hará el sistema y definen las restricciones sobre su operación e implementación.

La especificación de los requisitos debe ser completa, no tener definiciones contradictorias, y debe ser expresada con exactitud, ya que si hay ambigüedad se corre el riesgo de que sean interpretados de forma diferente por los usuarios y por los desarrolladores.

4.3.1 Requisitos funcionales

Estos requisitos constituyen la declaración de los servicios que el sistema debe proporcionar, cómo debe reaccionar ante una entrada particular y cómo se debe comportar ante situaciones particulares. En resumen, describen la funcionalidad del sistema y de qué forma va a utilizarlo el usuario.

A continuación se muestran los requisitos funcionales de la aplicación separados por paquetes, debido a que se trata de una línea de productos:

FRQ-0001	Controlar marcador
Descripción	El sistema deberá *controlar el marcador, tanto la puntuación de los equipos como los tiempos de las partes.*

FRQ-0002	Control tiempo
Descripción	El sistema deberá *controlar el tiempo de las partes. Reiniciando y parando cuando se le indique.*

FRQ-0005	Guardar datos del acta
Descripción	El sistema deberá *guardar los datos del acta del partido. En él constará la puntuación final del partido.*

FRQ-0012	Reinicio del partido
Descripción	El sistema deberá *permitir reiniciar el partido en cualquier momento*

4.3 Requisitos funcionales del paquete Marcador

FRQ-0003	Control de los jugadores
Descripción	El sistema deberá *controlar los jugadores, teniendo en cuenta tanto su nombre y apellidos como su número de jugador.*

FRQ-0006	Control de faltas
Descripción	El sistema deberá *llevar un control de faltas producidas durante el partido de ambos equipos.*

FRQ-0007	Seguimiento de anotaciones por jugador
Descripción	El sistema deberá *llevar en cuenta los puntos marcados por cada jugador, para ello necesitará el control de jugadores.*

4.4 Requisitos funcionales del paquete Estadísticas

FRQ-0008	Controlar datos de los equipos
Descripción	El sistema deberá *controlar los datos de los equipos participantes.*

4.5 Requisitos funcionales del paquete Gestor Equipo

FRQ-0004	Sincronización entre los usuarios
Descripción	El sistema deberá *sincronizar tanto la puntuación de los marcadores como el tiempo de las partes entre los usuarios de la aplicación.*

FRQ-0011	Conectar al Servidor
Descripción	El sistema deberá *conectarse al servidor al iniciar el partido.*

4.6 Requisitos funcionales del paquete Conexiones

4.3.2 Requisitos no funcionales

Los requisitos no funcionales nos especifican las propiedades de la aplicación que tienen que ver con el rendimiento, con la velocidad, el uso de la memoria, la fiabilidad, etc. Los requisitos no funcionales imponen restricciones a los funcionales.

NFR-0001	Plataforma
Descripción	El sistema deberá *estar construido en .NET y funcionar sobre Windows Mobile.*

NFR-0002	Alta flexibilidad
Descripción	El sistema deberá *ser altamente flexible ante nuevas funcionalidades añadidas mediante módulos o paquetes*

NFR-0003	Independencia entre paquetes
Descripción	El sistema deberá *garantizar la independencia total entre los distintos paquetes opcionales*

NFR-0004	Tiempo de reacción
Descripción	El sistema deberá *reaccionar a cualquier evento en tiempo inferior a 30 segundos.*

4.7 Requisitos no funcionales

4.4 Modelo de casos de uso

Utilizamos los casos de uso para describir el uso del sistema y cómo los usuarios interaccionan con nuestra aplicación. En los casos de uso se detallan paso a paso las acciones que realiza el sistema y que producen un resultado para un actor en particular.

4.4.1 Actores

Los actores son todas aquellas entidades externas que puedan interactuar con la aplicación. Son, por tanto, los distintos papeles que pueden tener los usuarios. Se debe tener en cuenta que el propio sistema puede ser un actor.

En nuestra aplicación los actores son los siguientes:

ACT-0001	Usuario
Descripción	Este actor representa *a la persona que va a utilizar la aplicación en modo máster*
Comentarios	Es quien pondrá en marcha la aplicación y el resto se unirán a ella

ACT-0002	Servidor
Descripción	Este actor representa *al sistema que contiene la aplicación*
Comentarios	Ninguno

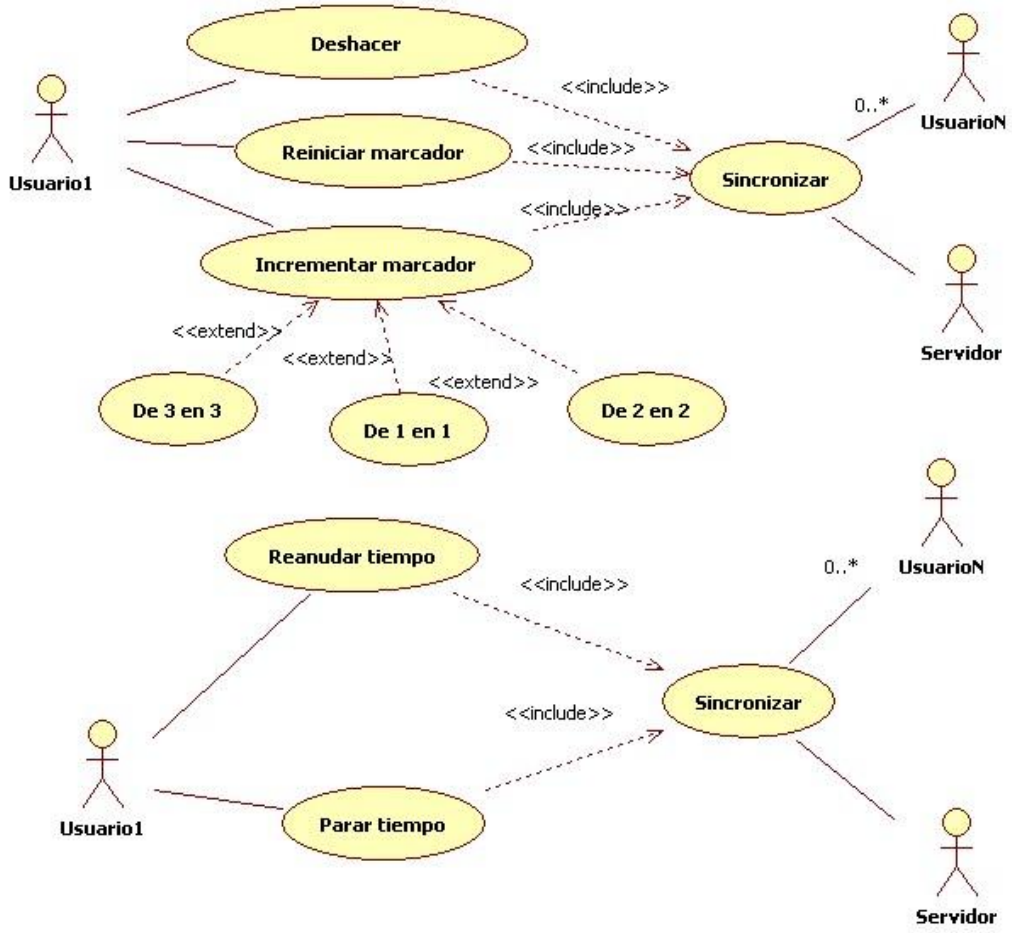
ACT-0003	Usuario_N
Descripción	Este actor representa *a la persona que va a utilizar la aplicación en modo esclavo*
Comentarios	Tipo de usuario que se conectará cuando la aplicación ya esté en funcionamiento

ACT-0004	Temporizador
Descripción	Este actor representa *al temporizador del sistema que se actualiza cada segundo*
Comentarios	Ninguno

4.8 Actores del sistema

4.4.2 Especificación de los casos de uso

En este apartado describiremos en detalle los casos de uso, pudiendo así observar cómo debería funcionar nuestro sistema, y reflejando la interacción entre usuario y sistema. Se podrán analizar las excepciones que tienen nuestros casos de uso. Presentaremos los casos de usos por paquetes, para un mejor entendimiento de los mismos.



4.9 Diagrama de casos de uso del paquete Marcador

UC-0004	**Deshacer**	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *se pulse el botón deshacer.*	
Precondición	Ninguna.	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) *selecciona el marcador (local o visitante) y pulsa el botón deshacer del marcador*
	2	El sistema *vuelve el marcador seleccionado a las unidades que constaban antes del último incremento y notifica de ello al servidor*
	3	El actor Servidor (ACT-0002) *notifica al resto de usuarios que el marcador ha sido restablecido*
	4	El actor Usuario N (ACT-0003) *recibe la notificación del servidor*
	5	El sistema *finaliza el caso de uso.*
Postcondición	El marcador quedara restablecido	

UC-0006	**Reiniciar marcador**	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *se quiera iniciar el control de un nuevo partido.*	
Precondición	Alguno de los marcadores no está a cero ó el tiempo no está en inicial.	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) *pulsa el botón de reiniciar el marcador*
	2	El sistema *pone a cero el marcador y lo notifica al servidor*
	3	El actor Servidor (ACT-0002) *notifica al resto de usuarios que el marcador se ha puesto a cero*
	4	El actor Usuario N (ACT-0003) *recibe la notificación del servidor*
	5	El sistema *finaliza el caso de uso*
Postcondición	El marcador, tanto para local como para visitante, se pondrá a cero. El tiempo también se reiniciará al estado inicial.	

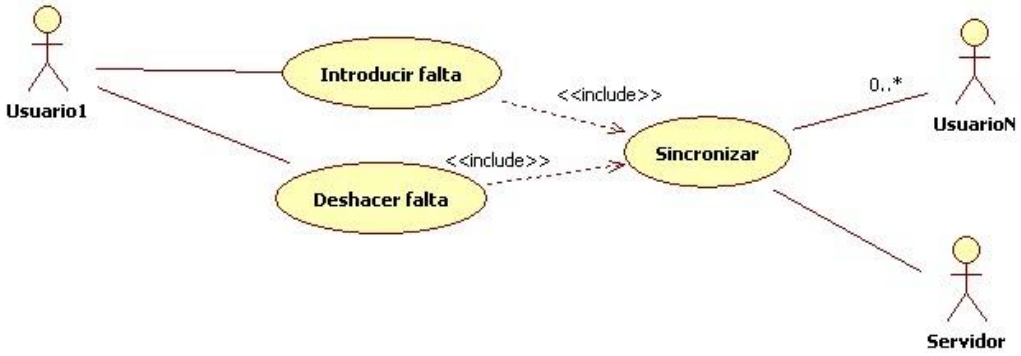
UC-0005	**Incrementar marcador**	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *se pulse el botón de incrementar punto en alguno de los marcadores de los equipos participantes.*	
Precondición	Ninguna.	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) *selecciona el marcador (local o visitante) y pulsa el botón incrementar marcador*
	2	Si *elige incrementar de uno en uno,* se realiza el caso de uso De 1 en 1 (UC-0022)
	3	Si *elige incrementar de dos en dos,* se realiza el caso de uso De 2 en 2 (UC-0023)
	4	Si *elige incrementar de tres en tres,* se realiza el caso de uso De 3 en 3 (UC-0024)
	5	El actor Servidor (ACT-0002) *notifica al resto de usuarios que el marcador ha sido incrementado*
	6	El actor Usuario_N (ACT-0003) *recibe la notificación del servidor*
	7	El sistema *finaliza el caso de uso*
Postcondición	El marcador incrementará en una unidad el marcador correspondiente.	

UC-0022	**De 1 en 1**	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando quiera incrementar en 1 el marcador o durante la realización de los siguientes casos de uso: [UC-0005] Incrementar marcador	
Precondición	Ninguna	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) *pulsa el botón de incrementar el marcador (local/visitante) en una unidad.*
	2	El sistema *incrementa el marcador en una unidad y notifica de ello al servidor*
	3	El actor Servidor (ACT-0002) *notifica al resto de usuarios que el marcador ha sido incrementado en una unidad*
	4	El actor Usuario_N (ACT-0003) *recibe la notificación del servidor*
	5	El sistema *da por finalizado el caso de uso*
Postcondición	Ninguna.	

UC-0023	De 2 en 2	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando quiera incrementar en 2 el marcador o durante la realización de los siguientes casos de uso: [UC-0005] Incrementar marcador	
Precondición	Ninguna.	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) *pulsa el botón de incrementar el marcador (local/visitante) en dos unidades.*
	2	El sistema *incrementa el marcador en dos unidades y notifica de ello al servidor*
	3	El actor Servidor (ACT-0002) *notifica al resto de usuarios que el marcador ha sido incrementado en dos unidades.*
	4	El actor Usuario_N (ACT-0003) *recibe la notificación del servidor*
	5	El sistema *da por finalizado el caso de uso*
Postcondición	Ninguna.	

UC-0024	De 3 en 3	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando quiera incrementar en 3 el marcador o durante la realización de los siguientes casos de uso: [UC-0005] Incrementar marcador	
Precondición	Ninguna.	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) *pulsa el botón de incrementar el marcador (local/visitante) en tres unidades*
	2	El sistema *incrementa el marcador en tres unidades y notifica de ello al servidor*
	3	El actor Servidor (ACT-0002) *notifica al resto de usuarios que el marcador ha sido incrementado en tres unidades.*
	4	El actor Usuario_N (ACT-0003) *recibe la notificación del servidor*
	5	El sistema *da por finalizado el caso de uso*
Postcondición	Ninguna.	

4.10: Casos de uso del paquete Marcador

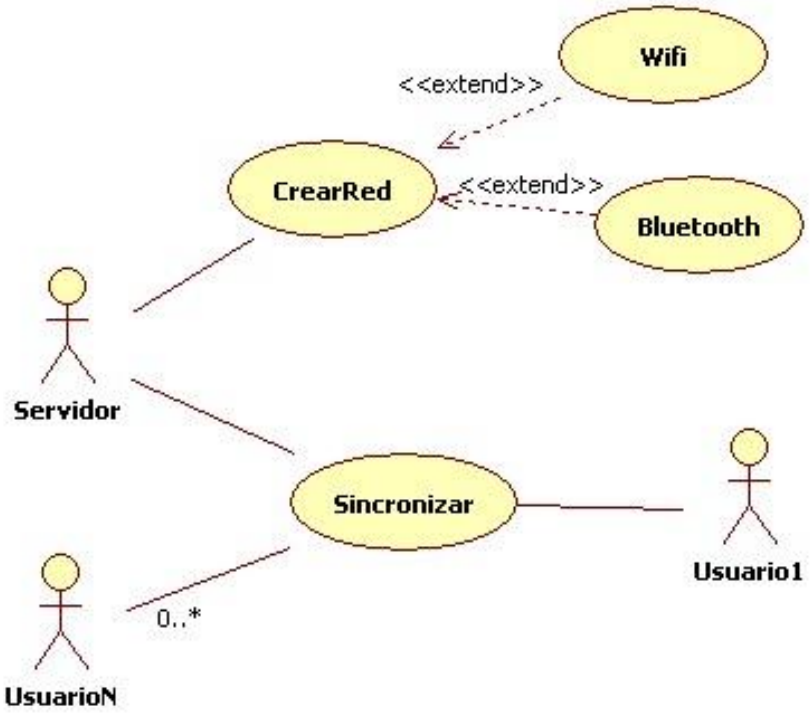


4.11: Diagrama de casos de uso del paquete Estadísticas

UC-0025	**Introducir falta**	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *se desee introducir una falta a un jugador.*	
Precondición	Ninguna.	
Secuencia normal	**Paso**	**Acción**
	1	El sistema *muestra todos los jugadores disponibles en el equipo*
	2	El actor Usuario (ACT-0001) *selecciona un jugador e introduce el tipo de falta.*
	3	El sistema *registra el tipo de falta cometida por el jugador.*
Postcondición	El jugador seleccionado tendrá una falta más.	

UC-0026	**Deshacer falta**	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *se desee quitar una falta a un jugador.*	
Precondición	El jugador seleccionado deberá de tener una falta añadida con anterioridad para deshacer la misma.	
Secuencia normal	**Paso**	**Acción**
	1	El sistema *muestra todos los jugadores disponibles en el equipo.*
	2	El actor Usuario (ACT-0001) *selecciona un jugador que tenga una falta anterior y lo selecciona.*
	3	El sistema *quita la falta al jugador seleccionado.*
Postcondición	El jugador contará con una falta menos tras realizar este paso.	

4.12: Casos de uso del paquete Estadísticas



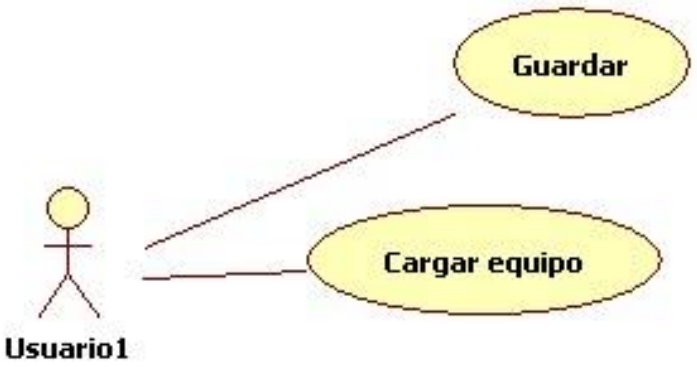
4.13: Diagrama de casos de uso del paquete Conexiones

UC-0013	Conexión Wifi	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el usuario selecciona conectarse por wifi a un servidor.*	
Precondición	El usuario tiene la posibilidad de conectarse por wifi. Existe un servidor al que conectarse.	
Secuencia normal	**Paso**	**Acción**
	1	El sistema *rastrea los alrededores y muestra los servidores activos*
	2	El actor Usuario (ACT-0001) *selecciona el servidor al cuál desea conectarse*
	3	Si *el servidor requiere una conexión segura,* el sistema *pide usuario y contraseña*
	4	Si *el servidor requiere una conexión segura,* el actor Usuario (ACT-0001) *introduce su usuario y contraseña*
	5	El sistema *establece una conexión con el servidor, muestra un mensaje de información y da por finalizado el caso de uso*
Postcondición	El usuario se encontrará conectado al servidor.	

UC-0014	Conexión Bluetooth	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el usuario selecciona conectarse por bluetooth a un servidor.*	
Precondición	El usuario tiene la posibilidad de conectarse por bluetooth. Existe un servidor al que conectarse	
Secuencia normal	**Paso**	**Acción**
	1	El sistema *rastrea los alrededores y muestra los servidores activos*
	2	El actor Usuario (ACT-0001) *selecciona el servidor al cuál desea conectarse*
	3	Si *el servidor necesita usuario y contraseña,* el sistema *pide usuario y contraseña*
	4	Si *el servidor necesita usuario y contraseña,* el actor Usuario (ACT-0001) *introduce su usuario y contraseña*
	5	El sistema *establece una conexión con el servidor, muestra un mensaje de información y da por finalizado el caso de uso*
Postcondición	El usuario se encontrará conectado al maestro.	

UC-0020	Crear red	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el usuario quiera crear una red.*	
Precondición	El usuario no estaba en otra red.	
Secuencia normal	Paso	Acción
	1	El actor Usuario (ACT-0001) *solicita al sistema crear una red.*
	2	El sistema *solicita al usuario que le proporciones los parámetros de la red que desea crear.*
	3	El actor Usuario (ACT-0001) *introduce los parámetros de la red.*
	4	El sistema *crea la red definida por el usuario.*
Postcondición	La red quedará creada.	

4.14: Casos de uso del paquete Conexiones

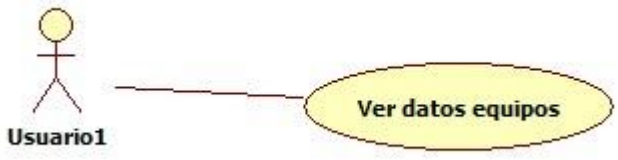


4.15: Diagrama de casos de uso del paquete Gestión Equipo

UC-0008	Cargar equipo	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el usuario pulse el botón de cargar equipo.*	
Precondición	No hay equipos cargados.	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) *pulsará el botón de cargar equipos*
	2	El sistema *notificará la servidor la petición del usuario*
	3	El actor Servidor (ACT-0002) *mostrará la usuario la lista de equipos para cargar*
	4	El actor Usuario (ACT-0001) *recibirá la lista de equipos y elegirá los equipos a cargar para el partido*
	5	El sistema *enviará los equipos escogidos por el usuario al propio servidor*
	6	El actor Servidor (ACT-0002) *recibirá la notificación de los equipos elegidos e informará de ello al resto de usuarios*
	7	El actor Usuario N (ACT-0003) *recibirá la notificación enviada por el servidor de los equipos cargados*
	8	El sistema *finaliza el caso de uso*
Postcondición	Los equipos cargados serán los que disputarán el partido.	
Excepciones	**Paso**	**Acción**
	5	Si *el usuario no ha escogido 2 equipos,* el sistema *le mostrará un mensaje de error y volverá al paso 4,* a continuación este caso de uso *continúa*

UC-0019	Guardar	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el usuario desee guardar un partido.*	
Precondición	El partido no estaba guardado con anterioridad.	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario (ACT-0001) *selecciona la opción de guardar un partido*
	2	El sistema *pregunta al usuario qué datos quiere guardar*
	3	El actor Usuario (ACT-0001) *elige una opción entre guardar solo el resultado o todas las incidencias*
	4	Si *el usuario eligió guardar solo los resultados,* el actor Servidor (ACT-0002) *guardará la información relativa al resultado entre los dos equipos*
	5	Si *el usuario eligió guardar todo,* el actor Servidor (ACT-0002) *guardará toda la información relativa a todas las incidencias del partido*
Postcondición	El partido quedará guardado.	
Excepciones	**Paso**	**Acción**
	4	Si *no se ha iniciado previamente un partido,* el sistema *informa con un mensaje de error,* a continuación este caso de uso *queda sin efecto*
	5	Si *no se ha iniciado previamente un partido,* el sistema *informa con un mensaje de error,* a continuación este caso de uso *queda sin efecto*

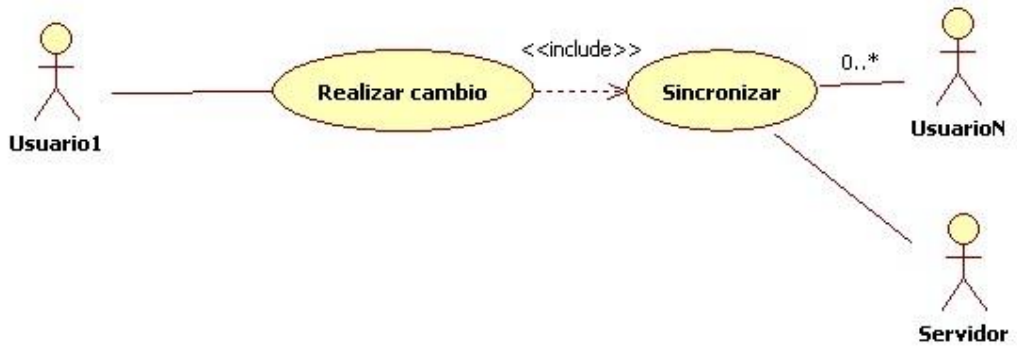
4.16: Casos de uso del paquete Gestión Equipo



4.17: Diagrama de casos de uso del paquete Ver Datos Equipo

UC-0017	Ver datos equipo	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el usuario desea ver los datos de un equipo.*	
Precondición	Ninguna.	
Secuencia normal	Paso	Acción
	1	El actor Usuario (ACT-0001) *selecciona la opción de ver los datos de un equipo*
	2	El sistema *le pide al usuario que introduzca el nombre del equipo del que desea ver sus datos*
	3	El actor Usuario (ACT-0001) *introduce el nombre del equipo*
	4	El sistema *muestra los datos del equipo seleccionado*
Postcondición	Se mostrarán los datos de equipo.	
Excepciones	Paso	Acción
	4	Si *el equipo introducido por el usuario no está disponible en la aplicación,* el sistema *informa del error y vuelve al paso 2.* a continuación este caso de uso *continúa*

4.18: Casos de uso del paquete Ver Datos Equipo



4.19: Diagrama de casos de uso del paquete Cambios

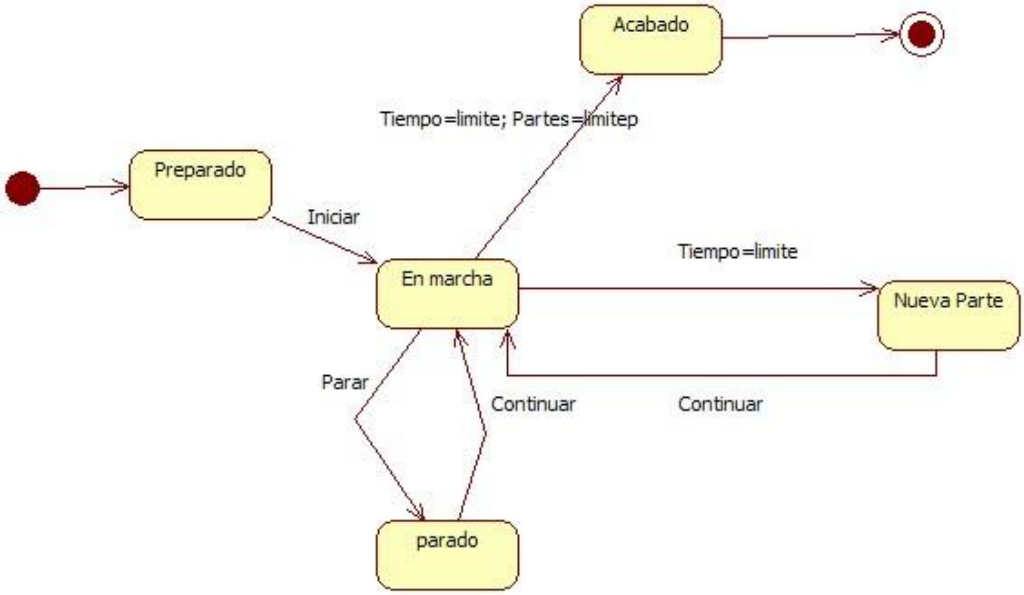
UC-0019	Guardar	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el usuario desee guardar un partido.*	
Precondición	El partido no estaba guardado con anterioridad.	
Secuencia normal	Paso	Acción
	1	El actor Usuario (ACT-0001) *selecciona la opción de guardar un partido*
	2	El sistema *pregunta al usuario qué datos quiere guardar*
	3	El actor Usuario (ACT-0001) *elige una opción entre guardar solo el resultado o todas las incidencias*
	4	Si *el usuario eligió guardar solo los resultados,* el actor Servidor (ACT-0002) *guardará la información relativa al resultado entre los dos equipos*
	5	Si *el usuario eligió guardar todo,* el actor Servidor (ACT-0002) *guardará toda la información relativa a todas las incidencias del partido*
Postcondición	El partido quedará guardado.	
Excepciones	Paso	Acción
	4	Si *no se ha iniciado previamente un partido,* el sistema *informa con un mensaje de error,* a continuación este caso de uso *queda sin efecto*
	5	Si *no se ha iniciado previamente un partido,* el sistema *informa con un mensaje de error,* a continuación este caso de uso *queda sin efecto*

4.20: Casos de uso del paquete Cambios

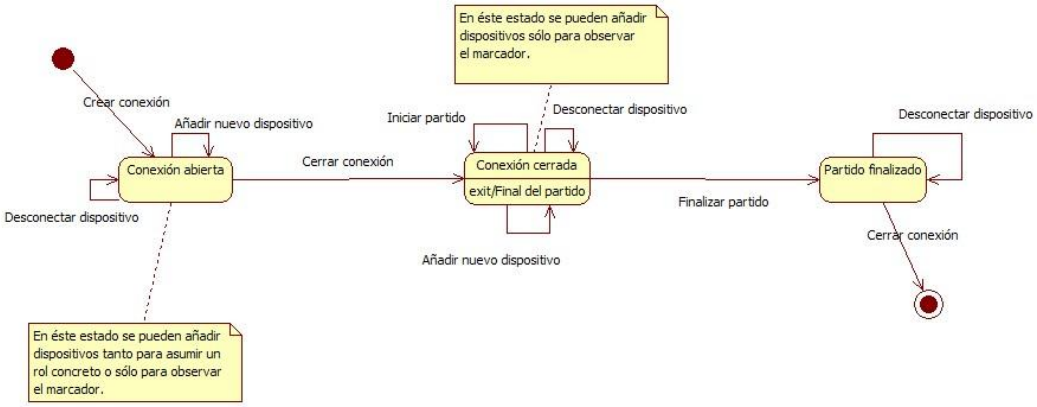
4.5 Diagramas de estado

En los diagramas de estado podemos ver el comportamiento de un objeto, es decir: todos aquellos estados por los que puede pasar un objeto en su vida, así como las transiciones que hacen pasar al objeto de uno a otro estado.

En esta fase del análisis nos centramos en representar los estados de un partido, así como los estados de la conexión. Ambos los pasamos a describir en los siguientes diagramas de estado:



4.21: Estados de un partido

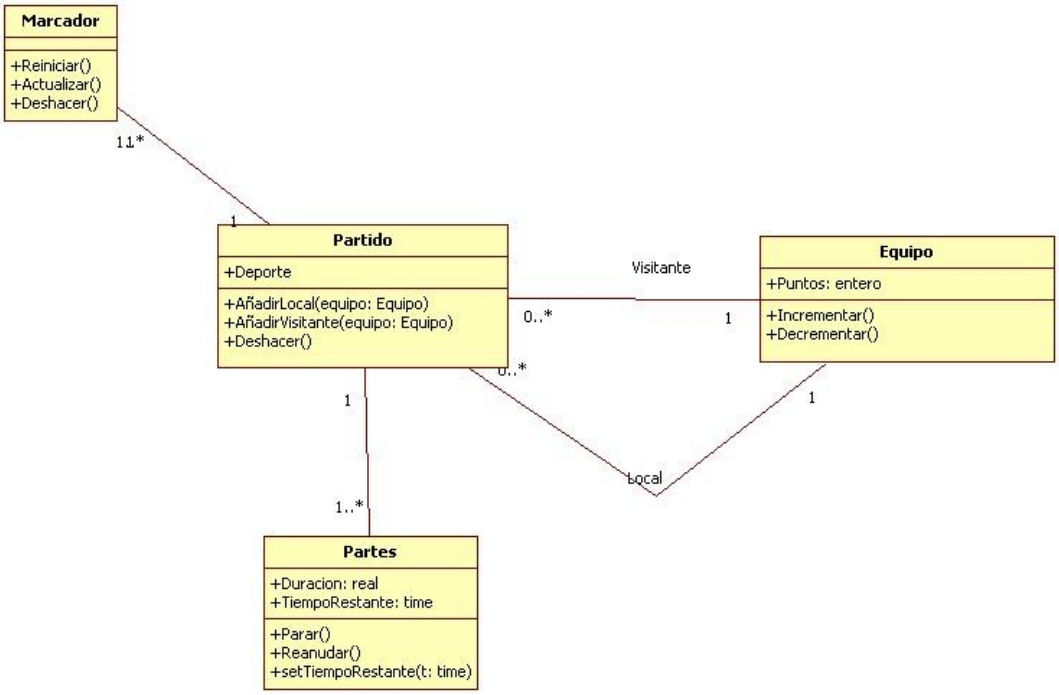


4.22: Estados de la conexión

4.6 Modelo del dominio

Como resultado del análisis del sistema resulta el modelo del dominio de nuestra línea de productos. En la siguiente figura podemos ver el diagrama inicial de clases que surge tras analizar el sistema.

Un diagrama de clases es una representación estática de las clases del sistema y de las relaciones entre las mismas. Es importante para ver los objetos del sistema, así como sus atributos y relaciones. Este diagrama que presentamos en este apartado es un modelo inicial que surge tras el estudio y análisis de la línea de productos. En la siguiente fase de estudio se decidirá si estas clases son suficientes, si alguna no es necesaria, o hay que añadir más. En el capítulo siguiente se explicará más con detalle el mismo.



4.23: Modelo inicial del dominio

CAPÍTULO 5

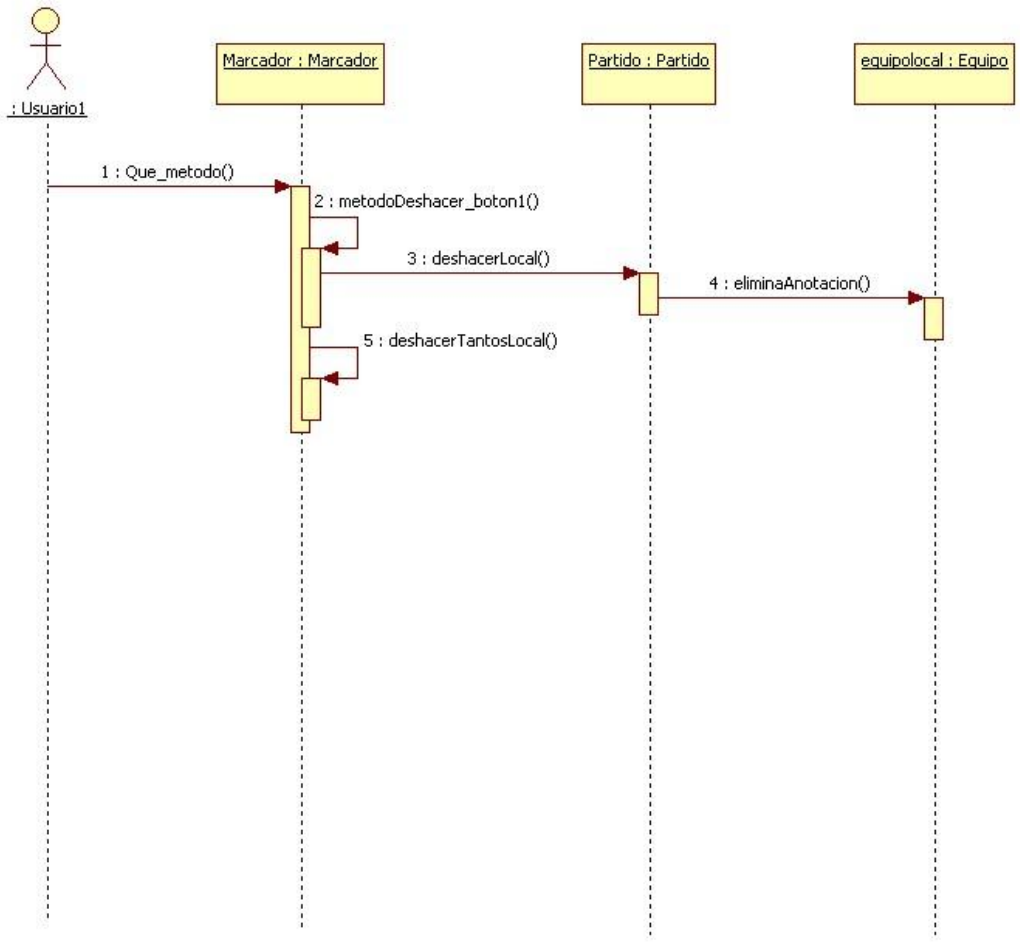
DISEÑO DE LA LÍNEA DE PRODUCTOS

La fase de diseño se define como el proceso por el cual se traducen las especificaciones de los requisitos en una representación del software que se quiere construir. El diseño es una especie de "puente" entre el análisis del problema y la implementación del mismo. A lo largo de esta fase de transforma el modelo de dominio de la información creado durante el análisis en las estructuras de datos necesarias para implementar el software. Además, el diseño puede entenderse como una materialización de los requisitos de la línea de productos.

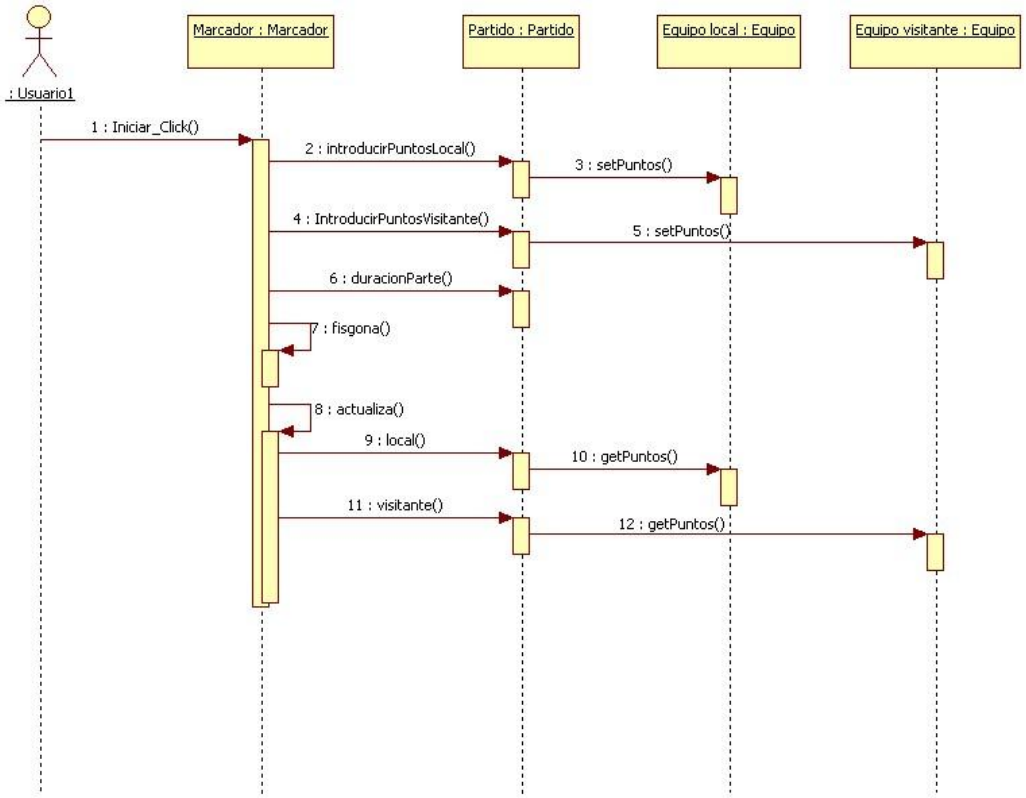
5.1 Diagramas de secuencia

Utilizaremos diagramas de secuencia en la fase de diseño para expresar cómo interaccionan usuario y aplicación.

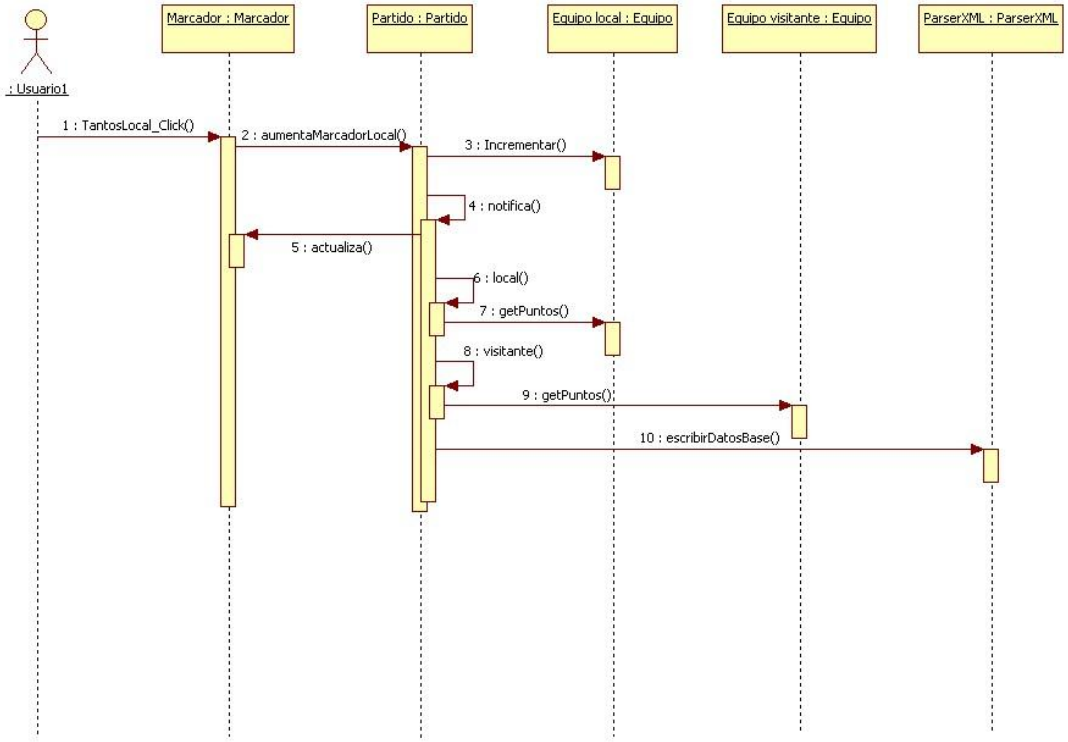
5.1.1 Paquete Marcador



5.1 Diagrama de secuencia del caso de uso "Deshacer local"

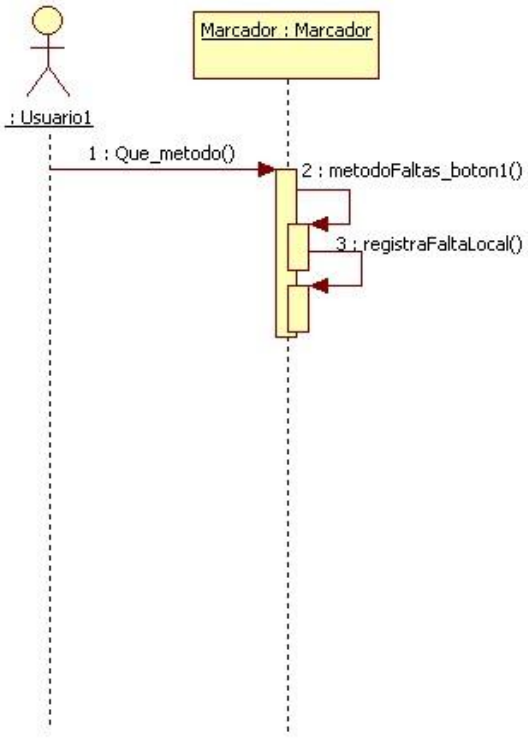


5.2: Diagrama de secuencia del caso de uso "Reiniciar Marcador"

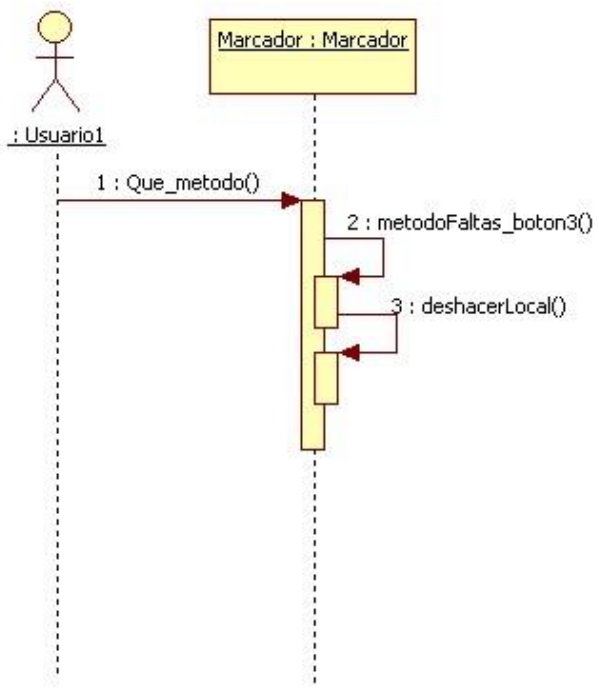


5.3: Diagrama de secuencia del caso de uso "incrementar marcador local de 1 en 1"

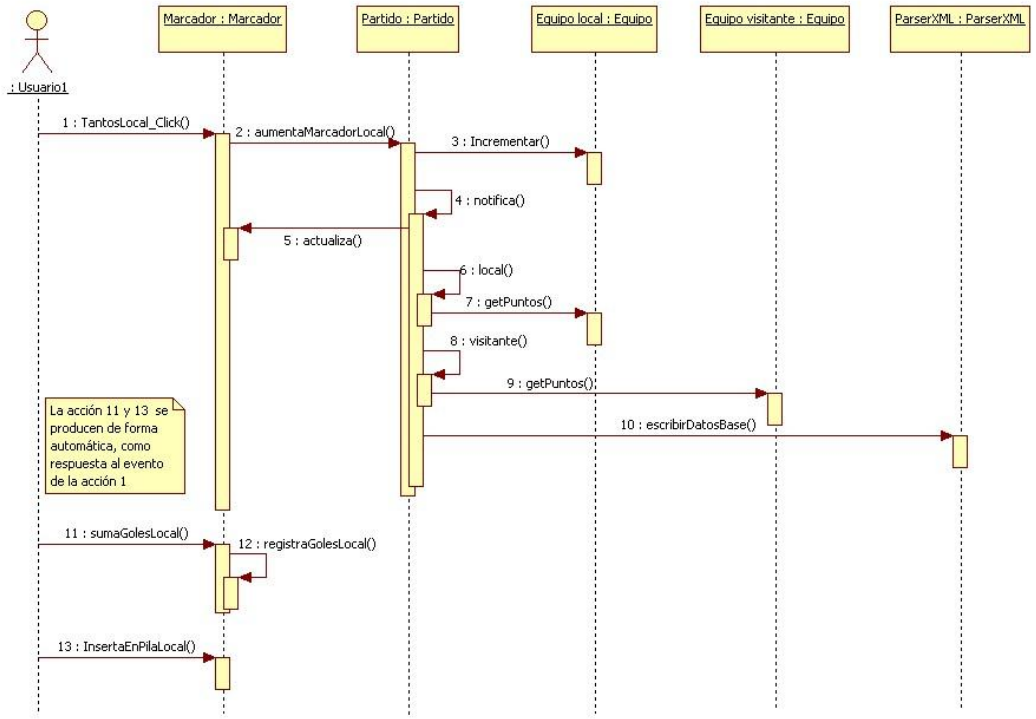
5.1.2 Paquete estadísticas



5.4: Diagrama de secuencia del caso de uso "Introducir falta local" (deporte: baloncesto)

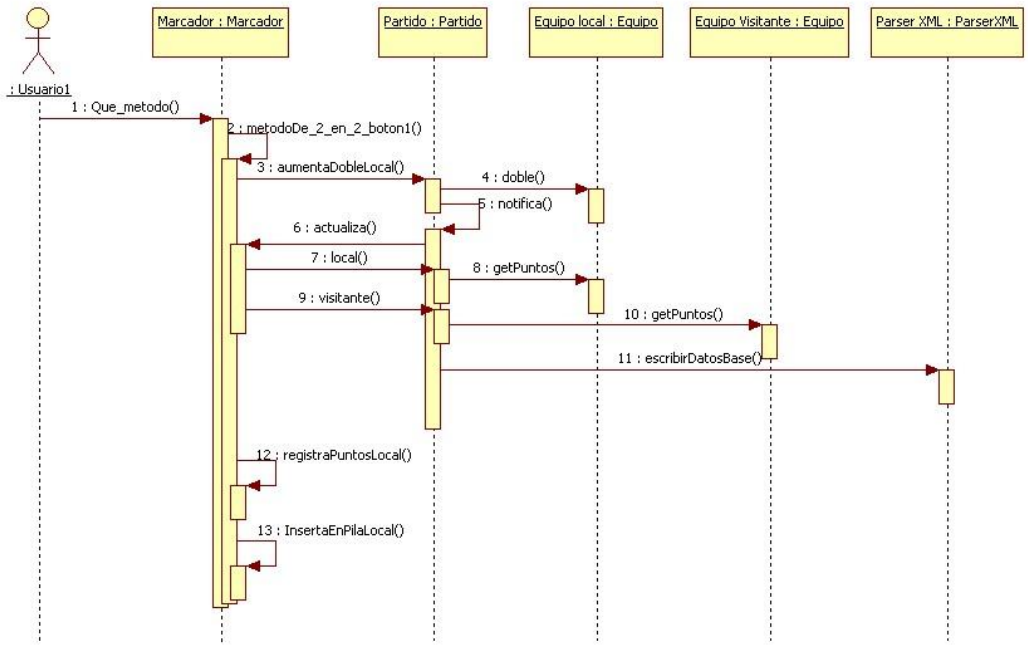


5.5: Diagrama de secuencia del caso de uso "Deshacer Falta local" (deporte: baloncesto)

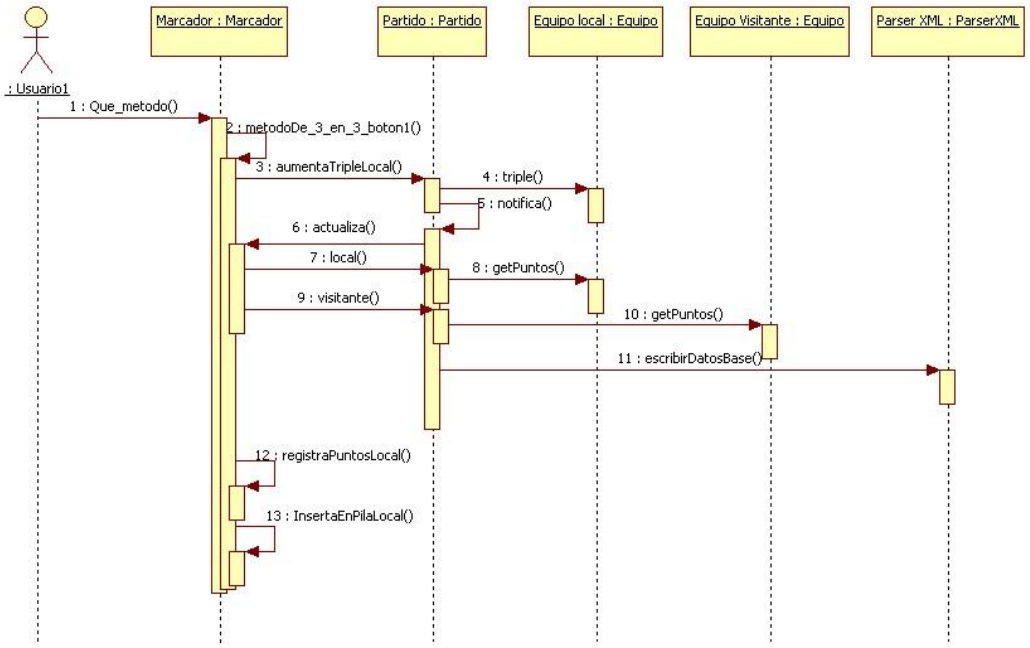


5.6: Diagrama de secuencia del caso de uso Incrementar marcador local de 1 en 1 con el paquete estadística incluido (deporte balonmano)

5.1.3 Paquete Anotación múltiple

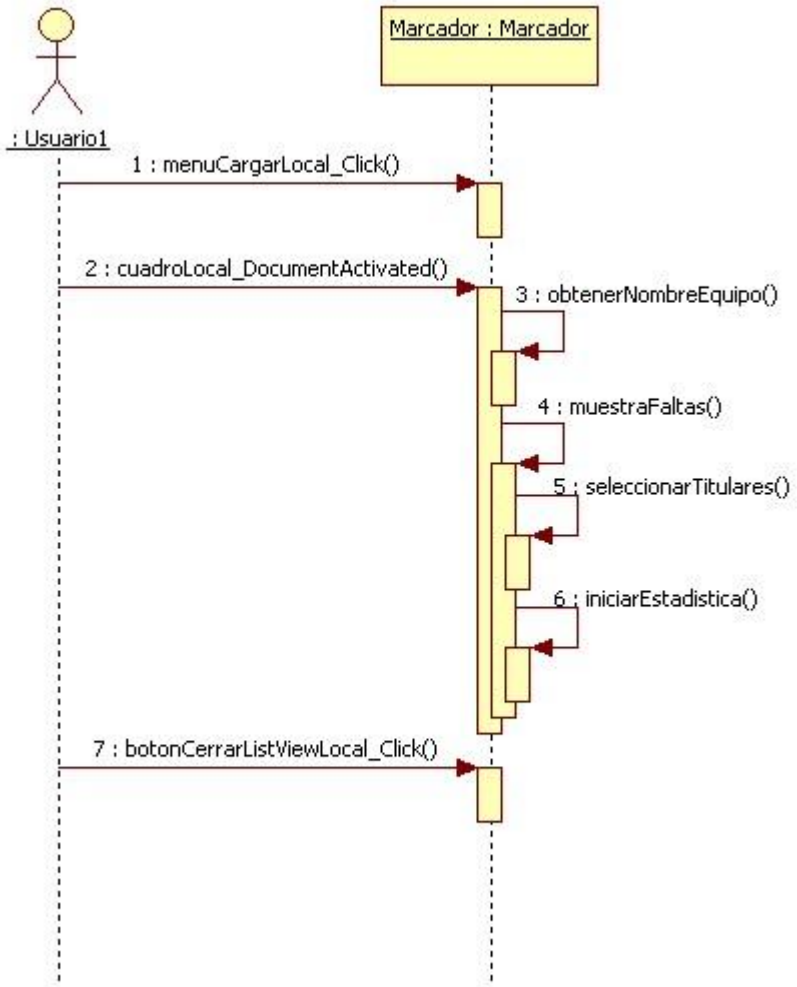


5.7: Diagrama de secuencia del caso de uso "Incrementar marcador local de 2 en 2" con paquete de estadísticas incluido



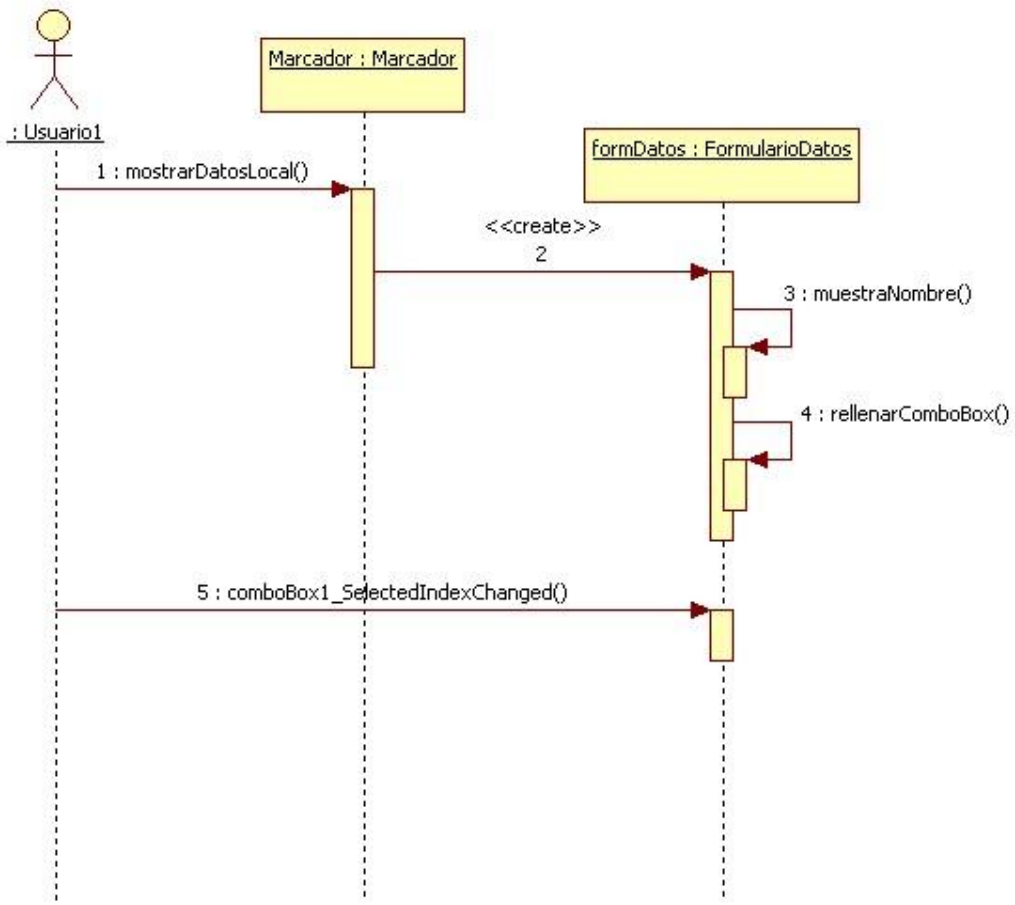
5.8: Diagrama de secuencia del caso de uso "Incrementar marcador local de 3 en 3" con paquete de estadísticas incluido

5.1.4 Paquete Gestión Equipo



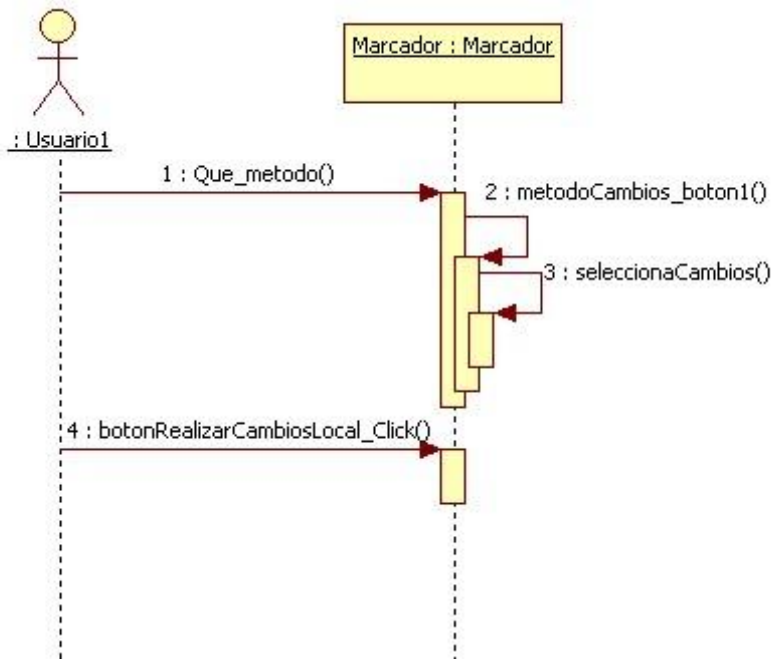
5.9: Diagrama de secuencia del caso de uso "Cargar Equipo Local"

5.1.5 Paquete Ver Datos Equipo



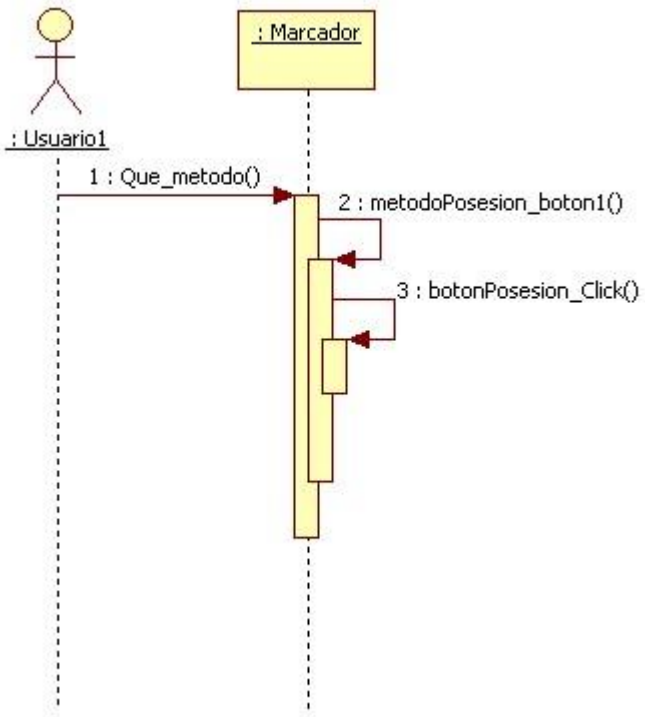
5.10: Diagrama de secuencia del caso de uso "Ver Datos Equipo Local"

5.1.6 Paquete cambios



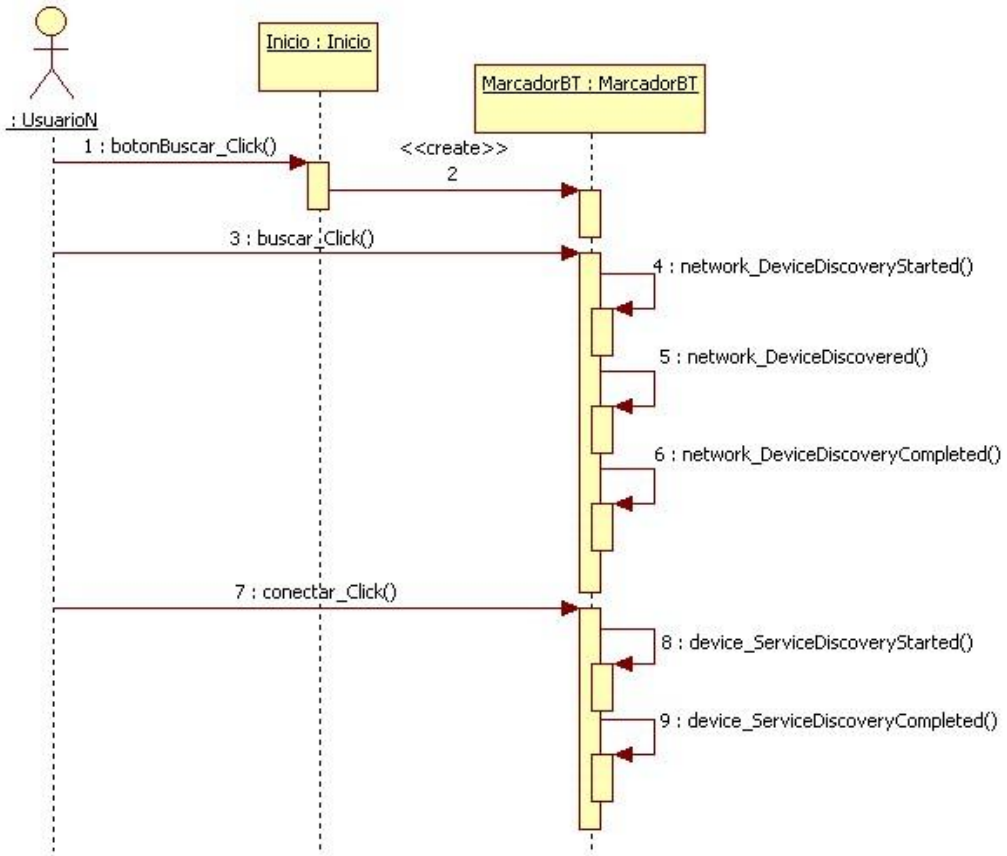
5.11: Diagrama de secuencia del caso de uso "Seleccionar cambios Equipo Local"

5.1.7 Paquete posesión



5.12: Diagrama de secuencia del caso de uso "Reiniciar Posesión"

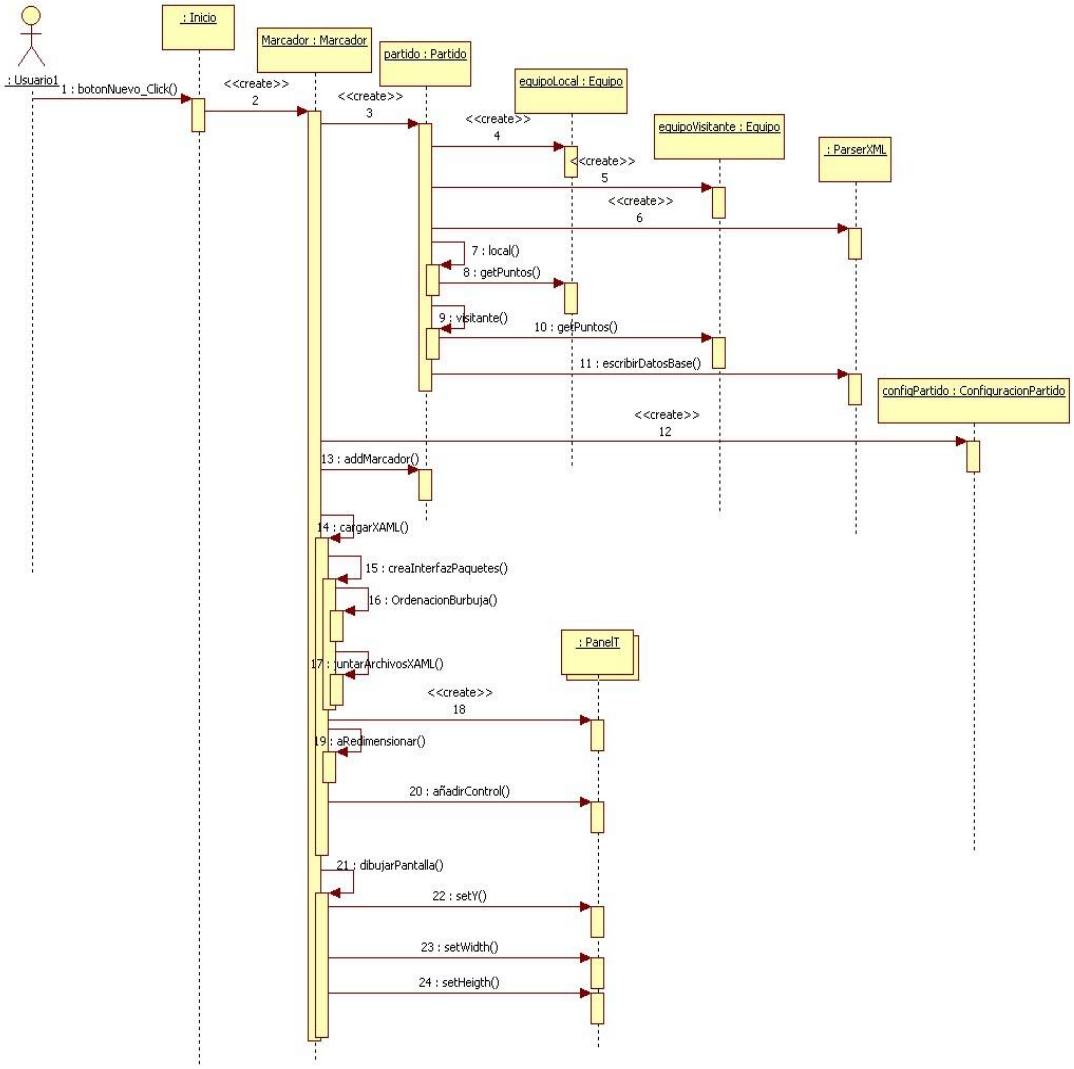
5.1.8 Paquete conexiones



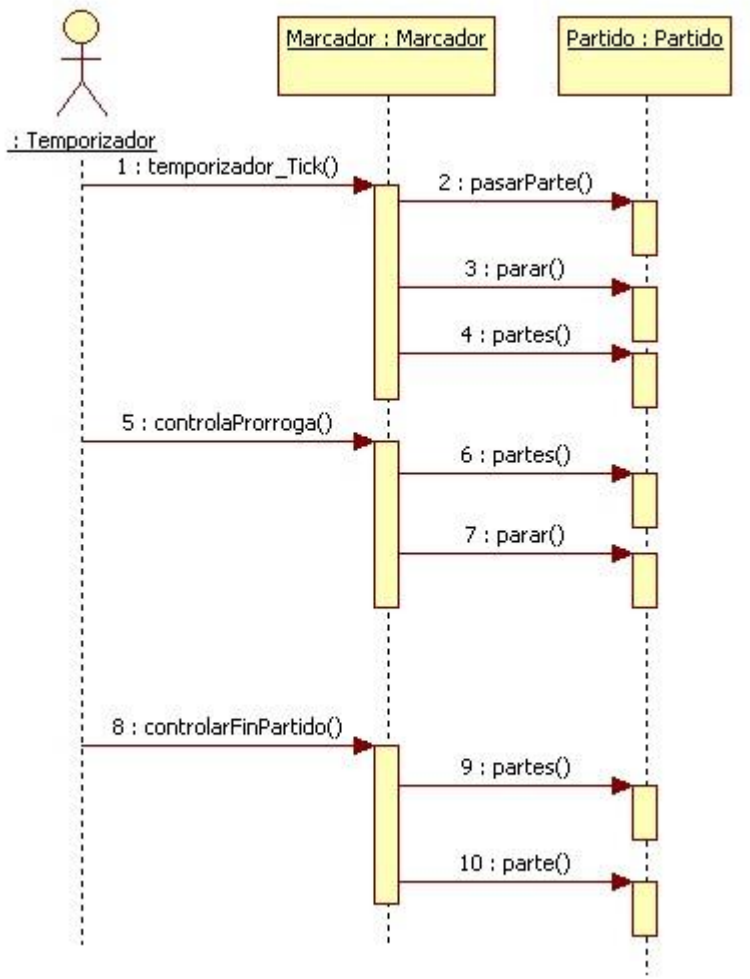
5.13: Diagrama de secuencia del caso de uso "Conexión Bluetooth"

5.1.9 Otros diagramas de secuencia

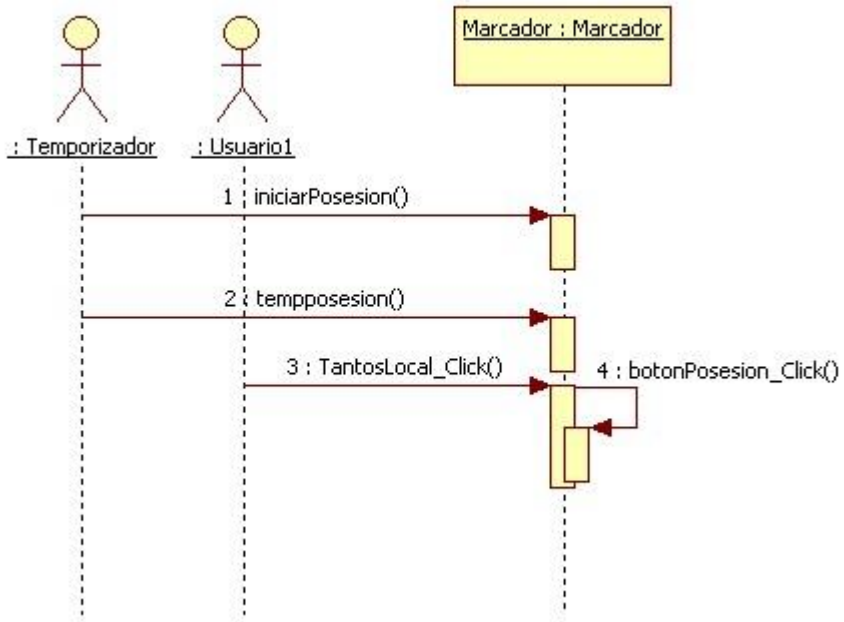
En este apartado se expondrán otros diagramas de secuencia de importancia para la aplicación. Algunos de ellos se inician automáticamente (sin la acción del usuario), lo cual está indicado en el propio diagrama.



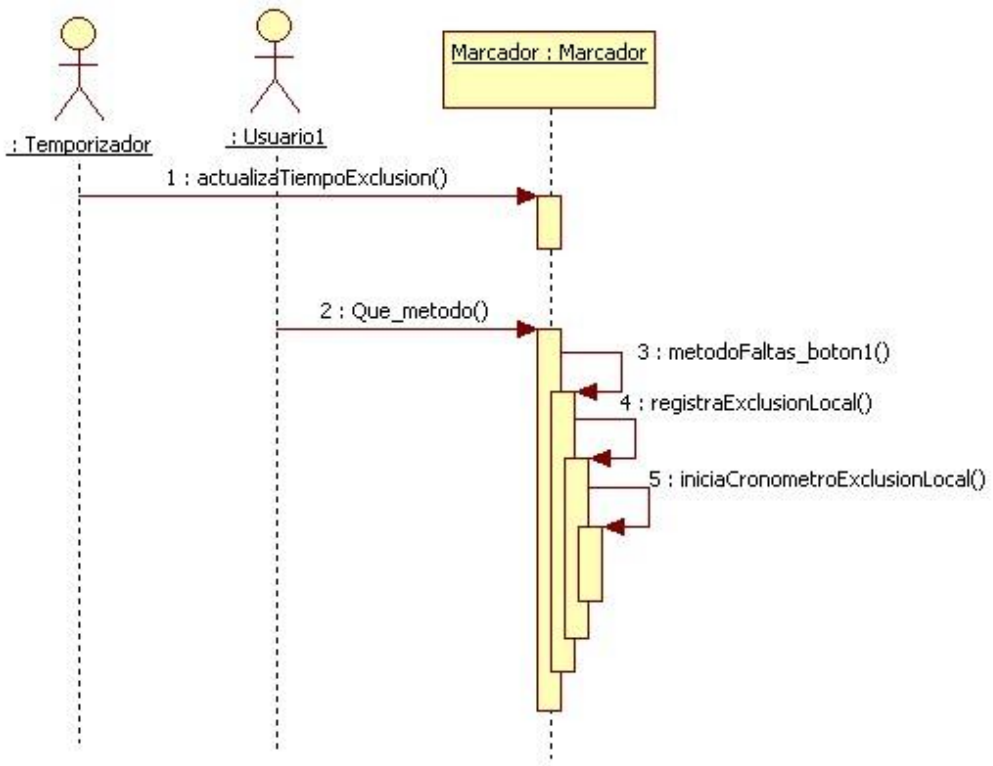
5.14: Diagrama de secuencia de inicio de la aplicación



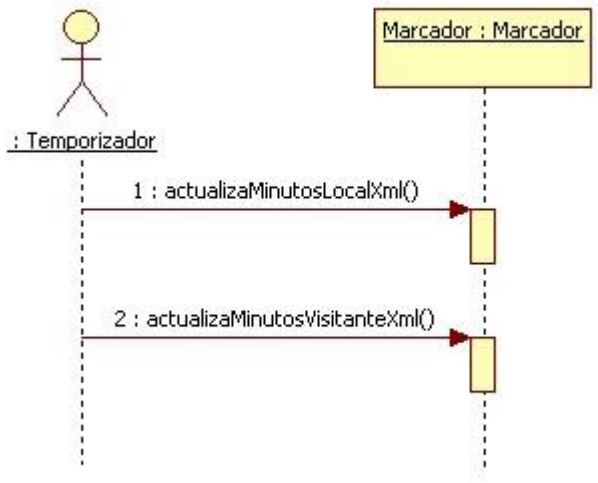
5.15: Diagrama de secuencia de los eventos de tiempo que suceden al llegar al final de un partido



5.16: Diagrama de secuencia de eventos de tiempo y posesión



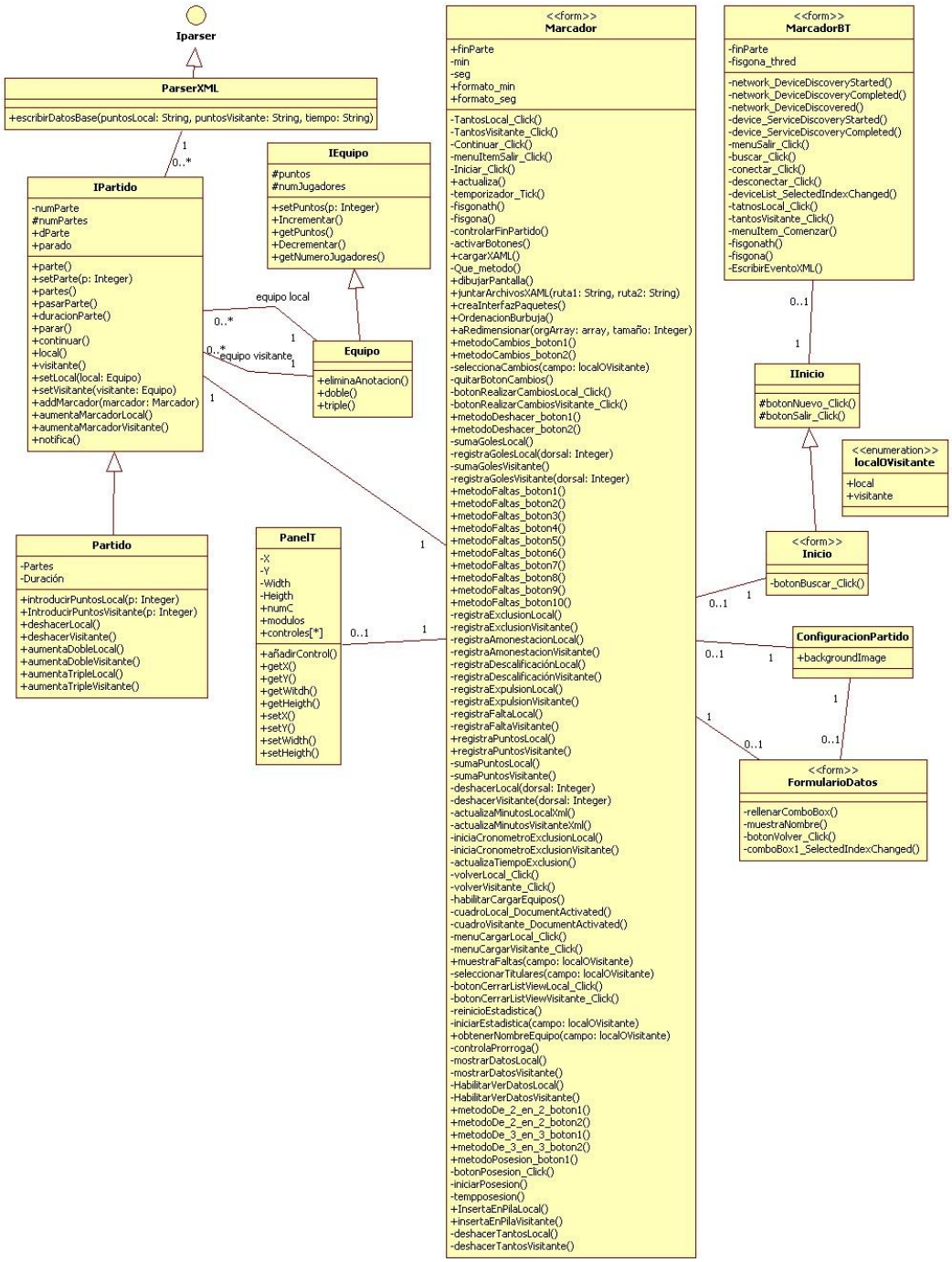
5.17: Diagrama de secuencia de eventos de tiempo de las etiquetas de exclusión (balonmano)



5.18: Diagrama de secuencia de actualización de estadísticas

5.2 Diagrama de clases

Tras realizar un primer análisis de las clases que se iban a necesitar en la línea de productos (como se puede ver en la figura 4.23), tras realizar un estudio más exhaustivo de la línea de productos se llegó a la conclusión de que dichas clases eran insuficientes, por lo que hubo que añadir alguna más. El diagrama final de clases de la línea de productos puede observarse en la figura 5.19. Se han agrupado todas las clases parciales mostrando todos los métodos posibles que puede tener dicha clase. Se adjunta una explicación detallada de todos los miembros de cada una de las clases, divididas por paquetes.



5.19: Diagrama de clases de la Línea de Productos

5.2.1 Paquete marcador

Class IEquipo	
Características genéricas de un equipo de cualquier deporte	
Atributos	**Métodos**
Puntos: Puntos acumulados en un partido por el equipo.	setPuntos(p:integer): Establece el número de puntos de un equipo
NumJugadores; Número de jugadores del equipo.	Incrementar():Incrementa en 1 los puntos del equipo
	getPuntos():Devuelve los puntos actuales por el equipo en un partido
	Decrementar():Disminuye en 1 los puntos actuales del equipo.
	getNumeroJugadores():Devuelve el número de jugagores del equipo.

Partial Class Equipo	
Implementación de la interfaz IEquipo para un deporte en particular	
Atributos	**Métodos**
	equipo(): Método creador de la clase

Class IPartido	
Clase que representa un partido de cualquier deporte.	
Atributos	**Métodos**
numParte: Número de parte en la que se encuenta el partido	parte(): Devuelve la parte del encuentro en la se encuentra el partido
numPartes: Número de partes del partido	setParte(int p): Fija la parte de partido
dParte: Duración de cada parte del partido en minutos	partes():Devuelve el número de partes de un partido.
parado: true si el tiempo está parado	pasarParte(): Pasa a la siguiente parte del partido
equipoLocal: Equipo que ejerce de local en el partido	duraciónParte():Devuelve la duración de cada parte del partido en minutos
equipoVisitante: Equipo que ejerce de visitante en el partido	parar():Para el partido
MarcadorLocal: Marcador principal. Se encuentra en el mismo dispositivo que el partido	continuar(): Continua el partido
parser: Procesador de archivos XML	local():Devuelve el equipo local del partido
	visitante(): Devuelve el equipo visitante del partido
	setLocal(): Asigna un equipo local al partido
	setVisitante(): Asigna un equipo visitante al partido
	addMarcador(Marcador marcador): Añade marcador como marcador local

	aumentaMarcadorLocal():Incrementa el marcador del equipo local
	aumentaMarcadorVisitante():Incrementa el marcador del equipo visitante
	notifica():Informa al sistema de los cambios

Partial Class Partido	
Clase que representa un partido de BALONMANO, implementa la interfaz de un partido y además añade las características propias de uno de balonmano	
Atributos	**Métodos**
PARTES: número de partes	partido():Método de creación de un partido. Sin asignar equipos.
DURACIÓN: duración del partido	introducirPuntosLocal(int p): Método para introducir el número de puntos del equipo local
	introducirPuntosVisitante(int p): Método para introducir el número de puntos del equipo visitante

Class ConfiguracionPartido	
Implementa la imagen de fondo del formulario	
Atributos	**Métodos**
backgroundImage: Fondo de imagen para la aplicación	ConfiguracionPartido(): Método creador de la clase

Partial Class Equipo	
Implementación de la interfaz IEquipo para un deporte en particular	
Atributos	**Métodos**
	equipo(): Método creador de la clase

Class IInicio	
Formulario de inicio de la aplicación, es la primera pantalla que se ve al iniciar la aplicación, y ofrece las dos posibilidades para controlar un partido: crearlo o unirse a uno ya creado	
Atributos	**Métodos**
	botonNuevo_Click(): Evento que sucede cuando se pulsa el botón "Crear partido"
	botonSalir_Click(): Evento que finaliza la aplicación

Partial Class Marcador (marcador.cs)	
Abstracción de un marcador	
Atributos	**Métodos**
partido: partido que estamos controlando	marcador(): inicializa el entorno y componentes
configPartido: imagen de fondo	TantosLocal_Click(): anotación equipo local
fisgona_thread: hilo que trata los eventos según llegan al partido	TantosVisitante_Click():anotación equipo visitante
	Continuar_Click(): reanuda el cronómetro
	Iniciar_Click(): inicia el partido
	menuItemSalir_Click(): salir de la aplicación
	actualiza(): actualiza los datos de marcador, puntuación y tiempo .
	temporizador_Tick(): evento del temporizador
	fisgonath(): el hilo de fisgona
	fisgona(): la función en sí

partial Class Marcador (marcadorInterfaz.cs)	
Métodos relacionados con la interfaz común de cualquier deporte	
Atributos	**Métodos**
vectorPaneles: paneles que tendrá la interfaz	controlarFinPartido(): determina si se ha llegado al final del partido
vectorBotones: botones que tendrá la interfaz	activarBotones(): evento que activa botones de la interfaz y reinicia las partes del partido
vectorLabels: etiquetas que tendrá la interfaz	cargarXAML(): carga la interfaz desde un archivo XAML
vectorListas: listas que tendrá la interfaz	Que_metodo(): evento asignado a todos los botones, que asocia a cada botón con su acción
	dibujarPantalla(): dibuja la interfaz
	juntarArchivosXAML(string ruta 1, string ruta2): une dos documentos XAML en uno
	creaInterfazPaquetes(): une todos los ficheros XAML de la carpeta interfaces en único archivo
	ordenacionBurbuja(): ordena un array
	aRedimensionar(Array array, int tamaño): establece la dimensión de un vector

Class ParserXml	
Procesador de archivos XML	
Atributos	**Métodos**
	EscribirDatosBase(string PuntosLocal, string PuntosVisitante, string Tiempo): guarda en el archivo XML el transcurso del partido.

Class PanelT	
Representa un control de tipo Panel Transparente	
Atributos	**Métodos**
numC: número de controles	PanelT(): constructor
modulos: porción del espacio del formulario	AñadirControl(Control elemento): añade un elemento al panel
X: distancia horizontal respecto del panel	getX()
Y: distancia vertical respecto del	getY()
Witdh: anchura del panel	getWitdh()
Heigth: altura del panel	getHeigth()
	setX(int p)
	setY(int p)
	setWitdh(int p)
	setHeigth(int p)
	getmodulos()
	setmodulos(int p)
	getnumC()
	setnumC(int p)

Class Program	
Clase de inicio de la aplicación	
Atributos	**Métodos**
	Main():Punto de entrada principal para la aplicación.

5.2.2 Paquete Bluetooth

Class MarcadorBT	
Marcador remoto que se conecta por bluetooth	
Atributos	**Métodos**
finParte: indica el final de una parte.	inicio(): método de creación del formulario de inicio.
fisgona_thread: función delegada	botonBuscar_Click(): evento que sucede cuando se pulsa el botón “Buscar Partido”.
m_manager	marcadorBT(): método de creación de la clase
m_network	networkDeviceDiscoveryStarted(): se llama cuando el bluetooth comienza a buscar dispositivos a su alrededor
m_serviceCurrent	networkDeviceDiscovered(): se llama al encontrar un dispositivo
m_streamCurrent	networkDeviceDiscoveryCompleted(): se llama cuando se ha acabado de buscar dispositivos
fileBrowser	deviceServiceDiscoveryStarted():
	deviceServiceDiscoveryCompleted():
	menuSalir_Click(): finaliza la aplicación

	buscar_Click(): búsqueda de otros dispositivos bluetooth a los que unirse
	conectar_Click(): emparejarse con otro dispositivo
	deviceList_Selected_Index_Changed(): evento para controlar la elección de un dispositivo
	desconectar_Click(): evento para terminar la conexión
	TantosLocal_Click(): anotación del equipo local
	TantosVisitante_Click(): anotación del visitante
	menuItemComenzar_Click()
	fisgonath():
	fisgona():
	EscribirEventoXml():

Class Inicio	
Implementación de la interfaz IInicio	
Atributos	**Métodos**
	inicio(): método de creación del formulario de inicio.
	botonBuscar_Click(): evento que sucede cuando se pulsa el botón “Buscar Partido”.

5.2.3 Paquete Cambios Baloncesto (igual para Balonmano y para Futbol Sala)

Partial Class Marcador (MarcadorCambios.cs)	
Métodos relacionados con el control de los cambios	
Atributos	**Métodos**
botonRealizarCambiosLocal	metodoCambios_boton1():Evento del botón de cambios local
botonRealizarCambiosVisitante	metodoCambios_boton2():Evento del botón de cambios visitante
	seleccionaCambios(localOvisitante campo): Crea una lista para marcar los cambios que se produzcan
	quitarBotonCambios(): Maneja el evento de destrucción de la lista, para destruir también los botones asociados
	botonRealizarCambiosLocal_Click(): Borra de la lista los jugadores que había antes e incluye los nuevos jugadores en el equipo local
	botonRealizarCambiosVisitante_Click(): Borra de la lista los jugadores que había antes e incluye los nuevos jugadores en el equipo visitante

5.2.4 Paquete De 2 en 2

Partial Class Equipo (Equipo.cs)	
Métodos para anotación múltiple X2	
Atributos	**Métodos**
	doble():Incrementa en 2 los puntos del equipo

Partial Class Marcador (Marcador.cs)	
Métodos para los eventos de la interfaz del paquete De 2 en 2	
Atributos	**Métodos**
	metodoDe_2_en_2_boton1():Evento del botón X2 local
	metodoDe_2_en_2_boton2():Evento del botón X2 visitante

Partial Class Partido (Partido.cs)	
Métodos para anotación múltiple X2	
Atributos	**Métodos**
	aumentaDobleLocal():Aumenta en dos el resultado del equipo local
	aumentaDobleVisitante():Aumenta en dos el resultado del equipo visitante

5.2.5 Paquete De 3 en 3

Partial Class Equipo (Equipo.cs)	
Métodos para anotación múltiple X3	
Atributos	**Métodos**
	doble():Incrementa en 3 los puntos del equipo

Partial Class Marcador (Marcador.cs)	
Métodos para los eventos de la interfaz del paquete De 3 en 3	
Atributos	**Métodos**
	metodoDe_3_en_3_boton1():Evento del botón X3 local
	metodoDe_3_en_3_boton2():Evento del botón X3 visitante

Partial Class Partido (Partido.cs)	
Métodos para anotación múltiple X3	
Atributos	**Métodos**
	aumentaTripleLocal():Aumenta en tres el resultado del equipo local
	aumentaTripleVisitante():Aumenta en tres el resultado del equipo visitante

5.2.6 Paquete DeshacerPunto

Partial Class Equipo (Equipo.cs)	
Métodos para deshacer anotación	
Atributos	**Métodos**
	eliminaAnotacion():Decrementa en uno la puntuación del equipo

Partial Class Marcador (MarcadorDeshacer.cs)	
Métodos para los eventos de la interfaz del paquete Deshacer	
Atributos	**Métodos**
	metodoDeshacer_boton1(): Evento del botón deshacer local
	metodoDeshacer_boton2(): Evento del botón deshacer visitante
	inicia8(): asocia eventos para guardar quién anota cada tanto
	insertaEnPilaLocal(): guarda en una pila quién anotó el tanto local.
	insertaEnPilaVisitante():guarda en una pila quién anotó el tanto visitante
	deshacerTantoLocal(): quita un tanto de las estadísticas al jugador local correspondiente
	deshacerTantoVisitante(): quita un tanto de las estadísticas al jugador visitante correspondiente

5.2.7 Paquete Estadísticas Baloncesto

Partial Class Marcador (MarcadorEstadistica.cs)	
Métodos para los eventos de la interfaz del paquete Estadísticas Baloncesto	
Atributos	**Métodos**
	inicia4(): Añade controladores de eventos a los controles de la interfaz del paquete Estadisticas Baloncesto
	sumaPuntosLocal(): Evento para guardar el anotador de la canasta local
	registraPuntosLocal(int dorsal, int tipoCanasta): Registra y guarda en estadísticas quien anotó la canasta local (del tipo que sea)
	sumaPuntosVisitante(): Evento para guardar el anotador de la canasta visitante
	registraPuntosVisitante(int dorsal, int tipoCanasta): Registra y guarda en estadísticas quien anotó la canasta local (del tipo que sea)
	metodofaltas_boton3():Evento del botón borrar falta local
	deshacerregistrafaltalocal(int dorsal): Quita una falta a un jugador local, eliminándola de las estadísticas
	metodofaltas_boton4():Evento del botón borrar falta visitante
	deshacerregistrafaltavisitante(int dorsal): Quita una falta a un jugador visitante, eliminándola de las estadísticas
	metodofaltas_boton1():Evento del botón falta local

	registrafaltalocal(int dorsal): Añade a las estadísticas del jugador local una falta
	metodofaltas_boton2():Evento del botón faltas visitante
	registrafaltavisitante(int dorsal): Añade a las estadísticas del jugador visitante una falta
	actualizaMinutosLocalXml(): Manejador de eventos que actualiza en el xml los minutos jugados por cada jugador del equipo local
	actualizaMinutosVisitanteXml(): Manejador de eventos que actualiza en el xml los minutos jugados por cada jugador del equipo visitante

5.2.8 Paquete Estadísticas Balonmano

Partial Class Marcador (MarcadorEstadistica.cs)	
Métodos para los eventos de la interfaz del paquete Estadísticas Balonmano	
Atributos	**Métodos**
	inicia4(): Añade controladores de eventos a los controles de la interfaz del paquete Estadisticas Balonmano
	sumaGolesLocal(): Evento para guardar el anotador del gol local
	registraGolLocal(int dorsal): Registra y guarda en estadísticas quien anotó el gol local
	sumaGolesVisitante(): Evento para guardar el anotador del gol visitante
	registraGolVisitante(int dorsal): Registra y guarda en estadísticas quien anotó el gol visitante.
	metodofaltas_boton1():Evento del botón exclusión local
	registraexclusionlocal(int dorsal): Añade una exclusión a las estadísticas del jugador local
	metodofaltas_boton2():Evento del botón exclusión visitante
	registraexclusionvisitante(int dorsal): Añade una exclusión a las estadísticas del jugador visitante
	metodofaltas_boton5():Evento del botón amonestación local

	registraamonestacionlocal(int dorsal): Añade una amonestación a las estadísticas del jugador local
	metodofaltas_boton6():Evento del botón amonestación visitante
	registraamonestacionvisitante(int dorsal): Añade una amonestacion a las estadísticas del jugador visitante
	metodofaltas_boton7():Evento del botón descalificación local
	registradescalificacionlocal(int dorsal): Añade una descalificación a las estadísticas del jugador local
	metodofaltas_boton8():Evento del botón descalificación visitante
	registradescalificacionvisitante(int dorsal): Añade una descalificación a las estadísticas del jugador visitante
	metodofaltas_boton9(): Evento del botón expulsión local
	registraexpulsionlocal(int dorsal): Añade una expulsión a las estadísticas del jugador local
	metodofaltas_boton10():Evento de botón expulsión visitante
	registraexpulsionvisitante(int dorsal): Añade una expulsiónn a las estadísticas del jugador visitante
	metodofaltas_boton3():Evento del botón borrar local
	deshacerLocal(int dorsal): Elimina la última

	sanción cometida por el jugador local
	metodofaltas_boton4():Evento del botón borrar visitante
	deshacerVisitante(int dorsal): Elimina la última sanción cometida por un jugador visitante.
	actualizaMinutosLocalXml(): Manejador de eventos que actualiza en el xml los minutos jugados por cada jugador del equipo local
	actualizaMinutosVisitanteXml(): Manejador de eventos que actualiza en el xml los minutos jugados por cada jugador del equipo visitante
	iniciaCronometroExclusionLocal():Pone a 2 minutos el cronómetro de exclusión local
	iniciaCronometroExclusionVisitante():Pone a 2 minutos el cronómetro de exclusión visitante
	actualizaTiempoExclusion(): Manejador de eventos para las etiquetas de exclusión

5.2.9 Paquete Estadísticas Futbol Sala

Partial Class Marcador (MarcadorEstadistica.cs)	
Métodos para los eventos de la interfaz del paquete Estadísticas Futbol Sala	
Atributos	**Métodos**
	inicia4(): Añade controladores de eventos a los controles de la interfaz del paquete Estadisticas Futbol Sala
	sumaGolesLocal(): Evento para registrar el anotador del gol local
	registraGolLocal(int dorsal): Registra y guarda en las estadísticas quién anotó el gol local.
	sumaGolesVisitante(): Evento para registrar el anotador del gol visitante
	registraGolVisitante(int dorsal): Registra y guarda en las estadísticas quién anotó el gol visitante
	metodofaltas_boton1():Evento del botón expulsión local
	registraexpulsionlocal(int dorsal): Añade una expulsión a las estadísticas del jugador local
	metodofaltas_boton2():Evento del botón expulsión visitante
	registraexpulsionvisitante(int dorsal): Añade una expulsión a las estadísticas del jugador visitante
	metodofaltas_boton5():Evento del botón amarilla local

	registraamonestacionlocal(int dorsal): Añade una amarilla a las estadísticas del jugador local
	metodofaltas_boton6():Evento del botón amarilla visitante
	registraamonestacionvisitante(int dorsal): Añade una amarilla a las estadísticas del jugador visitante
	metodofaltas_boton3():Método del botón borrar local
	deshacerLocal(int dorsal): Elimina la última sanción del jugador local
	metodofaltas_boton4(): Evento del botón borrar visitante
	deshacerVisitante(int dorsal): Elimina la última sanción del jugador visitante
	actualizaMinutosLocalXml(): Manejador de eventos que actualiza en el xml los minutos jugados por cada jugador del equipo local
	actualizaMinutosVisitanteXml(): Manejador de eventos que actualiza en el xml los minutos jugados por cada jugador del equipo visitante
	iniciaCronometroExpulsionLocal():Pone a 2 minutos el cronómetro de expulsión local
	iniciaCronometroExpulsionVisitante(): Pone a 2 minutos el cronómetro de expulsión visitante
	actualizaTiempoExpulsion(): Manejador de eventos para las etiquetas de expulsión

5.2.10 Paquete Gestor Equipo Baloncesto, Balonmano y Fútbol Sala

Partial Class Marcador (MarcadorGestorEquipo.cs)	
Elementos y Métodos para los eventos de la interfaz del paquete Gestor Equipo Baloncesto, Balonmano o Futbol Sala	
Atributos	**Métodos**
estadoCargarEquipo: Estructura compuesta de dos elementos booleanos llamado local y visitante.	inicia3():Inicializa el menú para cargar equipos
estructura: Estructura que controla si se han cargado los equipos local y visitante	volverLocal_Click(): Evento del botón local volver
menuGestorEquipos: Botón Gestor Equipo del menú	volverVisitante_Click(): Evento del botón visitante volver
menuCargarLocal: Botón Cargar Local en menú -> Gestor Equipo	habilitarCargarEquipos(): Evento para habilitar cargar equipos
menuCargarVisitante: Botón Cargar Visitante en menú -> Gestor Equipo	cuadrolocal_DocumentActivated(): Evento cuando se selecciona el archivo del equipo local
cuadrolocal: ventana de explorador de archivos para seleccionar equipo local	cuadrovisitante_DocumentActivated(): Evento cuando se selecciona el archivo del equipo visitante
cuadrovisitante: ventana de explorador de archivos para seleccionar equipo visitante	menuCargarLocal_Click(): Evento cuando se selecciona cargar local
volverLocal: botón para volver atrás en la ventana de equipo local	menuCargarVisitante_Click(): Evento cuando se selecciona cargar visitante
volverVisitante: botón para volver atrás en la ventana de equipo visitante	muestraFaltas(localOvisitante campo): Crea un documento xml de estadísticas de equipo que se utilizará posteriormente y llama al método para cargar los jugadores titulares.
eqlocal: documento XML con los datos del equipo local	seleccionarTitulares(localOvisitante campo): Método para seleccionar los jugadores titulares

eqvisitante: documento XML con los datos del equipo visitante	ListaTitulares_Disposed(): Evento que maneja la destrucción de la lista de jugadores
ListaTitulares: lista de selección de jugadores titulares	botonCerrarListViewLocal_Click(): Evento que maneja la selección de los jugadores locales titulares
elem1 a elem14: elementos de la lista de titulares	botonCerrarListViewVisitante_Click(object sender, EventArgs e): Evento que maneja la selección de los jugadores visitantes titulares
botonCerrarListViewLocal: botón para cerrar lista de selección de titulares local	reinicioEstadistica(object sender, EventArgs e): Evento para reiniciar las estadísticas
botonCerrarListViewVisitante: botón para cerrar lista de selección de titulares local	iniciarEstadistica(localOvisitante campo): Reinicia las estadísticas del equipo

5.2.11 Paquete Mostrar Nombre Equipo

Partial Class Marcador (MarcadorMostrarNombreEquipo.cs)	
Ampliación de la clase marcador para mostrar los nombre de los equipos	
Atributos	**Métodos**
nombreLocal: Etiqueta para el equipo local	inicia2(): Muestra el nombre del equipo en la pantalla.
nombreVisitante: Etiqueta para el equipo visitante	obtenerNombreEquipo(localOvisitante campo): Coloca el nombre del equipo en el marcador

5.2.12 Paquete Posesión

Partial Class Marcador (MarcadorPosesion.cs)	
Métodos y elementos para controlar la interfaz del paquete Posesión	
Atributos	**Métodos**
POSESION: Constate de valor 24	inicia1(): inicializa el marcador de posesión.
pos: indica el valor del contador en cada instante	metodoPosesion_boton1():Evento del botón reiniciar
	botonposesion_Click(): Reinicia el contador de posesión cuando se pulsa el botón.
	iniciarposesion(): permite iniciar la posesión a los botones de anotación.
	tempposesion(): Se ejecuta cada vez que el temporizador corre

5.2.13 Paquete Prórroga Baloncesto

Partial Class Marcador (MarcadorPrórroga.cs)	
Ampliación de la clase marcador para añadir prórrogas al tiempo normal	
Atributos	**Métodos**
	inicia7(): añade controlador de eventos al temporizador
	controlaProrroga(): Evento para controlar una prórroga en baloncesto. La prórroga es una parte de 5 minutos.

5.2.14 Paquete Prórroga Balonmano

Partial Class Marcador (MarcadorPrórroga.cs)	
Ampliación de la clase marcador para añadir prórrogas al tiempo normal	
Atributos	**Métodos**
	inicia7(): añade controlador de eventos al temporizador
	controlaProrroga(): Evento para controlar una prórroga en balonmano. La prórroga consta de 2 partes de 5 minutos cada una.

5.2.15 Paquete Prórroga Fútbol Sala

Partial Class Marcador (MarcadorPrórroga.cs)	
Ampliación de la clase marcador para añadir prorrogas al tiempo normal	
Atributos	**Métodos**
	inicia7(): añade controlador de eventos al temporizador
	controlaProrroga(): Evento para controlar una prórroga en fútbol sala. La prórroga es una parte de 10 minutos.

5.2.16 Paquete Ver Datos Baloncesto, Balonmano y Fútbol Sala

Class FormularioDatos	
Formulario en el que se muestran los datos de los equipos y sus jugadores	
Atributos	**Métodos**
configPartido: Contiene la configuración del partido que se controla	FormularioDatos(localOvisitante campo): Constructor del formulario
	rellenarComboBox(localOvisitante campo): Rellena el combobox del formulario con los nombres de los jugadores del equipo
	muestraNombre(localOvisitante campo): Muestra el nombre del equipo en la etiqueta superior
	BotonVolver_Click(): Evento para volver a la pantalla principal
	comboBox1_SelectedIndexChanged(): Evento para mostrar los datos de un jugador al seleccionarlo

Partial Class Marcador (MarcadorVerDatos.cs)	
Ampliación de la clase marcador para poder ver los datos de los equipos	
Atributos	**Métodos**
MenuVerDatosEquipos: botón Ver Datos del menu.	inicia5():Inicia en el menú principal submenús para ver los datos de los equipos
MenuEquipoLocal: botón Ver Datos -> Equipo Local	MostrarDatosLocal(): Inicializa el formulario para ver los datos del equipo local

MenuEquipoVisitante: botón Ver Datos -> Equipo Visitante	MostrarDatosVisitante(): Inicializa el formulario para ver los datos del equipo visitante
	HabilitarVerDatosEquipoLocal(Habilita la opción de menú de ver los datos de equipo local
	HabilitarVerDatosEquipoVisitante(): Habilita la opción de menú de ver los datos de equipo visitante

Capítulo 6

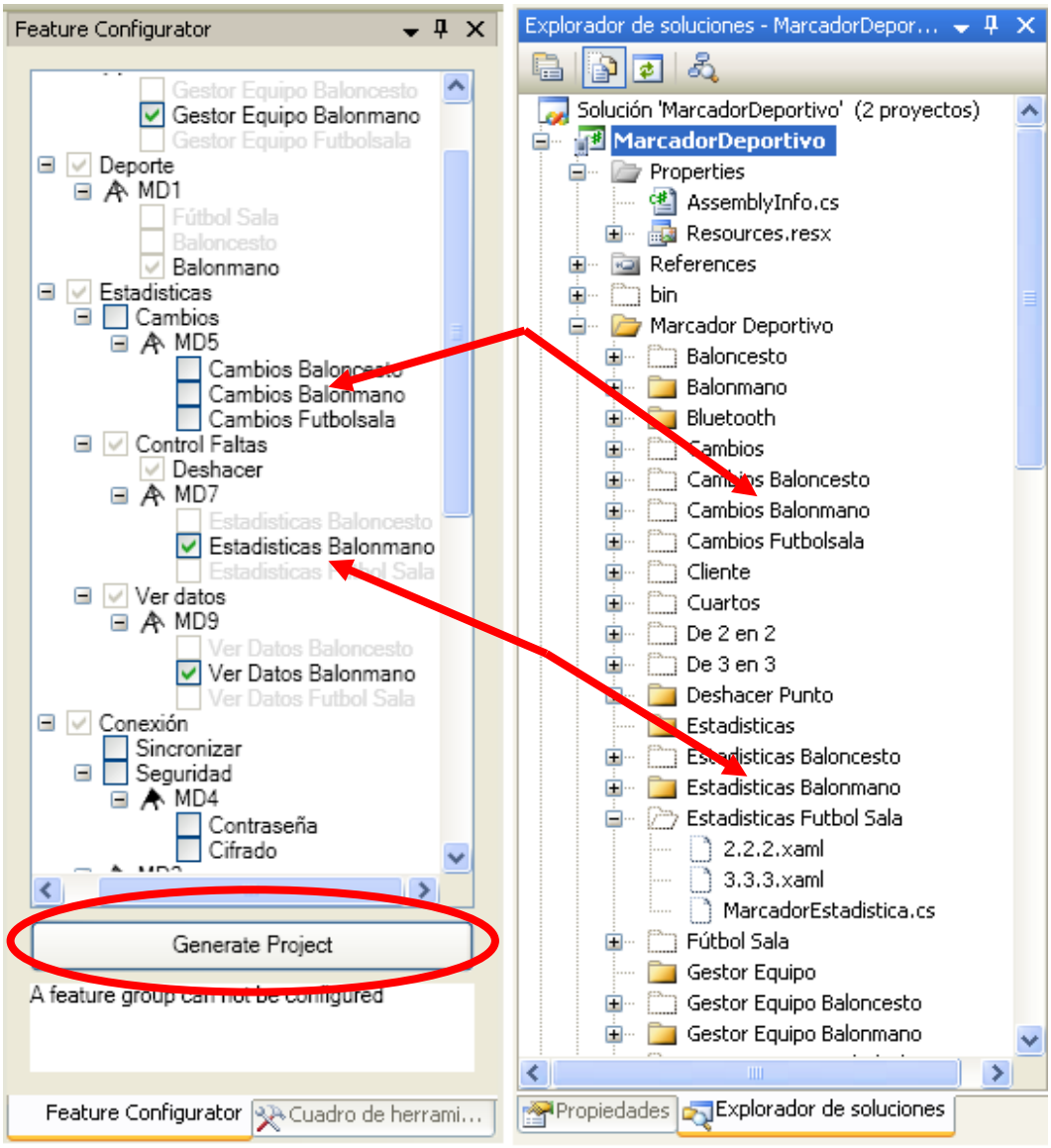
IMPLEMENTACIÓN DE LA LÍNEA DE PRODUCTOS

La implementación de la línea de productos consiste en la traducción del diseño realizado en el capítulo anterior a código de un lenguaje de programación, que en nuestro caso será el lenguaje C#.

6.1 Entorno de programación

Para el desarrollo del proyecto se nos propuso el lenguaje de programación C#. Este lenguaje está integrado en el entorno de desarrollo de Visual Studio .NET. Es un lenguaje orientado a objetos e ideal para la realización de nuestro proyecto. Dado que nuestra aplicación estaba orientada al funcionamiento en dispositivos móviles dentro del sistema operativo Windows Mobile 6.0, contar con un lenguaje de programación de Microsoft (empresa que desarrolla el sistema operativo Windows Mobile), podría ser una forma de facilitar la realización del proyecto. Más información sobre este lenguaje puede encontrarse en el apéndice B de esta memoria.

En nuestro modelo de desarrollo utilizamos el mecanismo de clases parciales de C# para implementar la línea de productos dentro de la estructura de soluciones y paquetes de la plataforma MS Visual Studio 2008. Si el framework que implementa la arquitectura de la línea de productos está organizado en paquetes de clases parciales (un paquete base y tantos paquetes auxiliares como variaciones existen), para derivar una aplicación concreta, basta con importar o referenciar los paquetes que se correspondan directamente con la configuración elegida en el modelo de características. Cada uno de estos paquetes está representado como una característica en el diagrama de características creado con la herramienta FMT. Para incluir o excluir uno de los paquetes opcionales en el proyecto que representa una aplicación concreta de la línea de productos, basta con marcar o desmarcar su característica correspondiente dentro de la ventana “feature configurator”, y pulsar el botón “Generate Project”, como se muestra en la figura 6.1:

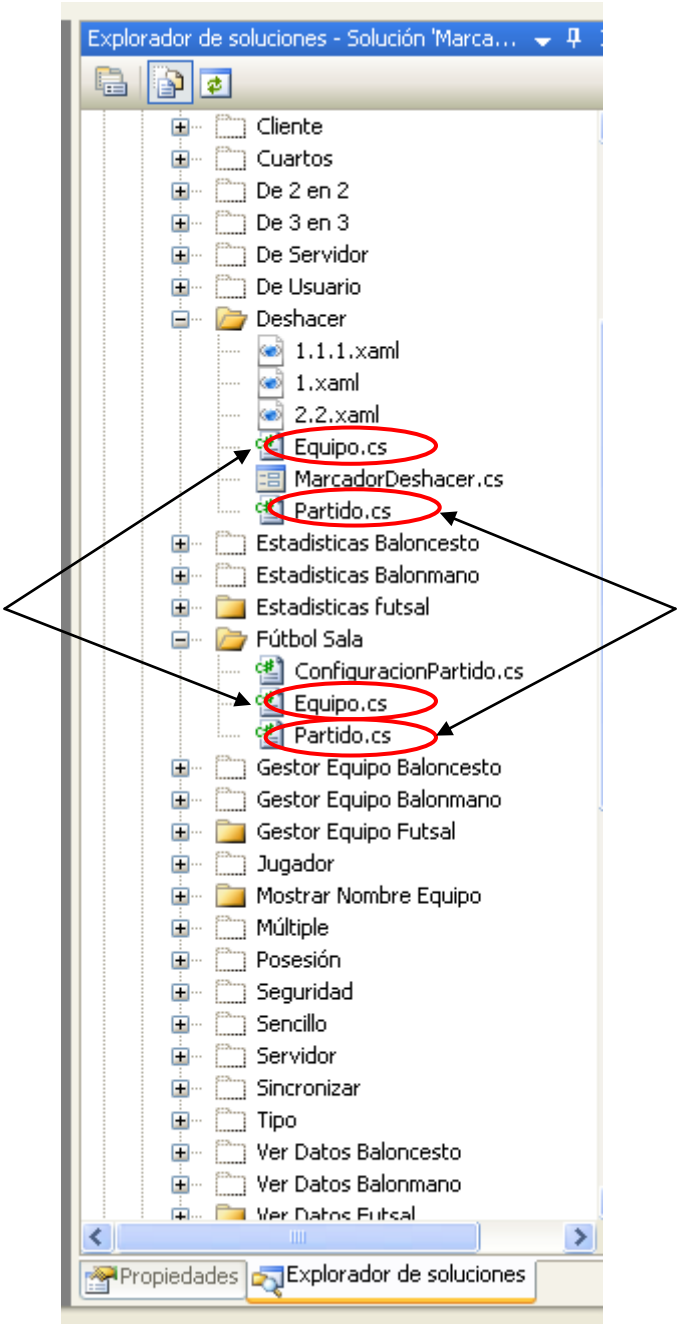


6.1: Generar proyecto

En la imagen 6.1 se ha seleccionado la característica “Estadísticas Balonmano” (que se representa en la ventana “Feature Configurator” con un tick sobre el cuadro de selección correspondiente, y en la ventana “Explorador de soluciones” como una carpeta activa en color amarillo), y se ha dejado sin marcar la característica “Cambios Balonmano” (que se muestra con un cuadro vacío en la ventana “Feature Configurator” y con una carpeta inactiva blanca en

"Explorador de soluciones"). Al pulsar sobre el botón "Generate Project", la herramienta FMT incluye en el proyecto todas las carpetas seleccionadas, así como todos los archivos que contienen, tal y como se haría de forma manual.

Nuestra aplicación está diseñada con clases parciales. Como ya se ha explicado, las clases parciales son porciones de código de una misma clase escritas en archivos diferentes. En tiempo de compilación, dichas "porciones" se unen formando una única clase. En la siguiente figura se puede ver la estructura en forma de clases parciales de nuestra aplicación:



6.2 Solución en clases parciales

6.2 Ficheros XML

Para guardar los datos de los partidos, con sus correspondientes estadísticas, así como para el uso de las aplicaciones que se conecten por bluetooth, se decidió el uso de ficheros XML como almacenamiento de información. El contenido de estos ficheros XML contiene la información acerca de los jugadores de los equipos, sus estadísticas, y también tendrán información sobre el estado del partido. La elección de guardar la información en ficheros XML se ha hecho por varias razones:

- Visual Studio .NET ofrece clases optimizadas tanto para la lectura como para la escritura de ficheros XML, facilitando así el almacenamiento y recuperación de datos de estos ficheros, y permitiendo el manejo óptimo de los datos de los ficheros.
- Visual Studio .NET proporciona un editor para la creación de documentos XML.
- Facilidad de crear una estructura jerárquica para organizar la información de los componentes de un equipo así como sus estadísticas.
- Propiedades de XML (extensibilidad, portabilidades, escalabilidad…). Ver apéndice B, apartado B.2.

6.2.1 Estructura de los ficheros XML.

La estructura de nuestros ficheros XML es diferente en función de la utilización que se le dará a cada fichero. Básicamente en nuestro proyecto hay 3 tipos de ficheros XML:

- Fichero *DatosBase.xml*: archivo utilizado para guardar los datos de un partido (puntuación de ambos equipos y tiempo de partido). Se utiliza para que los observadores que se conecten por bluetooth al dispositivo obtengan los datos del partido. Su estructura es la siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<partido>
  <equipo>
    <tantos></tantos>
  </equipo>
  <equipo>
    <tantos></tantos>
  </equipo>
  <tiempo> </tiempo>
</partido>
```

- Fichero *<nombreequipo>.xml*: archivo que contiene los nombres y dorsales de los jugadores del equipo correspondiente. Se usa para leer los nombres de los jugadores y utilizarlos en las estadísticas. Su estructura es la siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<equipo deporte="futbol">
      <nombre> </nombre>
      <jugadores>
            <jugador dorsal="">
                  <nombre> </nombre>
            </jugador>
            <jugador dorsal="">
                  <nombre> </nombre>
            </jugador>
            <!--Lista completa de jugadores del equipo-->
      </jugadores>
</equipo>
```

- Fichero *estadistica<nombreequipo>.xml*: archivo en el que se guardan los datos correspondientes a las estadísticas de los jugadores. Se usa como almacenamiento de las estadísticas y para mostrar los datos del jugador cuando sean requeridos por el usuario de la aplicación. Su estructura varía dependiendo del deporte que sea. Para un equipo de balonmano, por ejemplo, es la siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<equipo deporte="balonmano">
  <nombre> </nombre>
  <jugadores>
    <jugador dorsal="">
      <nombre> </nombre>
      <amonestaciones></amonestaciones>
      <exclusiones></exclusiones>
      <descalificaciones></descalificaciones>
      <expulsiones></expulsiones>
      <minutos></minutos>
    </jugador>
    <jugador dorsal="">
      <nombre></nombre>
      <amonestaciones></amonestaciones>
      <exclusiones></exclusiones>
      <descalificaciones></descalificaciones>
      <expulsiones></expulsiones>
      <minutos></minutos>
    </jugador>
  </jugadores>
 </equipo>
```

6.3 Interfaz de usuario

6.3.1 Distribución del espacio

Para la realización las interfaces de nuestros paquetes debíamos partir de la ya realizada en el paquete base, e ir añadiendo controles en el espacio restante según las funcionalidades de nuestros paquetes.

La manera de distribuir los botones de cada paquete consiste en dividir el espacio sobrante de la interfaz base en rectángulos imaginarios iguales llamados módulos, de tal forma que cada módulo conste del espacio suficiente para albergar una sucesión de controles colocados en horizontal. En este caso el espacio se divide en 7 módulos de 25 píxeles de alto por 240 píxeles de ancho cada uno.

Figura 6.3 División del espacio libre en módulos

Cada interfaz estará contenida en un panel contenedor de controles. Este panel será de un tamaño equivalente en módulos, al número de líneas de controles que necesite.

6.3.2 Aplicación de los archivos XAML

Para la descripción de la interfaz de cada paquete decidimos utilizar archivos XAML ya que, al tratarse de un "estándar" de Microsoft, disponíamos de un diseñador proporcionado por Visual Studio que nos facilitó el trabajo de composición de cada interfaz.

Otra razón por la que optamos por el uso de archivos XAML es porque se trata de un formato especial de archivos XML, lo cual nos facilita el trabajo gracias a las clases disponibles en Visual Studio para su manipulación, como ya hemos comentado en la sección 6.2. Para más información acerca de XAML ver apéndice A.

6.3.3 Estructura de los archivos XAML

Nuestros archivos XAML, para ser un XAML válido, constan de un nodo principal *window*. Este nodo principal contiene un nodo *Grid*, que representa el panel contenedor de la interfaz, que a su vez contiene todos los controles que necesita la interfaz.

Estos paneles *Grid* poseen los siguientes argumentos:

- Width: indica el ancho del panel, que en este caso siempre es de 240.
- Height: indica el alto del panel, que es igual al número de módulos que ocupa multiplicado por 25.
- VerticalAlignment: indica la situación en que está colocado el panel, que en nuestro caso es siempre "top" (arriba).
- Tag: este argumento es un argumento comodín que sirve para representar cualquier dato que no pueda ser expresado por los argumentos del control. En nuestro caso lo utilizamos para indicar el número de módulos que ocupa el panel.

En el caso de los paneles sólo utilizamos cuatro tipos de controles distintos que son: botones, etiquetas, listBox y listBoxItems. Estos controles poseen una serie de argumentos:

- Width: indica el ancho del control.
- Height: indica el alto del control, que debe ser menor que 25 (tamaño de un modulo).
- Name: nombre del control.
- Content: texto que muestra el control.

- Margin: este argumento consta de cuatro números separados por comas, que indican la distancia del control hasta el margen derecho, superior, izquierdo e inferior respectivamente.
- VerticalAlignment: indica la situación en que está colocado el control verticalmente, que en nuestro caso es siempre "top" (arriba).
- HorizontalAlignment: indica la situación en que está colocado el control horizontalmente, que en nuestro caso es siempre "left" (izquierda).

A continuación se muestra un ejemplo de interfaz en XAML:

```
<Window x:Class="WpfApplication1.faltas"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="faltas" Height="300" Width="300">
        <Grid Width="240" Height="50" VerticalAlignment="Top"
Tag="2">
         <ListBox Height="41" Margin="9,3,0,0" Name="ListaLocal"
                  Width="40" HorizontalAlignment="Left"
                  VerticalAlignment="Top" />
         <ListBox Height="41" Margin="0,3,13,0"
                  Name="ListaVisitante"          Width="40"
                  HorizontalAlignment="Right"
                  VerticalAlignment="Top" />
         <Button Height="20" Margin="52,3,0,0"
                  Name="faltas_boton1" Width="50" Content="Falta"
                  Click="metodo_faltas_boton1"
                  HorizontalAlignment="Left"
                  VerticalAlignment="Top"/>
         <Button Height="20" Margin="0,3,58,0"
                  Name="faltas_boton2" Width="50" Content="Falta"
                  HorizontalAlignment="Right"
                  VerticalAlignment="Top"/>
         <Button Height="20" Margin="52,24,0,4"
                  Name="faltas_boton3" Width="50" Content="Borrar"
                  HorizontalAlignment="Left" />
         <Button Height="20" Margin="0,24,58,6"
                  Name="faltas_boton4" Width="50" Content="Borrar"
                  HorizontalAlignment="Right" />
        </Grid>
            </Window>
```

Los distintos archivos de interfaz XAML son unidos posteriormente en un solo archivo mediante un método "juntarXAML". Este archivo resultante es el que finalmente se leerá para representar todas las interfaces en el espacio disponible.

La estructura de este archivo XAML resultante es la siguiente:

- Un nodo principal *Window*, que sirve para que el archivo sea un XAML valido.
- Un nodo *Grid*, que esta contenido dentro del nodo principal y representa al espacio disponible donde se van a representar las distintas interfaces.
- Nodos *Grid*, contenidos dentro del anterior, copia de los *Grid* provenientes de cada uno de los archivos de interfaz individuales. Estos *Grid* se disponen en orden de aparición de arriba abajo.

Para la ordenación de los *Grid* nos basamos en el nombre de los archivos que los contienen inicialmente. Estos archivos están nombrados con números, que representan su posición en el diagrama de características de la aplicación resultante.

A continuación se muestra un ejemplo del archivo resultante de unir dos interfaces:

```
<Window x:Class="WpfApplication1.cambios"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="faltas" Height="300" Width="300">
  <Grid xmlns="">
      <Grid Width="240" Height="25" VerticalAlignment="Top"
            Tag="1"
            xmlns="http://schemas.microsoft.com/winfx/2006/xaml/pr
            esentation">
        <Button HorizontalAlignment="Left" Margin="15,3,0,0"
            Name="De_2_en_2_boton1" Width="30" Content="X2"
            Height="20" VerticalAlignment="Top">
        </Button>
        <Button HorizontalAlignment="Right" Margin="0,3,45,0"
            Name="De_2_en_2_boton2" Width="30" Content="X2"
            Height="20" VerticalAlignment="Top">
        </Button>
      </Grid>
      <Grid Height="25" VerticalAlignment="Top" Tag="0"
         Margin="0,25,0,0"
         xmlns="http://schemas.microsoft.com/
                winfx/2006/xaml/presentation">
        <Button HorizontalAlignment="Left" Margin="45,-22,0,0"
            Name="De_3_en_3_boton1" Width="30" Content="X3"
        Height="20"
            VerticalAlignment="Top" />
```

```
        <Button HorizontalAlignment="Right" Margin="0,-22,15,0"
            Name="De_3_en_3_boton2" Width="30" Content="X3"
        Height="20"
            VerticalAlignment="Top" />
      </Grid>
  </Grid>
</Window>
```

6.3.4 Interfaz en tiempo de ejecución

Para cargar el archivo XAML resultante de la unión de todas las interfaces se utiliza un método "cargarXAML". Este método dispone de cuatro vectores de Paneles, Botones, ListBox y Labels. Dichos vectores, en su inicio, son vectores vacios que van creciendo y llenándose de forma dinámica.

Este método se encarga de leer el fichero XAML y a medida que encuentra en él un nodo *panel, botón, listbox o label*, crea un nuevo elemento en el vector correspondiente, con las características leídas de los argumentos del nodo, como por ejemplo el tamaño del elemento, el nombre o su contenido.

En el caso de los botones, además de cargar todas sus propiedades, se invoca un método "queMetodo", que se encarga de asignar a cada botón un evento de click en función de su nombre.

Otro caso especial es el de los listBoxItem, para los cuales se carece de un vector dinámico donde almacenarlo, por lo que se crean y se añaden directamente sobre su listBox correspondiente.

Además, cada control cargado se coloca en su panel correspondiente para después ser representado en su posición correcta.

6.3.5 El problema de los paneles transparentes.

Una vez cargado el archivo XAML, a continuación se ejecuta un método "dibujarPantalla", que es el encargado de representar todos los controles sobre el espacio disponible.

El problema que se plantea al realizar esta tarea es que, los paneles del espacio de nombres System.Windows.Form, poseen un color de fondo que nunca puede ser transparente y mostrar la imagen de fondo del formulario. Debido a esto, el espacio que ocupaban los paneles ocultaba la imagen de fondo de la aplicación con rectángulos de un color, consiguiendo una apariencia poco atractiva.

Para solucionar este problema creamos una clase nueva de paneles transparente o paneles imaginarios (clase PanelT), que utilizamos en lugar de los paneles clásicos de Windows Form. Dicha clase consta de las características básicas para almacenar información relativa al tamaño del panel, su posición, el número de módulos que ocupa y los controles que alberga; y de los métodos necesarios para poder leer o cambiar dichas características.

En el caso de los controles, utilizamos un vector de controles vacios en cada instancia de la clase, que irá creciendo y llenándose de manera dinámica al igual que se hacía en el método "cargarXAML".

Una vez solucionado el problema de los paneles, el método "dibujarPantalla" es el encargado de representar todos los controles en el espacio disponible, calculando su posición a partir de la posición relativa que ocupa cada control en su panel transparente correspondiente.

6.4 Solución al problema "Deshacer última acción"

Durante la implementación de la aplicación, nos surgió el problema de cómo deshacer la última acción. Éste es un apartado importante, ya que el usuario puede equivocarse en la introducción de una o varias faltas, y es algo que debe estar reflejado en la aplicación final. Tras estar investigando diferentes opciones, finalmente tomamos la determinación de implementar la estructura "Diccionario" facilitada por Visual Studio .NET. Dicha estructura consiste en un par de elementos:

- **Clave**: Valor que no debe modificarse mientras se utiliza el diccionario. La clave debe ser única en el diccionario. En nuestro caso como clave utilizamos el dorsal del jugador.
- **Valor**: dato al que se refiere una clave. En nuestra aplicación, el valor es una pila de elementos. En dicha pila se almacena el tipo de faltas que ha cometido el jugador. Para deshacer una falta, basta con extraer el último elemento insertado en la pila.

La estructura diccionario proporciona acceso al valor de una clave en un tiempo muy rápido (en O(1)), razón entre otras por la que escogimos esta estructura para el problema de deshacer la última acción.

6.5 Software utilizado

Para la elaboración de este proyecto así como de esta memoria se han utilizado los siguientes recursos software:

- Microsoft Windows XP Profesional Versión 2002. Service Pack 3.
- Microsoft Windows XP Home Edition Versión 2002. Service Pack 3.
- Microsoft Visual Studio 2008 Professional – ESN.
- Microsoft .NET Compact Framework 3.5.
- Microsoft ActiveSync 4.5.
- Windows Mobile 6 Professional SDK.
- Feature Modelling Tool.
- StarUML 5.0.2.1570.
- REM 1.2.2.
- Microsoft Office Word 2007.
- Microsoft Office PowerPoint 2007.
- Adobe Reader 9.1.2.
- PDF Creator.

6.6 Hardware utilizado

- Equipo de desarrollo 1:
  - Ordenador de sobremesa.
    - Procesador Intel Core 2 Duo E8400 a 3 GHz.
    - 3 GB memoria RAM.
    - 500 GB de disco duro.
    - S.O. Microsoft Windows XP Profesional Version 2002. Service Pack 3.
- Equipo de desarrollo 2:
  - Ordenador portátil.
    - Procesador Intel Pentium 4 a 1,73 GHz.
    - 1 GB memoria RAM.
    - 60 GB de disco duro.
    - S.O: Microsoft Windows XP Home Edition Versión 2002. Service Pack 3.
- Dispositivo móvil:
  - PDA Hewlett Packard.
    - Procesador SC32442-300MHz.
    - 23,71 MB memoria RAM
    - S.O: Microsoft Windows Mobile 5.0

Capítulo 7

PRUEBAS

7.1 Introducción

Uno de los últimos pasos realizados tras terminar la implementación del código fuente, ha sido la exhaustiva realización de pruebas para asegurarnos del correcto funcionamiento de la aplicación. El objetivo de esta fase no es convencerse de que el programa funciona bien sino ejercitarlo con la peor intención a fin de encontrarle fallos. Para ello hemos llevado a cabo una serie de casos de prueba diseñados para determinar si los requisitos establecidos durante el diseño del proyecto se han cumplido total o parcialmente.

En este apartado se presentarán las pruebas realizadas a las distintas aplicaciones obtenidas en la línea de productos desarrollada. La finalidad de las pruebas es detectar los posibles errores para su corrección.

Estas pruebas están centradas exclusivamente en el funcionamiento de cada paquete, sin tener en cuenta los posibles comportamientos que puede presentar cada uno de ellos al interactuar entre sí. Esto es debido a que las distintas combinaciones en una línea de productos software pueden ser muy numerosas. Por supuesto, estas otras pruebas deberán realizarse cada vez que se genere una aplicación distinta perteneciente a la línea de productos

Para la realización de las pruebas se ha utilizado, tanto el emulador de Pocket PC que ofrece el entorno de desarrollo de .NET, como una PDA. Las especificaciones técnicas de la misma pueden verse en el punto 6.6 de la memoria: “Hardware empleado”.

Se han realizado dos tipos de prueba:

- De comprobación de funcionalidad: Pruebas encaminadas a corregir los errores que hacen que no se cumpla la funcionalidad más básica pedida por el usuario.
- De casos límite: Pruebas encaminadas a encontrar errores que el desarrollador sospecha que pueden existir en un determinado caso.

7.2 Pruebas realizadas

7.2.1 Paquete Gestor Equipo Baloncesto

Descripción	Comprobación del botón CARGAR LOCAL
Acción realizada	Pulsación del botón CARGAR LOCAL en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar archivo de equipo
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.1 Comprobación del botón CARGAR LOCAL

Descripción	Comprobación del botón CARGAR VISITANTE
Acción realizada	Pulsación del botón CARGAR VISITANTE en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar archivo de equipo
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.2 Comprobación del botón CARGAR VISITANTE

Descripción	Comprobación de seleccionar equipo local (a)
Acción realizada	Selección archivo de equipo a cargar
Resultado esperado	Crea archivo local.xml
Resultado obtenido	Archivo creado correctamente
Observaciones	Correcto

Figura 7.3 Comprobación de seleccionar equipo local (a)

Descripción	Comprobación de seleccionar equipo visitante (a)
Acción realizada	Selección archivo de equipo a cargar sin paquete estadísticas incluido
Resultado esperado	Crea archivo visitante.xml
Resultado obtenido	Archivo creado correctamente
Observaciones	Correcto

Figura 7.4 Comprobación de seleccionar equipo visitante (a)

Descripción	Comprobación del botón OK en selección de jugadores (a)
Acción realizada	Pulsación del botón OK con número correcto de jugadores seleccionado
Resultado esperado	Actualiza jugadores titulares en la lista correspondiente
Resultado obtenido	Jugadores titulares añadidos
Observaciones	Correcto

Figura 7.5 Comprobación del botón OK en selección de jugadores (a)

Descripción	Comprobación del botón OK en selección de jugadores (b)
Acción realizada	Pulsación del botón OK con número incorrecto de jugadores seleccionado
Resultado esperado	Muestra mensaje con número correcto de jugadores
Resultado obtenido	Mensaje mostrado correctamente
Observaciones	Correcto

Figura 7.6 Comprobación del botón OK en selección de jugadores (b)

7.2.2 Paquete Ver Datos Equipo Baloncesto

Descripción	Comprobación del botón VER DATOS EQUIPO > EQUIPO LOCAL
Acción realizada	Pulsación del botón VER DATOS EQUIPO → EQUIPO LOCAL en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar jugador local
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.7 Comprobación del botón VER DATOS EQUIPO → EQUIPO LOCAL

Descripción	Comprobación del botón VER DATOS EQUIPO → EQUIPO VISITANTE
Acción realizada	Pulsación del botón VER DATOS EQUIPO → EQUIPO VISITANTE en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar jugador visitante
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.8 Comprobación del botón VER DATOS EQUIPO → EQUIPO VISITANTE

Descripción	Comprobación de selección de jugador en ventana VER DATOS EQUIPO
Acción realizada	Pulsación sobre el nombre de un jugador
Resultado esperado	Muestra datos estadísticos del jugador en cuadro de texto
Resultado obtenido	Datos mostrados correctamente
Observaciones	Correcto

Figura 7.9 Comprobación de selección de jugador en ventana VER DATOS EQUIPO

7.2.3 Paquete Estadísticas Baloncesto

Descripción	Comprobación del botón FALTA (a)
Acción realizada	Pulsación del botón FALTA con jugador seleccionado
Resultado esperado	Añade falta al jugador seleccionado
Resultado obtenido	Falta añadida correctamente
Observaciones	Correcto, muestra mensaje indicando la acción realizada

Figura 7.10 Comprobación del botón FALTA (a)

Descripción	Comprobación del botón FALTA (b)
Acción realizada	Pulsación del botón FALTA sin jugador seleccionado
Resultado esperado	No ejecuta ninguna acción
Resultado obtenido	Ninguna acción ejecutada
Observaciones	Correcto, muestra mensaje pidiendo seleccionar jugador

Figura 7.11 Comprobación del botón FALTA (b)

Descripción	Comprobación del botón BORRAR (a)
Acción realizada	Pulsación del botón BORRAR con jugador seleccionado
Resultado esperado	Borra una falta del jugador seleccionado
Resultado obtenido	Falta borrada correctamente
Observaciones	Correcto, muestra mensaje indicando la acción realizada

Figura 7.12 Comprobación del botón BORRAR (a)

Descripción	Comprobación del botón BORRAR (b)
Acción realizada	Pulsación del botón BORRAR sin jugador seleccionado
Resultado esperado	No ejecuta ninguna acción
Resultado obtenido	Ninguna acción ejecutada
Observaciones	Correcto, muestra mensaje pidiendo seleccionar jugador

Figura 7.13 Comprobación del botón BORRAR (b)

Descripción	Comprobación de seleccionar equipo local (b)
Acción realizada	Selección archivo de equipo a cargar con paquete estadísticas incluido
Resultado esperado	Muestra ventana con los miembros del equipo seleccionado y crea archivo local.xml y estadística<nombre_equipo>.xml
Resultado obtenido	Ventana mostrada y archivos creados correctamente
Observaciones	Correcto, aparece mensaje número de jugadores a seleccionar

Figura 7.14 Comprobación de seleccionar equipo local (b)

Descripción	Comprobación de seleccionar equipo visitante (b)
Acción realizada	Selección archivo de equipo a cargar con paquete estadísticas incluido
Resultado esperado	Muestra ventana con los miembros del equipo seleccionado y crea archivo visitante.xml y estadística<nombre_equipo>.xml
Resultado obtenido	Ventana mostrada y archivos creados correctamente
Observaciones	Correcto, aparece mensaje número de jugadores a seleccionar

Figura 7.15 Comprobación de seleccionar equipo visitante (b)

7.2.4 Paquete De 2 en 2

Descripción	Comprobación del botón X2 (a) local y visitante
Acción realizada	Pulsación del botón X2 sin jugador seleccionado
Resultado esperado	Añade 2 puntos al equipo correspondiente y actualiza el marcador
Resultado obtenido	Puntos añadidos y marcador actualizado correctamente
Observaciones	Correcto

Figura 7.16 Comprobación del botón X2 (a) local y visitante

Descripción	Comprobación del botón X2 (b) local y visitante
Acción realizada	Pulsación del botón X2 con jugador seleccionado
Resultado esperado	Añade 2 tantos al equipo y al jugador correspondiente, y actualiza el marcador
Resultado obtenido	Tantos añadidos y marcador actualizado correctamente
Observaciones	Correcto

Figura 7.17 Comprobación del botón X2 (b) local y visitante

7.2.5 Paquete De 3 en 3

Descripción	Comprobación del botón X3 (a) local y visitante
Acción realizada	Pulsación del botón X3 sin jugador seleccionado
Resultado esperado	Añade 3 tantos al equipo correspondiente y actualiza el marcador
Resultado obtenido	Tantos añadidos y marcador actualizado correctamente
Observaciones	Correcto

Figura 7.18 Comprobación del botón X3 (a) local y visitante

Descripción	Comprobación del botón X3 (b) local y visitante
Acción realizada	Pulsación del botón X3 con jugador seleccionado
Resultado esperado	Añade 3 tantos al equipo y al jugador correspondiente, y actualiza el marcador
Resultado obtenido	Tantos añadidos y marcador actualizado correctamente
Observaciones	Correcto

Figura 7.19 Comprobación del botón X3 (b) local y visitante

7.2.6 Paquete Cambios Baloncesto

Descripción	Comprobación del botón CAMBIO LOCAL y CAMBIO VISITANTE
Acción realizada	Pulsación del botón CAMBIO LOCAL o CAMBIO VISITANTE
Resultado esperado	Muestra ventana con los miembros del equipo seleccionado
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto, muestra mensaje con instrucciones para realizar el cambio

Figura 7.20 Comprobación del botón CAMBIO LOCAL y CAMBIO VISITANTE

7.2.7 Paquete Posesión

Descripción	Comprobación del botón REINICIAR
Acción realizada	Pulsación del botón REINICIAR
Resultado esperado	Reinicia a 24 el tiempo de posesión
Resultado obtenido	Tiempo reiniciado correctamente
Observaciones	Correcto

Figura 7.21 Comprobación del botón REINICIAR del paquete posesión

Descripción	Comprobación de finalización del tiempo de posesión
Acción realizada	Finalización del tiempo de posesión
Resultado esperado	Reinicia el tiempo de posesión y para el tiempo del partido
Resultado obtenido	Tiempo de posesión y tiempo de partido parado correctamente
Observaciones	Correcto

Figura 7.22 Comprobación de finalización del tiempo de posesión

7.2.8 Paquete Prórroga Baloncesto

Descripción	Comprobación de finalización del partido con empate
Acción realizada	Finalización del tiempo del partido con empate en el marcador
Resultado esperado	Inicia la prorroga con el tiempo correspondiente
Resultado obtenido	Prórroga iniciada correctamente
Observaciones	Correcto

Figura 7.23 Comprobación de finalización del partido con empate

7.2.9 Paquete Gestor Equipo Balonmano

Descripción	Comprobación del botón CARGAR LOCAL
Acción realizada	Pulsación del botón CARGAR LOCAL en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar archivo de equipo
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.24 Comprobación del botón CARGAR LOCAL

Descripción	Comprobación del botón CARGAR VISITANTE
Acción realizada	Pulsación del botón CARGAR VISITANTE en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar archivo de equipo
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.25 Comprobación del botón CARGAR VISITANTE

Descripción	Comprobación de seleccionar equipo local (a)
Acción realizada	Selección archivo de equipo a cargar
Resultado esperado	Crea archivo local.xml
Resultado obtenido	Archivo creado correctamente
Observaciones	Correcto

Figura 7.26 Comprobación de seleccionar equipo local (a)

Descripción	Comprobación de seleccionar equipo visitante (a)
Acción realizada	Selección archivo de equipo a cargar
Resultado esperado	Crea archivo visitante.xml
Resultado obtenido	Archivo creado correctamente
Observaciones	Correcto

Figura 7.27 Comprobación de seleccionar equipo visitante (a)

Descripción	Comprobación del botón OK en selección de jugadores (a)
Acción realizada	Pulsación del botón OK con número correcto de jugadores seleccionado
Resultado esperado	Actualiza jugadores titulares en la lista correspondiente
Resultado obtenido	Jugadores titulares añadidos
Observaciones	Correcto

Figura 7.28 Comprobación del botón OK en selección de jugadores (a)

Descripción	Comprobación del botón OK en selección de jugadores (b)
Acción realizada	Pulsación del botón OK con número incorrecto de jugadores seleccionado
Resultado esperado	Muestra mensaje con número correcto de jugadores
Resultado obtenido	Mensaje mostrado correctamente
Observaciones	Correcto

Figura 7.29 Comprobación del botón OK en selección de jugadores (b)

7.2.10 Paquete Ver Datos Equipo Balonmano

Descripción	Comprobación del botón VER DATOS EQUIPO → EQUIPO LOCAL
Acción realizada	Pulsación del botón VER DATOS EQUIPO → EQUIPO LOCAL en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar jugador local
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.30 Comprobación del botón VER DATOS EQUIPO → EQUIPO LOCAL

Descripción	Comprobación del botón VER DATOS EQUIPO → EQUIPO VISITANTE
Acción realizada	Pulsación del botón CARGAR VISITANTE en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar jugador visitante
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.31 Comprobación del botón VER DATOS EQUIPO → EQUIPO VISITANTE

Descripción	Comprobación de selección de jugador en ventana VER DATOS EQUIPO
Acción realizada	Pulsación sobre el nombre de un jugador
Resultado esperado	Muestra datos estadísticos del jugador en cuadro de texto
Resultado obtenido	Datos mostrados correctamente
Observaciones	Correcto

Figura 7.32 Comprobación de selección de jugador en ventana VER DATOS EQUIPO

7.2.11 Paquete Estadísticas Balonmano

Descripción	Comprobación del botón AMON. (a) local y visitante
Acción realizada	Pulsación del botón AMONESTACIÓN con jugador seleccionado
Resultado esperado	Añade amonestación al jugador correspondiente
Resultado obtenido	Amonestación añadida correctamente
Observaciones	Correcto

Figura 7.33 Comprobación del botón AMON. (a) local y visitante

Descripción	Comprobación del botón AMON.(b) local y visitante
Acción realizada	Pulsación del botón AMON. sin jugador seleccionado
Resultado esperado	Muestra mensaje indicando el error
Resultado obtenido	Mensaje mostrado correctamente
Observaciones	Correcto

Figura 7.34 Comprobación del botón AMON. (b) local y visitante

Descripción	Comprobación del botón EXCLUS. (a) local y visitante
Acción realizada	Pulsación del botón EXCLUS. con jugador seleccionado
Resultado esperado	Añade exclusión al jugador correspondiente y inicia el temporizador de exclusión
Resultado obtenido	Exclusión añadida y temporizador iniciado correctamente
Observaciones	Correcto

Figura 7.35 Comprobación del botón EXCLUS. (a) local y visitante

Descripción	Comprobación del botón EXCLUS. (b) local y visitante
Acción realizada	Pulsación del botón EXCLUS. sin jugador seleccionado
Resultado esperado	Muestra mensaje indicando el error
Resultado obtenido	Mensaje mostrado correctamente
Observaciones	Correcto

Figura 7.36 Comprobación del botón EXCLUS. (b) local y visitante

Descripción	Comprobación del botón DESC. (a) local y visitante
Acción realizada	Pulsación del botón DESC. con jugador seleccionado
Resultado esperado	Añade descalificación al jugador correspondiente e inicia el temporizador de descalificación.
Resultado obtenido	Descalificación añadida y temporizador iniciado correctamente.
Observaciones	Correcto

Figura 7.37 Comprobación del botón DESC. (a) local y visitante

Descripción	Comprobación del botón DESC. (b) local y visitante
Acción realizada	Pulsación del botón DESC. sin jugador seleccionado
Resultado esperado	Muestra mensaje indicando el error
Resultado obtenido	Mensaje mostrado correctamente
Observaciones	Correcto

Figura 7.38 Comprobación del botón DESC. (b) local y visitante

Descripción	Comprobación del botón EXPUL. (a) local y visitante
Acción realizada	Pulsación del botón EXPUL. con jugador seleccionado
Resultado esperado	Añade expulsión al jugador correspondiente.
Resultado obtenido	Expulsión añadida.
Observaciones	Correcto

Figura 7.39 Comprobación del botón EXPUL. (a) local y visitante

Descripción	Comprobación del botón EXPUL. (b) local y visitante
Acción realizada	Pulsación del botón EXPUL. sin jugador seleccionado
Resultado esperado	Muestra mensaje indicando el error
Resultado obtenido	Mensaje mostrado correctamente
Observaciones	Correcto

Figura 7.40 Comprobación del botón EXPUL. (b) local y visitante

Descripción	Comprobación de seleccionar equipo local (b)
Acción realizada	Selección archivo de equipo a cargar con paquete estadísticas incluido
Resultado esperado	Muestra ventana con los miembros del equipo seleccionado y crea archivo local.xml y estadística<nombre_equipo>.xml
Resultado obtenido	Ventana mostrada y archivos creados correctamente
Observaciones	Correcto, aparece mensaje número de jugadores a seleccionar

Figura 7.41 Comprobación de seleccionar equipo local (b)

Descripción	Comprobación de seleccionar equipo visitante (b)
Acción realizada	Selección archivo de equipo a cargar con paquete estadísticas incluido
Resultado esperado	Muestra ventana con los miembros del equipo seleccionado y crea archivo visitante.xml y estadística<nombre_equipo>.xml
Resultado obtenido	Ventana mostrada y archivos creados correctamente
Observaciones	Correcto, aparece mensaje número de jugadores a seleccionar

Figura 7.42 Comprobación de seleccionar equipo visitante (b)

7.2.12 Paquete Cambios Balonmano

Descripción	Comprobación del botón CAMBIO LOCAL y CAMBIO VISITANTE
Acción realizada	Pulsación del botón CAMBIO LOCAL o CAMBIO VISITANTE
Resultado esperado	Muestra ventana con los miembros del equipo seleccionado
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto, muestra mensaje con instrucciones para realizar el cambio

Figura 7.43 Comprobación del botón CAMBIO LOCAL y CAMBIO VISITANTE

7.2.13 Paquete Prórroga Balonmano

Descripción	Comprobación de finalización del partido con empate
Acción realizada	Finalización del tiempo del partido con empate en el marcador
Resultado esperado	Inicia la prorroga con el tiempo correspondiente
Resultado obtenido	Prórroga iniciada correctamente
Observaciones	Correcto

Figura 7.44 Comprobación de finalización del partido con empate

7.2.14 Paquete Gestor Equipo Futbol Sala

Descripción	Comprobación del botón CARGAR LOCAL
Acción realizada	Pulsación del botón CARGAR LOCAL en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar archivo de equipo
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.45 Comprobación del botón CARGAR LOCAL

Descripción	Comprobación del botón CARGAR VISITANTE
Acción realizada	Pulsación del botón CARGAR VISITANTE en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar archivo de equipo
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.46 Comprobación del botón CARGAR VISITANTE

Descripción	Comprobación de seleccionar equipo local (a)
Acción realizada	Selección archivo de equipo a cargar
Resultado esperado	Crea archivo local.xml
Resultado obtenido	Archivo creado correctamente
Observaciones	Correcto

Figura 7.47 Comprobación de seleccionar equipo local (a)

Descripción	Comprobación de seleccionar equipo visitante (a)
Acción realizada	Selección archivo de equipo a cargar
Resultado esperado	Crea archivo visitante.xml
Resultado obtenido	Archivo creado correctamente
Observaciones	Correcto

Figura 7.48 Comprobación de seleccionar equipo visitante (a)

Descripción	Comprobación del botón OK en selección de jugadores (a)
Acción realizada	Pulsación del botón OK con número correcto de jugadores seleccionado
Resultado esperado	Actualiza jugadores titulares en la lista correspondiente
Resultado obtenido	Jugadores titulares añadidos
Observaciones	Correcto

Figura 7.49 Comprobación del botón OK en selección de jugadores (a)

Descripción	Comprobación del botón OK en selección de jugadores (b)
Acción realizada	Pulsación del botón OK con número incorrecto de jugadores seleccionado
Resultado esperado	Muestra mensaje con número correcto de jugadores
Resultado obtenido	Mensaje mostrado correctamente
Observaciones	Correcto

Figura 7.50 Comprobación del botón OK en selección de jugadores (b)

7.2.15 Paquete Ver Datos Equipo Futbol Sala

Descripción	Comprobación del botón VER DATOS EQUIPO → EQUIPO LOCAL
Acción realizada	Pulsación del botón VER DATOS EQUIPO → EQUIPO LOCAL en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar jugador local
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.51 Comprobación del botón VER DATOS EQUIPO → EQUIPO LOCAL

Descripción	Comprobación del botón VER DATOS EQUIPO → EQUIPO VISITANTE
Acción realizada	Pulsación del botón VER DATOS EQUIPO → EQUIPO VISITANTE en cualquier configuración
Resultado esperado	Muestra ventana para seleccionar jugador visitante
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto

Figura 7.52 Comprobación del botón VER DATOS EQUIPO > EQUIPO VISITANTE

Descripción	Comprobación de selección de jugador en ventana VER DATOS EQUIPO
Acción realizada	Pulsación sobre el nombre de un jugador
Resultado esperado	Muestra datos estadísticos del jugador en cuadro de texto
Resultado obtenido	Datos mostrados correctamente
Observaciones	Correcto

Figura 7.53 Comprobación de selección de jugador en ventana VER DATOS EQUIPO

7.2.16 Paquete Estadísticas Futbol Sala

Descripción	Comprobación del botón AMARILLA (a) local y visitante
Acción realizada	Pulsación del botón AMARILLA con jugador seleccionado
Resultado esperado	Añade una tarjeta amarilla al jugador correspondiente
Resultado obtenido	Tarjeta amarilla añadida al jugador correctamente
Observaciones	Correcto, muestra mensaje indicando la acción realizada

Figura 7.54 Comprobación del botón AMARILLA (a) local y visitante

Descripción	Comprobación del botón AMARILLA (b) local y visitante
Acción realizada	Pulsación del botón AMARILLA sin jugador seleccionado
Resultado esperado	Muestra mensaje indicando el error
Resultado obtenido	Mensaje mostrado correctamente
Observaciones	Correcto

Figura 7.55 Comprobación del botón AMARILLA (b) local y visitante

Descripción	Comprobación del botón ROJA (a) local y visitante
Acción realizada	Pulsación del botón ROJA con jugador seleccionado
Resultado esperado	Añade una tarjeta roja al jugador correspondiente y inicia el temporizador de expulsión correspondiente
Resultado obtenido	Tarjeta roja añadida al jugador y temporizador iniciado correctamente
Observaciones	Correcto

Figura 7.56 Comprobación del botón ROJA (a) local y visitante

Descripción	Comprobación del botón ROJA (b) local y visitante
Acción realizada	Pulsación del botón ROJA sin jugador seleccionado
Resultado esperado	Muestra mensaje indicando el error
Resultado obtenido	Mensaje mostrado correctamente
Observaciones	Correcto

Figura 7.57 Comprobación del botón ROJA (b) local y visitante

Descripción	Comprobación de seleccionar equipo local (b)
Acción realizada	Selección archivo de equipo a cargar con paquete estadísticas incluido
Resultado esperado	Muestra ventana con los miembros del equipo seleccionado y crea archivo local.xml y estadística<nombre_equipo>.xml
Resultado obtenido	Ventana mostrada y archivos creados correctamente
Observaciones	Correcto, aparece mensaje número de jugadores a seleccionar

Figura 7.58 Comprobación de seleccionar equipo local (b)

Descripción	Comprobación de seleccionar equipo visitante (b)
Acción realizada	Selección archivo de equipo a cargar con paquete estadísticas incluido
Resultado esperado	Muestra ventana con los miembros del equipo seleccionado y crea archivo visitante.xml y estadística<nombre_equipo>.xml
Resultado obtenido	Ventana mostrada y archivos creados correctamente
Observaciones	Correcto, aparece mensaje número de jugadores a seleccionar

Figura 7.59 Comprobación de seleccionar equipo visitante (b)

7.2.17 Paquete Cambios Fútbol Sala

Descripción	Comprobación del botón CAMBIO LOCAL y CAMBIO VISITANTE
Acción realizada	Pulsación del botón CAMBIO LOCAL o CAMBIO VISITANTE
Resultado esperado	Muestra ventana con los miembros del equipo seleccionado
Resultado obtenido	Ventana mostrada correctamente
Observaciones	Correcto, muestra mensaje con instrucciones para realizar el cambio

Figura 7.60 Comprobación del botón CAMBIO LOCAL y CAMBIO VISITANTE

7.2.18 Paquete Prórroga Futbol Sala

Descripción	Comprobación de finalización del partido con empate
Acción realizada	Finalización del tempo del partido con empate en el marcador
Resultado esperado	Inicia la prórroga con el tiempo correspondiente
Resultado obtenido	Prórroga iniciada correctamente
Observaciones	Correcto

Figura 7.61 Comprobación de finalización del partido con empate

7.2.19 Paquete deshacer

Descripción	Comprobación del botón DESHACER (a) local y visitante
Acción realizada	Pulsación del botón DESHACER sin jugador seleccionado
Resultado esperado	Borra un tanto al marcador correspondiente y actualiza el marcador
Resultado obtenido	Tanto borrado y marcador actualizado correctamente
Observaciones	Correcto

Figura 7.62 Comprobación del botón DESHACER (a) local y visitante

Descripción	Comprobación del botón DESHACER (b) local y visitante
Acción realizada	Pulsación del botón DESHACER con jugador seleccionado
Resultado esperado	Borra un tanto al marcador y al jugador correspondiente, y actualiza el marcador
Resultado obtenido	Tanto borrado y marcador actualizado correctamente
Observaciones	Correcto

Figura 7.63 Comprobación del botón DESHACER (b) local y visitante

7.2.20 Paquete base

Descripción	Comprobación del botón INICIAR del menú
Acción realizada	Pulsación del botón INICIAR del menú
Resultado esperado	Inicia marcadores y activa todos los botones
Resultado obtenido	Marcadores iniciados y botones activos correctamente
Observaciones	Correcto

Figura 7.64 Comprobación del botón INICIAR del menú

Descripción	Comprobación del botón REINICIAR del menú
Acción realizada	Pulsación del botón REINICIAR del menú
Resultado esperado	Reinicia marcadores y pone a cero el temporizador y las estadísticas
Resultado obtenido	Marcadores reiniciados y temporizador y estadísticas puestos a cero
Observaciones	Correcto

Figura 7.65 Comprobación del botón REINICIAR del menú

PARTE 3

MANUAL DE USUARIO

Capítulo 8

MANUAL DE USUARIO

8.1 Introducción

En este capítulo se describirá el manual de usuario de las distintas aplicaciones que pueden obtenerse a través de la línea de productos del Marcador Deportivo. Se pretende que éste sea un manual de referencia para lograr un correcto funcionamiento de todas las funcionalidades que ofrece la aplicación, y para ello en cada paso se mostrará la pantalla que se podrá ver en nuestro dispositivo móvil. Además se ofrecerá una detallada explicación de todo lo que el marcador puede ofrecer, así como de los elementos que aparezcan en la interfaz para lograr un completo conocimiento de la aplicación.

Como se ha comentado, la línea de productos del Marcador Deportivo puede dar lugar a variadas y distintas aplicaciones. En este manual de usuario se explicarán todas las posibles soluciones, así como se mostrarán todas las pantallas que las distintas aplicaciones puedan ofrecer.

8.2 Descripción de la aplicación

Esta aplicación consiste en un Marcador Deportivo que permite la conexión por bluetooh de varios dispositivos a uno principal que ejerce las funciones de maestro. Los dispositivos que se conecten al mismo pueden observar todas las incidencias del partido.

Por su parte, el dispositivo maestro puede efectuar las siguientes acciones:

- Crear un partido de baloncesto/balonmano/fútbol sala (dependiendo de la aplicación generada de la línea de productos).
- Seleccionar los equipos local y visitante que disputarán el partido de entre los disponibles en el dispositivo móvil.
- Indicar qué jugadores serán los que disputen el encuentro desde el inicio del mismo, los titulares de cada equipo.
- Ejercer de cronometrador del partido, pudiendo iniciar y detener el tiempo cuando lo desee, así como llevar control de los segundos de posesión (en baloncesto), o del tiempo restante de exclusión (en balonmano y fútbol sala).
- Llevar un control de las estadísticas personalizadas de cada jugador: tantos o goles anotados, faltas cometidas con su correspondiente sanción, minutos disputados…
- Ver en todo momento los datos del equipo y de cada jugador en particular.

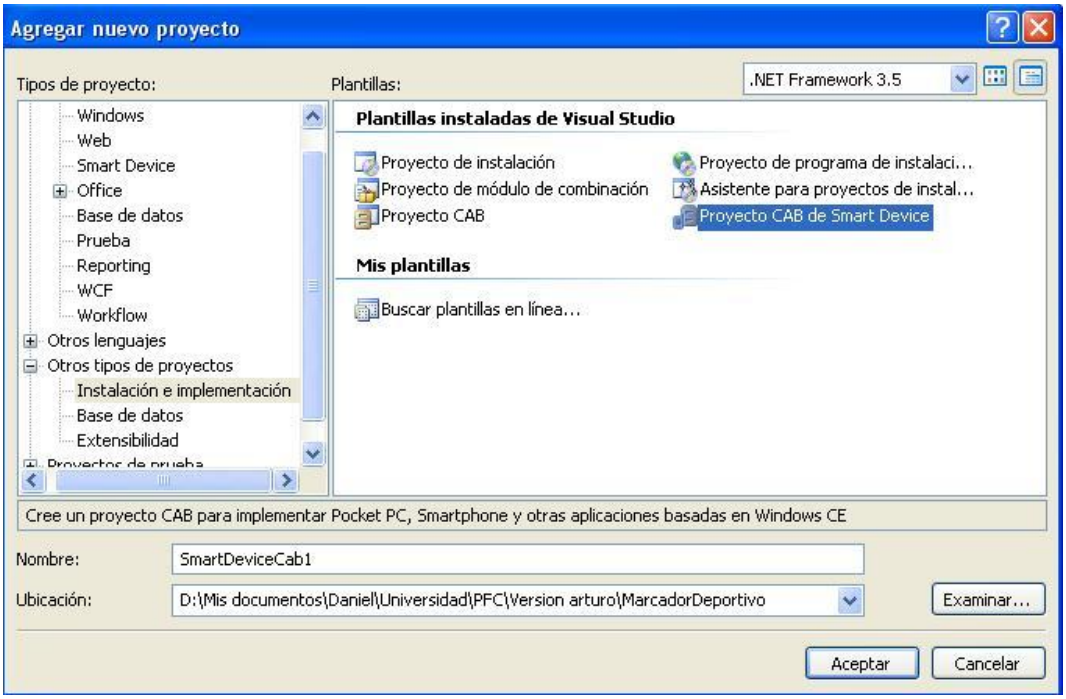
- Realizar los cambios que se sucedan durante el partido.
- Deshacer las acciones que por error haya podido efectuar en la aplicación.

En los siguientes apartados se procederá a detallar tanto la instalación como el uso de la aplicación.

8.3 Instalación de la aplicación

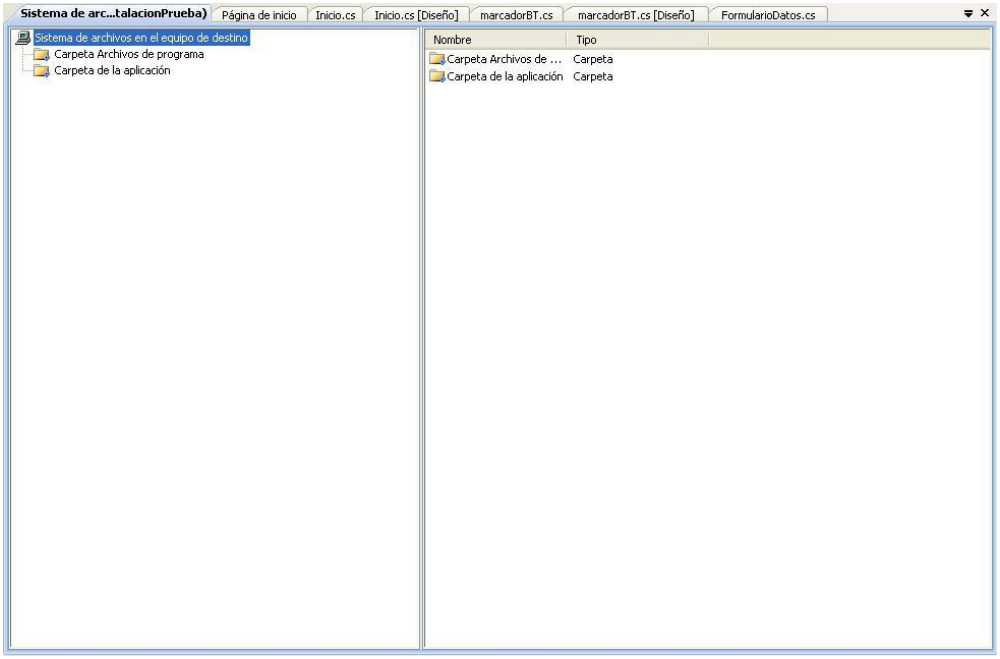
Para proceder a la instalación de la aplicación en primer lugar se debe generar el archivo .CAB correspondiente (si no está generado ya). Desde Microsoft Visual Studio 2008 este puede generarse siguiendo los siguientes pasos:

- Seleccionar el menú "archivo" de la barra de herramientas.
- Desplegar la opción "agregar" y posteriormente elegir "nuevo proyecto"
- Hacer click en la opción "Proyecto CAB de Smart Device". Se debe tener seleccionada en la lista desplegable que aparece en la esquina superior derecha la opción ".NET Framework 3.5".



8.1 Creación del proyecto CAB

- Tras esto, se nos mostrará una pantalla como la siguiente, en la que deberemos añadir las carpetas y archivos necesarios para llevar a cabo el proyecto de instalación:

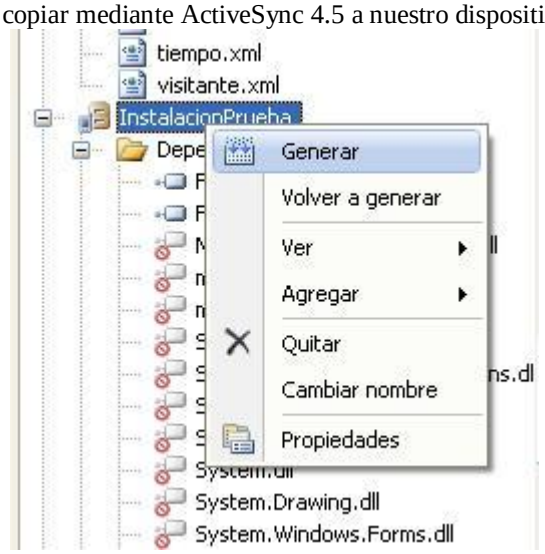


8.2: Sistema de archivos

- Dentro de la carpeta de la carpeta de la aplicación seleccionaremos "agregar" → "resultados de proyecto", seleccionando en la misma nuestro proyecto (MarcadorDeportivo), y todos los elementos que ahí aparecen.
- Con el botón derecho del ratón hacemos click en "Sistema de Archivos en el equipo de destino", para acceder al menú desplegable, en que deberemos seleccionar la opción "Agregar carpeta especial" → "Carpeta Mis Documentos". Dentro de esta carpeta crearemos un directorio MD. De la misma manera, deberán crearse dentro del directorio "MD" tantos nuevos archivos como equipos tengamos en nuestro proyecto de instalación (archivos .xml). Dentro de la carpeta "Interfaces" deberán incluirse los archivos .xaml que estén incluidos en las carpetas activas en el proyecto.

8.3: Aspecto final de los archivos

- Por último hay que generar el archivo .CAB para instalarlo en nuestro dispositivo móvil. Para ello seleccionamos nuestro archivo de instalación en el explorador de soluciones de Visual Studio 2008 (parte derecha de la pantalla), y en el menú desplegable que aparece al pinchar con el botón derecho seleccionamos "Generar". Con esto obtenemos el .CAB, que hay que copiar mediante ActiveSync 4.5 a nuestro dispositivo móvil, e instalarlo allí.



8.4: Generación del proyecto CAB

Una vez seguidos todos estos pasos (que pueden omitirse si ya disponemos del .CAB correspondiente), deberemos instalar el programa en nuestro dispositivo móvil y ejecutarlo. A continuación se explicará todo lo que puede hacerse con la aplicación.

8.4 Pantalla de bienvenida

La aplicación mostrará esta pantalla en la que puede observarse el título de la aplicación, así como los desarrolladores de la misma. Tiene un acceso al menú principal en la parte inferior izquierda, al que se puede acceder haciendo click en el mismo.

8.5: Pantalla de bienvenida

8.6: Menú pantalla de bienvenida

El menú principal tiene tres opciones:

- **Crear partido**: como su propio nombre indica, la utilidad de este botón es la de generar un nuevo partido del deporte que tenga cargada la aplicación.
- **Buscar**: utiliza el bluetooth para buscar dispositivos a los que conectarse que para visualizar otros partidos.
- **Salir**: sirve para salir de la aplicación

Pasaremos ahora a detallar la funcionalidad del marcador después de presionar la opción del menú "crear partido". Debido a que puede haber tres versiones diferentes correspondientes a los diferentes deportes que puede tener la aplicación, pasaremos a explicar en detalle una por una.

8.5 Baloncesto

8.5.1 Pantalla principal

Dentro de la propia línea de productos de baloncesto pueden generarse distintas aplicaciones en función de los paquetes que estén instalados. Es por ello que la pantalla principal puede variar. Algunas de las diferentes formas que puede adoptar son:

8.7: Pantalla principal sin botones

8.8: Pantalla principal con estadísticas

8.9: Pantalla principal con botón x2 y x3

8.10: Pantalla principal completa

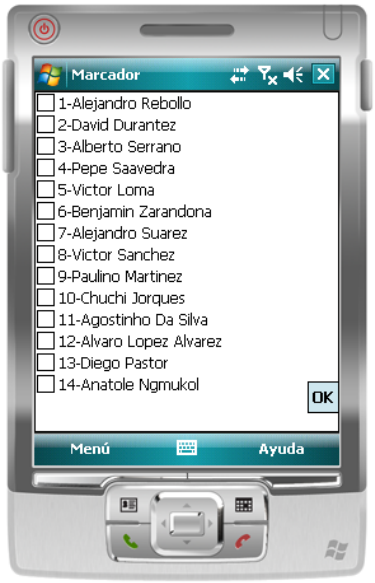
Para el desarrollo de este manual de usuario utilizaremos la versión de la línea de productos de baloncesto que tiene la interfaz completa (la de la figura 8.10) para poder así explicar así todas las funcionalidades.

8.5.2 Pantalla Cargar Equipos

Al entrar en la pantalla principal, todos los botones se hayan deshabilitados, excepto el del menú. El usuario debe entrar en el menú para cargar los dos equipos que vayan a jugar el partido (en caso de que el paquete de gestor equipos no esté incluido, el usuario debería seleccionar la opción de "iniciar", que estaría habilitada). El usuario, al pinchar bien en "cargar local" o "cargar visitante", podrá seleccionar de entre la lista de equipos que estén previamente cargados en el dispositivo móvil.

8.11: Menú para cargar equipos

8.12: Cuadro para elegir los equipos



8.13: Seleccionar titulares

Si el usuario se ha equivocado al elegir la opción bien de "cargar local" o bien "cargar visitante", siempre tiene la opción de volver a la pantalla principal presionando el botón "Volver".

Al elegir un equipo se pasa a la pantalla en la que se deben seleccionar los jugadores titulares del mismo. En el caso del baloncesto, solamente deben seleccionarse 5 jugadores (los que saltarán de inicio a la cancha). La aplicación no dejará continuar si no son 5 los seleccionados (informará de ello a través de un mensaje). Si la aplicación no ha implementado el paquete "estadísticas", no mostrará la pantalla de la figura 8.13, si no que volverá a la pantalla principal mostrando el nombre de los equipos en la parte superior del formulario.

8.5.3 Funcionalidad del Marcador de Baloncesto

Una vez seleccionados los dos equipos así como sus jugadores titulares, el programa habilitará automáticamente la opción del menú iniciar. El usuario debe de seleccionar esta opción para iniciar el partido.

La pantalla mostrará ahora:

- El nombre de ambos equipos en la parte superior (si está cargado el paquete para mostrar el nombre de los equipos. En caso contrario, mostrará "Local" y "Visitante").
- El marcador inicial (0 puntos para cada equipo).
- El tiempo que resta de la parte (10 minutos)
- La parte en la que nos encontramos de todas las que hay (en este caso, en la parte 1 de las 4 que tiene el partido).
- El cronómetro de posesión (si la aplicación implementa dicho paquete).
- Dos listas en las que están cargados los dorsales de los jugadores que se encuentran sobre el terreno de juego.
- Varios botones para realizar diversas acciones.

8.14: Pantalla principal con los equipos cargados

8.15: Botones para controlar la anotación

El partido no se pondrá en marcha hasta que el usuario no seleccione el botón "Continuar", momento en el cual empezará a correr el tiempo y se podrán anotar los puntos.

Una vez se ha pulsado el botón continuar, el usuario puede anotar los puntos haciendo click en los botones correspondientes. En baloncesto las canastas se dividen en sencillas (tiros libres, cuyo valor es de 1), canastas normales (su valor es de 2 puntos) y triples (cuentan por 3). Todo ello se puede tener en cuenta para seleccionar el tipo de canasta adecuado. Además, si se desea guardar en las estadísticas el jugador que ha anotado la canasta, se debe seleccionar en la lista que hay en el marcador.

Cada vez que hay una variación en el marcador el cronómetro de posesión (que se sitúa justo debajo del botón continuar), vuelve a los 24 segundos iniciales. El cronómetro de posesión puede reiniciarse cada vez que

cambie el equipo que tiene el balón sin necesidad de parar el tiempo. Para ello basta con pulsar el botón reiniciar. Para efectuar cualquier otra acción (una falta, un cambio), es necesario que el tiempo esté parado. Para parar y reanudar el partido se debe pulsar el botón continuar. Automáticamente el partido se para cuando la posesión llega a 0 segundos, o bien cuando una parte llega a su fin. Además, cuando restan menos de 24 segundos para llegar al final del partido, y la posesión se reinicia, desaparece el tiempo de posesión, al ser menor que lo que queda para terminar la parte.

Si el usuario se ha equivocado al introducir la puntuación puede presionar el botón "Deshacer", mediante el cual se restará un tanto a la puntuación seleccionada.

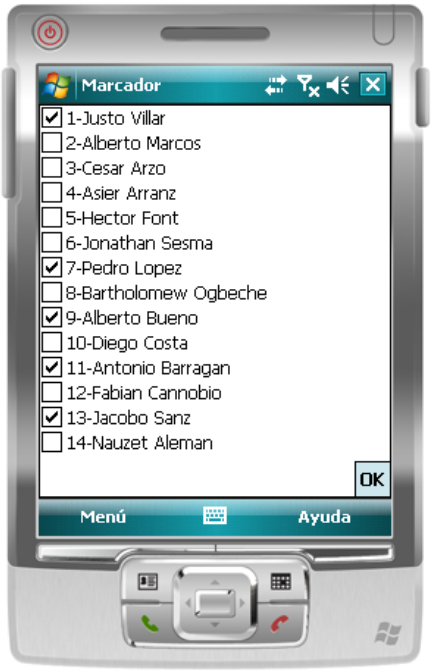
8.16: Anotación de una falta

Para anotar una falta, es preciso que el tiempo esté parado (según las reglas del baloncesto). Si se trata de anotar una falta a un jugador mientras el tiempo está corriendo, el sistema informa del error y no anota la falta. Además es necesario que haya un jugador seleccionado en la lista al anotar la falta. Si no es así el sistema también informa del error. En caso contrario el sistema anota la falta correctamente, como puede observarse en la figura 8.16.

El sistema también contempla la posibilidad de que el usuario se equivoque al anotar una falta. Para ello la interfaz tiene un botón "Borrar" al lado de cada lista (local o visitante), para eliminar una falta del jugador. Si el jugador no tiene ninguna falta anotada, el sistema avisa de que no puede eliminar ninguna falta del jugador.

8.5.4 Pantalla Cambios

Siempre que el tiempo esté parado el usuario tiene la posibilidad de introducir los cambios que se efectúen en los equipos. Para ello no tiene más que pulsar en el botón "Cambio local" o "Cambio Visitante" (dependiendo del equipo que realice el cambio de jugadores). Si dicho botón se presiona cuando el tiempo está corriendo, el sistema informa de que no se pueden realizar los cambios. En caso contrario, se indica al usuario que desmarque a los jugadores que salgan de la cancha y marque a aquellos que entren en la lista que se mostrará posteriormente (la de la figura 8.17). La lista que se muestra tiene seleccionados a aquellos jugadores que están en ese momento en el campo (bien porque salieron como titulares o bien porque han entrado en el terreno de juego a través de algún cambio). Así pues el usuario debe seleccionar los jugadores que tras los cambios estarán disputando el partido. El sistema comprueba que se han seleccionado 5 jugadores (si no no deja continuar), y vuelve a la pantalla principal.



8.17: Pantalla de cambios

8.5.5 Pantalla Ver Datos Equipo

8.18: Menú ver datos equipo

En cualquier momento del partido el usuario puede ver los datos de cualquiera de los dos equipos. Para ello el usuario debe ir al menú principal, y ahí seleccionar "Gestor Equipos" → "Ver Datos equipos" y posteriormente el equipo del que quiera ver los datos, el local o el visitante.

Tras esto, se abrirá una pantalla en la que podrá verse el nombre del equipo seleccionado. La pantalla tiene una lista desplegable en la que se encuentran todos los jugadores del equipo. El usuario puede seleccionar cualquiera de los jugadores, y una vez hecho esto se mostrarán sus datos del partido:

- Nombre.
- Dorsal.
- Puntos anotados durante el partido.
- Faltas cometidas durante el partido.
- Minutos disputados en el partido.

Para volver a la pantalla principal, el usuario debe presionar el botón "Volver" en la esquina inferior izquierda.

8.19: Aspecto de la pantalla Ver Datos

8.20: Datos del jugador seleccionado

Una vez que el partido finaliza se indica al usuario a través de un cuadro de información. A partir de ese momento ya no puede introducirse ningún nuevo dato a las estadísticas. Con el partido finalizado, el usuario tiene la opción de salir de la aplicación, o de reiniciar un nuevo partido (dicha opción de reiniciar está siempre presente a lo largo de la duración del partido).

8.21: Fin del partido

En baloncesto, si un partido acaba en empate a tantos, se disputan tantas prórrogas de 5 minutos como sean necesarias. Así pues, en nuestra aplicación, tras mostrar la pantalla anterior, en caso de acabar con el mismo resultado ambos equipos se pasaría a disputar la prórroga, mostrándose la misma en la pantalla del marcador. Si acabada la prórroga se volviera a quedar empate, nuevamente se iniciarían otros 5 minutos de prórroga. Si no, el partido finalizaría mostrando el mismo mensaje de la figura 8.21.

8.22: Prórroga

8.6 Balonmano

8.6.1. Pantalla principal

En balonmano, al igual que en los demás deportes que soporta la aplicación, la pantalla principal puede ser diferente en función de los paquetes que estén cargados en la aplicación. Así pues, la pantalla principal puede mostrarse de la siguiente forma:

8.23: Pantalla principal sin botones

8.24: Pantalla principal con estadísticas

8.25: Pantalla principal con etiquetas de exclusión

8.26: Pantalla principal completa

A lo largo de este manual de usuario utilizaremos la pantalla de la figura 8.26 para referirnos a las utilidades de la aplicación, ya que todo lo que contienen las demás pantallas de las diferentes aplicaciones de la línea de productos de balonmano lo tiene también dicha pantalla (al ser la que tiene todos los posibles paquetes integrados).

8.6.2 Pantalla Cargar Equipos

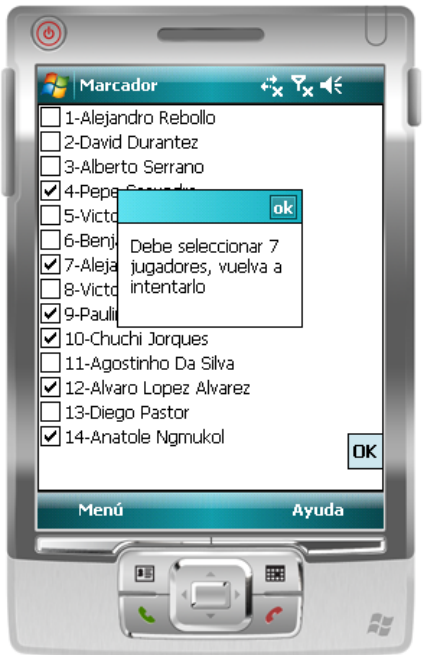
Al crear un partido de balonmano, el usuario deberá en primer lugar seleccionar la opción en el menú principal de "Gestor Equipos" → "Cargar Local" o "Cargar Visitante". La interfaz, como puede verse en las figuras 8.26 y 8.27, es la misma que en baloncesto. Únicamente es diferente la imagen de fondo que podemos observar en la aplicación.

8.27: Menú Gestor Equipos

8.28: Cuadro para seleccionar los equipos

U

na vez seleccionado el equipo se pasa a la pantalla para seleccionar los titulares (figura 8.13). En balonmano las reglas obligan a que haya 7 jugadores titulares por cada equipo. Es por ello que esta aplicación comprueba el número de jugadores seleccionados, y no deja continuar hasta que no haya 7 jugadores marcados. Mostraría un mensaje de error como el de la figura 8.29. Al igual que en baloncesto, si el paquete estadísticas no está cargado, la pantalla de seleccionar titulares no se mostrará.



8.29: Error en la selección de titulares

8.6.3 Funcionalidad del Marcador de Balonmano

Una vez que se han cargado los equipos, se habilita la opción de "iniciar" en el menú principal. Seleccionada esa opción, se nos presenta una pantalla como la de figura 8.30 que contiene lo siguiente:

- En la parte superior el nombre de los dos equipos cargados (si el paquete de Mostrar Nombre no está cargado, mostrará "Local" y "Visitante").
- La puntuación de los dos equipos.
- El tiempo que resta de la parte (en este caso, al inicio de la parte, restan 30 minutos).
- La parte del partido en la que nos encontramos, y el número de partes en que se divide el partido (en este caso, estamos en la primera parte de un total de 2).
- Dos recuadros azules vacíos (que servirán para llevar el tiempo de exclusión de los jugadores que se explicará más adelante).
- Dos listas en las que están cargados los dorsales de los jugadores titulares de ambos equipos.
- Una serie de botones (anotar, continuar, deshacer, amon., exclus., desc., expul., borrar y cambio) para llevar el control de la puntuación y de las estadísticas de cada jugador.

8.30: Pantalla principal con los equipos cargados

8.31: Distintos tipos de faltas

El marcador se pondrá en marcha cuando el usuario pulse el botón "Continuar". A partir de ese momento, el cronómetro comenzará a correr, y se podrán anotar los goles obtenidos por cada equipo. Para anotar los goles de los equipos basta con pulsar el botón "Anotar". Si hay algún jugador seleccionado en la lista correspondiente, el gol se añadirá a las estadísticas de dicho jugador. Si no hay ninguno seleccionado el gol se contará igualmente, aunque no se incluirá en las estadísticas del jugador. Si el usuario ha cometido algún error en la anotación, puede deshacer su acción haciendo click en el botón "Deshacer". De esta manera, se restará uno a los goles del equipo.

Debido a que en balonmano existen distintos tipos de faltas y cada una conlleva una sanción diferente,

la aplicación diferencia entre las diferentes faltas (amonestación, exclusión, descalificación y expulsión) para incluirlas en las estadísticas de cada jugador:

- **Amonestaciones**: Cada jugador solo debería recibir una como máximo, y el equipo 3.
- **Exclusiones**: El jugador que la reciba deberá abandonar el terreno de juego durante dos minutos. Cuando suceda se pondrá en marcha un cronómetro de 2 minutos, y el jugador no podrá entrar hasta que este acabe (ejemplo en la figura 8.31).
- **Descalificación**: El jugador que la reciba no podrá volver a entrar en el terreno de juego, y su equipo jugará con un jugador menos durante 2 minutos, activándose cronómetro de 2 minutos.
- **Expulsión**: El jugador receptor de la misma no puede volver al terreno de juego y su equipo jugará con un jugador menos lo que resta de partido.

Al anotar cualquiera de los 4 tipos de falta debe haber un jugador seleccionado en la lista. En caso de no haberlo, la aplicación mostraría un mensaje de error avisando de la causa del mismo y no registraría la falta.

El botón borrar que aparece debajo de las listas sirve para eliminar la última falta que se haya anotado a un jugador. La aplicación comprobará cuál es la última falta que ha cometido el jugador seleccionado y la eliminará de las estadísticas.

8.6.4 Pantalla Cambios

8.32: Cambios

Siempre que el partido esté parado, el usuario puede realizar cambios en el equipo local o en el visitante pulsando el botón correspondiente de cambios. Tras ello aparecerá una pantalla avisando de cómo deben realizarse los cambios (la figura de la derecha muestra este aviso). Tras esto se mostrará una pantalla como la de la figura 8.17, en la que el usuario deberá realizar los cambios correspondientes, seleccionando al final los 7 jugadores que seguirán en el campo tras los cambios (si se equivoca y no selecciona los 7, el sistema informa de ello e impide continuar).

8.6.5 Ver Datos Equipo

8.33: Datos de un jugador visitante

En cualquier momento del partido el usuario puede ver las estadísticas de cada jugador simplemente accediendo al menú principal, y seleccionando las opciones "Gestor Equipo" → "Ver Datos Equipo" → "Equipo local" o "Equipo visitante". Realizados estos pasos se muestra una pantalla en la que aparece el nombre del equipo seleccionado, y una lista desplegable con los jugadores. El usuario puede seleccionar a cualquier jugador, y obtener sus estadísticas. Las estadísticas que se muestran son las siguientes:

- Nombre del jugador.
- Dorsal.
- Goles marcados en el partido
- Amonestaciones recibidas en el partido.
- Exclusiones recibidas en el partido.
- Descalificaciones recibidas en el partido.
- Expulsiones recibidas en el partido.
- Minutos disputados.

El usuario puede volver en todo momento a la pantalla principal pulsando en el botón "Volver".

Una vez que el partido finaliza se indica al usuario a través de un cuadro de información. A partir de ese momento ya no puede introducirse ningún nuevo dato a las estadísticas. Con el partido finalizado, el usuario tiene la opción de salir de la aplicación, o de reiniciar un nuevo partido (dicha opción de reiniciar está siempre presente a lo largo de la duración del partido).

Al terminar un partido de balonmano puede ser necesario establecer un ganador, y si ha acabado en empate, dependiendo de las reglas del torneo específico que se esté jugando se disputará prórroga. Si tenemos instalado en nuestra aplicación el paquete de prórroga, al término del partido de balonmano, si ha acabado en empate, pasará a la prórroga. Una prórroga en balonmano consta de 2 partes de 5 minutos cada una. Al acabar la 2ª parte, si persiste el empate, volvería a iniciarse

8.34: Prórroga

otro período de dos partes de 5 minutos. Si no es así, el partido se dará por finalizado.

8.7 Fútbol Sala

8.7.1 Pantalla principal.

Como en los deportes citados anteriormente, la pantalla principal en fútbol sala puede variar dependiendo de los paquetes que estén implementados y sus correspondientes interfaces. Es por esto que la pantalla principal puede presentar diferentes aspectos:

8.35: Pantalla principal sin botones

8.36: Pantalla principal con estadísticas

8.37: Pantalla principal con expulsión

8.38: Pantalla principal completa

Utilizaremos la pantalla de la figura 8.38 a lo largo de este manual por ser la más completa la que dispone de todas las utilidades posibles en la aplicación.

8.7.2 Pantalla Cargar Equipos

El usuario tras crear un partido de fútbol sala tendrá todos los botones inhabilitados, excepto la opción de menú "Gestor Equipos", que es a la que deberá acceder para cargar los equipos local y visitante. La aplicación mostrará una pantalla igual a la de las figuras 8.12 y 8.28, y posteriormente pedirá al usuario que seleccione a los 5 jugadores que saldrán de inicio en un partido de fútbol sala (como podemos ver en la figura anexa a estas líneas). Tras esto se abrirá una pantalla en la cual se podrán ver todos los jugadores del equipo, y se deberán marcar 5 jugadores (si no la aplicación no permite continuar). En el caso de que el paquete estadísticas no estuviera implementado en la solución, la pantalla de titulares no aparecería.

8.39: Selección de jugadores titulares

En cualquier momento podemos volver a cargar los equipos, o si seleccionamos esta opción y nos hemos

equivocado, existe un botón de “Volver” para no seleccionar ningún equipo, como se muestra en la figura 8.28.

8.7.3 Funcionalidad del Marcador de Fútbol Sala

Una vez cargados el equipo local y el visitante, el usuario ya podrá iniciar un partido. Tras seleccionar esta opción en el menú principal de la aplicación se muestra una pantalla con los siguientes datos:

- En la parte superior se muestran los nombres de los dos equipos que disputan el partido (o bien “Local” y “Visitante”),
- La puntuación en goles de ambos equipos.
- El tiempo que resta de cada parte (en fútbol sala cada parte dura 20 minutos).
- La parte en la que nos encontramos y el número total de las mismas (en este caso nos encontramos en loa primera parte de un total de dos).
- Botones para llevar el control de la anotación (“Anotar” y “Deshacer”), del tiempo (“Continuar”) y de las estadísticas (Amarilla, Roja, Borrar, cambios…).
- Dos listas con los dorsales de los jugadores que están sobre el campo.
- Dos recuadros azules sobre los que se mostrará el tiempo de expulsión.

8.40: Pantalla principal con los equipos cargados

8.41: Etiquetas de tiempo de expulsión

Tras haber presionado "Iniciar" sobre el menú principal, el usuario deberá hacer click en "Continuar" para poner en marcha el cronómetro del tiempo. Para que los goles suban al marcador el usuario debe hacer click en el botón "Anotar" correspondiente (a local o visitante). Si además está seleccionado algún jugador en la lista del equipo, ese gol se contabilizará para las estadísticas personales del jugador. Si el usuario ha cometido un error al anotar el gol, siempre puede deshacer el gol pinchando en el botón "Deshacer" correspondiente.

Por último, para llevar un control de las estadísticas, el usuario puede anotar las faltas que conllevan sanción que han cometido los jugadores. En fútbol sala hay dos tipos de faltas que implican sanción:

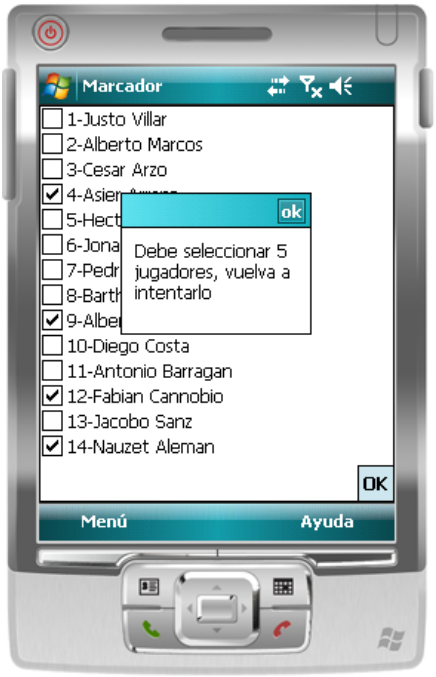
- **Tarjeta amarilla**: es una amonestación sencilla.
- **Tarjeta roja**: conlleva la expulsión definitiva del jugador del terreno de juego, dejando a su equipo con un jugador menos durante 2 minutos. Por ello, al seleccionarse la opción de "tarjeta roja", se iniciará el cronómetro de dos minutos.

El botón "Borrar" tiene su utilidad para suprimir la última falta del jugador seleccionado. En caso de no haber nada que borrar el marcador avisaría de ello con un mensaje, como podemos observar en la figura 8.42.

Siempre que se vaya a anotar una falta, o bien a borrar faltas de los jugadores, debe haber seleccionado en la lista algún jugador. En caso contrario no pasaría a engrosar las estadísticas.

8.42: Error en el botón borrar

8.7.4 Pantalla Cambios



8.43: Error en la selección de los cambios

En fútbol sala el usuario podrá realizar los cambios incluso en el caso de que el tiempo esté corriendo. Por tanto en cualquier momento del partido el usuario podrá presionar el botón de cambios (local o visitante) para hacer los cambios, marcando los 5 jugadores que seguirán en el campo una vez realizados los cambios. Si el usuario olvida seleccionar algún jugador, o bien selecciona más de 5, el programa muestra un mensaje de error como el de la figura 8.43.

8.8.4 Ver Datos Equipo

Para acceder a las estadísticas de jugador el usuario puede hacerlo seleccionando en el menú principal "Gestor Equipos"→"Ver Datos Equipo"→ "Equipo local" o "Equipo visitante". En la lista desplegable que aparece bajo el nombre del equipo que hemos elegido podemos escoger cualquier jugador para ver sus estadísticas. En ellas se ve:

- Nombre del jugador.
- Dorsal del jugador.
- Goles marcados en el partido.
- Tarjetas amarillas recibidas en el partido.
- Tarjetas rojas recibidas en el partido.
- Minutos disputados durante el partido.

8.44: Ver datos jugador

Para volver a la pantalla principal se debe hacer click en el botón “Volver”.

Una vez finalizado el partido, el usuario puede reiniciarlo, o bien salir de la aplicación, desde el menú principal.

8.45: Prórroga

En fútbol sala, al igual que en los demás deportes, si el partido acaba en empate a goles, puede ser necesario dirimir quién es el campeón del partido mediante una prórroga. Si está implementado este paquete, al acabar el tiempo reglamentario con empate se iniciaría una prórroga de 10 minutos. Al final de la misma, si ya no hay empate, el partido finaliza. Si no, se reinicia otra prórroga de 10 minutos.

PARTE 4

CONCLUSIONES

Capítulo 9
CONCLUSIONES

En este capítulo expondremos las conclusiones obtenidas a partir de la elaboración del presente proyecto, las principales dificultades encontradas y los conocimientos adquiridos con su realización. Además trataremos los objetivos alcanzados y las posibles futuras líneas de trabajo.

9.1 Dificultades encontradas

A continuación se detallan las principales dificultades que han surgido durante la elaboración y desarrollo del presente proyecto de fin de carrera:

- El primer problema que se nos planteó fue el partir de un proyecto realizado por otros compañeros, ya que, a pesar de haber realizado juntos las fases de análisis y de diseño, tuvimos que utilizar el paquete base implementado por ellos, el cual nos era totalmente desconocido. Por lo tanto, tuvimos que estudiar y comprender completamente el funcionamiento de su aplicación y los mecanismos utilizados por ellos para después aplicar métodos similares, implementar nuestra aplicación sin alterar el trabajo ya realizado, y hacer que el resultado final fuera lo más homogéneo posible.
- Para los autores del proyecto era la primera vez que afrontábamos un proyecto de gran envergadura, por lo que el desarrollo de las diversas fases del proyecto ha sido costoso, pero se ha superado a un nivel aceptable.
- Otra dificultad encontrada fue la incompatibilidad de WPF y XAML con Compact Framework y los dispositivos móviles, debido a lo cual tuvimos que diseñar una forma de adaptar las interfaces XAML a nuestra tecnología. La solución de este problema se describe más detalladamente en el capítulo de implementación.
- Además, la programación para dispositivos móviles en C# y el empleo de clases parciales nos era desconocida. Por suerte, la documentación sobre estos temas es extensa y pudimos adquirir conocimientos en estas áreas de un modo relativamente sencillo.
- La falta de medios como dispositivos móviles con tecnología bluetooth para la realización de pruebas fue otro de los problemas encontrados. Esta dificultad fue subsanada gracias a los emuladores de dispositivos de Microsoft y principalmente gracias a aplicaciones realizadas por desarrolladores particulares como las bibliotecas Franson BueTools, para el manejo de tecnología bluetooth, y la herramienta "Bluetooth for Microsoft Device Emulator", que permite la utilización de la antena bluetooth del ordenador como antena bluetooth del emulador de dispositivo.
- Por último, la planificación del trabajo ha sido una dura tarea, ya que no podíamos trabajar a la vez sobre la aplicación porque era difícil sincronizar las modificaciones hechas por separado. Además existían dificultades para reunirnos. Debido a ello, repartíamos las tareas de investigación, implementación y desarrollo de la memoria de forma que no fuera complicado poner los cambios en común.

9.2 Objetivos alcanzados

Durante la realización del proyecto se han ido logrando los objetivos que definíamos en la introducción de esta memoria:

- Se ha obtenido un conocimiento acerca de las Líneas de Producto Software: su funcionamiento, definición.
- Se ha desarrollado una Línea de Producto Software de Marcadores Deportivos aplicados a dispositivos móviles.
- Se ha utilizado el entorno de desarrollo de Visual Studio .NET para desarrollar la Línea de Producto, permitiendo crear con facilidad marcadores deportivos seleccionando las características deseadas en ese momento.
- Se ha logrado dotar de nueva funcionalidad al marcador deportivo creado en el Proyecto Fin de Carrera con el que hemos estado trabajando conjuntamente.
- Las nuevas funcionalidades del marcador deportivo se han logrado integrar perfectamente en la Línea de Producto.
- Se han desarrollado prototipos de marcadores deportivos a modo de ejemplo, para deportes diferentes.
- Se ha conseguido un conocimiento suficiente para permitir la implementación de aplicaciones para dispositivos móviles con Visual Studio 2008 y su entorno de trabajo, utilizando el lenguaje de programación C#.
- Se ha trabajado en equipo en las fases de análisis y diseño conjuntamente 4 personas (las implicadas en los dos proyectos acerca de la Línea de Productos de Marcadores Deportivos), así como en la fase de implementación, desarrollo, pruebas y documentación (en esta ocasión, los dos autores de este proyecto).
- Se han usado conocimientos adquiridos en la carrera de Ingeniería Técnica en Informática de Gestión para la realización de este proyecto.

9.3 Conocimientos adquiridos

Podemos decir que este proyecto nos ha enseñado a saber enfrentarnos a una aplicación de esta envergadura y a conocer nuestras capacidades y conocimientos. Hemos tenido que realizar las labores de análisis y desarrollo, así como la implementación y las pruebas y aunque ha habido dificultades hemos logrado organizar el tiempo y dividir el trabajo para que todo saliera correctamente.

También nos hemos familiarizado con el mundo de las aplicaciones para dispositivos móviles, que es una tecnología en auge en la época en la vivimos, ya que cada vez es mayor el número de personas que los utilizan, y son cada vez más accesibles para los usuarios domésticos.

Además hemos aprendido a procesar archivos XML, así como su utilización y su estructuración.

Hemos entrado en contacto con la tecnología WPF y los archivos de interfaz XAML, de los cuales desconocíamos incluso su existencia.

Hemos conocido, desarrollado y aplicado el concepto de línea de productos, y conocido sus características y beneficios.

Por último, resaltar la utilidad de las herramientas utilizadas, porque tanto el .NET (en especial el lenguaje C#) como el manejo de Visual Studio son muy demandados en el mundo laboral. Hemos partido desde cero en el uso de esta herramienta ya que las desconocíamos por completo, y hemos superado con creces el proceso de aprendizaje, familiarizándonos con dicha tecnología así como con el empleo de clases parciales y el desarrollo de aplicaciones para dispositivos móviles.

9.4 Futuras líneas de trabajo

Como hemos dicho anteriormente, este Proyecto Fin de Carrera es una ampliación de otro proyecto base. Por esta razón, al igual que la parte realizada por nosotros, existen otras muchas extensiones posibles a desarrollar. A continuación proponemos algunas de las posibles líneas de trabajo que podrían dar lugar nuevos proyectos dentro de la misma línea de productos:

- Implementar, tanto el paquete base, como el resto de paquetes opcionales, utilizando el lenguaje Java, para poder utilizar esta aplicación en dispositivos móviles más simples, como los teléfonos móviles convencionales.
- Posibilitar el reparto de roles entre los usuarios que se conectan en un mismo partido, de forma que cada usuario sea responsable de una parte del control del partido (control de faltas, control del tiempo, etc.), incluyendo la posibilidad de participar únicamente como observador.
- Adaptar la aplicación para la conexión vía Wifi.
- Aumentar las posibilidades de control de estadísticas como por ejemplo la posesión del balón de cada equipo, control de asistencias de cada jugador ó tipos de tantos anotados por cada jugador (triples y tiros libres en baloncesto, penaltis y lanzamientos de falta en fútbol sala…) .
- Creación de nuevos paquetes para otros deportes, tanto por equipos (rugby, hockey, voleibol…), como individuales (tenis, ciclismo, equitación…).
- Conectar la aplicación a un servidor para ser utilizada como un servicio web.

PARTE 5

REFERENCIAS

BIBLIOGRAFÍA

- Deitel, Harvey M.- Deitel, Paul J. Cómo programar en C# (2ª edición). Ed: Pearson Educación 2007
- Harold, Elliote Rusty. XML Bible. Ed: IDG Books Wordwide 1999
- Mayo, Joseph. C# Al descubierto. Ed: Prentice Hall. 2002
- Morrison, Michael - Et Al. XML Al descubierto. Ed: Prentice Hall. 2000
- Pressman, Roger S. Ingenieria del Software. Un enfoque práctico (6ª edicion). Ed: Mc Graw Hill 2005
- Sells, Chris. WPF. Ed: Anaya Multimedia. 2009

REFERENCIAS WEB

- http://www.elguille.com (última visita: agosto 2009) → información sobre temas de programación.
- http://www.feb.es (última visita: agosto 2009) → sitio oficial de la federación española de baloncesto → información acerca del reglamento del baloncesto.
- http://www.forosdelweb.com (última visita: agosto 2009) → foro de información general sobre dudas de programación.
- http://www.futsala.com (última visita: agosto 2009) → información sobre reglas del fútbol sala.
- http://giro.infor.uva.es (última visita: agosto 2009) → sitio web del grupo GIRO de la UVA
- http://msdn.microsoft.com (última visita: agosto 2009) → sitio web de Microsoft → información general sobre programación con Visual Studio, con tutoriales.
- http://rfebm.com (última visita: agosto 2009) → página oficial de la real federación española de balonmano.
- http://www.softwareproductlines.com (última visita: agosto 2009) → información acerca de líneas de producto.

APÉNDICES

Apéndice A.

XAML

A.1 Aspectos generales de XAML

XAML (*e**X**tensible **A**pplication **M**arkup **L**anguage*, *Lenguaje Extensible de Formato para Aplicaciones*) es el lenguaje de formato para la interfaz de usuario para la Base de Presentación de Windows (WPF por sus siglas en inglés) y Silverlight(wpf/e), el cual es uno de los "pilares" de la interfaz de programación de aplicaciones .NET en su versión 3.0.

XAML es un lenguaje declarativo basado en XML, optimizado para describir gráficamente interfaces de usuarios visuales ricas desde el punto de vista gráfico. En su uso típico, los archivos tipo XAML serían producidos por una herramienta de diseño visual, como Microsoft Visual Studio o Microsoft Expression. El XML resultante es interpretado en forma instantánea por un sub-sistema de despliegue de Windows Vista que reemplaza al GDI de las versiones anteriores de Windows. Los elementos de XAML se interconectan con objetos del Entorno Común de Ejecución para Lenguajes. Los atributos se conectan con propiedades o eventos de esos objetos.

XAML simplifica la creación de una interfaz de usuario para el modelo de programación de .NET Framework. Puede crear elementos de la interfaz de usuario visibles en el marcado XAML declarativo y, a continuación, separar la definición de la interfaz de usuario de la lógica en tiempo de ejecución utilizando archivos de código subyacente, que se unen al marcado mediante definiciones de clases parciales. La capacidad de combinar código con marcado en XAML es importante porque XML por sí solo es declarativo, y no sugiere realmente un modelo para el control de flujo. Un lenguaje declarativo basado en XML es muy intuitivo para crear interfaces que van desde el prototipo a la producción, sobre todo para las personas con entrenamiento en diseño y tecnologías web. A diferencia de la mayoría de los demás lenguajes de marcado, XAML representa directamente la creación de instancias de objetos administrados. Este principio de diseño general habilita el código simplificado y el acceso a la depuración para los objetos que se crean en XAML.

Elementos de objeto XAML

XAML tiene un conjunto de reglas que asignan elementos de objeto a clases o estructuras, atributos a propiedades o eventos, y espacios de nombres de XML a espacios de nombres de CLR (**C**ommon **L**anguage **R**untime). Los elementos de XAML se asignan a los tipos de Microsoft .NET tal y como se definen en los ensamblados a los que se hace referencia y los atributos se asignan a los miembros de dichos tipos.

Establecer propiedades

Las propiedades en XAML se establecen definiendo propiedades en un elemento de objeto utilizando alguna de las sintaxis posibles. Las sintaxis que se pueden utilizar para una propiedad

determinada variarán, dependiendo de las características de la propiedad que se está estableciendo. En XAML, las propiedades se pueden expresar a menudo como atributos. La sintaxis de atributo es la sintaxis de establecimiento de propiedades más optimizada y es la más intuitiva. Un ejemplo sería de esta sintaxis sería: <Grid Name="Panel"></Grid>

En algunas propiedades de un elemento de objeto, no es posible usar la sintaxis de atributo, ya que el objeto o la información necesaria para proporcionar el valor de la propiedad no se puede expresar correctamente como una cadena simple. En estos casos, se puede utilizar otra sintaxis conocida como sintaxis de elementos de propiedad. La sintaxis de elementos de propiedad establece la propiedad del elemento contenedor a la que se hace referencia en el contenido de la etiqueta. Generalmente, el contenido es un objeto del tipo que la propiedad toma como valor propio (con la instancia del valor normalmente especificada como otro elemento de objeto). La sintaxis para el elemento de propiedad en sí es <nombreDeTipo.Propiedad>. Después de especificar el contenido, se debe cerrar el elemento de propiedad con una etiqueta de cierre como cualquier otro elemento (con la sintaxis </nombreDeTipo.Propiedad>). En las propiedades en las que se admite tanto la sintaxis de atributo como la sintaxis de elementos de propiedad, ambas sintaxis tienen normalmente el mismo resultado, aunque algunos detalles como el control del espacio en blanco pueden variar ligeramente entre ellas. Si es posible utilizar una sintaxis de atributo, su uso es generalmente más conveniente, ya que habilita un marcado más compacto, pero es simplemente una cuestión de estilo, no una limitación técnica.

La sintaxis de elementos de propiedad para XAML representa una variación significativa de la interpretación de XML básica del marcado. Para XML, <nombreDeTipo.Propiedad> representa otro elemento, sin ninguna relación necesariamente implícita con un elemento primario TypeName aparte del hecho de ser un elemento secundario. En XAML, <nombreDeTipo.Property> implica directamente que Property es una propiedad de nombreDeTipo, establecida por el contenido del elemento de propiedad, y nunca será un elemento con nombre similar sino un elemento independiente con un punto en el nombre.

Las propiedades tal y como aparecen como atributos XAML en un elemento de WPF se heredan a menudo de las clases base.

El comportamiento de la herencia de clases de los elementos XAML de WPF es otra desviación significativa de la interpretación de XML básica del marcado. La herencia de clases (especialmente cuando las clases base intermedias son abstractas) es una de las razones por las que el conjunto de elementos de XAML y sus atributos permitidos resulta complicada de representar correcta y completamente utilizando los tipos de esquema que se usan normalmente para la programación en XML.

Valores de referencia y extensiones de marcado

Las extensiones de marcado son un concepto de XAML. En la sintaxis de atributo, las llaves ({ y }) indican un uso de la extensión de marcado. Este uso hace que el procesamiento de XAML se aparte del tratamiento general de valores de atributo como una cadena literal o un valor directamente convertible en cadena.

Cuando las propiedades toman un valor de tipo de referencia, requerirán a menudo una sintaxis de elementos de propiedad (lo que crea siempre una nueva instancia) o una referencia de objeto mediante una extensión de marcado. El uso de una extensión de marcado puede devolver una instancia existente, por lo que puede ser más versátil o incurrir en una menor sobrecarga de objetos.

Cuando se utiliza una extensión de marcado para proporcionar un valor de atributo, éste debería ser proporcionado por la lógica de la clase de respaldo para la extensión de marcado pertinente. Las extensiones de marcado más comunes utilizadas en la programación de aplicaciones de WPF son Binding, utilizada para las expresiones de enlace de datos, y las referencias de recursos StaticResource y DynamicResource. Las extensiones de marcado permiten utilizar la sintaxis de atributo para proporcionar valores de referencia para las propiedades, aun cuando dichas propiedades no admitan una sintaxis de atributo para la creación de instancias de objeto directas, o permiten habilitar un comportamiento concreto que difiera del comportamiento general que requiere que las propiedades de XAML se rellenen con valores del tipo de la propiedad.

Por ejemplo, si se establece el valor de la propiedad Style utilizando la sintaxis de atributo. La propiedad Style toma una instancia de la clase Style, un tipo de referencia que de forma predeterminada no se podía especificar dentro de una cadena de sintaxis de atributo. Pero en este caso, el atributo hace referencia a una extensión de marcado determinada, StaticResource. Cuando se procesa dicha extensión de marcado, devuelve una referencia a un estilo del que ya se han creado instancias como un recurso con clave en un diccionario de recursos.

Valores de atributo habilitados para el convertidor de tipos

Anteriormente, se ha dicho que el valor de atributo debe poder establecerlo una cadena. El control nativo básico de cómo se convierten las cadenas en otros tipos de objetos o valores primitivos está basado en el propio tipo String. Pero muchos tipos de WPF o miembros de esos tipos extienden el comportamiento básico del procesamiento de atributos de cadena, de tal forma que se pueden especificar instancias de tipos de objeto más complejos como valores de atributo mediante una cadena. En el nivel de código, este procesamiento se logra especificando un convertidor de tipos CLR que procesa el valor de atributo de cadena. El tipo de estructura Thickness, utilizado normalmente para indicar mediciones de una área rectangular, como una propiedad Margin, es un ejemplo de tipo que tiene una sintaxis de atributo especial habilitada para el convertidor de tipos expuesta para todas las propiedades que toman dicho tipo, con objeto de proporcionar facilidad de uso en el marcado XAML.

Tipos de colección y propiedades de la colección XAML

XAML especifica una característica del lenguaje por la que el elemento de objeto que representa un tipo de colección se puede omitir deliberadamente en el marcado. Cuando un procesador de XAML controla una propiedad que toma un tipo de colección, se crea implícitamente una instancia del tipo de colección adecuado, aun cuando el elemento de objeto para dicha colección no esté presente en el marcado.

Con la excepción del elemento raíz, cada elemento de objeto de una página que está anidado como elemento secundario de otro elemento es realmente un elemento que tiene una de las características siguientes o ambas: es un miembro de una propiedad de colección implícita de su elemento primario o un elemento que especifica el valor de la propiedad de contenido XAML para el elemento primario. En otras palabras, la relación entre elementos primarios y secundarios en una página de marcado es realmente un objeto único en la raíz, y cada elemento de objeto que está debajo de la raíz es una instancia única que proporciona un valor de propiedad del elemento primario o uno de los elementos dentro de una colección que es también un valor de propiedad de tipo de colección del elemento primario.

Propiedades del contenido XAML

XAML especifica una característica del lenguaje por la que cualquier clase que se puede utilizar como un elemento de objeto de XAML puede designar exactamente una de sus propiedades como la propiedad de contenido XAML para las instancias de la clase. Cuando un procesador de XAML controla un elemento de objeto que tiene una propiedad de contenido XAML, los elementos secundarios de XML de dicho elemento de objeto se procesan como si estuvieran contenidos en una etiqueta de elemento de propiedad implícita que representa esa propiedad de contenido. Dentro del marcado, se puede omitir la sintaxis de elementos de propiedad para la propiedad de contenido XAML. Cualquier elemento secundario especificado en el marcado se convertirá en el valor de la propiedad de contenido XAML.

Texto interno y propiedades del contenido XAML

Existen otras formas de especificar el texto de la presentación para diversos elementos. La propiedad Content se puede especificar normalmente en la sintaxis de atributo; sin embargo, la cadena de la presentación puede ser el texto interno que está dentro de, por ejemplo, un elemento de objeto Button. Esta sintaxis funciona porque Content es la propiedad de contenido XAML de la clase base ContentControl de Button. La cadena incluida en el elemento se evalúa basándose en el tipo de propiedad de la propiedad Content, que es Object. Object no intenta ninguna conversión de tipos de cadenas, por lo que el valor de la propiedad Content se convierte en el valor de cadena literal. Alternativamente, el contenido incluido dentro de Button podría haber sido cualquier Object único. Los controles como Button generalmente definen la propiedad de contenido XAML para la clase de manera que la propiedad de contenido XAML se puede utilizar para la interfaz de usuario y el texto de la presentación, para la composición de controles o para ambos.

La capacidad de colocar las cadenas dentro del elemento como contenido para generar marcado que se parece a otros lenguajes de marcado comunes es de especial importancia para el modelo de documento dinámico y para la localización.

Los valores de las propiedades de contenido XAML deben ser contiguos

El valor de una propiedad de contenido XAML se debe proporcionar exclusivamente antes o exclusivamente después de cualquier otro elemento de propiedad en ese elemento de objeto. Esto es cierto independientemente de si el valor de una propiedad de contenido XAML se especifica como una cadena o como uno o varios objetos.

El uso de mayúsculas y minúsculas y del espacio en blanco en XAML

XAML distingue entre mayúsculas y minúsculas. Los elementos de objeto, los elementos de propiedad y los nombres de atributo se deben especificar utilizando la escritura apropiada cuando se comparan por nombre con el tipo subyacente en el ensamblado, o con un miembro de un tipo. Los valores para los atributos no siempre distinguen entre mayúsculas y minúsculas. La distinción entre mayúsculas y minúsculas para los valores dependerá del comportamiento del convertidor de tipos asociado a la propiedad que toma el valor o del tipo de valor de propiedad. Por ejemplo, las propiedades que toma el tipo Boolean pueden tomar true o True como valores equivalentes, pero sólo porque la conversión de tipos de cadena predeterminada para Boolean ya los permite como equivalentes.

Los procesadores y serializadores de XAML omiten o quitan todos los espacios en blanco no significativos y normalizan cualquier espacio en blanco significativo. Este comportamiento generalmente sólo tiene importancia si se especifican cadenas dentro de las propiedades de contenido XAML. Es decir, XAML convierte los caracteres de espacio, avance de línea y tabulación en espacios y, a continuación, conserva un espacio si lo encuentra en cualquier extremo de una cadena contigua.

Elementos raíz en XAML y espacios de nombres XML

Para que un archivo XAML pueda ser tanto un archivo XML con un formato correcto como un archivo XAML válido, sólo debe tener un elemento raíz. Normalmente, se debe elegir un elemento que forme parte del modelo de aplicación (por ejemplo, Window o Page para una página, ResourceDictionary para un diccionario externo o Application para la raíz de la definición de la aplicación).

El elemento raíz también contiene los atributos xmlns y xmlns:x. Estos atributos indican a un procesador de XAML los espacios de nombres XML que contienen las definiciones de los elementos a los que el marcado hará referencia. El atributo xmlns indica específicamente el espacio de nombres XML predeterminado. Dentro este espacio de nombres XML, los elementos de objeto en el marcado se pueden especificar sin un prefijo. Para la mayoría de los escenarios de aplicaciones de WPF y para casi todos los ejemplos ofrecidos en las secciones de WPF del SDK, el espacio de nombres XML predeterminado está asignado al espacio de nombres http://schemas.microsoft.com/winfx/2006/xaml/presentation de WPF. El atributo xmlns:x indica un espacio de nombres XML adicional, que asigna el espacio de nombres del lenguaje XAML http://schemas.microsoft.com/winfx/2006/xaml. Los componentes del lenguaje necesarios definidos por la especificación XAML tienen el prefijo x: cuando se hace referencia a ellos en el marcado de un archivo con esta asignación. Este uso de xmlns para definir un ámbito de uso y asignación es coherente con la especificación XML 1.0. Tenga en cuenta que los atributos xmlns sólo son estrictamente necesarios en el elemento raíz de cada página y en la definición de aplicación si se proporciona ésta en el marcado. Las definiciones xmlns se aplicarán a todos los elementos secundarios de la raíz (este comportamiento sigue siendo coherente con la especificación XML 1.0 para xmlns). Los atributos xmlns también se permiten en otros elementos por debajo de la raíz y se aplicarían a los elementos secundarios del elemento que los define. Sin embargo, éste no

es un uso típico, porque una definición o redefinición frecuente de los espacios de nombres XML puede dar lugar a un estilo de marcado XAML que resulte difícil de leer.

Los ensamblados de WPF se sabe que contienen los tipos que admiten las asignaciones de WPF al espacio de nombres XML predeterminado debido a la configuración que forma parte del archivo de compilación del proyecto. Los ensamblados también se asignan en los archivos .targets. Por lo tanto, únicamente es necesario asignar el xmlns para poder hacer referencia a los elementos de XAML que vienen de los ensamblados de WPF. Para sus propios ensamblados personalizados o para los ensamblados fuera de WPF, puede especificar el ensamblado como parte de la asignación de xmlns. Normalmente, se elige un prefijo diferente, pero también es posible elegir un espacio de nombres XML diferente como valor predeterminado y, a continuación, asignar WPF a un prefijo.

El prefijo x:

En la mayor parte de los casos, se utiliza el prefijo x: para asignar el espacio de nombres XML en XAML http://schemas.microsoft.com/winfx/2006/xaml. Este prefijo x: se utiliza para asignar el espacio de nombres XML en XAML en las plantillas de los proyectos, en los ejemplos y en la documentación de este SDK. El prefijo x: o espacio de nombres XML en XAML contiene varias construcciones de programación que se utilizan con frecuencia en el XAML. A continuación se muestra una lista de las construcciones de programación prefijo x: o espacio de nombres en XAML más comunes que se utiliza:

- x:Key: establece una clave única para cada recurso de un elemento ResourceDictionary. x:Key representará probablemente el 90% de los usos de x: que se ven en el marcado de una aplicación.

- x:Class: especifica el espacio de nombres de CLR y el nombre de clase para la clase que proporciona código subyacente para una página XAML. Se debe disponer de esta clase para admitir el código subyacente; por esto casi siempre se verá x: asignado, aun cuando no haya ningún recurso.

- x:Name: especifica un nombre de objeto en tiempo de ejecución para la instancia que existe en el código en tiempo de ejecución una vez procesado un elemento de objeto. Se utiliza x:Name para aquellos casos de asignación de nombres a elementos en los que no se admite la propiedad de nivel de marco de trabajo de WPF Name equivalente. Esto sucede en ciertos escenarios de animación.

- x:Static: habilita una referencia de valor que obtiene un valor estático que de lo contrario no sería una propiedad compatible con XAML.

- x:Type: construye una referencia Type basada en un nombre de tipo. Esto se utiliza para especificar atributos que toman valores Type, como Style..::.TargetType , aunque en muchos casos la propiedad dispone de una conversión de cadena a Type nativa y el uso de x:Type es opcional.

Hay construcciones de programación adicionales en el prefijo x: o espacio de nombres en XAML que no son tan habituales.

Eventos y el código XAML subyacente

La mayoría de las aplicaciones de WPF constan de marcado y código subyacente. Dentro de un proyecto, el código XAML se escribe como un archivo .xaml y se utiliza un lenguaje de CLR, como Microsoft Visual Basic .NET o C#, para escribir un archivo de código subyacente. Cuando se compila un archivo XAML, la ubicación del archivo de código XAML subyacente para cada página XAML se identifica especificando un espacio de nombres y una clase como atributo x:Class del elemento raíz de la página XAML.

Si no se desea crear un archivo de código subyacente independiente, también se puede insertar el código dentro de un archivo XAML. Sin embargo, el código insertado es una técnica menos versátil que tiene importantes limitaciones.

Al especificar el comportamiento mediante eventos en el marcado, se utiliza normalmente la sintaxis de atributo para asociar los controladores. El elemento de objeto donde se especifica el atributo de evento se convierte en la instancia que escucha el evento y llama al controlador. El nombre del evento concreto que desea controlar es el nombre del atributo. El valor del atributo es el nombre de método del controlador que se definirá. A continuación, debe proporcionar la implementación del controlador en el código subyacente, controlador que estará basado en el delegado para ese evento. El controlador se escribe en código subyacente en un lenguaje de programación como Microsoft Visual Basic .NET o C#. Por ejemplo: <Button Click="manejador_de_evento_click"></Button>

Eventos enrutados

Una característica especial de los eventos determinada que es única y fundamental para WPF es un evento enrutado. Los eventos enrutados permiten a un elemento controlar un evento producido por otro elemento diferente, siempre que dichos elementos se relacionen entre sí a través de un árbol de elementos. Al especificar el control de eventos con un atributo XAML, se puede escuchar y controlar el evento enrutado en cualquier elemento, incluidos los elementos que no contienen ese evento en particular en la tabla de miembros de clase. Esto se consigue calificando el atributo de nombre de evento con el nombre de la clase propietaria.

Propiedades y eventos asociados

XAML incluye una característica de lenguaje que permite especificar ciertas propiedades o eventos en cualquier elemento, independientemente de si la propiedad o el elemento existe en la tabla de miembros para el elemento en el que se establece. La versión de esta característica para las propiedades se denomina propiedad asociada y la versión para los eventos se denomina evento asociado. Conceptualmente, las propiedades y los eventos asociados se pueden considerar como miembros globales que se pueden establecer en cualquier elemento o clase, independientemente de su jerarquía de clases.

Las propiedades asociadas en XAML se utilizan normalmente mediante la sintaxis de atributo. En la sintaxis de atributo, una propiedad asociada se especifica en la forma *tipoDePropietario.nombreDePropiedad*. A primera vista, esto se parece al uso de un elemento de propiedad, pero en este caso el *tipoDePropietario* que se especifica es siempre un tipo distinto del elemento de objeto en el que se establece la propiedad asociada. *tipoDePropietario* es el tipo que proporciona los métodos de descriptor de acceso que un procesador de XAML requiere para poder obtener o establecer el valor de la propiedad asociada. El escenario más común para las propiedades asociadas es permitir a los elementos secundarios enviar un valor de propiedad a su elemento primario.

Los eventos asociados utilizan una forma de sintaxis de atributo similar: *tipoDePropietario.nombreDeEvento*. Al igual que en los eventos no asociados, el valor del atributo para un evento asociado en XAML especifica el nombre del método controlador que se invoca cuando se controla el evento en el elemento.

Un escenario en el que se utilizan eventos asociados lo constituyen los eventos de entrada de dispositivo que se pueden controlar en cualquier elemento, como los botones del mouse. Un ejemplo de este tipo de evento asociado es *Mouse.MouseDown*. Sin embargo, la mayoría de los elementos del nivel de marco de trabajo de WPF pueden utilizar este evento sin recurrir a un evento asociado. Esto es debido a que la clase del elemento base UIElement crea un alias para el evento asociado *Mouse.MouseDown* y expone dicho alias en la tabla de miembros UIElement (como MouseDown). Como resultado, normalmente no es necesario especificar la sintaxis de eventos asociados en una página XAML o en la programación de aplicaciones de Windows Presentation Foundation (WPF). La excepción la constituye el uso de elementos personalizados o elementos de objeto que no se derivan de UIElement, pero que aun así tienen una representación visual (éstos son poco frecuentes). En WPF, todos los eventos asociados se implementan también como eventos enrutados. ContentElement también expone los alias para los eventos de entrada, con objeto de que los utilice el modelo de documento dinámico.

Seguridad XAML

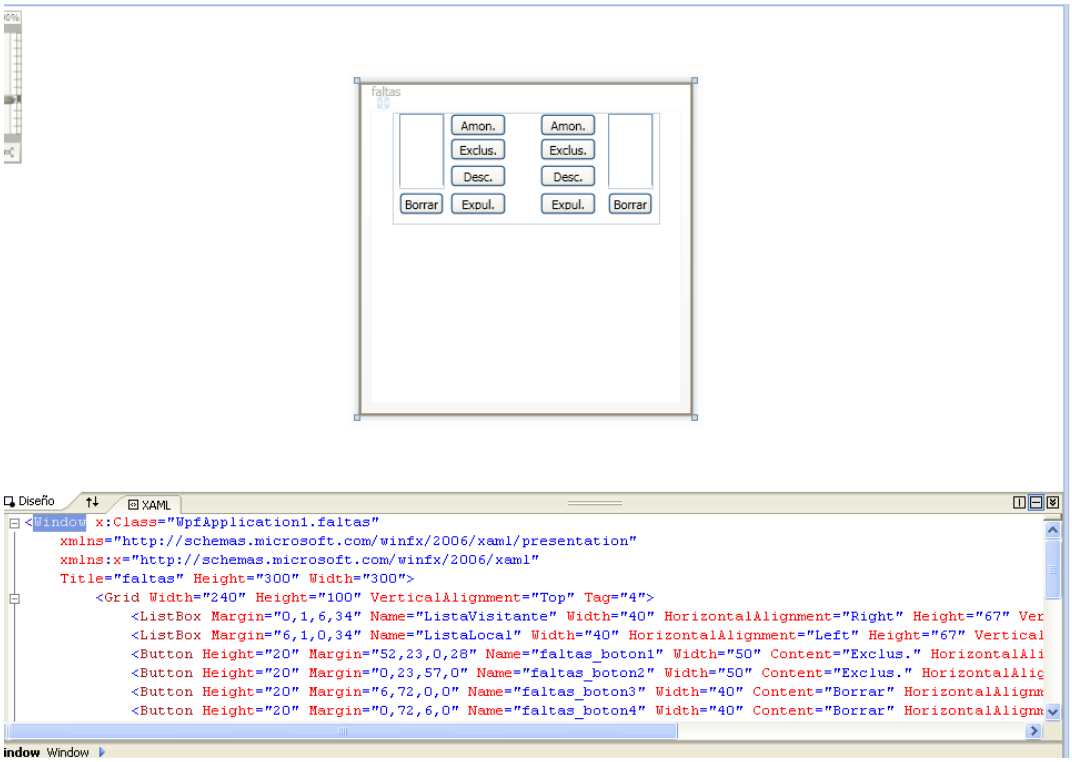
XAML es un lenguaje de marcado que representa directamente la creación de instancias y la ejecución de objetos. Por consiguiente, los elementos creados en XAML tienen la misma capacidad de interactuar con los recursos del sistema (por ejemplo, el acceso a la red y la E/S del sistema de archivos) que el código generado equivalente.

WPF admite el marco de trabajo de seguridad Seguridad de acceso a código (CAS) de .NET. Esto significa que el contenido de WPF que se ejecuta en la zona de Internet tiene permisos de ejecución reducidos. El "XAML dinámico" (páginas de XAML no compilado interpretadas en el momento de la carga por un visor de XAML) y las Aplicación del explorador XAML (XBAP) se ejecutan normalmente en esta zona de Internet y utilizan el mismo conjunto de permisos. Sin embargo, el XAML que se carga en una aplicación de plena confianza tiene el mismo acceso a los recursos del sistema que la aplicación host.

Se puede utilizar XAML para definir una interfaz de usuario completa, pero a veces también es conveniente definir sólo una parte de dicha interfaz en XAML. Esta función se podría

utilizar para habilitar una personalización parcial, para el almacenamiento local de información, utilizando XAML para proporcionar un objeto comercial, o para diversos escenarios posibles. La clave para estos escenarios es la clase XamlReader y su método Load. La entrada es un archivo XAML y el resultado es un objeto que representa el árbol de objetos en tiempo de ejecución completo que se ha creado a partir de ese marcado. A continuación, se puede insertar el objeto como una propiedad de otro objeto que ya existe en la aplicación. Siempre que la propiedad sea una propiedad adecuada en el modelo de contenido que tiene funciones de presentación y que notifique al motor de ejecución que se ha agregado el nuevo contenido a la aplicación, es posible modificar con bastante facilidad el contenido de una aplicación en ejecución cargando XAML.

A.2 Sintaxis de XAML



A.1: Imagen de un archivo XAML. En la parte superior la interfaz, y en la parte inferior el código del archivo

La terminología de la sintaxis de XAML definida aquí también se define o se hace referencia a ella dentro de la especificación del lenguaje XAML. XAML es un lenguaje basado en XML y sigue las reglas estructurales de XML. Parte de esta terminología la comparte con o está

basada en la terminología utilizada normalmente al describir el lenguaje XML o el Modelo de objetos de documento (DOM) XML.

Sintaxis de atributo

La sintaxis de atributo es la sintaxis de marcado de XAML que establece un valor para una propiedad o denomina un controlador de eventos para un evento declarando un atributo en un elemento. El elemento siempre se declara mediante la sintaxis de elementos de objeto. El nombre del atributo debe coincidir con el nombre de miembro de CLR de una propiedad o un evento. El nombre del atributo va seguido de un operador de asignación (=). El valor de atributo debe ser una cadena encerrada entre comillas (").

Para que una propiedad se pueda establecer mediante la sintaxis de atributo, dicha propiedad debe ser pública, de lectura y escritura, y tener un tipo de valor de propiedad del que se puedan crear instancias o al que pueda hacer referencia un procesador de XAML. Para los eventos, el evento debe ser público y tener un delegado público. La propiedad o el evento debe ser un miembro de la clase o estructura de la que el elemento de objeto contenedor crea una instancia.

El valor de cadena incluido entre las comillas de apertura y cierre es procesado por un procesador de XAML. En el caso de las propiedades, el comportamiento de procesamiento predeterminado está definido por el tipo de propiedad de CLR subyacente. Si la propiedad es un tipo primitivo, el valor de atributo se asigna basándose en la conversión implícita de la cadena al tipo primitivo pertinente. Si la propiedad es una enumeración, la cadena se trata como un nombre definido por dicha enumeración y se devuelve el valor correspondiente de esta. Si la propiedad no es un tipo primitivo ni una enumeración, el valor de atributo se deberá controlar mediante un convertidor de tipos declarado en la misma propiedad o en el tipo de destino. El convertidor de tipos debe proporcionar una conversión que acepte una cadena y debe producir una instancia del tipo de propiedad de CLR subyacente. El paso de la conversión también se podría aplazar mediante una extensión de marcado.

Los valores de enumeración en XAML son procesados intrínsecamente por los métodos nativos de la estructura Enum. Para los valores de enumeración sin marcadores, el comportamiento nativo consiste en procesar la cadena de un valor de atributo y resolverlo como uno de los valores de enumeración. La enumeración no se especifica en el formato *Enumeración.Valor*, tal y como se hace en el código. En su lugar, sólo se especifica *Valor*, y *Enumeración* se deduce a partir del tipo de la propiedad que se está estableciendo. Si se especifica un atributo en la forma *Enumeración.Valor*, no se resolverá correctamente.

Para las enumeraciones basadas en marcadores, el comportamiento está basado en el método *Enum.Parse*. Se pueden especificar varios valores para una enumeración basada en marcadores separando cada valor con una coma. Sin embargo, no es posible combinar valores de enumeración que no estén basados en marcadores.

Al especificar un atributo, se puede hacer referencia a cualquier propiedad o evento que exista como un miembro del tipo CLR cuya instancia se ha creado para el elemento de objeto contenedor. O bien, se puede hacer referencia a una propiedad o evento asociado, independiente del elemento de objeto contenedor. Asimismo, puede asignar un nombre a cualquier evento desde cualquier objeto que sea accesible a través del espacio de nombres predeterminado utilizando un nombre parcial *nombreDeTipo.evento*; esta sintaxis permite asociar controladores para eventos enrutados en los que el controlador está diseñado para controlar eventos con enrutamiento desde elementos secundarios, pero donde el elemento primario no tiene ese evento en su tabla de miembros. Esta sintaxis se parece a la sintaxis de eventos asociados, pero en este caso el evento no es un verdadero evento asociado. En su lugar, se hace referencia a un evento con un nombre completo.

Los nombres de las propiedades se proporcionan a veces como el valor de un atributo, no como el nombre de atributo, y dichos nombres también pueden incluir calificadores, como la propiedad especificada en la forma *tipoDePropietario.nombreDePropiedadDeDependencia*. Este escenario es habitual al crear estilos o plantillas en XAML. Las reglas de procesamiento para los nombres de propiedades proporcionados como un valor de atributo son diferentes, y están regidas por el tipo de propiedad que se va a establecer y por algunos factores del contexto, como que un estilo o una plantilla tenga o no un tipo de destino.

Sintaxis de elementos de propiedad

La sintaxis de elementos de propiedad es una sintaxis que difiere ligeramente de la sintaxis XML básica. En XML, el valor de un atributo es de hecho una cadena y la única variación posible es el formato de codificación de cadenas que se utiliza. En XAML, puede asignar otros elementos de objeto como valores de las propiedades. Esta función está habilitada por la sintaxis de elementos de propiedad. En lugar de especificar la propiedad como un atributo dentro de la etiqueta de elemento, la propiedad se especifica utilizando una etiqueta de elemento de apertura en la forma *nombreDeTipoElemento.nombreDePropiedad*, se especifica el valor de la propiedad y, a continuación, se cierra el elemento de propiedad.

Concretamente, la sintaxis comienza con un corchete angular izquierdo (<), seguido inmediatamente por el nombre de tipo de la clase o estructura en la que está incluida la sintaxis de elementos de propiedad. Esta va seguida inmediatamente de un punto (.), después por el nombre de una propiedad que debe existir dentro de la tabla de miembros del tipo especificado y, a continuación, por un corchete angular derecho (>). El valor que se va a asignar a la propiedad se incluye dentro del elemento de propiedad. Normalmente, el valor se proporciona como uno o más elementos de objeto, porque la especificación de objetos como valores es el escenario que la sintaxis de elementos de propiedad pretende resolver. Finalmente, se debe proporcionar una etiqueta de cierre equivalente que especifique la misma combinación *nombreDeTipoElemento.nombreDePropiedad*, anidada y equilibrada correctamente con otras etiquetas de elemento.

El valor contenido en un elemento de propiedad también se puede proporcionar como texto interno, en aquellos casos en que el tipo de propiedad que se especifica es un tipo de valor

primitivo, como String, o una enumeración en la que se especifica un nombre. Estos dos usos no son habituales, porque cada uno de estos casos también admite la sintaxis de atributo. Un escenario para rellenar un elemento de propiedad con una cadena es para las propiedades que no son la propiedad de contenido XAML pero que se utilizan para la representación de texto de la interfaz de usuario, en el que se requiere que aparezcan elementos de espacio en blanco determinados, como avances de línea. La sintaxis de atributo no puede conservar los avances de línea, pero la sintaxis de elementos de propiedad sí, siempre que esté activa la conservación de espacios en blanco significativos.

Un elemento de propiedad no se representa en el árbol lógico. Un elemento de propiedad no es más que una sintaxis determinada para establecer una propiedad, y no es ningún elemento que tenga una instancia o un objeto que lo respalde.

Sintaxis de contenido XAML

La sintaxis de contenido XAML es una sintaxis que sólo está habilitada en las clases que especifican ContentPropertyAttribute como parte de su declaración de clase. El objeto ContentPropertyAttribute requiere un parámetro que especifique la propiedad por nombre definida para ser la propiedad de contenido de ese tipo de elemento (incluyendo las clases derivadas). La propiedad así designada es la propiedad de contenido XAML de un elemento. Cuando lo procesa un procesador de XAML, los elementos secundarios o el texto interno que se encuentran entre las etiquetas de apertura y cierre del elemento se asignarán como el valor de esa propiedad de contenido XAML. Puede especificar elementos de propiedad para la propiedad de contenido si lo desea, y hacer explícito su marcado. Esta técnica resulta útil a veces para aumentar la claridad del marcado o por motivos de estilo de marcado; sin embargo, en general la intención de una propiedad de contenido es simplificar el marcado para que los elementos que poseen una relación intuitiva de tipo elemento primario-elemento secundario se puedan anidar directamente. Las etiquetas de los elementos de propiedad de las demás propiedades de un elemento no se asignan como "contenido"; se procesan previamente en el flujo de trabajo del analizador y no se las considera "contenido".

Al igual que cualquier otra propiedad, la propiedad de contenido XAML de un objeto será de un tipo específico. Éste puede ser el tipo Object. El tipo de esa propiedad de contenido ayuda a definir el modelo de contenido de un objeto. Por ejemplo, un tipo de Object es dinámico en el sentido de que cualquier objeto puede convertirse en el contenido, pero incluso esta asignación de tipos dinámica implica que el contenido debe ser un objeto único. El objeto único podría ser un objeto de colección, pero incluso en este caso sólo puede haber un objeto de colección de este tipo asignado como contenido.

Para aceptar varios elementos de objeto (o texto interno) como contenido, el tipo de la propiedad de contenido debe ser específicamente un tipo de colección. De forma similar a la sintaxis de elementos de propiedad para los tipos de colección, un procesador de XAML debe identificar los tipos que son tipos de colección. Si un elemento tiene una propiedad de contenido XAML y el tipo de la propiedad de contenido XAML es una colección, no es necesario especificar el tipo de colección implícito en el marcado como un elemento de objeto, ni especificar la propiedad de contenido XAML como elemento de propiedad. Por consiguiente, el modelo de

contenido aparente en el marcado puede tener ahora varios elementos secundarios asignados como contenido.

La especificación XAML declara que un procesador de XAML puede exigir que los elementos de objeto que se utilizan para rellenar la propiedad de contenido de XAML dentro de un elemento de objeto sean contiguos y no se mezclen. Esta restricción contra la mezcla de elementos de propiedad y contenido la aplica el procesador de WPFXAML.

Puede utilizar un elemento de objeto secundario como el primer marcado inmediato dentro de un elemento de objeto. A continuación, puede introducir elementos de propiedad. O bien, puede especificar uno o varios elementos de propiedad, seguidos del contenido y, por último, más elementos de propiedad. Pero una vez que un elemento de propiedad sigue al contenido, no se podrá introducir ningún otro tipo de contenido, únicamente se podrán agregar elementos de propiedad.

Este requisito de orden de elemento de contenido/propiedad no se aplica al texto interno utilizado como contenido. Sin embargo, mantener el texto interno contiguo es un buen estilo de marcado, ya que el espacio en blanco significativo será difícil de detectar visualmente en el marcado si los elementos de propiedad están intercalados en el texto interno.

Propiedades asociadas

Las propiedades asociadas son un concepto de programación introducido en XAML según el cual, un tipo puede ser el propietario de las propiedades y definirlas, pero cualquier elemento puede establecerlas. El escenario principal para el que están pensadas las propiedades asociadas es aquél que permite a los elementos secundarios de un árbol de elementos enviar información a un elemento primario sin necesidad de un modelo de objetos compartido por todos los elementos. Y a la inversa, los elementos primarios pueden utilizar las propiedades asociadas para enviar información a los elementos secundarios.

Las propiedades asociadas utilizan una sintaxis que a primera vista es similar a la de los elementos de propiedad, ya que también se debe especificar una combinación *nombreDeTipo.nombreDePropiedad*. Hay dos diferencias importantes:

- Puede utilizar la combinación *nombreDeTipo.nombreDePropiedad* incluso al establecer una propiedad asociada a través de la sintaxis de atributo. Las propiedades asociadas son el único caso en el que certificar el nombre de la propiedad es un requisito en una sintaxis de atributo.

- También puede utilizar la sintaxis de elementos de propiedad para las propiedades asociadas. Sin embargo, en la sintaxis de elementos de propiedad típica, el *nombreDeTipo* especificado es el elemento de objeto que contiene el elemento de propiedad. Si se está haciendo referencia a una propiedad asociada, el *nombreDeTipo* es la clase que define la propiedad asociada, no el elemento de objeto contenedor.

Eventos asociados

Los eventos asociados son otro concepto de programación introducido en XAML según el cual, un tipo puede definir los eventos, pero los controladores pueden estar asociados a cualquier objeto. A menudo, el tipo que define un evento asociado es un tipo estático que define un servicio, y algunas veces estos eventos asociados están expuestos por un alias de evento enrutado en los tipos que exponen el servicio. Los controladores para los eventos asociados se especifican mediante la sintaxis de atributo. Al igual que sucede con los eventos asociados, la sintaxis de atributo se expande para los eventos asociados con objeto de permitir el uso de *nombreDeTipo.nombreDeEvento*, donde nombreDeTipo es la clase que proporciona los descriptores de acceso de controlador de eventos Add y Remove para la infraestructura de evento asociada, y nombreDeEvento es el nombre del evento.

Espacios de nombres XML.

En ninguno de los ejemplos de sintaxis anteriores se especificó un espacio de nombres distinto del predeterminado. En las aplicaciones de WPF típicas se especifica el espacio de nombres WPF como espacio de nombres predeterminado. Puede especificar espacios de nombres distintos del predeterminado y seguir utilizando prácticamente los mismos tipos de sintaxis, pero siempre que se mencione el nombre de una clase a la que no se pueda tener acceso dentro del espacio de nombres predeterminado, dicho nombre debe ir precedido por el prefijo del espacio de nombres XML que se utilizó para asignar el espacio de nombres de CLR correspondiente. Por ejemplo, <custom:MyElement/> es la sintaxis de elementos de objeto para crear una instancia de la clase MyElement, donde el espacio de nombres de CLR que contiene dicha clase (y posiblemente el ensamblado externo que contiene ese espacio de nombres) se asignó previamente al prefijo custom.

Extensiones de marcado

XAML define una entidad de programación de extensión de marcado que ofrece una vía alternativa al control habitual de atributos o elementos de objetos del procesador de XAML, y cede el procesamiento a una clase de respaldo. La implementación de WPF de un procesador de XAML utiliza la clase abstracta MarkupExtension como base para todas las extensiones de marcado admitidas por WPF. El carácter que identifica una extensión de marcado ante un procesador de XAML al utilizar la sintaxis de atributo es la llave de la apertura ({), seguida por cualquier carácter distinto de una llave de cierre (}). La primera cadena que sigue a la llave de la apertura debe hacer referencia a la clase que proporciona el comportamiento de la extensión en particular, donde la referencia puede omitir la subcadena "Extensión" si dicha subcadena forma parte del nombre de clase verdadero. A partir de ese momento, puede aparecer un solo espacio, y la implementación de extensión utiliza cada carácter subsiguiente como entrada, hasta que se encuentra la llave de cierre. Las extensiones de marcado en WPF están diseñadas principalmente para proporcionar un medio de hacer referencia a otros objetos ya existentes, o referencias diferidas a los objetos que se

evaluarán en tiempo de ejecución, mientras se utiliza la sintaxis de atributo. Por ejemplo, un enlace de datos simple se consigue especificando la extensión de marcado {Binding} en lugar del tipo de valor que tomaría normalmente una propiedad determinada. Muchas de las extensiones de marcado permiten la existencia de una sintaxis de atributo para propiedades en las que, de lo contrario, no sería posible una sintaxis de atributo. Por ejemplo, un objeto Style es un tipo de referencia relativamente complejo que contiene varias propiedades, cada una de las cuales también toma objetos byref y no primitivas. Pero los estilos se crean normalmente como un recurso y, a continuación, se hace referencia a ellos mediante una de las dos extensiones de marcado que solicitan un recurso. La extensión cede la evaluación del valor de la propiedad a una búsqueda de recursos y habilita el ofrecer el valor de la propiedad Style, tomando el tipo Style, en la sintaxis de atributo de la manera siguiente:

```
<Button Style="{StaticResource MyStyle}">My button</Button>
```

Aquí, StaticResource identifica la clase StaticResourceExtension que proporciona la implementación de la extensión de marcado. La cadena siguiente, MyStyle, se utiliza como la entrada para el constructor StaticResourceExtension no predeterminado, donde el parámetro tal y como se toma de la cadena de extensión declara el objeto ResourceKey solicitado. Se espera que MyStyle sea el valor x:Key (atributo) de un objeto Style definido como recurso. El uso de Extensión de marcado StaticResource requiere que el recurso se utilice para proporcionar el valor de la propiedad Style mediante una lógica de búsqueda de recursos estáticos en el momento de la carga.

Apéndice B

.NET

B.1 Plataforma .NET

En este capítulo explicaremos en qué consiste la plataforma .NET y trataremos el estudio de la programación del lenguaje C# mostrando sus principales características.

.NET es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones.

Su propuesta es ofrecer una manera rápida, económica, segura y robusta de desarrollar aplicaciones permitiendo una integración más rápida y ágil entre empresas, y un acceso más simple y universal a toda la información desde cualquier tipo de dispositivo.

Con estos objetivos, Internet aparece como la base de un sistema operativo distribuido sobre el cual se ejecutarán aplicaciones que estarán preparadas para relacionarse entre sí de manera transparente. La programación del futuro se hará sobre un gran sistema operativo que residirá en Internet de forma que la información y las aplicaciones, servicios en este caso, ya no estarán en nuestro PC, sino en la Red.

Microsoft proporciona una plataforma que incluye los siguientes componentes básicos:

- Infraestructura de servidores, incluyendo Windows y .NET Enterprise Servers.
- Software de dispositivos .NET para hacer posible una nueva generación de dispositivos inteligentes (ordenadores, teléfonos, PDAs, consolas de juegos, etc) que puedan funcionar en .NET
- Herramientas de programación para crear servicios Web XML, con soporte multilenguaje: .NET Framework y Visual Studio.

B.1.1 .NET Framework

.NET Framework es un modelo de programación de Microsoft para desarrollar, implementar y ejecutar servicios Web XML y todos los tipos de aplicaciones (de escritorio, para dispositivos móviles o basadas en Web). Incorpora servicios Web XML, que integran aplicaciones y componentes poco complementados diseñados para el heterogéneo entorno informático actual

mediante la comunicación con protocolos de Internet estándar como SOAP, WSDL (Lenguaje de descripción de servicios Web) y UDDI (Integración, descubrimiento y descripción universal).

.NET Framework está formado por tres partes principales. La primera es Common Language Runtime (CRL), que asume la responsabilidad de ejecutar la aplicación. CRL garantiza que se cumplan todas las dependencias de la aplicación, administra la memoria y controla cuestiones como la seguridad y la integración de lenguajes. Common Language Runtime proporciona muchos servicios que simplifican el desarrollo del código y la implementación de la aplicación a la vez que aumenta la fiabilidad de la aplicación.

La segunda parte es la de las clases principales unificadas. Estas clases proporcionan todos los recursos que requiere un desarrollador para generar una aplicación moderna, incluida la compatibilidad con XML, las conexiones de red y el acceso a datos. Tener estas clases unificadas significa que un desarrollador que desarrolle cualquier tipo de aplicación, basada en Windows o en Web, utiliza las mismas clases. Esta coherencia aumenta la productividad del desarrollador y la reutilización de código.

La tercera y última parte es la de las clases de presentación, que incluyen ASP.NET para el desarrollo de aplicaciones Web, así como servicios WEB XML y Windows Forms para el desarrollo de aplicaciones basadas en Windows.

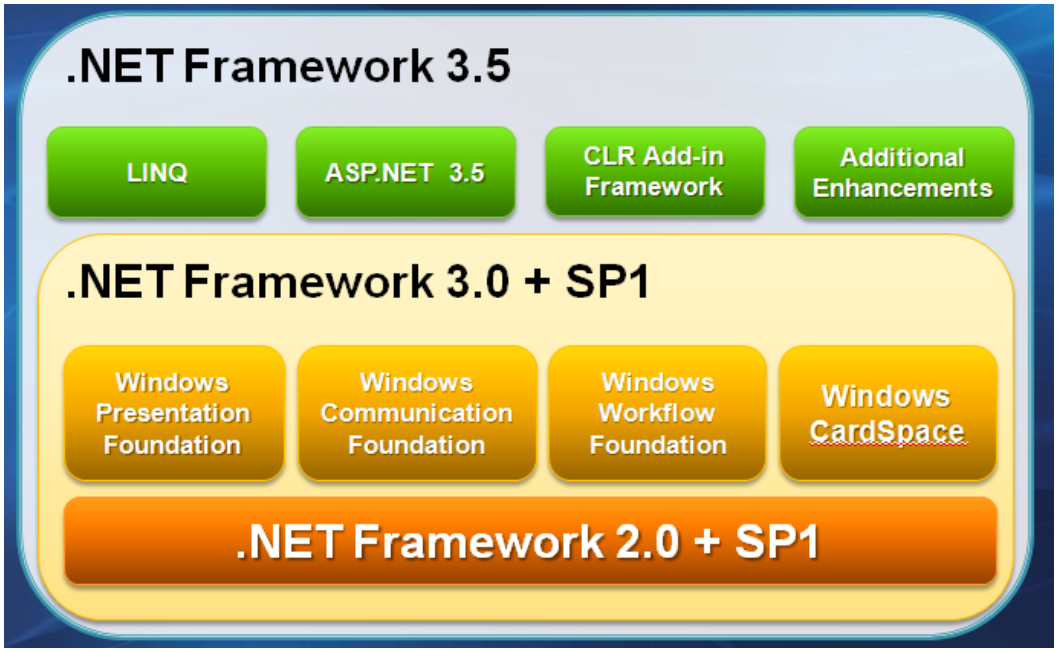
B.1.2 .NET Compact Framework

.NET Compact Framework aporta la eficacia del entorno de programación .NET Framework a los dispositivos compactos. Es un entorno independiente del hardware, para la ejecución de programas en dispositivos de computación con limitaciones de recursos, entre los que se encuentran los asistentes de datos personales (PDA) como Pocket PC, teléfonos móviles, dispositivos de computación para automóviles, etc.

.NET Compact Framework es un subconjunto de la biblioteca de clases .NET Framework y también contiene clases diseñadas expresamente para él. Hereda la arquitectura completa de Common Language Runtime y la ejecución de código administrado. Ofrece las siguientes funciones principales:

- Ejecuta programas independientes del hardware y el sistema operativo.
- Admite protocolos de red comunes y se conecta perfectamente con servicios XML Web.
- Proporciona a los desarrolladores un modelo para orientar sus aplicaciones y componentes ya sea a una amplia gama de dispositivos o a una categoría específica de éstos.
- Facilita el diseño y la optimización de los recursos de sistema limitados.
- Obtiene un rendimiento óptimo en la generación de código nativo cuando se utiliza compilación Just-In-Time (JIT).

B.1.3 .NET Framework 3.5



B.1: Elementos nuevos de .NET Framework 3.5 e integrados de versiones anteriores

Microsoft .NET Framework versión 3.5. .NET Framework es un componente integral de Windows que admite la creación y la ejecución de la siguiente generación de aplicaciones y servicios Web. Los componentes clave de .NET Framework son Common Language Runtime (CLR) y la biblioteca de clases .NET Framework, que incluye ADO.NET, ASP.NET, formularios Windows Forms y Windows Presentation Foundation (WPF). .NET Framework proporciona un entorno de ejecución administrado, un desarrollo e implementación simplificados y la integración con una gran variedad de lenguajes de programación.

.NET Framework versión 3.5 se basa en las versiones 2.0 y 3.0 y sus Service Pack correspondientes. .NET Framework versión 3.5 Service Pack 1 actualiza los ensamblados de la versión 3.5 e incluye nuevos Service Pack para las versiones 2.0 y 3.0.

.NET Compact Framework versión 3.5 amplía la compatibilidad con aplicaciones móviles distribuidas al incorporar la tecnología Windows Communication Foundation (WCF). También agrega nuevas características de lenguaje como LINQ, incluye nuevas API basadas en los comentarios de la comunidad y mejora la depuración con herramientas y características de diagnóstico actualizadas.

Complementos y extensibilidad

El ensamblado System.AddIn.dll de .NET Framework 3.5 proporciona un grado de compatibilidad eficaz y flexible a los programadores de aplicaciones extensibles. Introduce una nueva arquitectura y un nuevo modelo que ayudan a los programadores en las tareas preliminares al agregar extensibilidad a una aplicación y garantizar que sus extensiones siguen funcionando cuando la aplicación host cambia. El modelo proporciona las características siguientes:

- **Detección**: Puede buscar y administrar con facilidad conjuntos de complementos en diversas ubicaciones de un equipo con la clase AddInStore. Puede utilizar esta clase para buscar y obtener información sobre los complementos mediante sus tipos base sin tener que cargarlos.
- **Activación**: Una vez que una aplicación elige un complemento, la clase AddInToken facilita su activación. Sólo debe elegir un nivel de aislamiento y un recinto de seguridad, y el sistema se encargará de todo lo demás.
- **Aislamiento**: La compatibilidad con dominios de aplicación y el aislamiento de procesos de complementos está integrada. El nivel de aislamiento de cada complemento depende del host. El sistema se ocupa de cargar los dominios de aplicación y los procesos y de cerrarlos una vez que sus complementos detienen su ejecución.
- **Recintos de seguridad**: Resulta sencillo configurar los complementos con un nivel de confianza predeterminado o personalizado. Los conjuntos de permisos admitidos son los permisos de Internet, Intranet, plena confianza y el conjunto de permisos del host, así como las sobrecargas que permiten al host especificar un conjunto de permisos personalizados.
- **Composición de la interfaz de usuario**: El modelo de complementos admite la composición directa de controles de Windows Presentation Foundation (WPF) que traspasan los límites del dominio de aplicación. Puede hacer que los complementos contribuyan directamente en la interfaz de usuario del host a la vez que mantiene los beneficios que suponen el aislamiento, la capacidad de descarga, los recintos de seguridad y el control de versiones.
- **Control de versiones**: La arquitectura de los complementos hace posible que los hosts introduzcan nuevas versiones de su modelo de objetos sin interrumpir la compatibilidad con los complementos existentes y sin que esto afecte en modo alguno a la experiencia del desarrollador con los complementos nuevos.

B.1.4 Visual Studio

Es un conjunto complejo de herramientas de desarrollo para construir aplicaciones Web, servicios Web, aplicaciones Windows o de escritorio y aplicaciones para dispositivos móviles. Se pueden crear soluciones utilizando varios lenguajes y en las que la parte de diseño se implementa separadamente con respecto a la programación.

B.1.5 C#

Es el nuevo lenguaje diseñado por Microsoft para crear una amplia gama de aplicaciones que se ejecutan en .NET Framework. C# es simple, eficaz, con seguridad de tipos y orientado a objetos. Con él se pretende dar una alternativa a Java, que es su más directo competidor en el mundo de Internet.

C# combina elementos de múltiples lenguajes como C++, Java, Delphi o Visual Basic, pero es el único diseñado específicamente para ser utilizado en la plataforma .NET. Debido a esto, programar en C# resulta mucho más sencillo e intuitivo que con el resto de lenguajes.

C# es el lenguaje nativo para acceder a todos los servicios que proveerá .NET en el futuro, y sigue evolucionando continuamente para proporcionar mayor eficiencia y funcionalidad.

A continuación enumeramos las principales características que definen al lenguaje de programación C#. Algunas de estas características no son propias del lenguaje, sino de la plataforma .NET, aunque se listan aquí ya que tienen una implicación directa en el lenguaje:

- **Sencillez de uso**: C# elimina muchos elementos añadidos por otros lenguajes y que facilitan su uso y compresión. Es por ello que se dice que C# es autocontenido.
- **Modernidad**: Al ser C# un lenguaje de última generación, incorpora elementos que se ha demostrado a lo largo del tiempo que son muy útiles para el programador y que otros lenguajes habría que simularlos.
- **Orientado a objetos**: C# soporta todas las características del paradigma de la programación orientada a objetos, como la encapsulación, la herencia y el polimorfismo.
- **Orientado a componentes**: La propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular.
- **Recolección de basura**: Todo lenguaje incluido en la plataforma .NET tiene a su disposición el recolector de basura del CLR. Esto implica que no es necesario incluir instrucciones de destrucción de objetos en el lenguaje.
- **Seguridad de tipos**: C# incluye mecanismos de control de acceso a tipos de datos, lo que garantiza que no se produzcan errores difíciles de detectar. Para ello, el lenguaje provee de

una serie de normas de sintaxis, no se pueden usar variables no inicializadas previamente, y en el acceso a tablas se hace una comprobación de rangos.

- **Instrucciones seguras**: Para evitar errores comunes como se producían programando en otros lenguajes, en C# se han impuesto una serie de restricciones en el uso de instrucciones de control más comunes.
- **Unificación de tipos**: En C# todos los tipos derivan de una superclase común llamada System.Object, por lo que automáticamente heredarán todos los miembros definidos en esta clase. Es decir, son objetos.
- **Extensión de los operadores básicos**: C# permite redefinir el significado de la mayoría de los operadores (incluidos el de la conversión) cuando se apliquen a diferentes tipos de objetos.
- **Extensión de modificadores** : C# ofrece, a través de los atributos, la posibilidad de añadir a los metadatos del módulo resultante de la compilación de cualquier fuente información adicional a la generada por el compilador que luego podrá ser consultada en tiempo de ejecución a través de la biblioteca de reflexión de .NET.
- **Eficiente**: En C#, todo el código incluye numerosas restricciones para garantizar su seguridad, no permitiendo el uso de punteros. Sin embargo, y a diferencia de Java, existen modificadores para saltarse esta restricción, pudiendo manipular objetos a través de punteros.
- **Compatible**: Para facilitar la migración de programadores de C++ o Java a C#, no sólo se mantiene una sintaxis muy similar a la de los dos anteriores lenguajes, sino que el CLR también ofrece la posibilidad de acceder a código nativo escrito como funciones sueltas no orientadas a objetos.

B.2 XML

XML, siglas en inglés de *Extensible Markup Language* (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML, XAML.

XML no nació sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Las principales características de este lenguaje son:

- **Es ampliable**: es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- **La portabilidad de XML** es consecuencia de que es el propio desarrollador el que define las etiquetas y los atributos. No se necesitan bibliotecas ni servidores de aplicaciones especiales para leer un documento XML. Los documentos XML son archivos de texto normal, por lo que no requieren un software propietario para interpretarlos, como ocurre con la mayoría de los archivos binarios.
- **Estructura sencilla**: si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones.
- El **contenido es independiente de la presentación**: Sus clientes se concentrarán en usar HTML y CSS para definir el diseño y la presentación, lo cual no se verá afectado por cambios en la información, que se almacena por separado en un archivo XML.

XML proporciona a los programadores la capacidad de ofrecer datos estructurados desde muchas aplicaciones al sistema local con el fin de trabajar localmente con ellos. Además, por su parte, .NET Framework nos proporciona un conjunto de clases que están categorizadas de acuerdo a la funcionalidad que ofrecen, como lectura y escritura de documentos XML, validación de documentos XML, navegación y selección de nodos, administración de esquemas, y transformación de documentos XML.

.NET Framework ofrece la posibilidad de diseñar un conjunto integrado de clases XML e innovar en el entorno XML. Las clases XML que se suministran son elementos básicos de .NET Framework. Estas clases ofrecen una solución abierta, y compatible con estándares. Toda esta funcionalidad se encuentra dentro del espacio de nombres system.xml.dll. Entre los más usados tenemos: System.xml, System.Xml.Schema, System.Xml.XPath, System.Xml.Xsl.

El espacio de nombres System.XML, tiene un conjunto completo de clases XML para análisis, validación y manipulación de datos XML mediante sistemas de lectura, sistemas de escritura y componentes compatibles con el consorcio (W3C) DOM. También se explican las consultas XPath (XML Path Language) y las transformaciones XSLT (Extensible Stylesheet Language Transformations). Las clases principales del espacio de nombres XML son:

- La clase **XmlTextReader** proporciona acceso rápido de lectura, sin almacenamiento en caché y con desplazamiento sólo hacia delante a datos XML.
- La clase **XmlNodeReader** proporciona un objeto XmlReader a través del subárbol de nodo DOM dado.
- La clase **XmlValidatingReader** proporciona validación de esquemas DTD, XDR y XSD.

- La clase **XmlTextWriter** proporciona una forma rápida y de desplazamiento sólo hacia delante para generar código XML.
- La clase **XmlDocument** implementa las especificaciones W3C, Document Object Model Level 1, Core y Core DOM Level2.
- La clase **XmlDataDocument** proporciona una implementación de un objeto XmlDocument que se puede asociar a un objeto DataSet. Los datos XML estructurados se pueden ver y manipular simultáneamente a través de la representación relacional del objeto DataSet o de la representación de árbol del objeto XmlDataDocument.
- La clase **XPathDocument** proporciona una caché rápida y de alto rendimiento con el fin de procesar documentos XML para XSLT.
- La clase **XPathNavigator** proporciona un modelo de datos W3C XPath 1.0 sobre un almacén con un modelo de desplazamiento de tipo cursor.
- La clase **XslTransform** corresponde a un procesador XSLT compatible con la especificación W3C XSLT 1.0 con el fin de transformar documentos XML.
- Las clases del modelo de objetos **XmlSchema** proporcionan un conjunto de clases que se pueden examinar y que reflejan directamente la especificación W3C XSD. Proporcionan la capacidad de crear esquemas XSD mediante programación.
- La clase **XmlSchemaCollection** proporciona una biblioteca de esquemas XDR y XSD. Estos esquemas, almacenados en memoria, proporcionan validación rápida en tiempo de análisis para el objeto XmlValidatingReader.

B.2.1 Estructura

Para cumplir los requisitos de portabilidad y apertura, XML utiliza texto normal para representar datos incrustados en etiquetas que describen la estructura de los datos. Las etiquetas XML son los elementos encerrados entre paréntesis angulares (<, >). El primer elemento (<?xml version=" 1.0?>) que aparecería en nuestro fichero XML indica la versión de XML a la que es conforme el documento, el resto del documento contiene los datos.

Un documento XML válido y bien formado debe ajustarse a una serie de criterios. Solo debe existir un único elemento raíz actuando de contenedor para todos los datos, el cual contiene los elementos individuales como subelementos anidados. Para cada elemento *"inicio"* <*identificador*>, debe existir el correspondiente elemento fin denotado como <*/identificador*> que indica el final de los datos correspondientes a ese elemento. El sangrado y los saltos de línea no son significativos en XML pero se pueden utilizar para leer el documento con mayor facilidad.

APÉNDICE C

CONTENIDO DEL CD-ROM

El CD-ROM que se adjunta con la memoria contiene los siguientes directorios:

- **Código Fuente**: en esta carpeta se incluyen toda la solución de Visual Studio 2008 .NET para la línea de productos de marcadores deportivos, con el código fuente del proyecto desarrollado.
- **Memoria**: En este directorio se encuentra la versión de esta memoria en formato PDF.
- **Aplicaciones finales**: en esta carpeta se encuentran varios archivos .CAB con diversas soluciones de la línea de productos listos para ser instalados en un dispositivo móvil.
- **Software diverso**: en este directorio están otros programas software usados durante la elaboración del proyecto, como son el starUML (para la elaboración de diagramas), REM (para la fase de análisis).