UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

GUARDAMED: GIS DE UN ALMACEN DE MEDICAMENTOS

Tutor: Manuel Velasco de Diego

Autora: Almudena García Castilla

Noviembre, 2011

ÍNDICE

ÍNDICE 1

Contenido 1

Figuras 5

Tablas 6

1. INTRODUCCIÓN 7

1.1 Descripción general del problema 7

1.2 Descripción específica del problema 8

1.3 Terminología empleada 9

1.4 Índice del proyecto 12

2. ESTADO DEL ARTE 13

2.1 Bases de Datos 13

2.1.1 Introducción a las bases de datos 13

2.1.2 Definición de Base de Datos 14

2.1.3 Modelos de datos 17

2.1.4 Definición de Sistema Gestor Base de Datos 21

2.2 Bases de datos Orientadas a objetos 25

2.2.1 ¿Por qué surgen? 25

2.2.2 Orientación a Objetos: características generales 27

2.2.3 Definición Base de Datos Orientada a Objetos 31

2.2.4 Modelo de Objetos (OM) 31

2.2.5 Sistema Gestor de Base de Datos Orientada a Objetos .. 33

2.2.6 ODMG 36

2.3 Sistemas de Información Geográfica..... 37

2.3.1 Definición Sistemas de Información Geográfica 37

2.3.2 Componentes Sistemas de Información Geográfica..... 39

2.3.3 Tipo de información..... 41

2.3.3.1 Modelo raster 41

2.3.3.2 Modelo vectorial..... 42

2.3.4 Bases de Datos Geográficas 44

2.3.5 Funciones de los SIG 46

2.3.6 Aplicaciones de los SIG 47

3. OBJETIVOS..... 50

4. ENTORNO DE TRABAJO 52

4.1 Caché 52

4.1.1 Definición 52

4.1.2 Carácterísticas..... 52

4.1.2.1 Tecnología Post-relacional 52

4.1.2.2 Tecnología basada en objetos..... 54

4.1.2.3 Servidor de datos multidimensional de Caché 56

4.1.2.4 Servidor de aplicaciones Caché 58

4.1.2.5 Lenguajes soportados por Caché..... 61

4.1.3 Caché ObjectScript 65

4.1.3.1 Estructura general del lenguaje..... 66

4.1.3.2 Objetos 67

4.1.4 Caché Server Page (CSP) 69

4.1.4.1 %Request 72

4.1.5 Dynamic SQL 73

4.1.6 Caché Studio 73

5. MÉTODO DE RESOLUCIÓN 75

5.1 Requisitos 75

5.1.1 Requisitos Hardware 75

5.1.2 Requisitos de Usuario 76

5.1.3 Requisitos funcionales 77

5.1.3.1 Requisitos funcionales: Espaciales 77

5.1.3.2 Requisitos funcionales: Aplicación Web 78

5.2 Diseño de la Base de Datos 80

5.2.1 Modelo E/R 80

5.2.2 Modelo Relacional 83

5.2.3 Diseño Librería Espaciales 85

5.2.3.1 Funcionalidad detallada 91

5.3 Funcionalidad de la aplicación 108

5.3.1 Consultas 108

5.3.2 Introducción de datos 124

5.4 Manual de Usuario 129

5.4.1 Guardamed 129

5.4.1.1 Medicamento 132

5.4.1.2 Almacén 138

5.4.2 Introducción de datos.....144

5.4.2.1 Medicamentos.....147

5.4.2.2 Principios activos.....148

5.4.2.3 Almacén.....150

5.4.2.4 Relaciones.....159

6. EXPERIMENTACIÓN.....163

6.1 Introducción de un medicamento.....163

6.2 Consultas.....172

6.2.1 Tipo de medicamento en zona.....173

6.2.2 Zonas incompletas.....175

7. CONCLUSIONES.....179

7.1 Conclusiones personales.....180

8. PRESUPUESTO.....181

9. DESARROLLOS POSTERIORES.....182

10. BIBLOGRAFÍA.....185

Figuras

Figura 1: Modelos de bases de datos 20

Figura 2: SGBD 23

Figura 3: Ejemplo de Herencia 29

Figura 4: Ejemplo de herencia múltiple 29

Figura 5: Modelado de datos orientado a objetos 32

Figura 6: Características SGBDOO 35

Figura 7: Componentes de un SIG 39

Figura 8: Modelo Raster 42

Figura 9: Modelo Vectorial 43

Figura 10: Modelo Raster y Vectorial 43

Figura 11: Ejemplo forma geográfica 44

Figura 12: Distintas capas de datos 45

Figura 13: Ejemplo de almacenamiento multidimensional 53

Figura 14: Accesos a Caché 54

Figura 15: Acceso multidimensional 57

Figura 16: Servidor de aplicaciones Caché 60

Figura 17: Conexiones Java 63

Figura 18: Conexiones .NET – Caché 64

Figura 19: Caché Server Page 71

Figura 20: Cajoneras 78

Figura 21: Explicación de una cajonera 78

Figura 22: Sistema de coordenadas 86

Figura 23: Punto 86

Figura 24: Línea 87

Figura 25: Polígono 87

Figura 26: Cubo 89

Figura 27: Distancia entre dos puntos 91

Figura 28: Pertenece punto a polígono 95

Figura 29: Proyecciones del polígono 96

Figura 30: Polígono dentro de polígono 100

Figura 31: Parámetros del cubo 105

Figura 32:Tratamiento de las consultas 110

Figura 33: Tratamiento de la introducción de datos 125

Tablas

Tabla 1: Tipos de datos espaciales y sus métodos 90

1. INTRODUCCIÓN

Una primera introducción hará referencia a las bases de datos orientadas a objetos y a los Sistemas de Información Geográfica (SIG).

1.1 Descripción de general del problema

Desde la introducción de los ordenadores, muchos han sido los cambios que éstos han sufrido. Durante este tiempo los avances tecnológicos han sido muchos y variados, aunque una de las funciones principales de éstos siga en pie: almacenar información. La información de las empresas es un gran activo de las mismas, por lo que gestionarlo de forma rápida y eficiente es algo que les interesa y mucho. Así surgieron las bases de datos, con el origen de almacenar grandes cantidades de datos y gestionarlos de la mejor manera posible.

Las bases de datos se utilizan normalmente para guardar gran variedad de información dependiendo del dominio de la aplicación elegida. Los mecanismos de almacenamiento de datos actuales en una base de datos utilizan modelos formales que garantizan la consistencia, la seguridad, reducen la redundancia y permiten su uso concurrente. Debido a estas exigencias una base de datos modela datos de una manera distinta a su representación real. Tal es el caso del modelo relacional. Como es bien conocido, si los datos que se manejan son complejos y estructurados se necesitan crear mecanismos de armado y desarmado de datos para que aplicaciones de explotación y bases de datos puedan interactuar.

Con la llegada del paradigma de programación orientado a objetos y la aparición de estructuras de datos no atómicos, sugieren nuevos modelos para organizar información en una base de datos. Bases de datos orientadas a

objetos. Las nuevas aplicaciones necesitan un nuevo modelo de base de datos para suplir las necesidades actuales.

Por otro lado, cada vez existen un mayor número de aplicaciones cuyo objetivo es solucionar problemas de índole geográfica. Por ejemplo, cuando se prepara y planifica un viaje, se suele buscar temas de nuestro interés que se encuentren dentro de una determinada región como puedan ser museos, monumentos históricos… etc. Se buscan los que puedan estar cerca para no perder mucho tiempo desplazándonos de un sitio a otro. También son comunes las aplicaciones que buscan dentro de una determinada población hoteles o lugares de alojamiento.

Todo este tipo de aplicaciones manejan un determinado tipo de información: información espacial. Éste tipo de información y la gestión del mismo, se utiliza en los SIG: Sistemas de Información Espacial.

1.2 Descripción específica del problema

Todas las actividades que realizan las aplicaciones anteriormente descritas se han realizado de una forma manual con mapas, guías, listados…etc. Y el análisis, es decir, la combinación de dichas informaciones para extraer conclusiones. Sin embargo, en este terreno, la utilización de herramientas informáticas ha proporcionado nuevos medios para abordar el problema: los Sistemas de Información Geográficos. Los Sistemas de Información Geográficos tienen dos componentes fundamentales:

- Un modelo de datos en el que se almacenan las características de los objetos geográficos de forma similar a como se almacenan en una base de datos convencional, junto con información espacial (coordenadas) y las relaciones entre los distintos objetos (qué está conectado a qué, o junto a).

- Una colección de funciones que nos permiten interrogar a la base de datos y obtener respuestas, bien en base a listados o a imágenes (mapas).

Una característica esencial de los sistemas de información geográficos es que intentan capturar en su modelo de datos la realidad, y no una imagen determinada de ésta. Por ejemplo, en una ciudad tendremos el contorno de la misma, junto con el contorno de sus barrios, calles… etc. Así, con la unión de un SIG y una base de datos orientada a objetos, se crean objetos geográficos similares a los reales. Para ello y gracias al modelo orientado a objetos, se crean datos básicos a partir de los cuales, gracias a la jerarquía, se pueden crear objetos muchos más sofisticados. Así hacen que sea compatible con la jerarquía que se utiliza en los lenguajes de programación orientada a objetos.

1.3 Terminología empleada

Librería: conjunto de procedimientos y funciones (subprogramas) agrupadas en un archivo con el fin de que puedan aprovecharlas otros programas.

S.G.B.D (Sistema gestor de base de datos): tipo de software específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta.

Memoria caché: tipo de memoria en la que se almacenan una copia de datos originales. Cuando se accede por primera vez a un dato, se hace una copia a esta memoria, los siguientes accesos al dato se realizaran a la copia en lugar de al original. Esto se hace porque acceder a datos en memoria caché es mucho más rápido que hacerlo a otro tipo de dispositivos.

Aplicación informática: programa informático diseñado para facilitar al usuario la realización de determinados tipos de trabajo. Suele resultar una solución informática para la automatización de ciertas tareas complicadas como puede ser la contabilidad de una empresa o la gestión de un almacén.

Aplicación Web: sistema informático que puede ser utilizado por usuarios de todo el mundo. Acceden a la aplicación, a través de Internet, a un servidor que lo contiene. Dichas aplicaciones son muy populares debido a la facilidad para actualizar y mantener la aplicación sin necesidad de distribuirla ni instalarla. Un ejemplo de éstas es el correo electrónico o las tiendas on-line.

Sql (Structured Query Language): lenguaje utilizado para el acceso a bases de datos de tipo relacional. Permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo así realizar consultas para recuperar información de la base de datos.

Servidor de aplicaciones: aplicación informática o programa, capaz de ejecutar distintas tareas (distintas aplicaciones).

Lenguajes de script: lenguaje de programación diseñado para ejecutarse por medio de un intérprete (programa que analiza y ejecuta otros programas) y no mediante un compilador como el resto. El modo de ejecución, por medio de intérprete, del programa escrito en el lenguaje es independiente del lenguaje mismo.

ODBC (Open DataBase Connectivity): estándar de acceso a Bases de Datos desarrollado por Microsoft Corporation. El objetivo de ODBC es hacer posible el acceso a cualquier dato de cualquier aplicación, sin importar qué Sistema Gestor de Bases de Datos almacene los datos.

JDBC (Java Database Connectivity): un API (conjunto de funciones y métodos, que ofrece cierta biblioteca para ser utilizado por otro software) que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

Máquina virtual: software que crea un entorno virtual entre la plataforma de la computadora y el usuario final, permitiendo que este ejecute un software determinado.

URL (Uniform Resource Locutor): secuencia de caracteres, de acuerdo a un formato estándar, que se usa para nombrar recursos, como documentos e imágenes en Internet, por su localización. El formato general de un URL es: *Protocolo://máquina/directorio/fichero*

1.4 Índice del proyecto

El proyecto se ha estructurado en los siguientes capítulos:

Capítulo 1*:* se trata una introducción general del proyecto, donde se da una visión a grandes rasgos de los temas tratados en los capítulos siguientes, así como la terminología utilizada.

Capítulo 2: se expone el estado de la cuestión, es decir, los fundamentos teóricos que han dado lugar al desarrollo del proyecto.

Capítulo 3: se describen los objetivos que se pretenden conseguir con el proyecto.

Capítulo 4: se especifica el entorno de trabajo, es decir, las herramientas y lenguajes de programación utilizados para el desarrollo del proyecto.

Capítulo 5: se exponen con detalle el análisis y diseño de la aplicación. Se especifican las funcionalidades, requisitos y el diseño de este proyecto.

Capítulo 6: se muestran algunos ejemplos y sus correspondientes resultados, los cuales se realizan en la aplicación creada para verificar su correcto comportamiento y mostrar que cumple los requisitos definidos.

Capítulo 7: se exponen las conclusiones obtenidas tras la realización del proyecto.

Capítulo 8: se proponen desarrollos futuros o mejoras que podrían realizarse tras la finalización de este proyecto, para ampliar o mejorar sus funcionalidades.

Capítulo 9: contiene la bibliografía que se ha empleado para la realización del proyecto.

2. ESTADO DEL ARTE

2.1 Bases de Datos

A continuación se expone toda la información relativa a las bases de datos.

2.1.1 Introducción a las bases de datos

Una base de datos o banco de datos (en inglés: database) es una colección de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

En cualquier ámbito, ya sea empresarial o personal, surge la necesidad de almacenar datos de forma que nos resulte fácil su acceso en cada momento y, por consiguiente, facilite nuestras actividades. Algunos ejemplos de datos almacenables serían: teléfonos, fotos, facturas, clientes…

Antes de la llegada del ordenador se almacenaban datos en papel y estos se guardaban en grandes muebles archivos. Para acceder a los datos de forma eficiente era totalmente necesario que se guardaran de forma ordenada, si no era un trabajo casi imposible. Otra pega era relacionar datos de diferentes archivos ya que era muy costoso en tiempo y trabajo.

La llegada de los ordenadores hizo que el proceso de almacenar datos, fuese mucho más sencillo ya que permitía manipularlos fácilmente y mostrarlos de diversas formas. Con el considerable ahorro en trabajo y espacio de almacén.

Cada día la cantidad de información manejada es mayor, por lo que se hace imprescindible almacenarla, gestionarla y recuperarla de forma ágil,

eficiente y precisa. Para esto son necesarios ordenadores con mayor capacidad de memoria y de proceso.

Llegados a éste punto cabe destacar la diferencia entre datos e información. Con datos se refiere a textos, letras o números. Sin embargo, cuando se habla de información se refiere a datos que pueden ser relacionados entre sí, de alguna manera, con el objetivo de sacar alguna conclusión que permita tomar decisiones en una empresa entre otros.

2.1.2 Definición de Base de Datos

Definir el termino 'Base de datos' no es algo sencillo, por ello en este apartado se intenta hacerlo de una manera clara y nombrando diferentes aspectos, para llegar a una correcta definición del mismo.

De una forma muy general y sencilla, una base de datos se puede definir como un conjunto de datos y las relaciones que existen entre estos. Por ejemplo, la base de datos de una empresa podría contener:

- Elementos: tales como clientes, proveedores, ventas, artículos y compras.

- Relaciones: son las relaciones que existen entre los diferentes elementos. Algunos ejemplos serían: Clientes a los que le hemos realizado una venta, artículos de la venta, artículos adquiridos en una compra y compras realizadas a un proveedor.

Sin embargo, existen multitud de definiciones de base de datos, que han ido cambiando a lo largo del tiempo. Desde la introducción de la expresión 'base de datos' a comienzo de los años 60 hasta ahora, ha pasado mucho tiempo y muchas han sido las definiciones que este concepto ha tenido. Algunas de ellas son:

"*Colección de datos interrelacionados almacenados en conjunto sin redundancias perjudiciales o innecesarias. Su finalidad es servir a una aplicación o más, de la mejor manera posible; los datos se almacenan de modo que resulten independientes de los programas que los usan; se emplean métodos bien determinados para incluir nuevos datos y para modificar o extraer los datos almacenados*"

(Martin, 1975)

"*Colección de datos, donde los datos están lógicamente relacionados entre sí, tienen una definición y descripción comunes y están estructurados de una forma particular. Una base de datos es también un modelo del mundo real y, como tal, debe poder servir para toda una gama de usos y aplicaciones*"

(Conference des Statisticiens Européens, 1977)

"*Colección no redundante de datos que son compartidos por diferentes sistemas de aplicación*"

(Howe, 1983)

"*Colección integrada y generalizada de datos, estructurada atendiendo a las relaciones naturales de modo que suministre todos los caminos de acceso necesarios a cada unidad de datos con objeto de poder atender a todas las necesidades diferentes de los usuarios*"

(Deen, 1985)

"*Colección de datos interrelacionados*"

(Elsmari y Navathe, 1989)

A continuación se explican algunas de las características esenciales de una base de datos.

Todas las definiciones anteriores coinciden en definir a una base de datos como una colección de datos almacenados en un soporte informático. Los *datos* están *interrelacionados* y *estructurados* de acuerdo a un modelo (explicados más adelante) que intenta recoger toda la información posible sin que esta sea redundante. Para recoger información del mundo real y representarla en un modelo lógico hace falta tener ciertas restricciones semánticas, que son inherentes al modelo a las que hay que prestarles mucha atención.

Los datos no deben ser *redundantes,* no se debe repetir información. Esta redundancia debe ser controlada, de forma que no existan duplicidades innecesarias. Por tanto se debe controlar que no exista información repetida. Esto hace que la adquisición de datos sea más efectiva y exista un mayor control del espacio de almacenamiento.

Las bases de datos deben atender a *múltiples usuarios* y *a diferentes aplicaciones*. Su estructura y almacenamiento debe permitir el acceso desde diferentes aplicaciones para atender a todas las necesidades que se planteen respecto a los datos. Se consigue un mayor valor informativo por ser una información recogida de forma más universal, evitando así las antiguas formas de almacenamiento de datos orientados a los procesos, los llamados ficheros.

Uno de los aspectos fundamentales es la *independencia* (tanto física como lógica) entre datos y tratamiento. Es decir, se debe asegurar que sea cual sea el dispositivo de almacenamiento, el usuario o la aplicación se podrá acceder y manipular los datos de la base.

La estructura o *esquema* de la base de datos, debe reflejar las interrelaciones y las restricciones del mundo real. La estructura es la definición o descripción del conjunto de datos contenidos en la base de datos, deben ser *únicas* y estar integradas con los mismos datos. En las bases de datos, la descripción y la definición se almacena junto con los datos, de modo que

cualquier cambio que se produzca en dicha documentación se ha de reflejar y quedar recogido en el sistema.

De acuerdo con las características anteriores se puede dar una definición correcta y completa de una Base de Datos:

"*Colección o depósito de datos integrados, almacenados en soporte secundario (no volátil) y con redundancia controlada. Los datos, que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de ellos, y su definición (estructura de base de datos) única y almacenada junto con los datos, se ha de apoyar en un modelo de datos, el cual ha de permitir captar las interrelaciones y restricciones existentes en el mundo real. Los procedimientos de actualización y recuperación, comunes y bien determinados, facilitarán la seguridad en el conjunto de los datos*"

(Fundamentos y modelos de Bases de datos, Ed: Ra-ma)

2.1.3 Modelos de datos

Una de las características fundamentales de los sistemas de bases de datos es que proporcionan cierto nivel de abstracción de datos, al ocultar las características sobre el almacenamiento físico que la mayoría de usuarios no necesita conocer. Los modelos de datos son el instrumento principal para ofrecer dicha abstracción. Un *modelo de datos* es un conjunto de conceptos que sirven para describir la estructura de una base de datos: los datos, las relaciones entre los datos y las restricciones que deben cumplirse sobre los datos. Los *modelos de datos* contienen también un conjunto de operaciones básicas para la realización de consultas (lecturas) y actualizaciones de datos. Además, los modelos de datos más modernos incluyen conceptos para especificar comportamiento, permitiendo especificar un conjunto de operaciones definidas por el usuario.

Los modelos de datos se pueden clasificar dependiendo de los tipos de conceptos que ofrecen para describir la estructura de la base de datos:

- Modelos de datos de alto nivel, o *modelos conceptuales*, disponen de conceptos muy cercanos al modo en que la mayoría de los usuarios percibe los datos.

- Modelos de datos de bajo nivel, o *modelos físicos*, proporcionan conceptos que describen los detalles de cómo se almacenan los datos en el ordenador. Los conceptos de los modelos físicos están dirigidos al personal informático, no a los usuarios finales.

- Entre estos dos extremos se encuentran los *modelos lógicos*, cuyos conceptos pueden ser entendidos por los usuarios finales, aunque no están demasiado alejados de la forma en que los datos se organizan físicamente. Los modelos lógicos ocultan algunos detalles de cómo se almacenan los datos, pero pueden implementarse de manera directa en un ordenador.

Los modelos de datos también, y más comúnmente, se clasifican de acuerdo al modelo de administración de los datos. Estos son:

- Modelo jerárquico: utiliza jerarquías o árboles para la representación lógica de los datos. Los archivos son organizados en jerarquías, y normalmente cada uno de ellos se corresponde con una de las entidades de la base de datos.

- Modelo de red: intenta superar las deficiencias del enfoque jerárquico, permitiendo el tipo de relaciones de muchos a muchos.

- Modelo relacional: usa una colección de tablas para representar tanto los datos como sus relaciones. Cada tabla tiene varias columnas y cada columna tiene un nombre único.

- Modelo Entidad-Relación (E/R): utiliza entidades, que representan "elementos" del mundo real que son distinguibles de otros, y las *relaciones* entre ellos.

- Modelo de datos orientado a objetos: aquellos que utilizan para representar los datos objetos, con todas sus características y lo que conlleva la utilización de los mismos: herencia, encapsulación, polimorfismo… etc. Es el que se ha seguido en este proyecto.

- Modelo de datos Objeto/Relacionales: comparten características de los modelos relacionales y de los orientados a objetos. Almacena Objetos en bases de datos relacionales.

- Modelo de datos semiestructurados: permite la especificación de datos donde los elementos de datos individuales del mismo tipo pueden tener diferentes conjuntos de atributos. En los modelos anteriormente descritos, cada elemento de datos de un tipo particular debe tener el mismo conjunto de datos.

Los modelos han ido avanzando cronológicamente y forman parte de distintas generaciones de modelos:

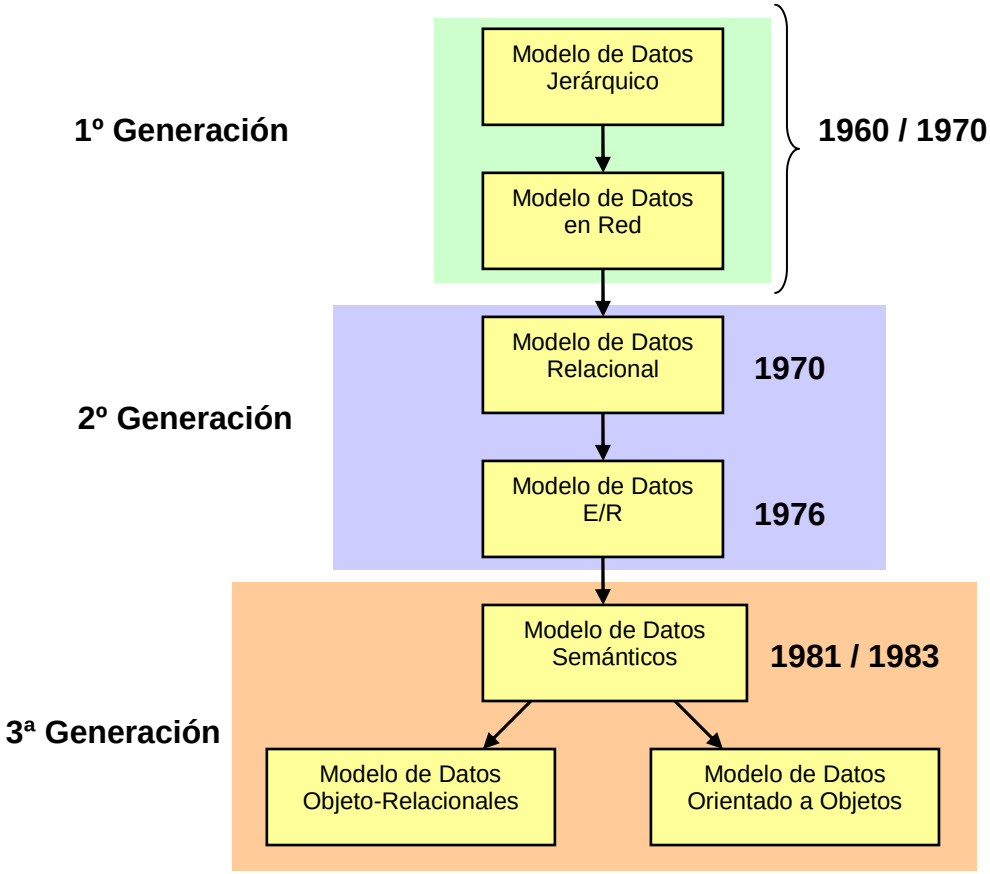


Figura 1: Modelos de bases de datos

2.1.4 Definición S.G.B.D

A continuación se explica lo que es un Sistema Gestor de Base de datos (S.G.B.D).

El propósito general de los sistemas de gestión de bases de datos es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante para una organización.

Se define como un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. De una forma más formal:

"Conjunto coordinado de programas, procedimientos, lenguajes, etc. Que suministra a los distintos tipos de usuarios los medios necesarios para describir y manipular los datos almacenados en la base, garantizando su integridad, confidencialidad y disponibilidad"

(Grupo de bases de datos avanzadas, Universidad Carlos III de Madrid)

"El Sistema de Gestión de Bases de Datos (S.G.B.D) es el conjunto de programas que permiten la implantación, acceso y mantenimiento de la base de datos."

(Fundamentos y modelos de Bases de datos, Ed: Ra-ma)

Todo S.G.B.D debe poseer una serie de características indispensables que se exponen a continuación:

- Lenguaje de definición de datos (LDD): es utilizado para definir la estructura lógica de la BD (nivel lógico), las estructuras externas

requeridas para el desarrollo de las diferentes aplicaciones (nivel externo) así como la estructura interna (nivel físico). Es decir, se utiliza para especificar el esquema de la base de datos. Por tanto permite realizar la función de descripción del SGBD.

- Lenguaje de manipulación de datos (LMD): una vez se ha descrito la BD, ésta ya está preparada para cargar los datos en las estructuras definidas y para su utilización. Así, el LMD permite añadir, suprimir, modificar y buscar datos en la BD. es aquel que usan los usuarios directos para ejecutar sus operaciones sobre la base de datos. Estas operaciones son principalmente inserción, eliminación, modificación y consulta. Este lenguaje permite la función de manipulación del SGBD.

- Lenguaje de control de datos (LCD): es aquel que permite desarrollar su labor al administrador de la base de datos. el administrador de la BD utiliza este lenguaje para especificar los aspectos de seguridad física (copias de seguridad, arranque de la BD en caso de caída, etc.) así como de protección frente a accesos no permitidos (autorizaciones y contraseñas, perfiles de usuarios, etc.). El LCD también se requiere para definir los interfaces que necesitan los distintos usuarios para comunicarse con la BD. Este lenguaje permite realizar la función de utilización del SGBD.

- Permitir almacenar grandes cantidades de información: Las bases de datos surgieron para cubrir esta necesidad de una forma fácil. Así, el S.G.B.D debe ser capaz de almacenar elevadas cantidades de información sin que el usuario perciba que el rendimiento de su ordenador es menor.

- Gestión segura de los datos: debe garantizar la seguridad y privacidad de los datos almacenados, así como su integridad. Es uno de los elementos más importantes a tener en cuenta.

- Permitir el acceso concurrente: es decir, debe permitir acceder simultáneamente a varios usuarios, y controlar que por efecto de lo anterior no se produzcan errores o inconsistencias en los datos.

- Independencia física de los datos: se debe garantizar el objetivo de acceder a los datos, independientemente de en qué lugar estén guardados éstos.

En la siguiente figura se representan un S.G.B.D, los distintos usuarios que pueden trabajar con la base de datos (cada uno con un lenguaje diferente), el acceso concurrente y la independencia física de los datos.

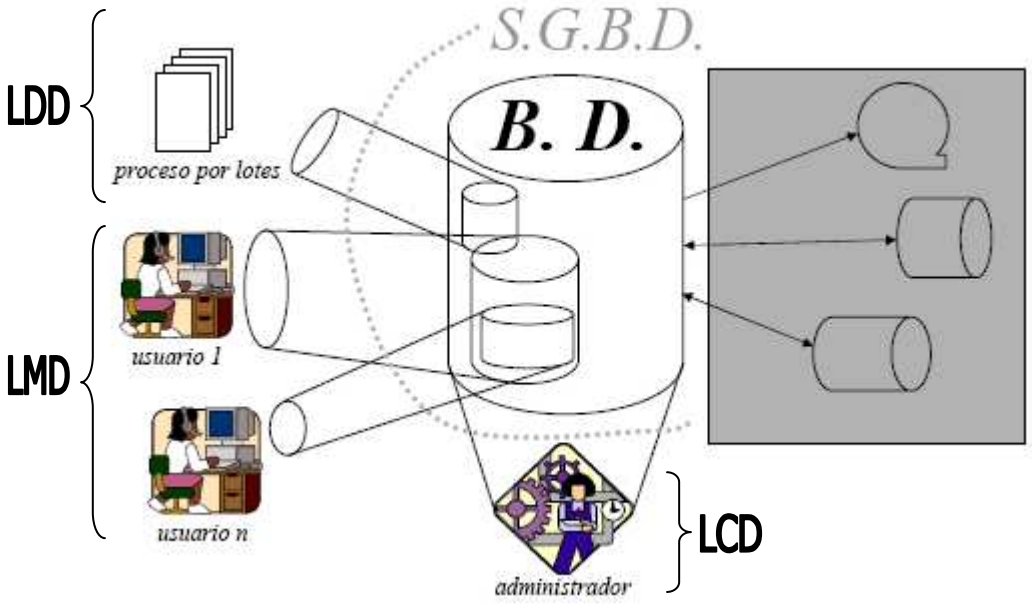


Figura 2: SGBD

Las ventajas de utilizar un S.G.B.D son muchas, entre ellas las que se muestran a continuación.

- Independencia con respecto de los datos: esto es que los datos no sean dependientes de la aplicación, es decir que la información no sea dependiente del S.G.B.D, permitiendo así que se pueda acceder a ésta desde diferentes aplicaciones.

- Acceso eficiente a los datos: es importante que se pueda acceder a los datos siempre y de la mejor manera posible, siendo rápido y útil.

- Integridad y seguridad de los datos: controlar la integridad de los datos es fundamental. Si se accede a través del S.G.B.D a los datos, es importante que éste controle que los datos que se manejan sean correctos. Por ejemplo, si se van a introducir números de teléfono se debe controlar que todos tengan nueve cifras, o si se van a asignar asignaturas que va a cursar un estudiante, comprobar antes de hacerlo, que no sean las que ya tiene aprobadas. Otro de los aspectos fundamentales es la seguridad de los datos. Permite por tanto, controlar el acceso a los mismos, qué se puede ver y quién lo puede ver. No sólo es una ventaja, sino una característica obligatoria de todo S.G.B.D.

- Administración de los datos: los S.G.B.D permiten que múltiples aplicaciones y usuarios tengan acceso a los datos. Por ello es necesario que alguien se encargue de la administración de los mismos, controlando entre otros, la creación de usuarios y roles para el acceso a los datos, organizar la representación de los datos para minimizar la redundancia o mejorar el almacenamiento de los datos para facilitar así su recuperación.

- Acceso concurrente y recuperación: los S.G.B.D hacen que el usuario tenga la sensación de tener acceso a los datos un usuario a la vez. Del mismo modo, protegen a los usuarios de los efectos que pueda tener un fallo en el sistema.

2.2 Bases de datos Orientadas a objetos

En este capitulo se hace una introducción al concepto de orientación a objetos, a las bases de datos orientadas a objetos y a los S.G.B.DOO (Sistemas de Bases de Datos orientadas a Objetos).

2.2.1 ¿Por qué surgen?

Los Sistemas Gestores de Bases de Datos Relacionales (S.G.B.D.R) o los de 2º Generación son los más extendidos. Sin embargo, éstos presentan ciertos problemas:

- Pobre representación de las entidades del mundo real: durante el proceso de normalización, las entidades del mundo real se fragmentan en muchas relaciones, con una representación física que refleja esta estructura. Esto es ineficiente y hace que para realizar una consulta se realicen a su vez muchas combinaciones innecesarias. Se pierde semántica y se ralentiza la recuperación.

- Sobrecarga semántica: el modelo relacional consta de una única estructura para representar tanto datos como relaciones: las tablas. Por ello, no se puede distinguir entre datos y operaciones.

- Soporte inadecuado para las restricciones de integridad: por integridad se entiende la validez y coherencia de los datos almacenados. Se expresa en términos de restricciones, es decir una serie de reglas que la base de datos debe respetar y no permitir que éstas no se cumplan. Muchos sistemas no soportan ciertos tipos de restricciones lo que supone un enorme problema. Además estas restricciones deben ser incluidas en las aplicaciones.

- Estructura de datos homogénea: cada tupla de una relación debe estar compuesta de los mismos atributos. De la misma manera, los valores de una columna concreta deben pertenecer todos al mismo dominio. Además la intersección de una fila y una columna debe tener un único valor. Esto en el mundo real no se da. Es muy restrictivo para ciertos elementos del mundo real que no siguen este comportamiento.

- Dificultades para gestionar las consultas recursivas: es muy difícil gestionar las consultas recursivas, es decir, aquellas sobre relaciones de una tabla consigo misma.

- Operaciones Limitadas: el modelo relacional tiene un conjunto fijo de operaciones y no permite definir operaciones nuevas.

Con todo esto, surge la necesidad de crear nuevos modelos de datos, los modelos de datos de 3ª generación, que diesen solución a los problemas anteriormente expuestos. Para este proyecto se utiliza el modelo de datos orientado a objetos que a continuación se explica.

2.2.2 Orientación a Objetos: características generales

Antes de ver lo que son las bases de datos orientadas a objetos, vamos a definir los conceptos de la orientación a objetos.

La orientación a objetos es un paradigma de programación. A continuación se describen de una manera breve sus características:

Abstracción: se llama así al proceso de identificar los aspectos esenciales de una entidad, separando el diseño de los detalles de implementación.

Encapsulación: es la separación de la implementación y la interfaz de una clase. Con *interfaz* se refiere a las características externas que pueden ver otras clases y la *implementación* es la realización interna de las estructuras de datos y funciones.

Ocultación de la información: proporciona independencia de los datos al ocultar los detalles internos de cada objeto.

Objeto: representación o signo de una entidad concreta. Los objetos tienen las siguientes características:

1. Comportamiento: operaciones que puede realizar un objeto.
2. Estado: valores almacenados en un objeto como atributos que lo definen.
3. Identidad: todo objeto tiene una identidad única.

En un sistema orientado a objetos, cada objeto se caracteriza por un identificador, (OID, Object Identifier). Este identificador hace que cada objeto del sistema sea diferente de todos los demás, aunque la información de ambos sea la misma. Las características del OID son:

- Lo crea automáticamente el sistema.
- Es exclusivo de dicho objeto.
- Es invariable en el tiempo. No se puede modificar. Si un objeto se borra, ningún otro objeto tendrá el OID del objeto borrado (no reutilizable).
- Es independiente del estado en el que se encuentre el objeto.
- Es "invisible" al usuario.

Método: Función asociada a un objeto. Define el comportamiento de un objeto. Pueden cambiar o consultar el estado del objeto.

Mensaje: es el medio por el que los objetos se comunican. Es una solicitud que un objeto envía a otro pidiendo la ejecución de algún método.

Clases: son patrones que sirven para definir conjuntos de objetos similares. Los objetos que tienen los mismos atributos y que responden a los mismos mensajes pueden agruparse para formar una clase. Es decir, una clase es la especificación de un conjunto de objetos con estructura y comportamiento comunes.

Herencia: Algunos objetos pueden tener atributos y métodos similares, pero no iguales. Cuando se dan éstos casos, es útil agrupar las características comunes. Este hecho se permite gracias a la herencia, la cual, permite definir una clase como un caso especial (subclase) de otra más general (superclase).
De esta manera se crea una relación básica entre una superclase y una subclase. La subclase hereda la estructura y comportamiento de la superclase (atributos y métodos) y además puede añadir nuevos atributos y métodos.

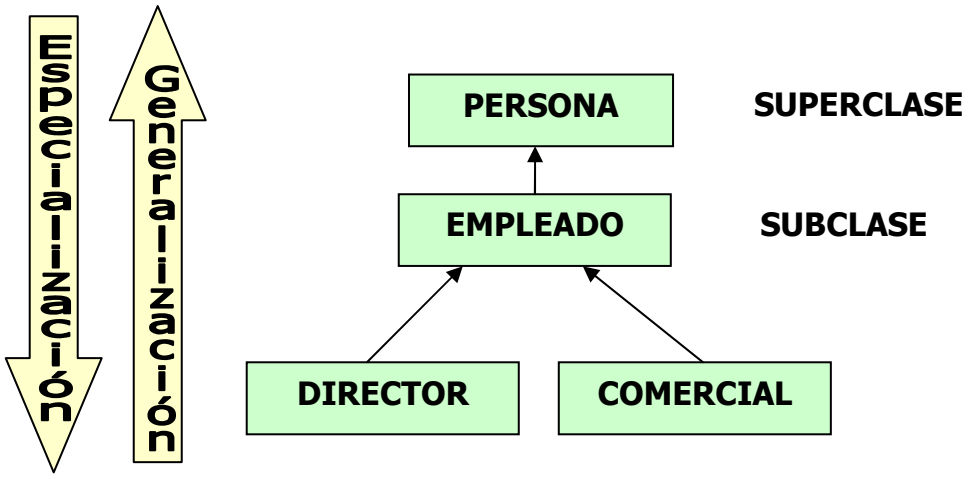


Figura 3: Ejemplo de Herencia

Donde empleado heredaría atributos y métodos de persona. Además empleado podría definirse nuevos atributos y métodos.

Existen dos tipos de herencia:

- Herencia simple: la subclase hereda de una superclase. La figura 3 es un ejemplo de herencia simple.

- Herencia múltiple: la subclase hereda de varias superclases.

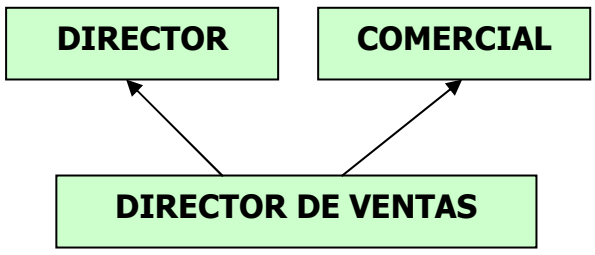


Figura 4: Ejemplo de Herencia múltiple

Anulación: proceso de redefinir propiedades de manera que pasen de ser de las subclases a las superclases y puedan ser utilizadas por las primeras. Una ventaja es la eliminación de redundancias.

Sobrecarga: utilización de un mismo nombre de método dentro de una clase o entre varias definiciones de clase pero con distintos parámetros. El método se selecciona en tiempo de ejecución.

Polimorfismo: capacidad de ejecutar distintos métodos en respuesta al mismo mensaje. Un método polimórfico es aquel que tiene muchas implementaciones. Una subclase puede modificar la implementación de los métodos heredados. El mismo nombre de un método puede designar implementaciones distintas. Hay distintos tipos de polimorfismo:

- Polimorfismo de operación (ad hoc) : sobrecarga
- Polimorfismo de inclusión : método definido en una superclase y heredado por sus subclases
- Polimorfismo paramétrico o de genericidad: utiliza tipos como parámetros dentro de la declaración genérica de tipos de clase.

Enlace dinámico: mecanismo por el que se difiere la selección del método apropiado basándose en el tipo de objeto, hasta el momento de la ejecución (se evitan recompilaciones).

Objetos Complejos: es aquel objeto que está compuesto de subobjetos o componentes. La relación que existe entre el objeto y sus componentes se expresa mediante una jerarquía 'forma parte de'.

2.2.3 Definición Base de Datos Orientada a Objetos

Como se ha explicado en el apartado 2.2.1 dada la complejidad del mundo real, el modelo relacional (el más extendido) resulta inadecuado para ciertas aplicaciones. Por ello se necesitaba definir un nuevo modelo de datos que fuese capaz de soportar aquellas complejidades que el modelo relacional no podía, así nacieron modelos de datos de 3ª generación entre los que está incluido el modelo orientado a objetos.

Se puede definir el modelo de datos orientado a objetos como un modelo de datos (lógico) que captura la semántica de los objetos soportados en la programación orientada a objetos.

De esta forma se puede definir una base de datos orientada a objetos como una base de datos (cuya definición se encuentra en el apartado 2.1.2) cuyo modelo de datos es el modelo de datos orientado a objetos.

2.2.4 Modelo de Objetos (OM):

A continuación se va a exponer las principales características del modelo de datos orientado a objetos. La definición del modelo de datos de una base se encuentra en el apartado 2.1.3.

- Primitivas de modelado: objeto (identificadores únicos) y literal.

- Los objetos y literales pueden clasificarse en tipos. Todos los objetos y literales de un mismo tipo exhiben un estado y comportamiento comunes.

- El estado de un objeto está definido por los valores de sus propiedades (atributos y relaciones). Estas propiedades suelen cambiar en el tiempo.

    - Propiedad: atributo o relación entre objetos, los valores pueden cambiar a lo largo del tiempo. Las relaciones pueden ser uno a uno, uno a muchos, muchos a muchos.

- El comportamiento de un objeto está definido por las operaciones ejecutadas por o en el objeto. Las operaciones pueden tener una lista de parámetros de entrada - salida con un tipo específico y puede devolver un resultado tipado.

- Una DB almacena *objetos* que pueden ser compartidos por múltiples usuarios y aplicaciones. El esquema de la BD se define en lenguaje de definición de objetos (ODL) y contiene instancias de los tipos definidos por su esquema.

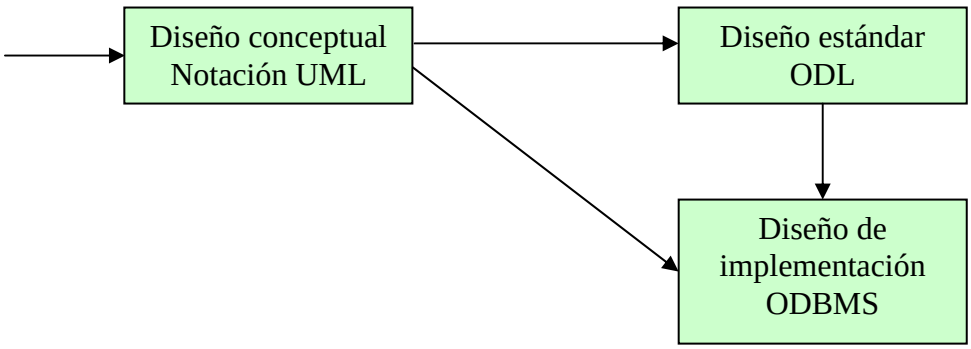


Figura 5: Modelado de datos orientado a objetos

2.2.5 Sistema Gestor de Base de Datos Orientada a objetos

Con toda la información expuesta hasta ahora, ya se puede exponer el concepto de Sistema Gestor de Base de Datos Orientada a Objetos (SGBDOO), en inglés ODBMS, *object database management system*.

De manera más formal: El Sistema de Gestión de Bases de Datos Orientada a Objetos (SGBDOO) es el conjunto de programas que permiten la implantación, acceso y mantenimiento de la base de datos, siendo ésta una base de datos orientada a objetos.

Así el SGBDOO debe satisfacer de igual manera las características de un SGBD y las de un modelo de datos orientado a objetos.

A continuación se enumeran las características reflejadas en el Manifiesto de los SGBDOO (Atkinson et al. 1989):

- Deben soportarse objetos complejos aplicando una serie de constructores a los objetos básicos (SET, TUPLE, LIST/ARRAY).
- Deben soportarse mecanismos de identidad de objetos.
- Debe soportarse la encapsulación: Acceso a la especificación de la interfaz de los métodos.
- Deben soportarse los tipos o clases.
- Los tipos o clases deben ser capaces de heredar de sus ancestros.
- Debe soportarse el enlace dinámico: Anulación y sobrecarga.

- El DML debe ser computacionalmente completo: Lenguaje de programación de propósito general.

- El conjunto de todos los tipos de datos debe ser ampliable

- Debe proporcionarse persistencia a los datos.

- El SGBD debe ser capaz de gestionar BD de gran tamaño: Índices y buffers.

- El SGBD debe soportar usuarios concurrentes

- El SGBD debe ser capaz de recuperarse de fallos hardware y software.

- El SGBD debe proporcionar una forma simple de consultar los datos: Mecanismo de consulta ad hoc que sea de alto nivel.

- Características opcionales: herencia múltiple, comprobación de tipos,…..

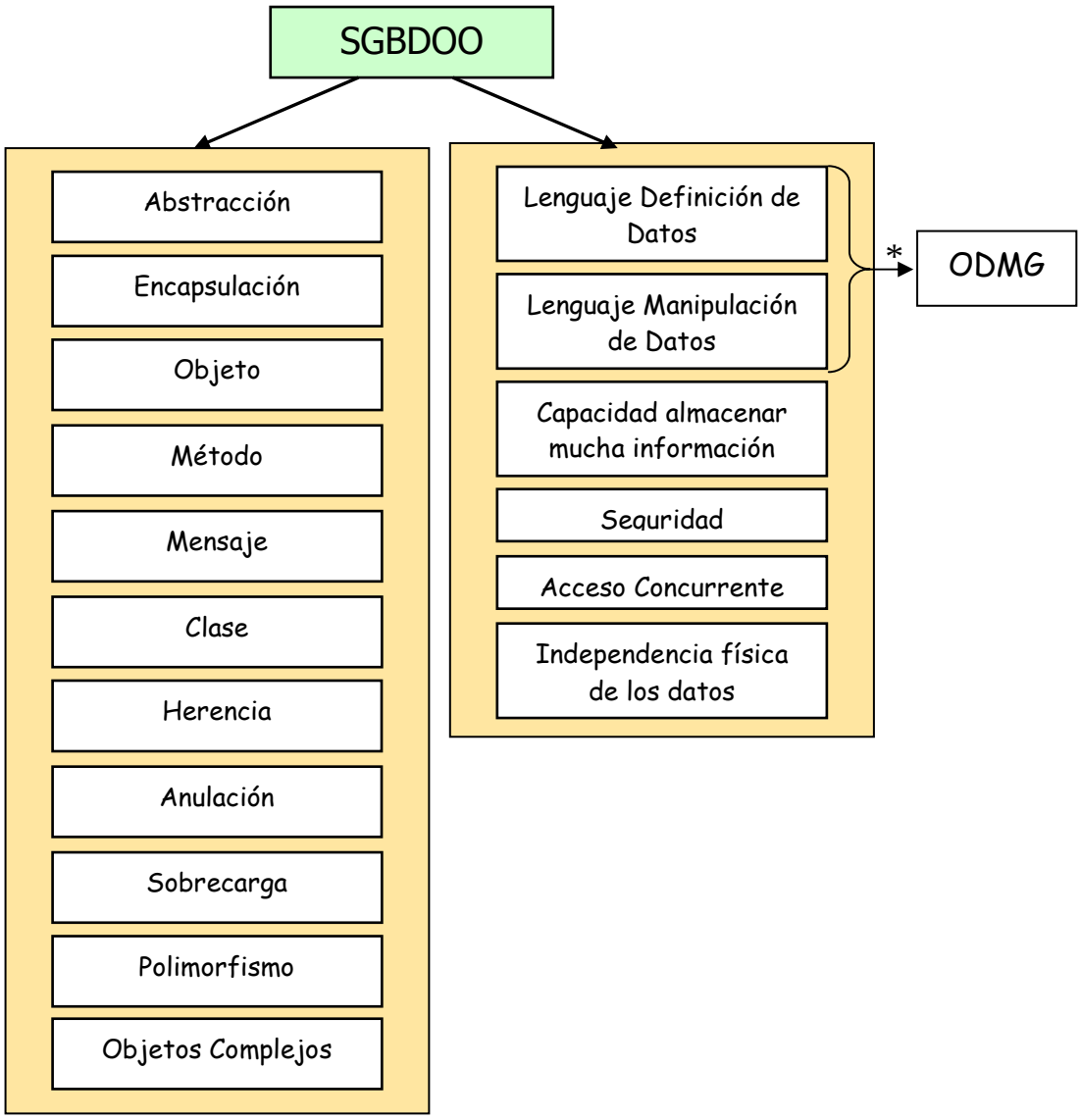


Figura 6: Características SGBDOO

*: Los lenguajes, LDD y LMD, se exponen en el dibujo por ser una característica que tiene que soportar los SGBDOO. Sin embargo en éstos, estos lenguajes ya no reciben este nombre y su estructura es diferente. Existe un nuevo lenguaje para ellos ODMG.

2.2.6 ODMG

Object Database Management Group (ODMG) se trata del grupo de personas y empresas encargadas de desarrollar el modelo de objetos para persistencia, así como para la definición de dicho estándar. Este grupo está formado por importantes fabricantes como Sun Microsystems, POET Software, Computer Asssociates y Versant Corporation entre otros.

Este modelo especifica los elementos que se definirán, y en qué manera se hará, para la consecución de persistencia en las Bases de Datos Orientadas a Objetos que soporten el estándar. Consta de un lenguaje de definición de objetos, ODL, que especifica los elementos de este modelo.

Los principales componentes de la arquitectura ODMG para un S.G.B.D.O.O son:

- El modelo de objetos: anteriormente explicado en el apartado 2.2.4.

- Lenguaje de definición de objetos: es un lenguaje para definir las especificaciones de los objetos en los sistemas que sean compatibles con ODMG. El objetivo es facilitar la portabilidad de los esquemas entre sistemas compatibles con ODMG.

- Lenguaje de consulta de objetos: proporciona un acceso declarativo a la base de datos de objetos utilizando una sintaxis parecida a la de SQL.

- Enlaces con los principales lenguajes orientados a objetos: C++, Java y Smalltalk.

La versión inicial del estándar fue creada en 1993. Durante los años 1997 1999, se fue modificando el estándar añadiendo nuevas primitivas. No será hasta 2001 cuando ODMG publique el estándar definitivo. Después de publicarlo la organización se deshizo.

La herramienta utilizada en este proyecto (Caché), se basa en el estándar ODMG.

2.3 Sistemas de Información Geográfica

2.3.1 Definición Sistemas de Información Geográfica

Dentro de la necesidad de guardar todo tipo de información, surge la necesidad de almacenar datos de tipo geográfico con el fin de resolver problemas complejos de planificación y gestión.

La definición de los SIG (Sistemas de Información Geográfica) no es algo fácil, a pesar de que este término está ampliamente difundido. Desde la aparición de éstos, a mediados de los años sesenta hasta ahora, este término ha tenido multitud de definiciones.

A continuación se muestran algunas de las definiciones que un SIG ha tenido a lo largo de los años:

"*Un conjunto de procedimientos manuales o computerizados usado para almacenar y tratar datos referenciados geográficamente*"
(Aronoff 1989)

"*Una base de datos computerizada que tiene información espacial*"
(Cebrián 1988)

"*Un sistema computerizado para la captura, almacenamiento, recuperación, análisis y presentación de datos espaciales*"
(Clarke 1986)

"*Un sistema de hardware, software, datos y aplicaciones que es usado para registrar digitalmente, editar, modelizar y analizar datos espaciales, y presentarlos en forma alfanumérica y gráfica*"
(NCGIA 1990)

"*Sistema de información diseñado para trabajar con datos georreferenciados mediante coordenadas espaciales o geográficas. En otras palabras, un SIG es a la vez una base de datos con funcionalidades específicas para datos referenciados espacialmente y un conjunto de operaciones para trabajar con los datos*"
(Star y Estes 1990)

De todas estas definiciones se pueden sacar varias ideas:

- Un SIG trabaja con datos espaciales, es decir, con datos que estén geográficamente referenciados.
- Necesita una base de datos en la que almacenar la información anterior.
- Su objetivo es gestionar y analizar información espacial.

Una vez claras todas estas ideas, puede pasar a definirse de una forma general un SIG:

Un Sistema de Información Geográfica (SIG) es una tecnología de manejo de información geográfica formada por hardware, software, datos geográficos y personal, diseñado para capturar, almacenar, manipular, analizar y desplegar en todas sus formas la información geográficamente referenciada con el fin de resolver problemas complejos de planificación y gestión.

2.3.2 Componentes Sistemas de Información Geográfica

Cómo se puede leer en la definición anterior, un SIG consta de varios elementos: Software, Hardware, Datos y Personal. A continuación se pasa a describir de forma breve cada uno de ellos.

Figura 7: Componentes de un SIG

❖ PROGRAMAS (SOFTWARE)

Es el soporte lógico. Proveen las herramientas y funcionalidades necesarias para almacenar, analizar y desplegar la información geográfica. Los componentes principales son:

- Herramientas para la entrada y manipulación de la información geográfica.
- Sistema de base de datos.
- Una interfaz gráfica para el usuario, para el fácil acceso a las herramientas.
- Herramientas para soporte de consultas, análisis y visualización de datos geográficos.

- EQUIPOS (HARDWARE)

Es el soporte físico, la plataforma sobre la que se ejecutan este tipo de aplicaciones. Los SIG son compatibles con un amplio rango de tipos de ordenadores, desde servidores hasta computadores personales usados o no en red.

- DATOS GEOGRÁFICOS

Probablemente sea la parte más importante de un SIG. Constituyen una representación simplificada del mundo real. El mantenimiento y la correcta gestión de los mismos, es de vital importancia, ya que el sistema se basa en ellos para resolver las cuestiones planteadas, y si éstos no son correctos, el SIG tampoco lo será. El principal problema al adquirir un SIG, es que normalmente éste viene sin datos, y debe ser el usuario el que los introduzca.

- PERSONAL

Una pieza clave para el correcto funcionamiento de los SIG, es el personal encargado de manejarlo. Debe ser personal experto en el desarrollo, ya que si no es así, las amplias funciones de un SIG se verían ampliamente limitadas. El personal se encarga de operar, desarrollar y administrar el sistema, además establece planes para aplicarlo en problemas del mundo real.

- PROCEDIMIENTOS

Cada SIG operará de acuerdo con un plan bien diseñado y con unas reglas claras, que son los modelos y las prácticas operativas características de cada uno.

2.3.3 Tipo de información

Un SIG almacena información geográfica: cualquier elemento relativo a la superficie terrestre con una dimensión física y una localización espacial, o una posición medible en el espacio.

Además de información no geográfica (alfanumérica): corresponden a las descripciones, cualificaciones o características que nombran y determinan los objetos o elementos geográficos.

Sin embargo la forma en que almacena los datos geográficos no es única, existiendo dos modelos para la información almacenada: raster y vectorial.

2.3.3.1 Modelo raster

Se puede definir este modelo de una forma sencilla diciendo que se basa en la "digitalización" de un mapa.

Y de un modo más formal: este modelo se basa en una concepción implícita de las relaciones de vecindad que existen entre los objetos geográficos. Su forma de proceder es dividir la zona de afección de la base de datos en una malla regular de pequeñas celdas (píxeles) y atribuir un valor numérico a cada celda como representación de su valor temático. Dado que la malla es regular, el tamaño del píxel es constante y se conoce la posición en coordenadas del centro de una de las celdas, se puede decir que todos los píxeles están georreferenciados.

Para tener una descripción precisa de los objetos geográficos contenidos en la base de datos el tamaño del píxel debe ser reducido en función de la escala, lo que dotará a la malla de una resolución alta; sin embargo, a mayor número de filas y columnas en la malla, mayor esfuerzo en el proceso de

captura de la información y mayor costo computacional al momento de procesarla.

El modelo de datos raster es útil cuando tenemos que describir objetos geográficos con límites no muy claros, como por ejemplo puede ser la dispersión de una nube de contaminantes, o los niveles de contaminación de un acuífero subterráneo, donde los contornos no son absolutamente nítidos; en esos casos, el modelo raster es más apropiado que el vectorial.

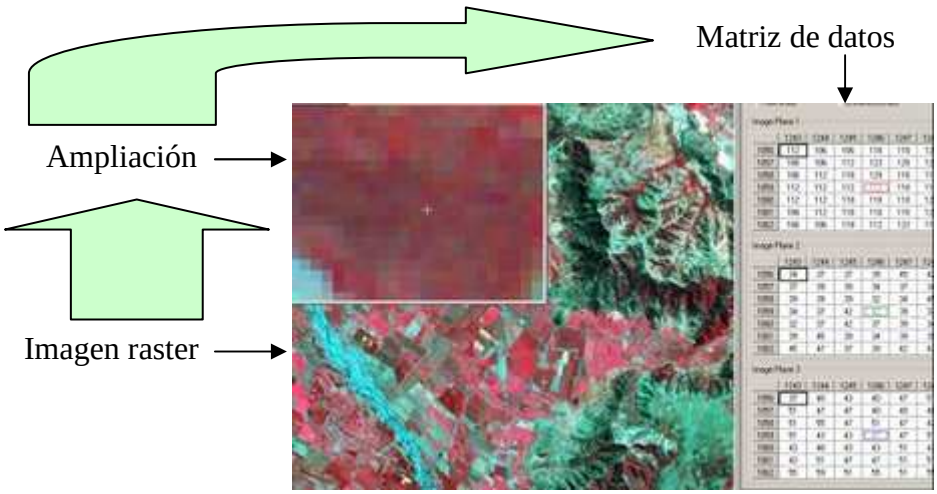


Figura 8: Modelo Raster

2.3.3.2 Modelo vectorial

Son aquellos Sistemas de Información Geográfica que para la descripción de los objetos geográficos utilizan vectores (líneas) definidos por pares ordenados de coordenadas relativas a algún sistema cartográfico.

Con un par de coordenadas se define un punto, con dos puntos se genera una línea, y con una agrupación de líneas se forman polígonos. A estos objetos de dibujo ya se les puede asociar las diversas capas de información que se relacionan con el modelo espacial generado a través de puntos y líneas.

Este tipo de modelo es el recomendado cuando se trabaja con objetos geográficos que tienen muy definidos sus límites, como puede ser carreteras o ciudades.

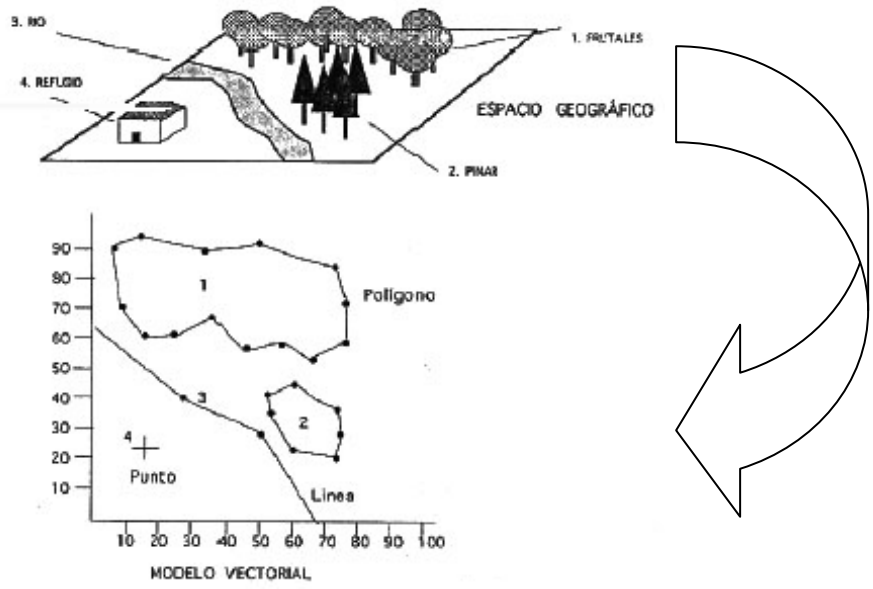


Figura 9: Modelo Vectorial

Así, se utiliza uno u otro en función del objetivo que se desee alcanzar. En este gráfico se ve claramente las diferencias entre los dos modelos:

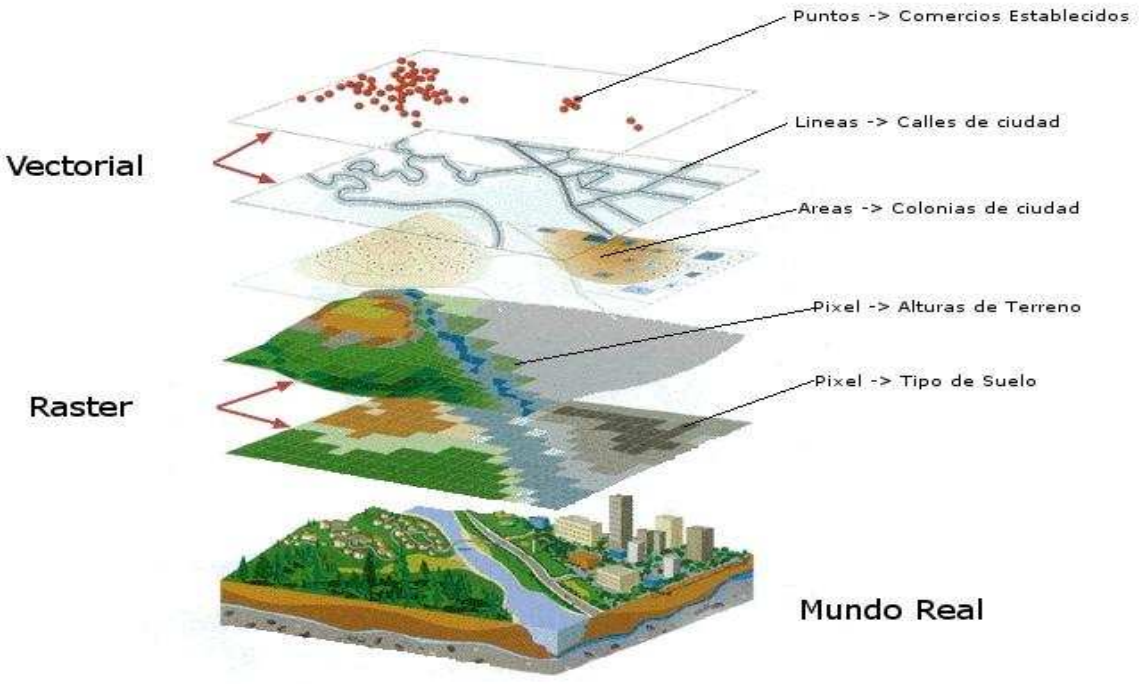


Figura 10: Modelo Raster y Vectorial

2.3.4 Bases de Datos Geográficas

La información geográfica es el elemento diferenciador y esencial de un Sistema de Información Geográfica frente a otro tipo de Sistemas de Información; así, la naturaleza de este tipo de información contiene dos vertientes diferentes: por un lado está la vertiente espacial y por otro la vertiente temática de los datos (alfanumérica). Mientras otros Sistemas de Información contienen sólo datos alfanuméricos (nombres, direcciones, números de cuenta, etc.), las bases de datos de un SIG integran además la delimitación espacial de cada uno de los objetos geográficos.

Por ejemplo, un lago que tiene su correspondiente forma geométrica plasmada en un plano, tiene también otros datos asociados como niveles de contaminación, flora, fauna, pesca y niveles de captación en relación a la temporada del año.

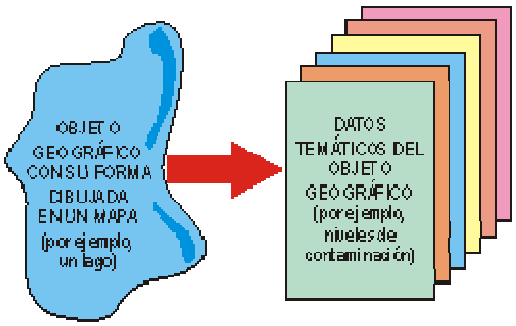


Figura 11: Ejemplo forma geográfica

Por ello, un SIG tiene que trabajar a la vez con dos tipos de información: la topografía definida en plano y sus atributos temáticos asociados. Es decir, tiene que trabajar datos espaciales y datos no espaciales, uniendo ambos y constituyendo con todo ello una sola base de datos geográfica.

La construcción de una base de datos geográfica implica un proceso de abstracción para pasar de la complejidad del mundo real a una representación simplificada que pueda ser procesada por el lenguaje de los ordenadores

actuales. Este proceso de abstracción tiene diversos niveles y normalmente comienza con la concepción de la estructura de la base de datos, generalmente en capas; en esta fase, y dependiendo de la utilidad que se vaya a dar a la información a compilar, se seleccionan las capas temáticas a incluir.

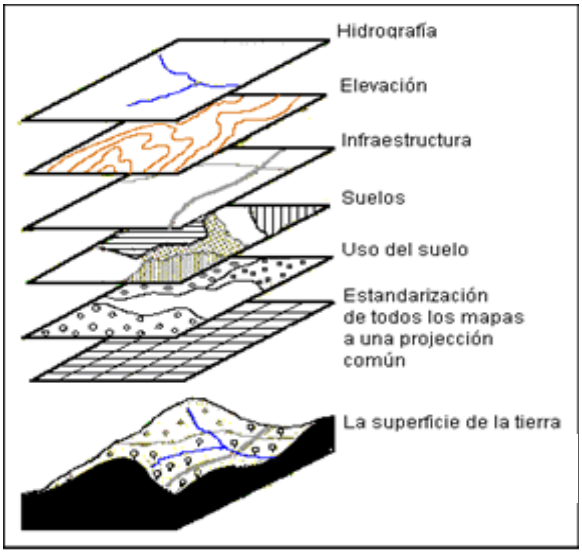


Figura 12: Distintas capas de datos

Estructurar la información espacial en capas del mundo real, trae consigo dos problemas principales. En primer lugar, la necesidad de abstracción que requieren las máquinas implica trabajar con elementos básicos de dibujo, de tal forma que toda la complejidad de la realidad debe ser reducida a puntos, líneas o polígonos.

En segundo lugar, existen relaciones espaciales entre los objetos geográficos que el sistema no puede obviar; la topología, que en realidad es el método matemático-lógico usado para definir las relaciones espaciales entre los objetos geográficos puede llegar a ser muy compleja, ya que son muchos los elementos que interactúan sobre cada aspecto de la realidad.

La topología de un SIG reduce sus funciones a cuestiones mucho más sencillas, como por ejemplo conocer el polígono (o polígonos) a que pertenece una determinada línea, o bien saber qué agrupación de líneas forman una determinada carretera.

2.3.5 Funciones de los SIG

Su funcionalidad general es la gestión de información geográfica. Sin embargo dentro de esta funcionalidad general, se pueden detallar otras de una forma muy general:

- Captura, registro y almacenamiento de datos: el paso de información analógica, en papel, a formato digital de un ordenador.

- Estructuración de datos y manipulación: creación de bases de datos, de nueva cartografía.

- Proceso, análisis y gestión de datos: topología, consultas gráficas, alfanuméricas, combinadas, superposición de planos e información.

- Creación de salidas: impresión de informes, graficación de planos y publicación en diversos formatos electrónicos.

Con la funcionalidad anteriormente detallada, se puede dar respuesta a ciertas preguntas muy comunes en las aplicaciones actuales:

- Localización *¿Qué hay en...?*
- Condición *¿Dónde sucede que...?*
- Tendencias *¿Qué ha cambiado...?*
- Rutas *¿Cuál es el camino óptimo…?*
- Pautas *¿Qué pautas existen...?*
- Modelos *¿Qué ocurriría si...?*

Gracias a la respuesta de estas preguntas, los SIG se han implantado en multitud de sectores para muy diversas tareas, aunque su función principal es la ayuda a la gestión y la toma de decisiones.

2.3.6 Aplicaciones de los Sistemas de Información Geográfica

Como se ha dicho antes los SIG se han implantado en multitud de sectores, a continuación se describen brevemente algunas de sus aplicaciones principales:

APLICACIONES FORESTALES

Esta es una aplicación característica en la que los SIG suponen una gran ayuda para la conservación y explotación del bosque. Indicar qué áreas forestales merecen la máxima preservación y dónde resulta más adecuada en cada momento la tala de árboles o analizar las pautas de difusión de incendios forestales.

ESTUDIOS DE IMPACTO AMBIENTAL

En países muy urbanizados los conflictos entre usos del suelo son muy frecuentes. El SIG puede mostrar cuáles son los usos del suelo en el espacio que va a ser ocupado físicamente por cualquier tipo de infraestructura, indicar si resulta afectado alguna formación vegetal, algún yacimiento arqueológico o contestar a preguntas del tipo cuántas personas van a verse afectadas por el ruido del tráfico futuro.

CATASTRO

En varios países se está realizando la ambiciosa tares de informatizar el catastro con un SIG. El catastro de bienes inmuebles se convierte así en una base de datos computerizada que contiene información territorial del terreno nacional. El catastro contiene información espacial (localización, límites, superficie) y temática (cultivos, calidades, valores) sobre las parcelas. Aunque

su función principal es la de servir de base para la gestión de diversos impuestos y subvenciones.

MANTENIMIENTO Y CONSERVACIÓN DE INFRAESTRUCTURAS

La utilización en este campo tiene como funcionalidad principal crear un inventario sobre redes de carreteras y ferrocarriles basados en la tecnología SIG. Se puede obtener información del tipo cuáles han sido los tramos de carretera que no han sido asfaltados en un tiempo, información sobre accidentes o estado de conservación de las carreteras entre muchos otros.

SISTEMAS DE NAVEGACIÓN PARA AUTOMÓVILES

Uno de los más difundidos. La mayoría de los vehículos son fabricados con sistemas de navegación. Con esta tecnología el conductor de un automóvil dispone de una pantalla en la que aparece un mapa digital de la zona en la que se encuentra, indicando las calles, las direcciones prohibidas o los giros prohibidos. Además de información turística.

PROTECCIÓN CIVIL: RIESGOS, DESASTRES, CATÁSTROFES

Los SIG constituyen una herramienta eficaz para la prevención de riesgos de muy distintos tipos y para la toma de decisiones ante catástrofes. Con la ayuda de los SIG se pueden abordar cuestiones como la determinación de la distribución exacta de los focos y zonas de riesgo, también la identificación de la población potencialmente afectada y la selección de las redes de transporte utilizables para facilitar posibles evacuaciones.

ÁNALISIS DE MERCADOS

El análisis de mercados trata sobre los clientes (reales o potenciales) de las empresas y la satisfacción de sus necesidades mediante la oferta de los

bienes o servicios apropiados. En el marco de la competencia, el análisis de mercados resulta un aspecto clave no sólo para la expansión y el crecimiento de las compañías, sino para garantizar su propia supervivencia. Dado que tanto los clientes como los puntos de oferta tienen una localización en el espacio, la consideración de ese componente espacial en los análisis de mercado resulta fundamental. A esto se le ha venido llamando análisis espacial de mercados, geomarketing o incluso geodemografía.

PLANIFICACIÓN URBANA

Cada vez más los municipios poseen SIG en los que almacenan y gestionan información relativa al planeamiento, propiedad de los bienes inmuebles, los impuestos que recaen sobre éstos, infraestructuras… etc. Los SIG se utilizan en este campo para diversas tareas como la gestión de los impuestos municipales, el control del cumplimiento de la norma urbanística, la localización de nuevos equipamientos o la mejora del transporte entre otros.

CARTOGRAFÍA DIGITAL 3D

Este tipo de información tridimensional de construcciones civiles, es requerida para realizar, por ejemplo, la planeación de la cobertura de las ondas de radio en una población ubicando los rebotes de ondas radiales entre antenas, optimización de redes, ubicación de antenas, interferencias de radio frecuencia, tendido de líneas de transmisión en 3D; o en el caso de la planeación de un aeropuerto este modelado tridimensional permitiría realizar el estudio de los espacios aéreos que intervienen en el proceso de diseño referenciado, en su caso, la viabilidad técnica de su construcción.

3. OBJETIVOS

El propósito principal de este proyecto es realizar, sobre el S.G.B.D Caché, una librería que gestione datos de tipo espacial y una aplicación que utilice dicho sistema. A continuación se muestran de una forma más específica los objetivos del proyecto.

Uno de los objetivos principales es la utilización de la herramienta Caché. Un sistema gestor de base de datos de tipo post-relacional.

Otro objetivo fundamental es la creación de una librería encargada de manejar y gestionar información de tipo espacial. Primero se analizan los posibles tipos de datos espaciales para llevar a cabo la correcta implementación de los mismos. Una vez definidos dichos tipos de datos, se pasa a definir las diferentes operaciones que se pueden realizar con los datos anteriormente descritos. La librería a crear para la gestión de datos espaciales, debe ser lo más general posible, para que pueda reutilizarse en futuras aplicaciones del mismo tipo. Es decir, no se creará una librería exclusiva para esta aplicación, sino que se definirá con carácter general para poder utilizarla en proyectos de la misma índole.

Por último, la elección del tipo de aplicación que usará la librería, anteriormente descrita. El tema de la aplicación será la gestión del almacenamiento en una cajonera de medicamentos. La aplicación muestra las diferentes alternativas de almacenamiento, indica donde se encuentra cada elemento e informa sobre los elementos almacenados, en este proyecto medicamentos.

Los tres puntos anteriores conforman los pilares del proyecto. Una vez claros y definidos los objetivos principales, se deciden los objetivos de la aplicación.

Como anteriormente se expone, la aplicación es relativa a la gestión del espacio de almacenamiento en una farmacia. La aplicación debe mostrar los lugares donde se puede almacenar, qué se tiene almacenado y dónde.

La aplicación contiene información de cada uno de los medicamentos que almacene la base de datos. Aparte de datos como nombre, principio activo o forma farmacéutica de cada medicamento, se debe almacenar información de tipo espacial como su situación física dentro de las cajoneras y sus dimensiones (para saber lo que ocupa). Es decir, unas coordenadas que sitúen al medicamento en la cajonera. Este hecho permite que se puedan consultar los medicamentos almacenados en una determinada zona, como puede ser una puerta o un cajón.

Almacenar información de los medicamentos. Esto permite consultar por nombre comercial, existencias, medicamentos que contengan un determinado principio activo…

Realizar una pequeña herramienta, que haga que la introducción de información en la base de datos sea lo más sencilla e intuitiva posible.

En resumen, se intenta por tanto, crear una base de datos con información espacial para usarla en una aplicación que muestre el almacenamiento de medicamentos en una farmacia.

4. ENTORNO DE TRABAJO

A continuación se explica la herramienta de trabajo utilizada en este proyecto: Caché.

4.1 Caché

4.1.1 Definición

Caché es una base de datos post-relacional que proporciona, como característica fuera de lo común, tres opciones integradas de acceso a los datos que pueden utilizarse de forma simultánea sobre los mismos datos: una robusta base de datos de objetos, SQL de alto rendimiento y un acceso directo al motor multidimensional.

No se precisa mapeo entre las vistas de datos de objetos, relacionales y multidimensionales, lo que da como resultado una gran reducción en el tiempo de desarrollo y proceso. Caché permite el desarrollo rápido de aplicaciones Web, proporciona una extraordinaria velocidad de proceso de transacciones, escalabilidad masiva y consultas en tiempo real sobre datos transaccionales.

4.1.2 Características

A continuación se profundiza en la definición anteriormente dada de caché explicando cada una de sus características.

4.1.2.1 Tecnología post-relacional

Caché esta basada en una tecnología post-relacional. La mayoría de las bases de datos, utilizan una tecnología relacional. Esto quiere decir, que los datos se organizan mediante un conjunto de tablas estructuradas en registros y campos, que se vinculan entre sí por un campo en común. Este tipo de organización es bidimensional. Al hacer transacciones, con este tipo de

organización, es necesario realizar muchas uniones ("joins") de tablas para ir hilando los datos. Esto hace que el proceso se sobrecargue y la aplicación se ralentice.

Caché, al ser de tipo post-relacional, no obliga a que los datos se almacenen como el modelo anteriormente descrito, sino que lo hace en estructuras multidimensionales (arrays multidimensionales denominados "globals"). Además de permitir un modelado de datos realista, las arrays multidimensionales ofrecen un acceso mucho más rápido porque eliminan la sobrecarga asociada a los "saltos entre tablas" y los "joins" de la tecnología relacional.

Estas estructuras son las mejores para describir los datos del mundo real, además de ofrecer un acceso mucho más rápido a los datos que en el modelo relacional. Dichas estructuras son arrays de datos. Por ello la ventaja de este nuevo tipo de almacenamiento es el rendimiento, ya que se puede acceder a los datos rápidamente.

Aunque los datos se almacenan en formato multidimensional, Caché ofrece a los desarrolladores la libertad de modelar sus datos de la forma que elijan: como objetos, como tablas o como arrays multidimensionales (denominados "globals" en Caché).

En el siguiente ejemplo en cada nodo del array se almacenan instancias de Empleado:

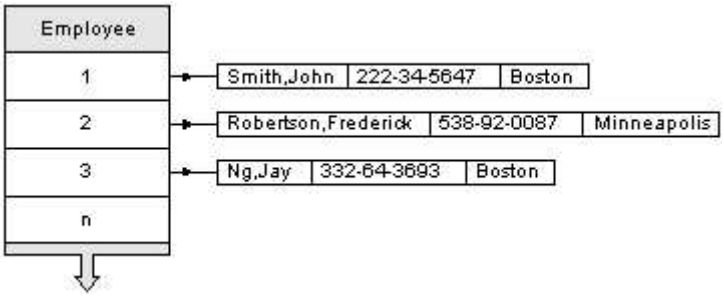


Figura 13: Ejemplo de almacenamiento multidimensional

Caché ObjectScript soporta todos los métodos de acceso a datos: objetos, SQL, arrays multidimensionales e incluso HTML embebido. Esta es otra característica de la tecnología post-relacional. Estas formas de acceso no son excluyentes y se pueden utilizar de forma concurrente.

Figura 14: Accesos a caché

Con todo esto, vemos que lo que define la tecnología post-relacionalson las múltiples formas de acceso a los datos y la estructura de datos multidimensional.

4.1.2.2 Tecnología basada en objetos

Para las nuevas aplicaciones de base de datos, la mayoría de los desarrolladores prefiere utilizar tecnología basada en objetos porque pueden desarrollar aplicaciones complejas de una forma más rápida y modificarlas posteriormente con facilidad. La tecnología de objetos permite muchas ventajas:

- Modelo de datos intuitivo La tecnología de objetos permite a los desarrolladores pensar en la información y utilizarla (incluso información tremendamente compleja) de forma sencilla y cercana a la realidad, acelerando el proceso de desarrollo de la aplicación.

- Desarrollo rápido de aplicaciones Los conceptos de encapsulación, herencia y polimorfismo de objetos permiten reutilizar, volver a planificar y compartir clases entre aplicaciones, y los programadores pueden aprovechar su trabajo para otros muchos proyectos. Programar es más sencillo; es más fácil hacer el seguimiento de lo que se está haciendo y de lo que se está modificando.

- Las versiones personalizadas de las clases pueden sustituir fácilmente a las estándares, facilitando la personalización de una aplicación.

- El enfoque de "caja negra" en la encapsulación significa que el programador puede mejorar el funcionamiento interno de los objetos sin que afecte al resto de la aplicación.

- Los objetos proporcionan un método sencillo de conectar distintas tecnologías y distintas aplicaciones.

- Java y las interfaces de usuario basadas en GUI utilizan de forma natural la tecnología de objetos.

- Muchas de las nuevas herramientas incorporan tecnología de objetos.

- Los objetos proporcionan un buen aislamiento entre la interfaz de usuario y el resto de la aplicación. De este modo, cuando es necesario adoptar una nueva tecnología de interfaz de usuario se puede reutilizar la mayor parte del código.

Aunque muchas aplicaciones se escriben ahora con lenguajes de programación de objetos, se intenta implementar a la fuerza los datos de los objetos en tablas relacionales planas. Esto afecta seriamente a las ventajas de la tecnología de objetos.

Caché proporciona una estructura de datos multidimensional que almacena de forma natural datos complejos de objetos. El resultado: más rapidez en el acceso a los datos y en la programación.

Por supuesto, muchas herramientas (como los generadores de informes) utilizan SQL, no tecnología de objetos, para acceder a los datos. Una característica única de Caché es que siempre que se define una clase de objetos de base de datos, proporciona automáticamente acceso SQL completo a esos datos. Por tanto, sin trabajo adicional, las herramientas basadas en SQL funcionarán inmediatamente con los datos de Caché, manteniendo el alto rendimiento del Servidor de Datos Multidimensional de Caché.

El proceso inverso también es válido. Cuando se importa una definición DDL de una base de datos relacional, Caché genera automáticamente una descripción de objetos de los datos, habilitando su acceso inmediato como objetos y a través de SQL. La Arquitectura de Datos Unificada de Caché (Unified Dictionary) mantiene sincronizadas estas vías de acceso; sólo hay una descripción de datos que editar.

4.1.2.3 Servidor de datos multidimensional de Caché

La base de datos de alto rendimiento de Caché utiliza un motor de datos multidimensional que permite el almacenamiento eficaz y compacto de datos en una estructura de datos compleja. Los objetos y SQL se implementan especificando un diccionario de datos unificado que define las clases y tablas y

proporciona un mapeo con las estructuras multidimensionales, mapeo que puede generarse automáticamente.

Caché proporciona a los programadores la libertad de almacenar y acceder a los datos mediante acceso a objetos, SQL o acceso directo a las estructuras multidimensionales. Independientemente del método de acceso, todos los datos de la base de datos de Caché se almacenan en las arrays multidimensionales. Una vez almacenados los datos, se pueden utilizar simultáneamente los tres métodos de acceso sobre los mismos datos, con concurrencia total.

Una característica única de Caché es su Arquitectura de Datos Unificada. Siempre que se define una clase de objetos en la base de datos, Caché genera automáticamente una descripción relacional SQL de dicha clase. De forma similar, cuando se importa en el Diccionario de Datos una descripción DDL de una base de datos relacional, Caché crea automáticamente junto con la descripción relacional, la definición de la clase equivalente, lo que posibilita el acceso inmediato como objetos. Caché mantiene estas descripciones unidas: en la práctica existe una única definición de datos.

Caché crea automáticamente un mapeo de cómo se almacenan los objetos y tablas en las estructuras multidimensionales, aunque también se puede controlar el mapeo explícitamente.

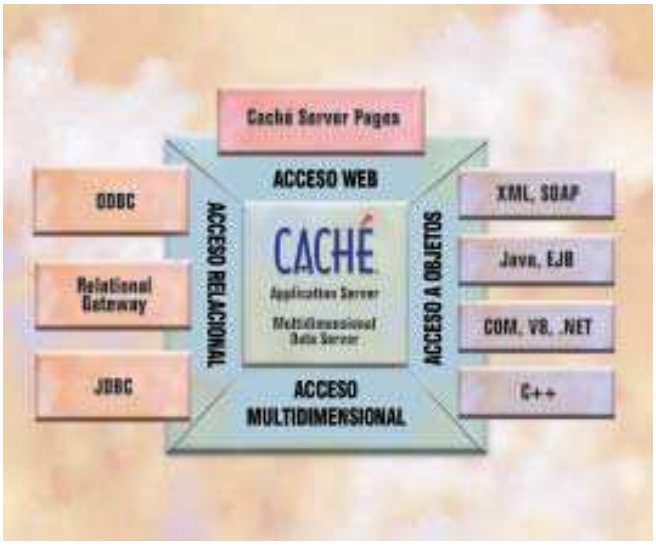


Figura 15:

Acceso multidimensional

4.1.2.4 Servidor de aplicaciones de Caché

El Servidor de Aplicaciones de Caché ofrece posibilidades avanzadas de programación de objetos, un almacenamiento sofisticado de datos en memoria intermedia cache y fácil acceso a diversas tecnologías. El Servidor de Aplicaciones de Caché permite desarrollar rápidamente aplicaciones de base de datos sofisticadas, obtener alto rendimiento al operar con ellas y darles soporte con facilidad.

El Servidor de Aplicaciones de Caché proporciona:

- La Máquina Virtual de Caché que ejecuta dos lenguajes de script integrados: Caché ObjectScript y Basic.

- Acceso a los Servidores de Datos Multidimensionales de Caché en el mismo ordenador, o en otros, de forma transparente.

- Software de conectividad con almacenamiento en memoria intermedia cache en la parte cliente, para permitir el acceso rápido a Caché Objects desde todas las tecnologías utilizadas habitualmente, incluyendo Java, C++, C#, COM, .NET, Visual Basic y Delphi. Caché se encarga automáticamente de los procesos en red entre el cliente y el Servidor de Aplicaciones.

- Compatibilidad con SOAP y XML.

- Acceso SQL mediante ODBC y JDBC, incluyendo almacenamiento sofisticado en memoria intermedia cache en el cliente y el servidor de aplicaciones para conseguir mayor rendimiento.

- Acceso a bases de datos relacionales.

- Caché Server Pages para crear aplicaciones Web de alto rendimiento y fáciles de programar.

- Caché Studio, una herramienta para desarrollar y depurar rápidamente aplicaciones con Caché.

- El código de los lenguajes de script se almacena en la base de datos y puede modificarse en línea, propagándose los cambios automáticamente a todos los servidores de aplicaciones.

El núcleo del Servidor de Aplicaciones de Caché es la Máquina Virtual de Caché, que soporta los dos lenguajes de script de Caché: Caché ObjectScript y Basic.

- Caché ObjectScript es un potente lenguaje orientado a objetos, fácil de aprender, que incorpora estructuras de datos muy flexibles.

- Basic es similar a Visual Basic y a VBScript, Basic se ha ampliado para disponer de acceso directo a las arrays multidimensionales de Caché.

- Caché MVBasic es la variante de los dos lenguajes de programación Basic utilizada en aplicaciones MultiValue.

La interoperatividad entre los lenguajes es total al estar implementados en la misma Máquina Virtual de Caché. Esto hace que:

- Los métodos de objeto pueden escribirse en cualquier lenguaje, la misma clase puede utilizar ambos lenguajes.

- Cada lenguaje puede realizar llamadas a código escrito en el otro lenguaje.

- Comparten variables, arrays y objetos.

El código que se ejecuta en la Máquina Virtual de Caché se puede ejecutar en otro hardware y sistema operativo sin cambios. El código se almacena en la base de datos y se propaga automáticamente a los Servidores de Aplicaciones.

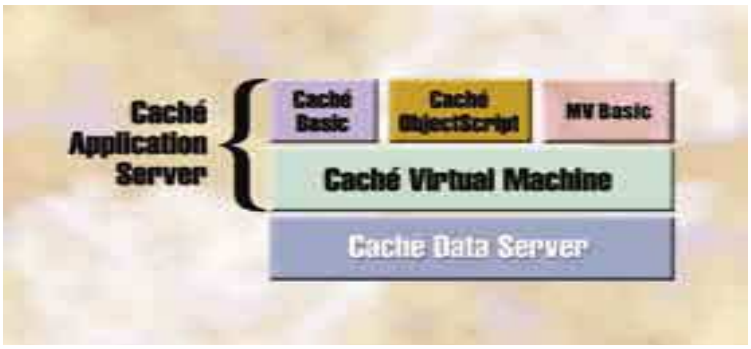


Figura 16: Servidor de Aplicaciones Caché

El acceso a la base de datos desde la Máquina Virtual de Caché está muy optimizado. Cada proceso de usuario en la Máquina Virtual de Caché tiene acceso directo a las estructuras de datos multidimensionales mediante la realización de llamadas a la memoria compartida que accede a la memoria cache compartida de la base de datos. Las otras tecnologías (Java, C++, ODBC, JDBC, etc.) se conectan a través de la Máquina Virtual de Caché para acceder a la base de datos.

4.1.2.5 Lenguajes soportados por Caché

Dado que Caché tiene acceso multidimensional a los datos, el software de conectividad permite el acceso a Caché Objects desde todas las tecnologías utilizadas habitualmente, como se ha visto anteriormente (figura 15). A continuación se describen las mismas:

- **CACHÉ OBJECTSCRIPT**

Caché ObjectScript es un potente lenguaje de programación orientado a objetos diseñado para el desarrollo rápido de aplicaciones de base de datos. Es el utilizado en el proyecto, por lo tanto se describirá más adelante.

- **BASIC**

Quizás Basic sea el lenguaje de programación de aplicaciones más conocido del mundo. En Caché, Basic se ha ampliado para dar soporte al acceso directo de las estructuras de datos del núcleo del Servidor de Aplicaciones (arrays multidimensionales) y a otras características del Servidor de Aplicaciones de Caché. Soporta directamente el modelo de objetos de Caché utilizando sintaxis Visual Basic y se ejecuta en la Máquina Virtual de Caché.

Basic se puede utilizar tanto en métodos de clases como en rutinas de Caché (consulte la descripción de rutinas de Caché ObjectScript). Basic puede llamar a Caché ObjectScript y viceversa, accediendo ambos lenguajes a las mismas variables, arrays y objetos en la memoria de proceso.

- **HTML Y SQL**

Caché es totalmente compatible con HTML. Para aplicaciones Web HTML y SQL pueden estar embebidos en el código de Caché ObjectScript. Además

Caché cuenta con Caché Server Page, CSP, una herramienta que permite la creación de páginas Web para un acceso fácil a los datos.

- **MVBAsic**

MVBasic es otro lenguaje de script incluido en Caché, y se trata de una variante de Basic. Sin embargo, está pensado para ejecutar aplicaciones escritas para sistemas MultiValue (Pick) y, por tanto, soporta características adicionales, que incluyen la posibilidad de acceder y manipular ficheros MultiValue. MVBasic se puede utilizar tanto en métodos de clases como en rutinas de Caché. MVBasic puede llamar a Caché ObjectScript o a Basic, y viceversa, accediendo los tres lenguajes a las mismas variables, arrays y objetos de la memoria de proceso. Caché MVBasic tiene las mismas extensiones que Caché Basic, incluyendo el acceso a objetos.

- **C++**

Todas las clases de Caché pueden proyectarse como una clase C++, con métodos que se corresponden con cada propiedad y método de la clase Caché. Para los programas C++ estas clases son iguales que cualquier otra clase local C++.

- **JAVA**

Java es una tecnología de programación muy conocida, pero la conexión de las aplicaciones de Java con una base de datos puede ser todo un reto. La conexión de una base de datos relacional requiere mucho código SQL, un proceso lento que reduce muchas de las ventajas de la tecnología de objetos de Java. Normalmente es preferible la sintaxis de objetos directa para acceder a la base de datos. Caché soporta todos estos enfoques:

- Todas las clases de Caché pueden proyectarse como una clase Java y puede accederse a las propiedades y métodos como objetos de Java.

- Las clases de Caché también pueden proyectarse como Enterprise Java Beans (EJB).

- JDBC proporciona acceso SQL de alto rendimiento utilizando un controlador basado completamente en Java

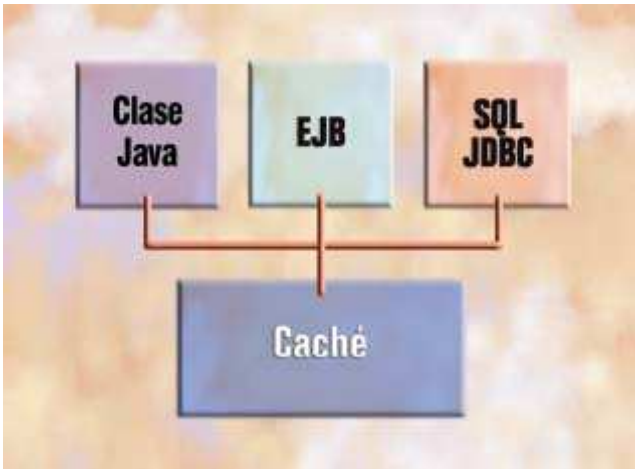


Figura 17: Conexiones de Java

❖ **JALAPEÑO**

Jalapeño (JAva LAnguage PErsistence with NO mapping) es una tecnología de que facilita a los desarrolladores de Java el almacenamiento de sus objetos en la base de datos de objetos Caché, utilizando el acceso a objetos y proporcionando a la vez acceso SQL de alto rendimiento a los mismos datos. Los objetos se almacenan en la base de datos como objetos auténticos con propiedades, relaciones, etc. (es decir, no se almacenan simplemente por su estado serie), por tanto, no se requiere mapeo relacional de objetos.

❖ **.NET**

Caché funciona perfectamente con .NET debido a su acceso a los datos abierto y flexible. Existen varias formas de conectarlos, incluidos objetos, SQL, XML y SOAP. Los desarrolladores pueden crear aplicaciones con las tecnologías que prefieran, beneficiándose todas ellas del rendimiento y escalabilidad superiores de Caché.

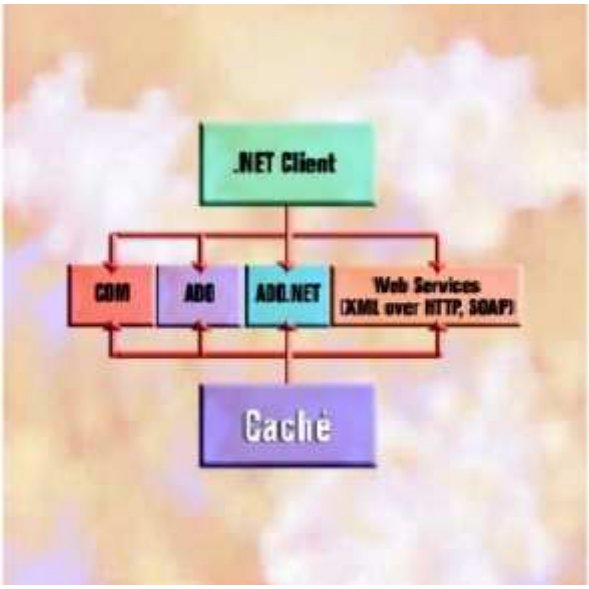


Figura 18: Conexiones .NET - Caché

❖ **XML**

Al igual que HTML es un lenguaje de marcas compatible con Internet para mostrar datos, XML es un lenguaje de marcas para intercambiar datos entre aplicaciones. Mediante XML, distintas aplicaciones (dentro de una misma empresa o en diferentes) pueden compartir datos a través de una red. La estructura de los datos XML es jerárquica y multidimensional, convirtiéndose en una unión natural con el motor de datos multidimensional de Caché. Caché proporciona una interfaz bidireccional fácil de utilizar con XML que evita a los desarrolladores la necesidad de crear manualmente una "capa de mapeo" de proceso entre los datos XML y la base de datos.

4.1.3 Caché ObjectScript

Caché ObjectScript es un potente lenguaje de programación orientado a objetos diseñado para el desarrollo rápido de aplicaciones de base de datos. Este lenguaje es utilizado por la máquina virtual de Caché. Estas son algunas de las características clave de este lenguaje:

- Soporta métodos y objetos, incluyendo características típicas de éstos como son la herencia y el polimorfismo.
- Soporta de una manera eficiente código SQL. Igualmente ocurre con HTML.
- Gran número de funciones predefinidas que facilitan al usuario el trabajo.
- Comandos que permiten manejar dispositivos de Entrada y Salida (I/O).
- Soporte para la depuración y evaluación de los comandos en tiempo de ejecución y evaluación.
- Lenguaje flexible y dinámico.
- Los datos se pueden almacenar en :
    - propiedades de objetos.
    - variables.
    - arrays multidimensionales
    - archivos de base de datos (“globals”)

4.1.3.1 Estructura general del lenguaje

Caché ObjectScript está orientado a los comandos; por tanto su sintaxis es similar a:

```
set x=a+b
do rotate(a,3)
if (x>3)
```

Existe un conjunto de funciones de sistema integradas que son especialmente potentes para el trabajo con texto. Sus nombres empiezan con el carácter '$' para distinguirlas de los nombres de variables y de arrays. Por ejemplo:

```
$extract(cadena,desde,hasta) // extrae un conjunto de caracteres de una cadena
$length(cadena) // determina la longitud de una cadena
```

Las expresiones utilizan la precedencia de operadores de izquierda a derecha, al igual que la mayoría de las calculadoras de mano, excepto en los casos en que los paréntesis cambian el orden de evaluación.

Los operadores, al igual en que otros tipos de lenguajes se clasifican en:

- Operadores de Comparación:
  - Igual → =
  - Distinto → '=
  - Mayor → >
  - Menor → <
  - Mayor o igual → >=
  - Menor o igual → <=

- Operadores Lógicos:

  - AND → &&
  - OR → ||
  - NOT → ‘

4.1.3.2 Objetos

En Caché todos los objetos tiene un identificador que los distingue del resto. Este identificador es único y distinto para cada objeto. Si un objeto se elimina el identificador de ese objeto no volverá a tenerlo ningún otro. Se utiliza para recuperar y almacenar objetos en la base de datos.

Para acceder a un objeto es necesario hacerlo a través de una referencia “oref”. Es decir, es necesario recuperar el objeto de la base de datos y asignárselo a una variable, un *oref.* Una vez asignado se podrá manipular el objeto.

Ejemplos:

Crear un objeto nuevo:

Set nuevaprenda = ##class(Prenda.tipo).%New()

*(1) *(2)

*(1) nuevaprenda es el oref del objeto

*(2) Especifica la clase y el paquete al que va a pertenecer el Nuevo objeto.

Se introducen los valores del nuevo objeto:

Set nuevaprenda.nombre = "CamisaRocio"

Set nuevaprenda.color = 57

Set nuevaprenda.talla = 40

Se guarda un objeto:

Do nuevaprenda.%Save()

Para recuperar objetos es necesario el id (como anteriormente se explica):

Set Pantalon = ##class(Prenda.tipo).%OpenId(2)

Se asigna el oref Pantalon la prenda abierta, cuyo id es 2. Así mediante este oref se puede acceder al objeto recientemente abierto.

Para llamar a métodos es necesario hacerlo desde el objeto que implementa dicho método:

Do Pantalon.mostrarPatrones()

Donde mostrarPatrones () es un método de la clase ciudad, que muestra todos los patrones de una prenda.

4.1.4 Caché Server Page (CSP)

CSP es una tecnología que se proporciona como parte del Servidor de Aplicaciones de Caché. Es el medio más rápido para que las aplicaciones de Caché interactúen con la Web y proporciona:

- Una solución avanzada de desarrollo orientado a objetos
- Rendimiento y escalabilidad muy altos en tiempo de ejecución.

CSP soporta HTML, XML y otros lenguajes orientados a la Web. CSP no es una herramienta de diseño Web, aunque se puede utilizar con ellas. Mientras las herramientas de diseño Web generalmente se concentran sólo en la producción de HTML estático, CSP va más allá del aspecto de las páginas para facilitar el desarrollo de la lógica de la aplicación. También proporciona la configuración que permite la ejecución rápida del código en el Servidor de Aplicaciones de Caché.

CSP da la capacidad de desarrollar rápidamente aplicaciones Web muy sofisticadas.

Estas son algunas de las características de Caché Server Pages:

- Páginas de servidor dinámicas: Como las páginas se crean dinámicamente en el servidor de aplicaciones mediante el código de la aplicación, en lugar de tener un servidor Web que simplemente devuelve HTML estático, las aplicaciones responden rápidamente a distintos tipos de solicitudes y adaptan las páginas resultantes que se envían de vuelta al navegador.
- Modelo de sesión: Todo el proceso relacionado con las páginas de un solo navegador se considera parte de una sesión, desde la

primera solicitud del navegador hasta que la aplicación termine o exceda el tiempo de respuesta (timeout).

- Conservación del estado del servidor: En una sesión, los datos de la aplicación que están en el servidor (e incluso el contexto completo de la aplicación) se pueden conservar automáticamente en las solicitudes del navegador, facilitando el desarrollo y la ejecución de aplicaciones complejas.

- Arquitectura de objetos: Como cada página corresponde a una clase, el código y otras características comunes a muchas páginas se pueden incorporar fácilmente mediante herencia. Normalmente también se hace referencia a los datos mediante objetos con todas las ventajas de la programación orientada a objetos.

- XML: Caché soporta totalmente XML, como alternativa potente a HTML para crear páginas Web y como formato universal para mover datos entre aplicaciones y sistemas.

- Caché Application Tags para generación automática de código en el servidor de aplicaciones: Estas etiquetas HTML extendidas son fáciles de utilizar como etiquetas HTML tradicionales. Cuando se añaden a un documento HTML, generan código de aplicación sofisticado que proporciona amplia funcionalidad, como abrir objetos, ejecutar consultas y controlar el flujo del programa. Estas etiquetas son extensibles: cada uno puede crear las suyas propias para adaptarlas a sus necesidades específicas.

- Integración con herramientas conocidas de diseño Web: CSP funciona con varias herramientas que facilitan la organización visual de una página como Dreamweaver.

- Métodos del servidor que se pueden llamar desde el navegador: Para facilitar el desarrollo de aplicaciones interactivas más dinámicas, CSP simplifica la llamada a métodos del servidor. Cuando se produce un evento en el navegador, generalmente porque el usuario lleva a cabo una acción, se puede llamar al código de la aplicación en el servidor y generar una respuesta al evento, todo ello sin la carga que supone transmitir y cargar una nueva página entera.

- Cifrado: Caché cifra automáticamente a todos los datos del URL, para ayudar a autentificar las solicitudes y evitar la manipulación. La clave de cifrado se conserva únicamente en el servidor, y sólo es válida para el transcurso de una sesión.

El funcionamiento de una página CSP es el siguiente:

- El navegador hace una petición a una página CSP.

- El servidor Web, al detectar la extensión CSP, pasa la petición directamente al servidor de Caché.

- Caché ejecuta la petición y genera una página HTML que se devuelve al servidor Web.

- EL servidor Web envía la página al solicitante.

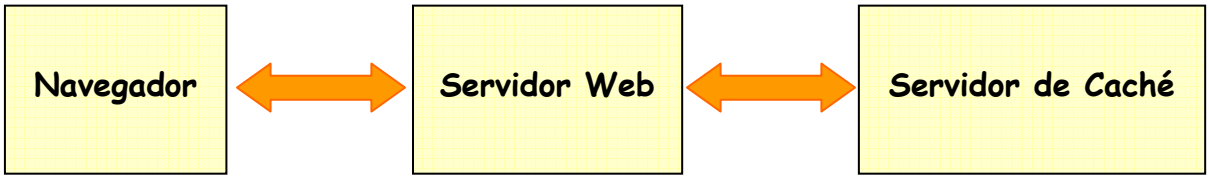


Figura 19: Caché Server Page

CSP sigue una arquitectura de objetos, esto es que cada página corresponde a una clase. Esto permite que características comunes se puedan incorporar de una forma fácil mediante la herencia. También permiten insertar objetos dentro de las páginas.

CSP incorpora "Caché Application Tags", etiquetas parecidas a las de HTML, que incorporan una variada funcionalidad a la página. Estas etiquetas permiten entre otras: acceder a objetos de la base de datos, ejecutar consultas, controlar el flujo del programa o ejecutar código en el servidor de aplicaciones de Caché. Este tipo de tags no están embebidas dentro del código HTML que se envía a un navegador, sólo están en el archivo CSP que lee el compilador Web de Caché. Es éste último es que las transforma automáticamente en código HTML estándar, para que así pueda procesarlo cualquier navegador.

CSP consta de un asistente de formularios que permite crear formularios sencillos de una forma rápida y fácil.

4.1.4.1 %Request

CSP proporciona diferentes objetos, uno de ellos es %Request. El objeto %Request permite el acceso a toda la información que pasa desde el navegador del cliente al servidor. Esto significa que contiene todos los pares nombre/valor y datos del formulario. Por ello es de gran utilidad para las aplicaciones Web y en concreto para la que se ha desarrollado en éste proyecto (ya que se basa en consultas). Así, se almacenan los valores introducidos por el usuario, en los distintos campos de la aplicación, para realizar las consultas y es muy fácil acceder a dichos valores.

4.1.5 Dynamic SQL

Otra de las herramientas que proporciona Caché y que es Dynamic sql. Esta herramienta permite preparar y ejecutar consultas en tiempo de ejecución. Permite programar consultas dentro del lenguaje de programación de Caché, Caché ObjectScript y Basic. Los valores de salida de la consulta se recuperan en un objeto de tipo %Library.ResultSet.

En el siguiente ejemplo se muestra su estructura:

```
Set rset = ##class(%ResultSet).%New()
set l = "SELECT * FROM Almacen.Med where Med.codnac=? "
Do rset.Prepare(l)
Do rset.Execute(codigonac)
```

Se crea un nuevo objeto de tipo ResultSet. Se prepara la consulta, donde el valor de entrada se identifica con el carácter '?'. Por último se ejecuta, introduciendo el código nacional del medicamento a consultar (en este caso por variable). Los valores de salida se almacenan en la variable rset, donde podrán ser tratados.

4.1.6 Caché Studio

Caché Studio es la aplicación de Caché que permite utilizar todo lo explicado anteriormente; crear, depurar y probar aplicaciones caché de forma rápida. Es una aplicación visual que permite crear bases de datos orientadas a objetos y aplicaciones Web. Morfológicamente es como una aplicación estándar de Windows. Utiliza diferentes ventanas para mostrar y editar diferentes aspectos.

A continuación se explica cada una de las partes de las que está compuesto:

- Editor de clases: Editor para la definición de clases. En esta misma ventana se editan otros tipos de archivos como CSP o archivos JavaScript.

- Ventana de proyecto: Muestra el contenido del actual proyecto. Tiene diferentes formas de verlo:

  - Los archivos de un proyecto. Los divide en clases, rutinas, archivos CSP y otros (donde se almacenan otro tipo de archivos como por ejemplo los JavaScript).

  - Ventanas abiertas. Muestra todas las ventanas abiertas que tiene Studio.

  - Namespaces. Muestra las clases, rutinas, archivos CSP y otros, de un namespace.

- Inspector de clases: muestra y permite la modificación de las propiedades y métodos de la definición de clases.

- Ventana de salida: muestra la salida del servidor de Caché, como los mensajes generados durante la compilación.

5. METODO DE RESOLUCIÓN

5.1 Requisitos

A continuación se exponen una serie de requisitos de sistema y de usuario para la correcta compresión y ejecución de la herramienta Caché.

5.1.1 Requisitos Hardware

Los requerimientos del ordenador para que se pueda instalar Caché (versión 4.1) son:

- Windows Server 2003 (SP1). Es compatible con sistemas tanto de 32 como de 64 bits. Windows XP Pro y Windows 2000.

- También es compatible con otros Sistemas Operativos como: Mac OS X 10.3, 10.4. , Red Hat Enterprise 4 Linux (32 y 64 bits), Sun Solaris 9, 10, SUSE Linux 9 (SP2) Enterprise Server (32 y 64 bits)

- 450 MB en el disco duro para la instalación de la herramienta. Esto no incluye los datos, por lo que se necesitará espacio adicional para el almacenamiento de los mismos.

- 10 MB libres en el sistema de Windows para la instalación.

- Acceso al CD-ROM para la instalación o bien acceso a la red.

- El funcionamiento de Caché mejora visiblemente cuanto mayor sea la velocidad del procesador y del disco.

5.1.2 Requisitos de Usuario

Para entender la herramienta y su funcionamiento el usuario debe tener conocimientos de los siguientes temas:

- Conocimientos de SQL. Caché soporta SQL y es muy útil para realizar diversas operaciones.

- Conocimientos de bases de datos relacionales. SQL está basado en bases de datos relacionales, y por ello para poder usar SQL es necesario conocer el funcionamiento de este tipo de bases de datos.

- Conocimientos sobre la programación orientada a objetos. Caché se basa en la programación orientada a objetos, por lo que carecer de dichos conocimientos haría muy complicado trabajar con la herramienta.

- Conocimientos de HTML, para la creación de páginas Web.

- Nivel medio de inglés. Puede parecer una obviedad, pero toda la documentación de Caché está en inglés, y por ello es necesario tener ciertos conocimientos del idioma.

Caché soporta, como se ha mencionado varias veces a lo largo de éste documento, SQL y programación orientada a objetos. Podría realizarse el trabajo utilizando sólo una de las dos opciones, pero el proceso sería mucho más largo y complicado. Utilizando las dos, el trabajo se hace más fácil y la aplicación permite muchas más opciones.

5.1.3 Requisitos funcionales

La realización de este proyecto se puede dividir en dos partes igualmente importantes: la realización de una librería para el tratamiento de datos espaciales y la creación de una aplicación que utilice la librería anterior. Por ello los requisitos funcionales se pueden también, dividir en dos partes.

5.1.3.1 Requisitos funcionales: Espaciales

En los objetivos del documento se mencionaba que la creación de la librería que gestione datos de tipo espacial debería ser general, es decir, que pudiera usarse para futuras aplicaciones que utilizaran al igual que ésta datos espaciales. Después de realizar un análisis de los diferentes tipos de datos que se pueden crear que representen fielmente datos espaciales, se llega a la conclusión de que es necesario crear al menos cuatro tipos de datos diferentes.

Estos tipos de datos son:

- Punto: representa un punto del espacio.
- Línea: representa un segmento en el espacio
- Polígono: formado por un conjunto de líneas unidas entre sí, que forman un polígono conexo.
- Cubo: formado por un conjunto de polígonos (rectángulos) unidos entre sí, que forman un cubo conexo.

En principio todos los tipos de datos deben tener operaciones básicas como nuevoPunto, nuevaLinea, nuevoPoligono y nuevoCubo.

Lo interesante de los datos espaciales, es la interacción entre ellos. Por ejemplo, saber si un punto pertenece o no a un polígono, por lo que se realizarán diferentes métodos que aseguren dicha interacción. Esta parte se explica con detenimiento en la sección 5.2 del documento.

5.1.3.2 Requisitos funcionales: aplicación Web

La aplicación que debe utilizar la librería creada, trata sobre el almacenamiento de medicamentos en cajoneras. Primero se va a explicar como son las cajoneras de una farmacia con un gráfico:

Figura 20: Cajoneras

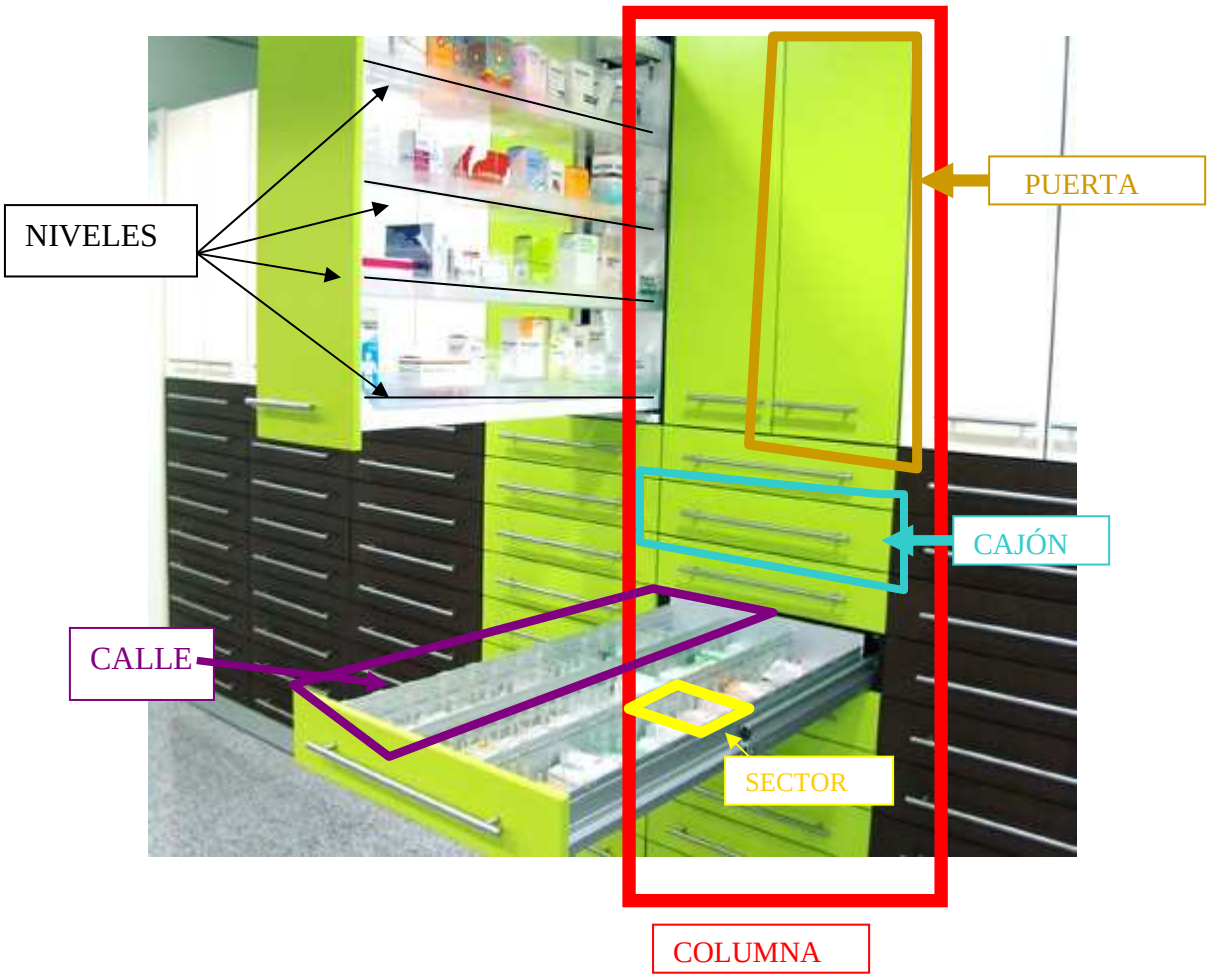


Figura 21: Explicación de una cajonera

A continuación se especifican los requisitos funcionales que debe cumplir dicha aplicación.

Todos los medicamentos se guardan en un lugar concreto. A este lugar en concreto, se le denomina sector. Por ello, la aplicación almacena información sobre los distintos sectores y lo que contienen. Relacionado con los mismos, la aplicación debe permitir:

- Obtener la localización de cualquier medicamento. Se introduce el medicamento a consultar y la aplicación muestra información del mismo y los sectores en los que esta almacenado.

- Obtener los medicamentos de una zona determinada. Dentro de una zona concreta, como por ejemplo un cajón, obtener todos los medicamentos que que se guardan en éste.

- Obtener el hueco libre dentro de un sector. Útil, para saber si se puede almacenar algún medicamento más.

- Obtener los sectores libres en los que se puede almacenar medicamentos y de cuanto espacio disponen.

Los medicamentos son los artículos que se almacenan en las cajoneras. Por ello la aplicación almacena información de los mismos. Con la información almacenada, la aplicación debe permitir:

- Obtener información sobre cualquier medicamento. Se introduce en la aplicación el medicamento a consultar y ésta devuelve la información sobre el mismo.

- Obtener qué tipo de medicamentos se guarda en cada zona.

- Obtener medicamentos dependiendo de la forma farmacéutica. Se pide un medicamento y se comprueba si existe en la forma farmacéutica requerida. Por ejemplo: buscar amoxicilina en jarabe (ni en capsulas, ni en sobres)

- Obtener medicamentos que contengan un determinado principio activo. Se introduce un principio activo a consultar y devuelve todos los medicamentos que lo contienen y sus posiciones.

5.2 Diseño de la Base de Datos

En este apartado se expone con detenimiento el diseño que se ha seguido para la creación de la base de datos del actual proyecto**.**

5.2.1 Modelo E/R

A continuación se explica cada una de las entidades y las relaciones existentes en este modelo.

Una columna tiene puertas y se divide en cajones, se identifica unívocamente con su id (id_col), además tiene un nombre y una posición. La posición viene dada por sus coordenadas (x, y, z), ya que queremos representarlo como un cubo tridimensional (así se definen las posiciones de todos los elementos).

Una puerta se distingue unívocamente por su id (id_puerta), su nombre y su posición. Pertenece a una única columna y se divide en niveles.

Cada nivel se identifica por su id (id_nivel), nombre y posición. Pertenece a una única columna y se divide en sectores_n (sectores de nivel).

Un cajón se distingue unívocamente por su id (id_cajón), su nombre y su posición. Pertenece a una única columna y se divide en calles.

Cada calle se identifica por su id (id_calle), nombre y posición. Pertenece a un único cajón y se divide en sectores_c (sectores de cajón).

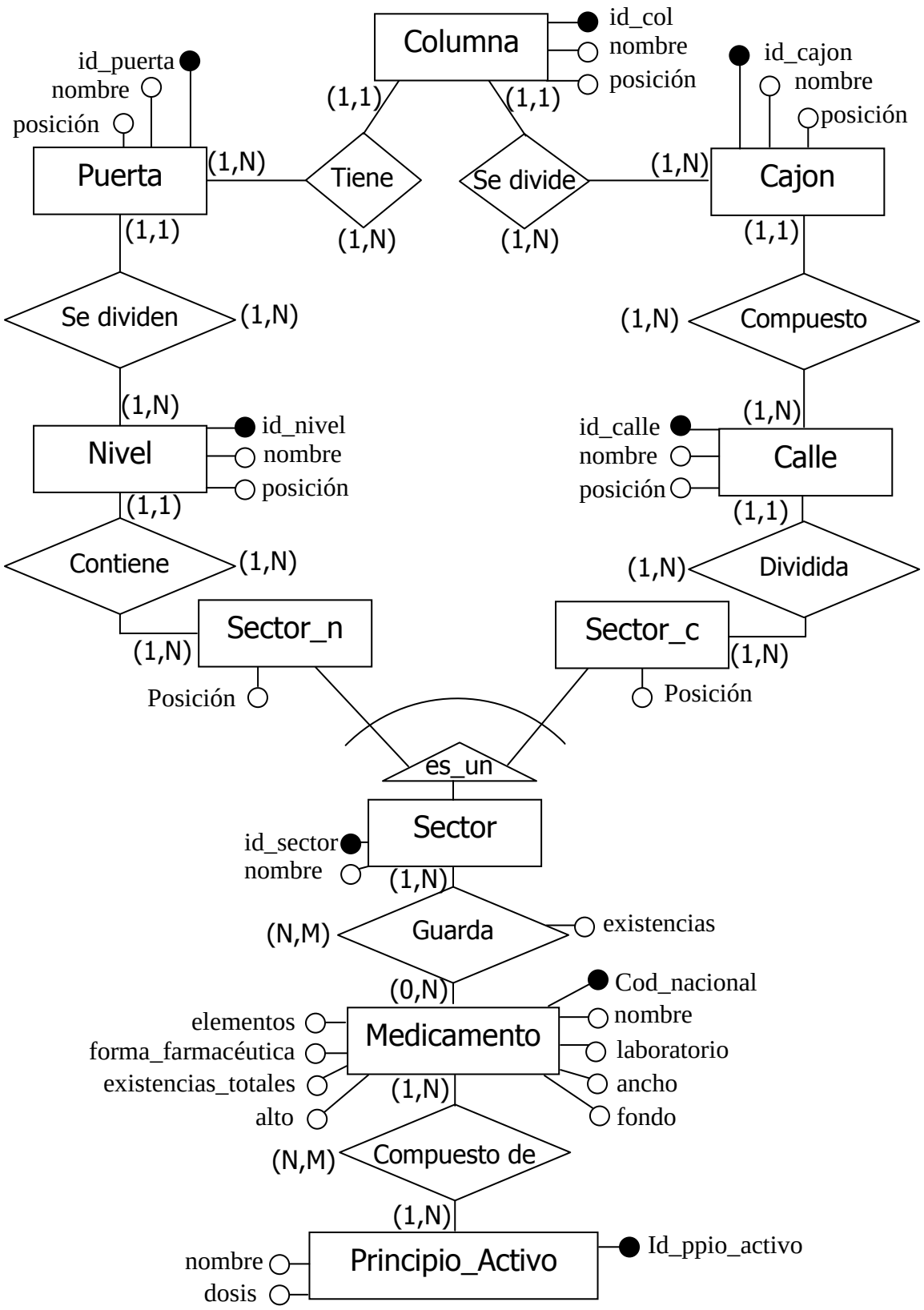
Como hemos visto un sector puede ser de dos tipos: un sector de un cajón o un sector de una puerta, pero sólo puede ser de un tipo.

En un sector se guardan medicamentos, de un tipo o de varios para aprovechar el espacio al máximo. Y cada medicamento se puede guardar en un único sector o en varios si se tienen más existencias. Por tanto, las existencias reflejadas en la relación "guarda" son las que se tienen en un sector concreto de un medicamento, y las "existencias_totales" de un medicamento son las que se tienen en total independientemente de los sectores en los que se encuentre.

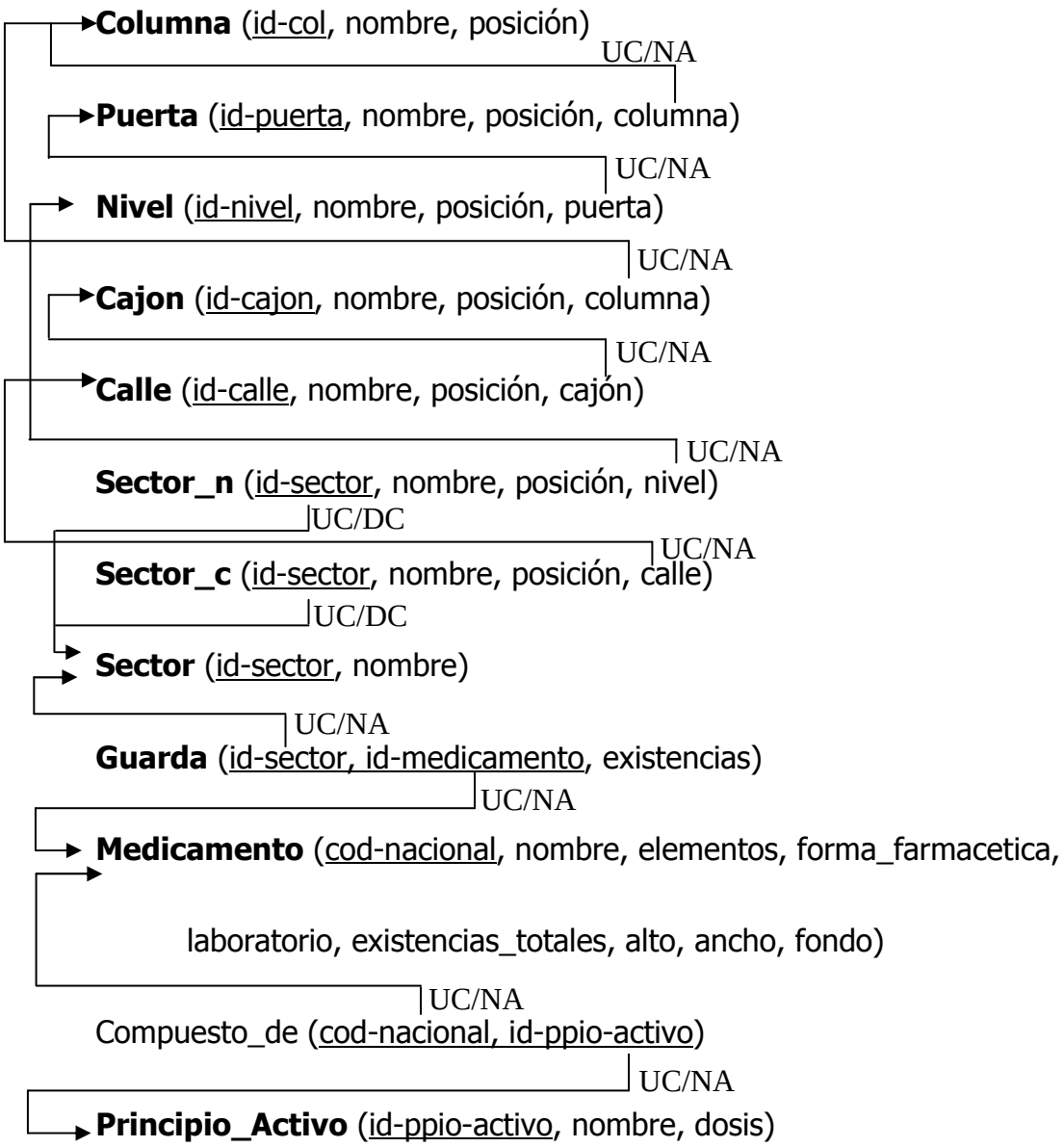
De un medicamento se guarda su identificador para diferenciarlo de los demás, en este caso su código nacional, compuesto de 6 cifras. Su nombre (nombre comercial del medicamento), laboratorio que lo fabrica, su forma farmacéutica (comprimidos, cápsulas, sobres, solución, gotas, supositorios, óvulos, inyectables…), los elementos que contiene (si un medicamento tiene 12 sobres, sus elementos serían 12), las existencias totales que hay en stock y su dimensión (ancho, alto y fondo). En este caso la dimensión se refiere a la de la caja del medicamento (suponemos que todas son cúbicas) para conocer el volumen que ocupa.

Un principio activo es una sustancia con acción farmacológica, cada medicamento puede estar compuesto de uno o varios principios activos y en distintas dosis. Por ejemplo: un "gelocatil" tiene como principio activo el paracetamol, pero será distinto el que tenga una dosis de 650 mg que el que tenga 1 gr. Un "augmentine" tiene dos principios activos cada uno con su dosis:

amoxicilina 500 mg y ácido clavulánico 125 mg ó amoxicilina 875 mg y ácido clavulánico 125 mg.



5.2.2 Modelo Relacional



Las restricciones de integridad del anterior esquema relacional son las siguientes:

[FK Columna-Puerta] UC/NA → Si se actualiza una columna, la tabla puerta también lo haría. En cambio no se puede borrar una columna si tiene puertas asignadas.

[FK Puerta-Nivel] UC/NA → Al actualizar una puerta, también se actualizará el nivel correspondiente. Sin embargo no se puede borrar una puerta que tenga niveles asignados.

[FK Columna-Cajón] UC/NA → Si se actualiza una columna, la tabla cajón también lo haría. En cambio no se puede borrar una columna si tiene cajones asignados.

[FK Cajón-Calle] UC/NA → Al actualizarun cajón, también se actualizará la calle correspondiente. Sin embargo no se puede borrar un cajón que tenga calles asignadas.

[FK Nivel-Sector_N] UC/NA → Si se actualiza un nivel, la tabla Sector_n también lo haría. En cambio no se puede borrar un nivel si tiene sectores_n asignados.

[FK Calle-Sector_C] UC/NA → Si se actualiza una calle, la tabla Sector_c también lo haría. En cambio no se puede borrar una calle si tiene sectores_c asignados.

[FK Sector-Sector_N] UC/DC → Cualquier actualización realizada en sector, se verá reflejada en la tabla Sector_N. Del mismo modo, si se elimina un sector, también se eliminará el Sector_N.

[FK Sector-Sector_C] UC/DC → Cualquier actualización realizada en sector, se verá reflejada en la tabla Sector_C. Del mismo modo, si se elimina un sector, también se eliminará el Sector_C.

[FK Sector-Guarda] UC/NA → Si se actualiza un sector, la tabla guarda también lo haría. En cambio no se puede borrar un sector si tiene objetos en la tabla guarda asignados.

[FK Medicamento-Guarda] UC/NA → Si se actualiza un medicamento, la tabla guarda también lo haría. En cambio no se puede borrar un medicamento si tiene objetos en la tabla guarda asignados.

[FK Medicamento-Compuesto_de] UC/NA → Si se actualiza un medicamento, la tabla "compuesto_de" también lo haría. En cambio no se puede borrar un medicamento si tiene objetos en la tabla "compuesto_de" asignados.

[FK Principio_Activo-Compuesto_de] UC/NA → Si se actualiza un principio activo, la tabla "compuesto_de" también lo haría. En cambio no se puede borrar un principio activo si tiene objetos en la tabla "compuesto_de" asignados.

5.2.3 Diseño Librería Espaciales

Uno de los objetivos fundamentales del proyecto era la creación de una librería que fuese capaz de gestionar datos espaciales. Se pedía que esta librería se diseñase lo más general posible para que pudiese reutilizarse en proyectos futuros, que utilicen al igual que éste, datos espaciales.

Después de analizar los posibles tipos de datos que fuesen capaces de representar y soportar datos espaciales, se crearon los siguientes:

Punto: Caracterizado por tres coordenadas (x,y,z). Las coordenadas que se almacenan en la base de datos se toman considerando el cero como el extremo inferior izquierdo del fondo. Por ello se ha creado un tipo de datos punto, caracterizado por tres coordenadas numéricas x, y, z.

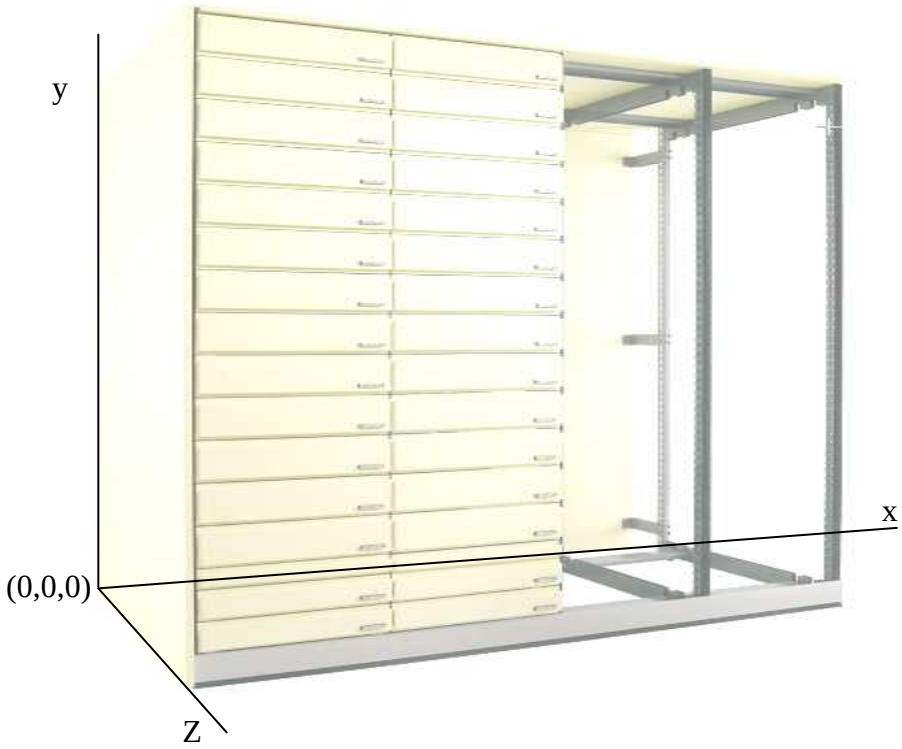


Figura 22: Sistema de coordenadas

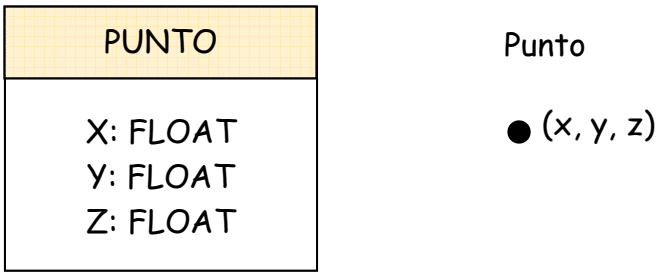


Figura 23: Punto

Línea: Caracterizado por dos elementos: origen y destino. Éstos son a su vez dos puntos, como los definidos anteriormente. La línea queda por tanto definida de forma similar a un segmento, con un punto inicial y otro final.

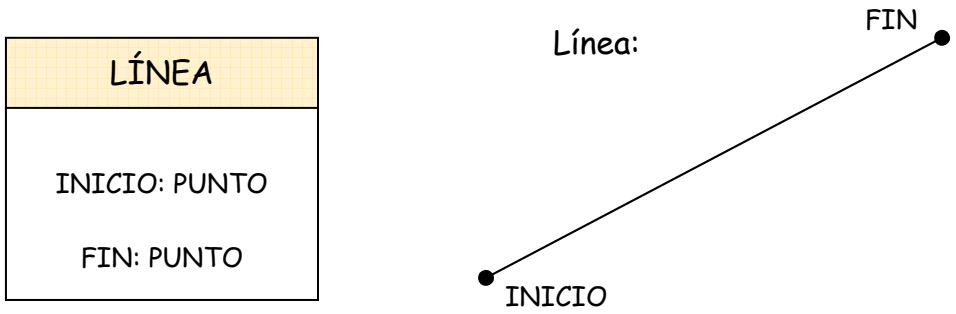


Figura 24: Línea

Donde INICIO (x1, y1, z1) y FIN (x2, y2, z2)

Polígono: figura geométrica formada por aristas que dibuja una forma conexa. Las aristas, son líneas como las anteriormente definidas. Los polígonos que vamos a usar para este proyecto son rectángulos, ya que se van a tomar como las caras de los cubos a representar. Por lo tanto An será 4 para este caso concreto.

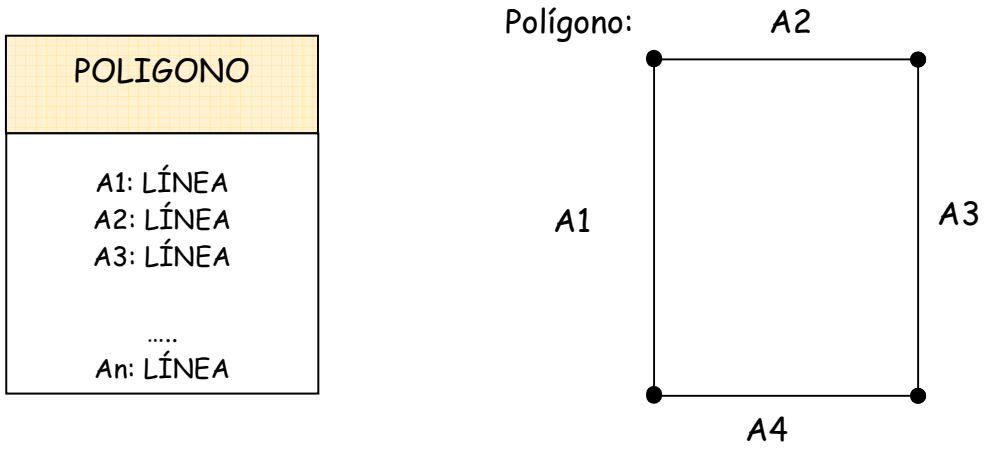
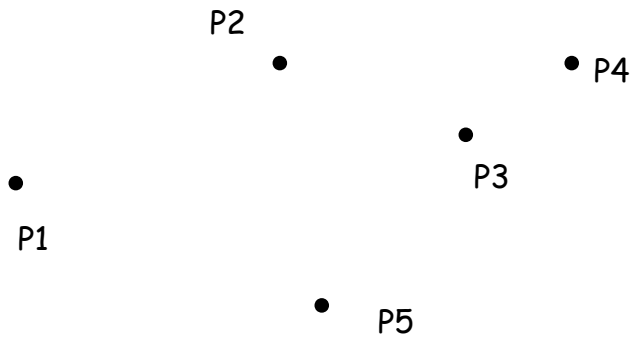
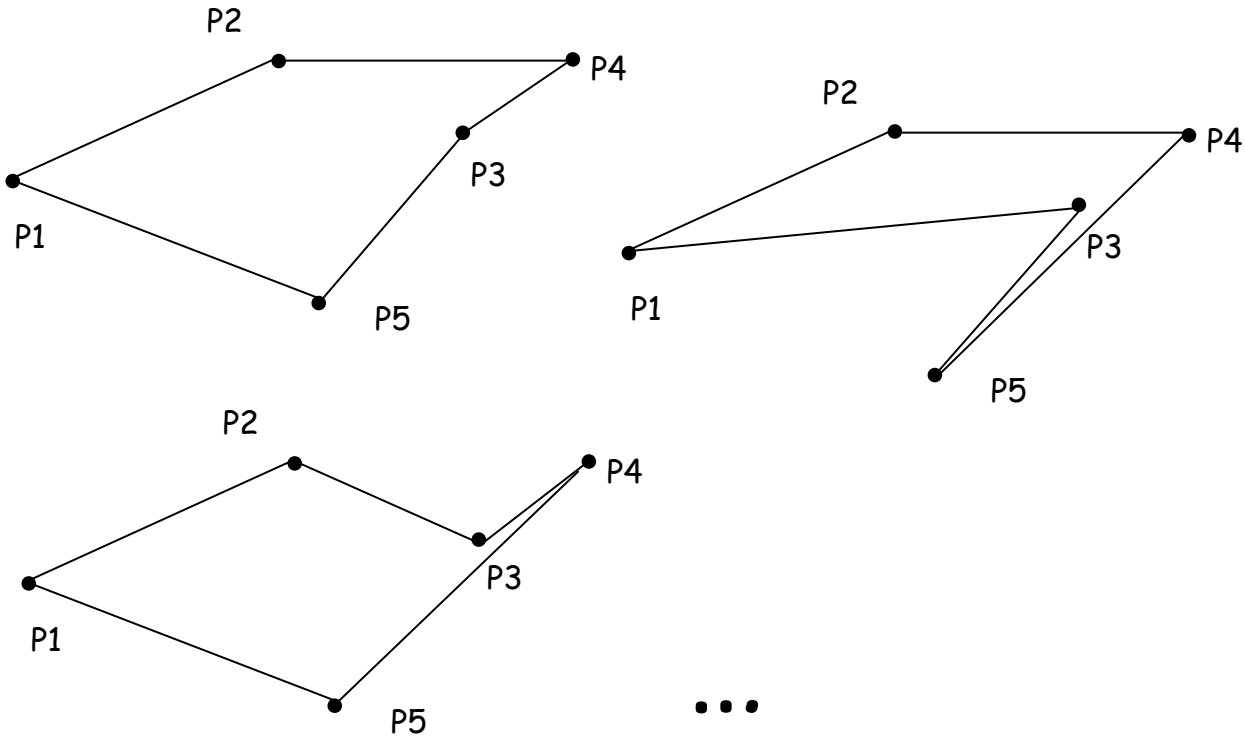


Figura 25: Polígono

El polígono se podría haber definido como un conjunto de puntos, pero se ha definido como un conjunto de aristas (líneas) porque para la implementación de métodos del mismo, resulta más fácil hacerlo por líneas. Si sólo se introdujesen puntos, no se sabría con exactitud que formas tienen los polígonos. En el siguiente ejemplo se introducen cinco puntos:



Para esta combinación de puntos, existen varias formas de interpretarlos como polígono:



Es decir, que para los mismos cinco puntos, existen multitud de formas de interpretar el polígono. Esto se podría solucionar introduciendo los puntos en orden. Sin embargo, al hacerlo en líneas no es necesaria esta restricción. Por tanto se define una polígono, como un conjunto de líneas en lugar de puntos, porque la implementación de los métodos resulta más sencilla y además no se requiere como restricción que la introducción de las mismas sea en orden.

Cubo: figura geométrica formada por seis polígonos que dibuja una forma conexa. Las caras, son polígonos como los anteriormente definidos.

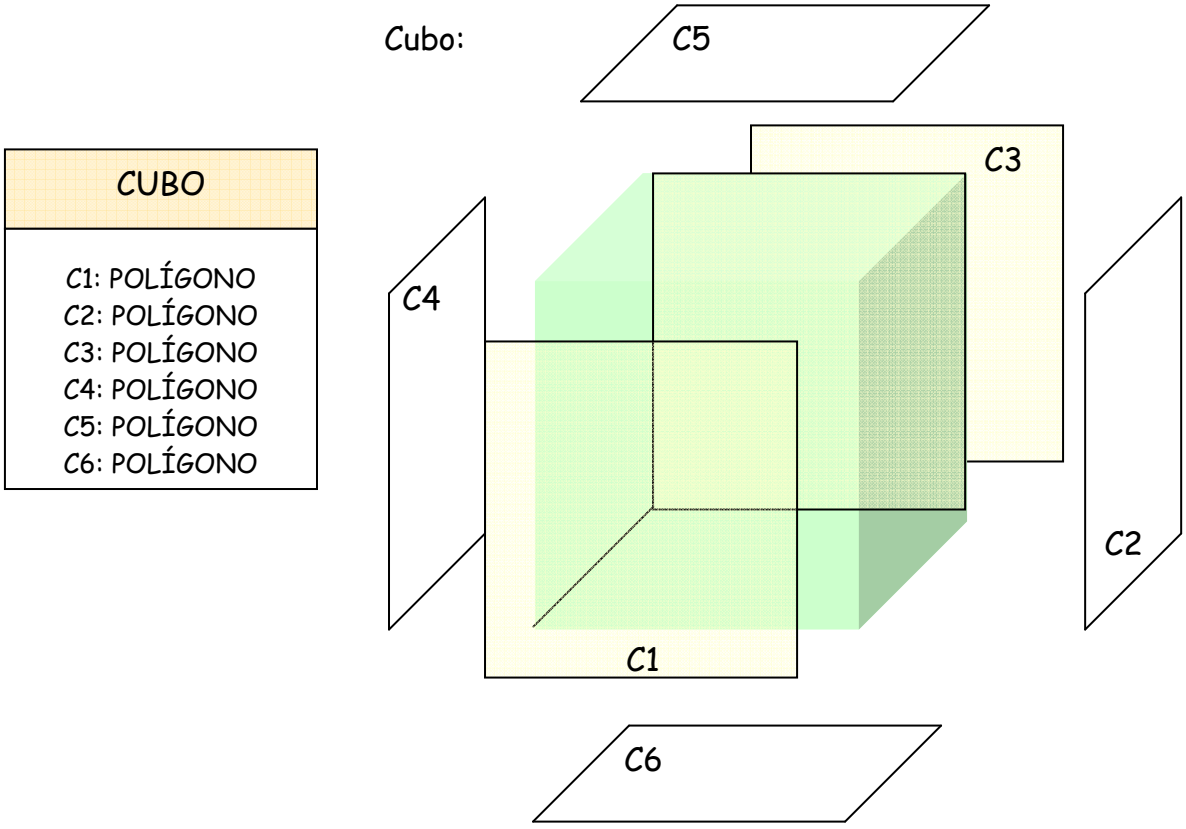


Figura 26: Cubo

Con un cubo se definen todas las estructuras del proyecto: columna, puerta, nivel, cajón, calle, sector y medicamento (la dimensión de la caja que lo contiene).

Una vez definidos todos los tipos de datos, se pasan a definir los métodos que implementará cada uno de ellos. Los métodos implementados se muestran a continuación en la siguiente tabla:

	PUNTO	LINEA	POLÍGONO	CUBO
PUNTO	Distancia entre puntos	Pertenece punto a línea	Punto dentro de polígono	Punto dentro de cubo
LÍNEA	Pertenece punto a línea			
POLÍGONO	Punto dentro de polígono		Polígono dentro de polígono	
CUBO	Punto dentro de cubo			Cubo dentro de cubo Volumen de cubo

Tabla 1: Tipos de datos espaciales y sus métodos

5.2.3.1 Funcionalidad detallada

En esta sección se explica con detenimiento los métodos mostrados en la tabla anterior, los cuales permiten la interacción entre los distintos tipos de datos definidos.

Distancia entre puntos: Como su propio nombre indica, muestra la distancia entre dos puntos dados. Se introducen las coordenadas de los puntos y el método devuelve la distancia entre ambos, en metros.

Como son tres dimensiones primero se calcula la distancia entre las proyecciones (x1, z1) y (x2, z2) y con la diferencia entre las coordenadas y de cada puno se crea un triángulo rectángulo donde la hipotenusa es la distancia entre los puntos en tres dimensiones. Para entenderlo se utiliza el gráfico siguiente:

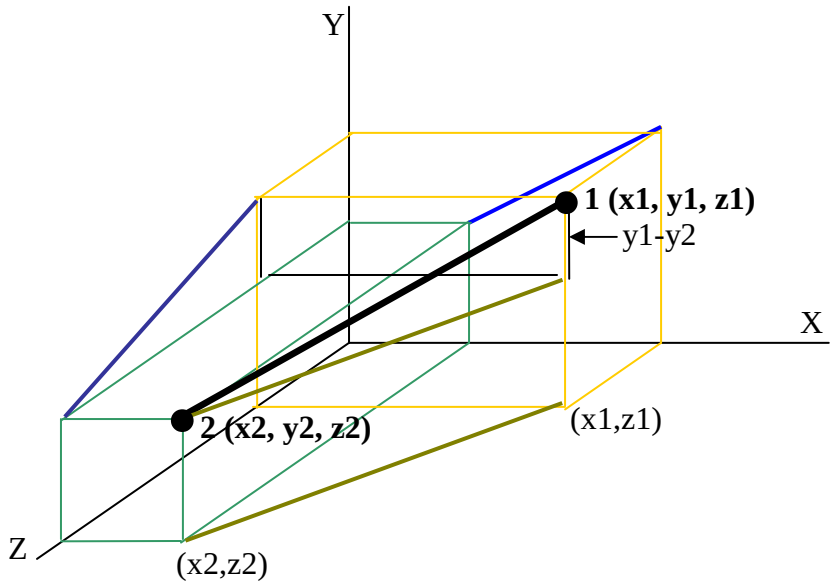


Figura 27: Distancia entre dos puntos

La distancia entre dos puntos en dos dimensiones viene dada por la raíz cuadrada de: (x2-x1)^2 + (z2-z1)^2. Para conocer el valor absoluto de la distancia entre las proyecciones y de los puntos, la diferencia se eleva al cuadrado y se hace la raiz cuadrada.

```
ClassMethod distancia(x1 As %Float, y1 As %Float, z1 As %Float, x2 As %Float, y2 As %Float, z2 As %Float) As %Float
{
	set ax1 = (x2-x1)
	set az2 = (z2-z1)

	set cax1 = ax1*ax1
	set caz2 = az2*az2

	set sum = cax1 + caz2
	set cat1 = $ZSQR(sum)

	set ay2 = (y2-y1)

	set cay2 = ay2*ay2

	set cat2 = $ZSQR(cay2)

	set hip = (cat1*cat1)+(cat2*cat2)
	set sol = $ZSQR(hip)

	set solRednd = $FNUMBER(sol,".",3)

	quit solRednd
}
```

Pertenece punto a línea: Comprueba si un punto pertenece o no a una línea. Para esto, si el punto está en dos de las rectas proyectadas en los

ejes de coordenadas, entonces se puede asegurar que el punto pertenece a la recta representada en tres dimensiones. Recibe como argumentos los identificadores del punto y la línea respectivamente. Devuelve un entero: 0 si no pertenece y 1 en caso contrario.

```
ClassMethod pertenecePuntoLinea2d(punto As %ObjectIdentity, linea As
%ObjectIdentity) As %Integer
{

	set p = ##class(Espaciales.punto).%OpenId(punto)

	set l = ##class(Espaciales.linea).%OpenId(linea)

	set l1 = ##class(Espaciales.punto).%OpenId(l.origen)

	set l2 = ##class(Espaciales.punto).%OpenId(l.destino)


	If (##class(Espaciales.linea).compruebaRango(p,l1,l2)= 1)
	{
		set a1 = (l2.y -l1.y)

		set a2 = (l2.x -l1.x)

		If (a1=0)  //La recta es horizontal
		{
			if p.y = l2.y
			{
				set pert = 1
			}
			else {quit 0}

		}


		elseif (a2=0) //La recta es vertical
		{
			if p.x = l2.x
			{
				set pert = 1
			}
			else {quit 0}
		}

		else // sino se calcula la ecuación de una recta y se sustituye
		    // el  punto en ella
		{
			set m = a1/a2
			set c = ((m*(-l1.x)) + l1.y)
			if (m*p.x -p.y+c = 0)
			{
				set pert = 1
			}
```

```
                else {quit 0}
            }

        }
        else {quit 0} //El punto esta fuera del rango

        If (pert=1)// el punto pertenece a la primera proyeccion y se comprueba la
                   //segunda proyeccion

            {
            set a1 = (l2.z -l1.z)

            set a2 = (l2.x -l1.x)

            If (a1=0)  //La recta es perpendicular al eje z
            {
                if p.z = l2.z
                {
                    write "El punto pertenece a la recta"
                    Quit 1
                }
                else {quit 0}

            }
            elseif (a2=0) //La recta es vertical
            {
                if p.x = l2.x
                {
                    write "El punto pertenece a la recta"
                    Quit 1
                }
                else {quit 0}
            }

            else // sino se calcula la ecuación de una recta y se sustituye
                 // el  punto en ella
            {
                set m = a1/a2
                set c = ((m*(-l1.x)) + l1.z)
                if (m*p.x - p.z + c = 0)
                {
                    write "El punto pertenece a la recta"
                    {Quit 1}
                }
                else {quit 0}
            }

}

}
```

Punto dentro de polígono: Comprueba si un punto está dentro de un polígono. Recibe como argumentos las coordenadas del punto y el polígono.

Devuelve 0 si el punto está dentro y 1 si está fuera del polígono. La comprobación se realiza extendiendo, desde el punto a comprobar, una recta a lo largo del eje x. El número de intersecciones entre ésta y el polígono indicará si el punto pertenece o no al mismo. Si el número de intersecciones es par ó cero, el punto está fuera. Por el contrario si el número de intersecciones es uno, el punto está dentro del polígono.

En la siguiente figura, se muestra la idea anteriormente expuesta:

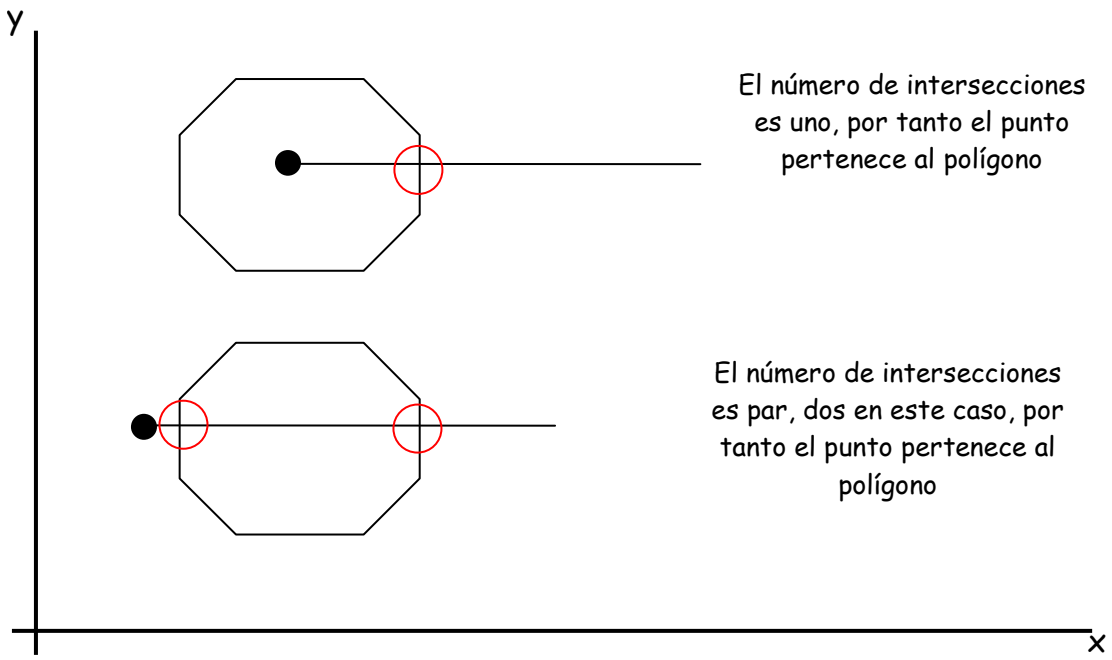


Figura 28: Pertenece punto a polígono

Como nuestro polígono está en tres dimensiones tendremos que hacer como en el caso anterior, comprobar que el punto está dentro de al menos dos de los polígonos proyectados en los ejes de coordenadas, en este caso comprobaremos el proyectado en los ejes x-y y el proyectado en los ejes x-z. Y en la proyección del plano que forma el polígono sobre los ejes x-y, para esto comprobaremos que el punto pertenece a la línea proyectada.

Para esta proyección se necesita saber los dos puntos en los que z es 0 (puntos en los que laslíneas cortan el plano x-y) para dos líneas alternas del polígono (ya que hemos explicado anteriormente que los polígonos que vamos

a utilizar en este proyecto son de cuatro lados) y con esos dos puntos se crea la línea que proyecta el plano del polígono en el plano x, y.

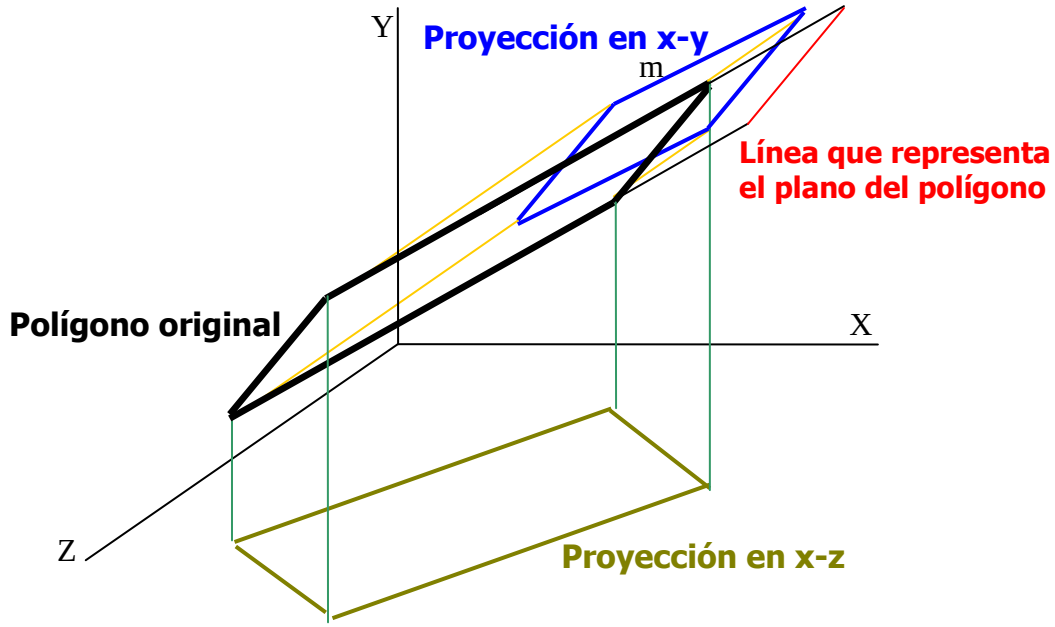


Figura 29: Proyecciones del polígono

Con la ecuación de la recta en tres dimensiones haciendo z=0 se saca el punto en el que cada recta del polígono interseca con el plano x, y y con esos dos puntos se tiene la recta que proyecta el plano del polígono.

Ecuación de la recta tridimensional: $\frac{x-x_o}{a}=\frac{y-y_o}{b}=\frac{z-z_o}{c}$, donde a, b y c son las coordenadas del vector dirección de la recta. Estas se hallan restando las coordenadas de dos puntos pertenecientes a la recta:

a = x1 - x2

b = y1 - y2

c = z1 - z2

Haciendo z = 0

(x-x1)/a = (0-z1)/c despejando nos queda: x= (c*x1 – a*z1)/c

(y-y1)/b = (0-z1)/c despejando nos queda: y= (c*y1 – b*z1)/c

donde (x,y) es el punto donde corta la ecuación el plano x,y.

A continuación, el método que implementa la idea anterior:

```
ClassMethod puntoDentroPoligono(x, y, z As %Float, polg As poligono) As %Integer
{
	//Me los pasan abiertos

	set interseccion = 0
	set n = polg.numAristas

	for cont=1:1:n
	{
		set l = polg.aristas.GetAt(cont)

		set linea = ##class(Espaciales.linea).%OpenId(l)

		set inicio = ##class(Espaciales.punto).%OpenId(linea.inicio) //v[i]
		set fin = ##class(Espaciales.punto).%OpenId(linea.fin) //v[i+1]


		 if (((inicio.y <= y) && (fin.y > y)) // comprueba hacia arriba
		|| ((inicio.y > y) && (fin.y <= y))) // comprueba hacia abajo
		{
			// comprueba la arista actual con la coordenada x
			 set vt = (y - inicio.y) / (fin.y -inicio.y)

				set t = vt * (destino.x - origen.x)
				set t2 = t + origen.x

				 //SI (p.x < origen.x + vt * (destino.x - origen.x)) //
				P.x < interseccion

				if (x < t2)
					{
					set interseccion = interseccion +1 }
					}

		 }


	If (interseccion = 1)// seguimos comprobando la siguiente proyeccion (x,z)
	{

	   set interseccion = 0
	   set n = polg.numAristas

	   for cont=1:1:n
		{
		set l = polg.aristas.GetAt(cont)
```

```
set linea = ##class(Espaciales.linea).%OpenId(l)

set inicio = ##class(Espaciales.punto).%OpenId(linea.inicio) //v[i]
set fin = ##class(Espaciales.punto).%OpenId(linea.fin) //v[i+1]


 if (((inicio.z <= z) && (fin.z > z)) // comprueba hacia arriba
|| ((inicio.z > z) && (fin.z <= z))) // comprueba hacia abajo
 {
        // comprueba la arista actual con la coordenada x
         set vt = (z - inicio.z) / (fin.z -inicio.z)

                set t = vt * (destino.x - origen.x)
                set t2 = t + origen.x

                 //SI (p.x < origen.x + vt * (destino.x - origen.x)) //
                P.x < interseccion

                if (x < t2)
                        {
                        set interseccion = interseccion +1 }
                        }
                }

If (interseccion = 1) // buscamos la recta que proyecta el plano
{
for cont=1:1:2
{

  set l1 = polg.aristas.GetAt(cont)
  set cont = cont + 2
  set l2 = pol.arista.GetAt(cont)
  set cont = cont - 2

  set linea1 = ##class(Espaciales.linea).%OpenId(l1)
  set linea2 = ##class(Espaciales.linea).%OpenId(l2)

  set inicio1 = ##class(Espaciales.punto).%OpenId(linea1.inicio)
  set fin1 = ##class(Espaciales.punto).%OpenId(linea1.fin)

  set a1 = inicio1.x – fin1.x
  set b1 = inicio1.y – fin1.y
  set c1 = inicio1.z – fin1.z

  If a = 0 // la linea no corta el eje x probamos con las otras dos
                 //aristas
                { set cont = cont +1}
  Else
        { set cont = 3}
}



set inicio2 = ##class(Espaciales.punto).%OpenId(linea2.inicio)
set fin2 = ##class(Espaciales.punto).%OpenId(linea2.fin)
```

```
                set a2 = inicio2.x – fin2.x
                set b2 = inicio2.y – fin2.y
                set c2 = inicio2.z – fin2.z

        set p1x = ((c1*inicio1.x) – (a1*inicio1.z)) / c1
        set p1y = ((c1*inicio1.y) – (b1*inicio1.z)) / c1

        set p2x = ((c2*inicio2.x) – (a2*inicio2.z)) / c2
        set p2y = ((c2*inicio2.y) – (b2*inicio2.z)) / c2

        set a1 = (p2.y -p1.y)

        set a2 = (p2.x p1.x)

        If (a1=0)  //La recta es horizontal
        {
                if y = p2.y
                {
                        write "El punto pertenece al polígono"
                        Quit 0
                }
                else {quit 1}

        }


        elseif (a2=0) //La recta es vertical
        {
                if x = l2.x
                {
                        write "El punto pertenece al polígono"
                        Quit 0
                }
                else {quit 1}
        }

        else // sino se calcula la ecuación de una recta y se sustituye
             // el  punto en ella
        {
                set m = a1/a2
                set c = ((m*(-p1.x)) + p1.y)
                if (m*x -y+c = 0)
                {
                        write "El punto pertenece al polígono"
                        Quit 0

                }
                else {quit 1}
        }
}


        else// Si las intersecciones son 0 ó 2 ,
        {quit 1}
        }
```

```
        else // Si las intersecciones son 0, si esta a la dcha, o 2 ,si esta a la izqda,
                {quit 1}
        }
```

Polígono dentro de polígono: Comprueba si un polígono está contenido en otro. Recibe como argumentos los dos polígonos. Devuelve 1 si el está contenido y 0 si está fuera.

La comprobación se realiza viendo que todos los puntos que delimitan las aristas del polígono a mirar están dentro del otro. Para esto se utiliza el método "*puntoDentroPoligono"*.

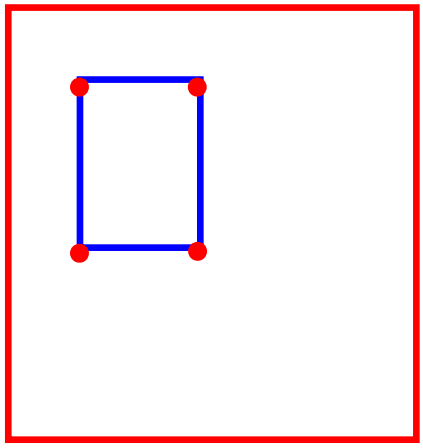


Figura 30: Polígono dentro de polígono

```
        ClassMethod pertenecePoligonoAPoligono(pol1 As %ObjectIdentity, pol As
%ObjectIdentity) As %Integer
{

        set poli = ##class(Espaciales.poligono).%OpenId(pol1)
        set num = poli.numLineas
        set contador=1
        set ok = 0

        While (contador<= num)& (ok=0)
```

```
        {
                set l = poli.lineas.GetAt(contador)
                set p1 = ##class(Espaciales.punto).OpenId(l.inicio)
                set p2 = ##class(Espaciales.punto).OpenId(l.fin)


set sol1 =##class(Espaciales.poligono).puntoDentroPoligono(p1.x,p1.y,p1.z,pol)
set sol2 =##class(Espaciales.poligono).puntoDentroPoligono(p2.x,p2.y,p2.z,pol)

                If ((sol1 = 0) && (sol2 = 0)) // El punto pertenece al polígono
                        {
                        set ok = 1
                        }

                set contador = contador + 1
        }

        do poli.%Close()
        quit ok
}
```

Punto dentro de cubo: Comprueba si un punto está dentro de un cubo. Recibe como argumentos las coordenadas del punto y el cubo. Devuelve 0 si el punto está dentro y 1 si está fuera del polígono.

La comprobación se realiza viendo que el punto esta dentro de las proyecciones del cubo sobre los planos de coordenadas (con esto es suficiente ya que las zonas del proyecto y las dimensiones de los medicamentos son rectangulares, nunca oblícuas respecto a los planos de coordenadas).

Para que el punto este contenido en el cubo tiene que estarlo en 3 proyecciones o mas, ya que si solo lo estuviese en dos podrían ser caras paralelas del cubo y por tanto no estaría contenido ya que no lo está en las demás.

```
ClassMethod puntoDentroCubo (x, y, z As %Float, cub As cubo) As %Integer
{
	//Me los pasan abiertos

	set interseccion = 0
	set n = 6
	set num = 0


	for cont=1:1:n
	{
		set c = cubo.cara.GetAt(cont)

		set polg = ##class(Espaciales.poligono).%OpenId(c)

		set intersecc = 0
		set n2 = polg.numAristas

	for cont2=1:1:n2
	{
		set l = polg.aristas.GetAt(cont2)

		set linea = ##class(Espaciales.linea).%OpenId(l)

		set inicio = ##class(Espaciales.punto).%OpenId(linea.inicio)
		set fin = ##class(Espaciales.punto).%OpenId(linea.fin)



		 if (((inicio.y <= y) && (fin.y > y)) // comprueba hacia arriba
	    || ((inicio.y > y) && (fin.y <= y))) // comprueba hacia abajo
	    {
			// comprueba la arista actual con la coordenada x
			 set vt = (y - inicio.y) / (fin.y -inicio.y)

				set t = vt * (destino.x - origen.x)
				set t2 = t + origen.x

				 //SI (p.x < origen.x + vt * (destino.x - origen.x)) //
				P.x < intersecc

				if (x < t2)
					{
					set intersecc = intersecc +1 }
					}

	     }

	If (intersec = 1)// seguimos comprobando la siguiente proyeccion (x,z)
	{

	   set intersecc = 0
	   set n2 = polg.numAristas
```

```
for cont2=1:1:n
        {
        set l = polg.aristas.GetAt(cont)
        set linea = ##class(Espaciales.linea).%OpenId(l)

        set inicio = ##class(Espaciales.punto).%OpenId(linea.inicio) //v[i]
        set fin = ##class(Espaciales.punto).%OpenId(linea.fin) //v[i+1]


         if (((inicio.z <= z) && (fin.z > z)) // comprueba hacia arriba
      || ((inicio.z > z) && (fin.z <= z))) // comprueba hacia abajo
         {
                // comprueba la arista actual con la coordenada x
                 set vt = (z - inicio.z) / (fin.z -inicio.z)

                        set t = vt * (destino.x - origen.x)
                        set t2 = t + origen.x

                         //SI (p.x < origen.x + vt * (destino.x - origen.x)) //
                        P.x < intersecc

                        if (x < t2)
                                {
                                set intersecc = intersecc +1 }
                                }
                        }
                If intersecc = 1
                        {set num = num + 1}
         }
   }
}// termina de mirar todos los polígonos

If (num >= 3) // esta contenido en mas de dos proyecciones

        {quit 0}

else // no esta contenido

        {quit 1}

}
```

Cubo dentro de cubo: Comprueba si un cubo está contenido en otro cubo. Recibe como argumentos los dos cubos. Devuelve 0 si está contenido y 1 si no lo está.

La comprobación se realiza viendo que todos los vértices del cubo a comprobar se encuentran dentro del otro cubo. Para esto se utilza el método "*puntoDentroCubo*".

```
ClassMethod perteneceCuboACubo(cu1 As cubo, cu2 As cubo) As %Integer
{

set cub = ##class(Espaciales.cubo).%OpenId(cu1)
set nume = cub.numcaras
set cont=1

While (cont<= nume)& (ok=0)
{
        set poli = cub.cara.GetAt(contador)
        set num = poli.numLineas
        set contador=1
        set ok = 0

        While (contador<= num)& (ok=0)
        {
        set l = poli.lineas.GetAt(contador)
        set p1 = ##class(Espaciales.punto).OpenId(l.inicio)
        set p2 = ##class(Espaciales.punto).OpenId(l.fin)

set sol1 =##class(Espaciales.cubo).puntoDentroCubo(p1.x,p1.y,p1.z,cu2)
set sol2 =##class(Espaciales.cubo).puntoDentroCubo(p2.x,p2.y,p2.z,cu2)

        If ((sol1 = 0) && (sol2 = 0)) // El punto pertenece al cubo
                {
                set ok = 1
                }
        set contador = contador + 1
        }
set cont = cont + 1
}

do cub.%Close()
quit ok
}
```

Volumen de cubo: Calcula el volumen de un cubo dado. Recibe como argumento el cubo. Devuelve el volumen. A continuación se describen algunos de los parámetros utilizados en el método:

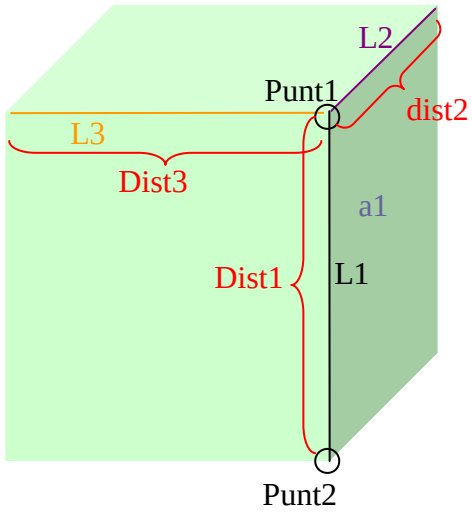


Figura 31: Parámetros del cubo

```
ClassMethod volumenCubo(cu1 As cubo) As %Integer
{

set cub = ##class(Espaciales.cubo).%OpenId(cu1)

set nume = cub.numcaras
set cont= 2

set ok = 0

set poli1 = cub.cara.GetAt(1)// poligono que voy a comparar
set l1 = poli1.lineas.GetAt(1) //linea del poligono 1 que comparo


set punt1 = ##class(Espaciales.punto).OpenId(l1.inicio)
set punt2 = ##class(Espaciales.punto).OpenId(l1.fin)

While (cont<= nume)& (ok=0)
{
        set poli2 = cub.cara.GetAt(cont)
```

```
        set num = poli2.numLineas
        set contador = 1
        set ok = 0

        While (contador<= num)& (ok=0) //miro todas las lineas del poligono
        {
        set l = poli2.lineas.GetAt(contador)
        set p1 = ##class(Espaciales.punto).OpenId(l.inicio)
        set p2 = ##class(Espaciales.punto).OpenId(l.fin)

                If (((punt1 = p1) && (punt2 = p2))
                  || ((punt1 = p2) && (punt2 = p1)))
                        {
                        set ok = 1
                        set contador = contador – 1
                        }
        set contador = contador + 1
        }
set cont = cont + 1
}

// busco la linea de poli1 y poli2 distinta de la que los une que tiene punt1

set num = poli2.numLineas
set contador = 1
set ok = 0

While (contador<= num)& (ok=0) //miro todas las lineas del poligono
        {
        set l2 = poli2.lineas.GetAt(contador)
        set p1 = ##class(Espaciales.punto).OpenId(l.inicio)
        set p2 = ##class(Espaciales.punto).OpenId(l.fin)

                If ((punt1 = p1) || (punt1 = p2))
                {
                set ok = 1
                set contador = contador – 1
                }
```

```
                set contador = contador + 1
                }

        set num = poli1.numLineas
        set contador = 1
        set ok = 0

        While (contador<= num)& (ok=0) //miro todas las lineas del poligono
                {
                set l3 = poli1.lineas.GetAt(contador)
                set p1 = ##class(Espaciales.punto).OpenId(l.inicio)
                set p2 = ##class(Espaciales.punto).OpenId(l.fin)

                        If ((punt1 = p1) || (punt1 = p2))
                        {
                        set ok = 1
                        set contador = contador – 1
                        }
                set contador = contador + 1
                }
        // Calculo la distancia entre lospuntos de cada linea
        set p1 = ##class(Espaciales.punto).OpenId(l1.inicio)
        set p2 = ##class(Espaciales.punto).OpenId(l1.fin)
set dist1 = ## class(Espaciales.punto).distancia(p1.x,p1.y,p1.z,p2.x,p2.y,p2.z)

        set p1 = ##class(Espaciales.punto).OpenId(l2.inicio)
        set p2 = ##class(Espaciales.punto).OpenId(l2.fin)
set dist2 = ## class(Espaciales.punto).distancia(p1.x,p1.y,p1.z,p2.x,p2.y,p2.z)

        set p1 = ##class(Espaciales.punto).OpenId(l3.inicio)
        set p2 = ##class(Espaciales.punto).OpenId(l3.fin)
set dist3 = ## class(Espaciales.punto).distancia(p1.x,p1.y,p1.z,p2.x,p2.y,p2.z)

        set a1 = dist1*dist2// area del poligono

        set vol = a1*dist3// volumen del cubo

        do cub.%Close()
        quit vol
}
```

La funcionalidad principal de esta librería se ha explicado detalladamente, sin embargo no son éstos los únicos métodos desarrollados en la misma. Existen otros métodos como auxiliares o métodos relacionados con la gestión de los datos, que permiten el correcto funcionamiento de la librería.

5.3 Funcionalidad de la aplicación

A continuación se muestra de forma detallada la funcionalidad que cumple la aplicación y el código que implementa dicha funcionalidad.

5.3.1 Consultas

La aplicación permite una gran variedad de consultas. La página principal se encarga de recoger los datos que introduce el usuario, y dependiendo de éstos y la consulta que quiera ejecutar, realizar una llamada u otra a los métodos que implementan la funcionalidad de la aplicación. Sin embargo entre la página principal y la base de datos donde se encuentran los métodos, existe otra página auxiliar que actúa entre ambas. Esto se hace, para crear una interfaz sencilla y agradable para el usuario, que es la página principal que el usuario ve. Sin embargo, para facilitar la labor de mostrar los datos correctamente y agrupar métodos, quedando así más claro y sencillo, se utiliza una página auxiliar, que es totalmente transparente para el usuario, y que sin embargo hace más legible y claro el código.

Por tanto cuando un usuario ejecuta una consulta, la página principal envía los datos de la consulta, vía URL, a la página auxiliar que es la encargada de llamar al método correspondiente de la base de datos. La información devuelta por los métodos, no tiene formato ninguno, por tanto, es devuelta a la página auxiliar, donde se le dará el correcto formato para que sea mostrada en la página principal.

La página auxiliar contienes diferentes scripts, encargados de realizar diferentes cometidos:

1. Controlar que los datos que introduce el usuario para consultar en la aplicación sean correctos.

2. Realizar llamadas a los métodos, dependiendo de los datos introducidos por el usuario.

3. Mostrar los valores que devuelven los métodos anteriores, siguiendo el estilo de la página.

En todos los métodos, los valores que devuelven, son tratados en scripts, donde se les da un formato y donde es más fácil cambiárselo si se desease en un futuro.

En los métodos no se contempla el caso en el que una columna, puerta o cajón introducido no exista, ya que en todos los casos, en la aplicación, los datos no son introducidos a mano por el usuario, sino que los selecciona de un desplegable (combo box), en el cual si están todos los cajones, puertas y columnas que almacena la base de datos. La elección del desplegable es muy correcta porque evita hacer casos especiales en los métodos para contemplar errores como la inexistencia o la incorrecta introducción del nombre de columnas, cajones o puertas.

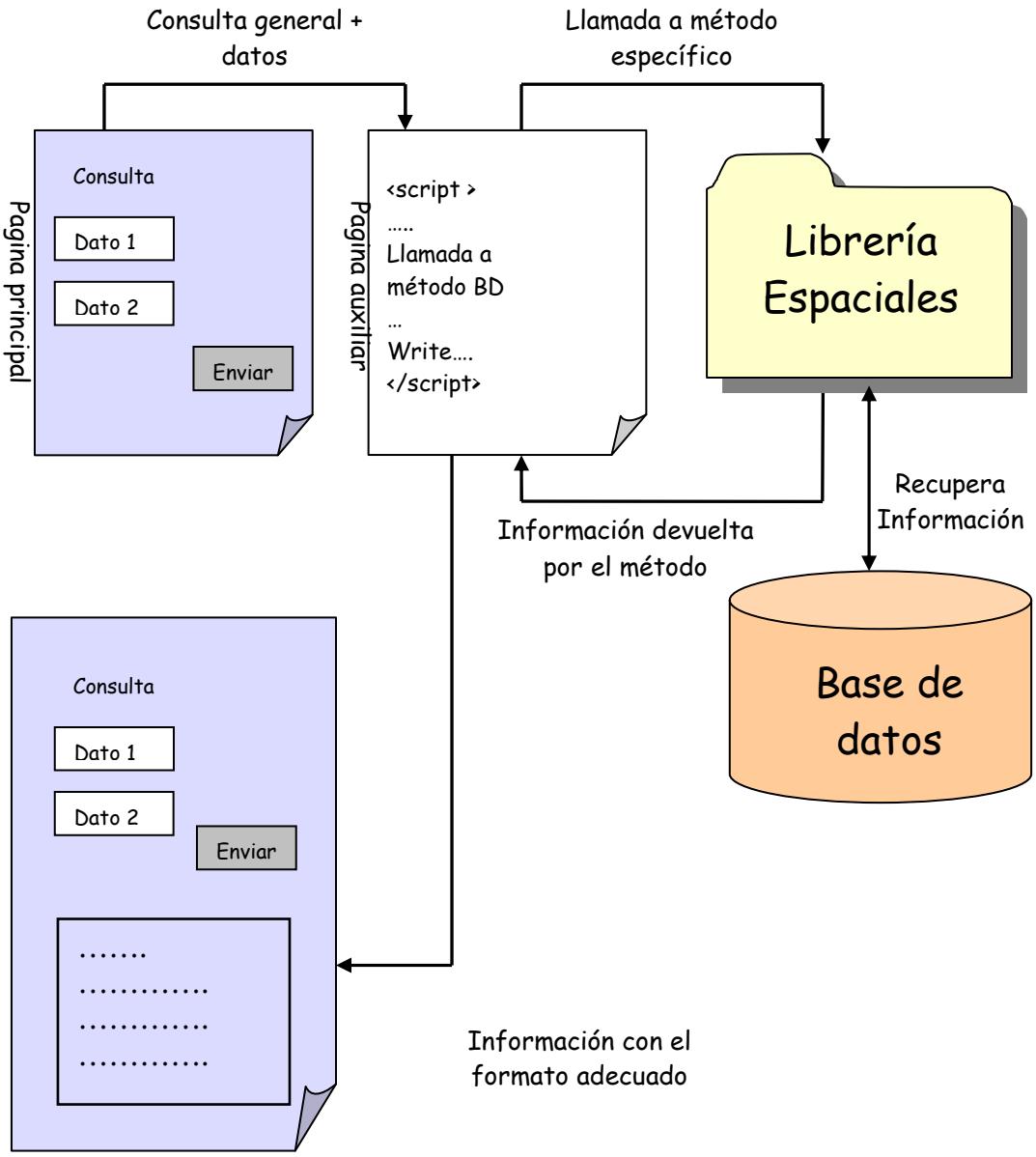


Figura 32: Tratamiento de las consultas

Información medicamentos: Muestra información de los medicamentos almacenados en la base de datos.

```
ClassMethod infoMed(cn As %Float) As %ObjectHandle
{
	Set rset = ##class(%ResultSet).%New()
	set l = "SELECT * FROM Almacen.Med where Med.codnac =? "
	Do rset.Prepare(l)
	Do rset.Execute(cn)
	do rset.Next()
	set id = rset.GetData(1)
	set med = ##class(Almacen.Med).%OpenId(id)
	do med.%Close()
	quit med
}
```

Éste método es de un medicamento. Cada método ejecuta una consulta dinámica (DYNAMIC SQL, explicado en la sección 4.1.5) y devuelve el identificador del objeto del que se busca información (en este caso de un medicamento). Como se ha explicado más arriba, en el script de la página CSP se tratará correctamente el valor devuelto, para darle un formato más adecuado.

Posición de un medicamento: Muestra la posición de los medicamentos almacenados en la base de datos. Devuelve el punto en el que se encuentra el medicamento.

```
ClassMethod PosMed(cn As %Float) As %ObjectHandle
{
	Set rset = ##class(%ResultSet).%New()
	set l = "SELECT * FROM Almacen.Med where Med.codnac =? "
	Do rset.Prepare(l)
	Do rset.Execute(cn)
	do rset.Next()
	set sit = rset.Data("situacion")
	Set pos = ##class(Espaciales.punto).%OpenId(sit)
	do pos.%Close()
	quit pos
}
```

Medicamento por principio activo: Muestra los medicamentos que tiene asociados un principio activo, es decir, que en su composición tienen ese principio activo. Recibe como parámetro el nombre del principio activo en cuestión. Llama a un método que le devuelve el identificador del principio activo. Con éste dato, comprueba si el principio activo tiene medicamentos asociados, ejecutando una consulta. Si es así, almacena en una lista el medicamento (su código nacional, nombre, elementos y forma farmacéutica), que será la que devuelva el método para su posterior tratamiento en la página CSP.

```
ClassMethod medPioActivo(nom As %String) As %ListOfDataTypes
{
	set idPpio = ..compruebaPpioAct(nom)
	Set rset = ##class(%ResultSet).%New()
	set l = "SELECT * FROM Almacen.compuestoDe where ppioAct =?"
```

```
        Do rset.Prepare(l)

        Do rset.Execute(idPpio)

        set cont = 1

        set list = ##class(%Library.ListOfDataTypes).%New()

        while ( rset.Next())
        {


                set m = rset.Data("Medicamento")

                set med = ##class(Almacen.Med).%OpenId(m)
                        do list.Insert(med.cn)
                        do list.Insert(med.nombre)
                        do list.Insert(med.elementos)
                        do list.Insert(med.formaFarmac)

}

        quit list
}
```

Composición de un medicamento: Muestra el nombre del medicamento y los principios activos, asociados al mismo, con sus dosis. Recibe como parámetro el código nacional del medicamento. Llama a un método que devuelve el identificador del medicamento. Con él comprueba todos los principios activos que tiene asociados e introduce en la lista su nombre y su dosis. El método devuelve la lista para su posterior tratamiento en la página CSP.

```
ClassMethod composicion(cn As %Float) As %ListOfDataTypes
{

        set idMed = ..compruebaMed(cn)

        Set rset = ##class(%ResultSet).%New()

        set l = "SELECT * FROM Almacen.compuestoDe where medicamento =?"

        Do rset.Prepare(l)

        Do rset.Execute(idMed)
```

```
        set cont = 1

        set list = ##class(%Library.ListOfDataTypes).%New()

        set med = ##class(Almacen.Med).%OpenId(idMed)

        do list.Insert(med.nombre)//Primer dato el nombre del medicamento

        while ( rset.Next())
        {

                set ppio = rset.Data("PpioActivo")

                set ppioAct = ##class(Almacen.PpioAct).%OpenId(ppio)

                        do list.Insert(PpioAct.nombre)
                        do list.Insert(PpioAct.dosis)


}

        quit list
}
```

Tipo de medicamento en zona: Muestra los medicamentos de una determinada forma farmacéutica que están guardados en un determinado cajón. Por parámetro se introduce el tipo de forma farmacéutica y el nombre del cajón. Se obtiene la posición de los medicamentos y se comprueba que estén dentro del cubo que representa el cajón. Si están dentro, se comprueba que sea un medicamento que tenga la forma farmacéutica deseada; si es así se introduce en la lista que devuelve el método.

Este método es para cajones, existe uno similar para niveles de puertas.

```
ClassMethod medEnZona(tipo, cajon As %String) As %ListOfDataTypes
{

        //Busca el cajón

        set act = 0

        set idC = ##class(Almacen.Cajon).buscaCajon(cajon)
```

```
If (idC'="")
{
        //Se abre el cajon
        set caj = ##class(Almacen.Cajon).%OpenId(idC)

        //Se abre la posición del cajón

        set x = caj.pos

        set pos = ##class(Espaciales.punto).%OpenId(x)

        set alto = caj.alto//Se abre el alto del cajón

        set ancho = caj.ancho//Se abre el ancho del cajón

        set fondo = caj.fondo//Se abre el fondo del cajón

       //Obtengo el cubo del cajon con sus parámetros

set cubo = ##class(Espaciales.cubo).devuelveCubo(pos,alto,ancho,fondo)

        //Busco los medicamentos

        Set rset = ##class(%ResultSet).%New()

        set l = "SELECT * FROM Almacen.Med "

        Do rset.Prepare(l)

        Do rset.Execute(idC)

        set list = ##class(%Library.ListOfDataTypes).%New()

        While (rset.Next())
        {

        set idMed = rset.GetData(1)

        set sit = rset.Data("posicion")

        set pto = ##class(Espaciales.punto).%OpenId(sit)

        set formaFar = rset.GetData(5)

set sol = ##class(Espaciales.cubo).puntoDentroCubo(pto.x,pto.y,pto.z,cubo)

        If (sol = 0)//El punto está contenido en el cubo
                {

                If (formaFar = tipo)  //La forma farmacéutica del medicamento
                                      //es igual a la de la consulta
                        {
                        set act = 1

                        set nombre = rset.GetData(3)
                        set cn = rset.GetData(2)
                        set elementos = rset.GetData(4)
```

```
                    do list.Insert(idMed) //El id del medicamento
                    do list.Insert(cn) //Codigo nacional
                    do list.Insert(nombre) //Nombre de la sala
                    do list.Insert(elementos) //Elementos
                    do list.Insert(sit)//posición del medicamento
                    do pto.%Close()

                                        }
                                    }
                                }
                    }

                    quit list
          }
```

Información de una zona: Muestra información de una zona, en este caso de un cajón. Se buscan los sectores correspondientes al cajón dado y se guarda de cada uno de ellos su posición y su contenido en una lista. El contenido puede ser "Vacío" o un medicamento, del que se guarda en la lista su código nacional, su nombre sus elementos, sus forma farmacéutica y sus existencias.

```
ClassMethod infoZona(cajon As %String) As %ListOfDataTypes
{
        //Busca el cajón

        set act = 0

        set idC = ##class(Almacen.Cajon).buscaCajon(cajon)
        If (idC'="")
        {
                //Se abre el cajon
                set caj = ##class(Almacen.Cajon).%OpenId(idC)

                // buscar en todos los sectores del cajon

                Set l = "SELECT * FROM Almacen.sector INNER JOIN (Almacen.calle
inner join almacen.cajon on (cajon.nombre = ?) and (calle.id = cajon.calle)) on
calle.sector=sector.id"

                Do rset.Prepare(l)

                Do rset.Execute(cajon)
```

```
                set list = ##class(%Library.ListOfDataTypes).%New()

                While (rset.Next())
                {

                        set idSect = rset.GetData(1)

                        set nom = rset.Data("nombre")

                        set pos = rset.Data("posicion")

                        do list.Insert(nom) //El nombre del sector

                        do list.Insert(pos) //La posición del sector


                        //Si el sector está vacío

                        If (##class(Almacen.sector).%EstaVacio(idSect) = 1)
                        {
                                do list.Insert("Vacío")
                        }

                        Else //se busca el medicamento guardado
                        {
                             set idMed = ##class(Almacen.relacion).buscaMed(idSect)

                                set medi = ##class(Almacen.Med).%OpenId(idMed)

                                do list.Insert(medi.cn)

                                do list.Insert(medi.nombre)

                                do list.Insert(medi.elementos)

                                do list.Insert(medi.formaFar)

                                do list.Insert(medi.existencias)

                                do medi.%Close()
                        }
                }

        quit list
        }
}
```

Zonas vacías: Muestra las zonas vacías de una columna dada. Por parámetro se introduce el nombre de la columna. Se buscan primero todas las puertas y de estas todos sus niveles para ver los que están vacíos e introducirlos en una lista que se devuelve. Lo mismo ocurre para los cajones de la columna.

```
ClassMethod zonasVacias(columna As %String) As %ListOfDataTypes
{

	//Busca la columna

	set act = 0

	set idC = ##class(Almacen.Columna).buscaColumna(columna)
	If (idC'="")
	{
		//Se abre la columna
		set col = ##class(Almacen.Columna).%OpenId(idC)



		// buscar en todos las puertas de la columna

		Set l = "SELECT * FROM Almacen.puerta INNER JOIN
(Almacen.columna inner join almacen.puerta on (columna.nombre = ?) and
(puerta.id = columna. puerta)) on columna. puerta = puerta.id"

		Do rset.Prepare(l)

		Do rset.Execute(columna)

		set list = ##class(%Library.ListOfDataTypes).%New()

		While (rset.Next())
		{

			set idPuer = rset.GetData(1)

			//Se abre el cajon

			set puer = ##class(Almacen.Puerta).%OpenId(idPuer)


		// buscar en todos los niveles de la puerta

		Set l1 = "SELECT * FROM Almacen.nivel INNER JOIN (Almacen.puerta
inner join almacen.nivel on (puerta.nombre = ?)) on nivel.id=puerta.nivel"

		Do rset.Prepare(l1)

		Do rset.Execute(puerta.nombre)
```

```
While (rset.Next())
{
        set idNiv = rset.GetData(1)

        //Si el nivel está vacío

        If (##class(Almacen.nivel).%EstaVacio(idNiv) = 1)
        {
                set nom = rset.Data("nombre")

                set pos = rset.Data("posicion")

                do list.Insert(nom) //El nombre del nivel

                do list.Insert(pos) //La posición del nivel
        }
}}



// buscar en todos los cajones de la columna

Set l = "SELECT * FROM Almacen.cajon INNER JOIN
(Almacen.columna inner join almacen.cajon on (columna.nombre = ?) and (cajon.id
= columna.cajon)) on columna.cajon=cajon.id"

Do rset.Prepare(l)

Do rset.Execute(columna)

While (rset.Next())
{

        set idCaj = rset.GetData(1)

        //Se abre el cajon

        set caj = ##class(Almacen.Cajon).%OpenId(idCaj)


// buscar en todos los sectores del cajon

Set l1 = "SELECT * FROM Almacen.sector INNER JOIN (Almacen.calle
inner join almacen.cajon on (cajon.nombre = ?) and (calle.id = cajon.calle)) on
calle.sector=sector.id"

Do rset.Prepare(l1)

Do rset.Execute(caj.nombre)

While (rset.Next())
{
```

```
                    set idSect = rset.GetData(1)


                    //Si el sector está vacío

                    If (##class(Almacen.sector).%EstaVacio(idSect) = 1)
                    {
                            set nom = rset.Data("nombre")

                            set pos = rset.Data("posicion")

                            do list.Insert(nom) //El nombre del sector

                            do list.Insert(pos) //La posición del sector
                    }
            }}

        quit list
        }
}
```

Zonas incompletas: Muestra las zonas que tienen guardado un medicamento, pero que les queda volumen vacío. El volumen vacío debe ser como mínimo el que se da por parámetro. Se devuelve una lista con los sectores a los que les ocurre esto y el volumen que les queda sin ocupar.

```
ClassMethod zonasIncompletas(cajon As %String, volumen As %Float) As
%ListOfDataTypes
{

        //Busca el cajón

        set act = 0

        set idC = ##class(Almacen.Cajon).buscaCajon(cajon)
        If (idC'="")
        {
                //Se abre el cajon
                set caj = ##class(Almacen.Cajon).%OpenId(idC)

                // buscar en todos los sectores del cajon

                Set l = "SELECT * FROM Almacen.sector INNER JOIN (Almacen.calle
inner join almacen.cajon on (cajon.nombre = ?) and (calle.id = cajon.calle)) on
calle.sector=sector.id"

                Do rset.Prepare(l)

                Do rset.Execute(cajon)
```

```
set list = ##class(%Library.ListOfDataTypes).%New()

While (rset.Next())
{
        set idSect = rset.GetData(1)
        set nom = rset.Data("nombre")

        //Si el sector no está vacío

        If (##class(Almacen.sector).%EstaVacio(idSect) = 0)
        {
          //Se abre la posición del sector

                set x = rset.Data("posicion")

                set pos = ##class(Espaciales.punto).%OpenId(x)

                set alto = rset.Data("alto")//Se abre el alto del sector

                set ancho = rset.Data("ancho")//Se abre el ancho

                set fondo = rset.Data("fondo")//Se abre el fondo

        //Obtengo el cubo del sector con sus parámetros
set cuboS = ##class(Espaciales.cubo).devuelveCubo(pos,alto,ancho,fondo)

        //se busca el medicamento guardado

              set idMed = ##class(Almacen.relacion).buscaMed(idSect)

                set medi = ##class(Almacen.Med).%OpenId(idMed)

                set exist = rset.Data("existencias")

                set altom = rset.Data("alto")//Se abre el alto del medic

                set anchom = rset.Data("ancho")//Se abre el ancho

                set fondom = rset.Data("fondo")//Se abre el fondo

        // obtengo el volumen que ocupa el medicamento
set volMed = ##class(Almacen.Med).calculaVol(exist,alto,ancho,fondo)

        // obtengo el volumen que ocupa el sector

        set volSect = ##class(Espaciales.Cubo).calculaVol(cuboS)

        set volVacio = ..restaVol(volMed,volSect)

        If ((volMed<volSect)and(volVacio>=volumen))
        {
                do list.Insert(pos) //La posición del sector

                do list.Insert(nom) //El nombre del sector
```

```
                        do list.Insert(volVacio) //Volumen vacío

                        do list.Insert("cm3")
                }

                do medi.%Close()
                }
        }

    quit list
    }
}
```

Tamaño de una zona: Muestra el tamaño de la zona en la que se encuentran las coordenadas introducidas. El tamaño total, el ocupado y el vacío. Estos datos los devuelve en una lista.

```
ClassMethod tamanoZona(x,y,z As %Float) As %ListOfDataTypes
{
    set pos = ##class(Espaciales.punto).obtenerPunto(x,y,z)

    Set l = "SELECT * FROM Almacen.Sector"

    Do rset.Prepare(l)

    Do rset.Execute()

    set list = ##class(%Library.ListOfDataTypes).%New()

    While (rset.Next())
    {
                set idSect = rset.GetData(1)

                set nom = rset.Data("nombre")

                 //Se abre la posición del sector

                set x = rset.Data("posicion")

                set pos = ##class(Espaciales.punto).%OpenId(x)

                set alto = rset.Data("alto")//Se abre el alto del sector

                set ancho = rset.Data("ancho")//Se abre el ancho

                set fondo = rset.Data("fondo")//Se abre el fondo

                //Obtengo el cubo del sector con sus parámetros
```

```
set cuboS = ##class(Espaciales.cubo).devuelveCubo(pos,alto,ancho,fondo)
        set volSect = ##class(Espaciales.Cubo).calculaVol(cuboS)
        //Si el sector no está vacío
        If (##class(Almacen.sector).%EstaVacio(idSect) = 0)
        {
        //se busca el medicamento guardado
            set idMed = ##class(Almacen.relacion).buscaMed(idSect)
              set medi = ##class(Almacen.Med).%OpenId(idMed)
              set exist = rset.Data("existencias")
              set altom = rset.Data("alto")//Se abre el alto del medic
              set anchom = rset.Data("ancho")//Se abre el ancho
              set fondom = rset.Data("fondo")//Se abre el fondo
        // obtengo el volumen que ocupa el medicamento
    set volMed = ##class(Almacen.Med).calculaVol(exist,alto,ancho,fondo)
        // obtengo el volumen que ocupa el sector
        set volVacio = ..restaVol(volMed,volSect)
              do list.Insert(pos) //La posición del sector
              do list.Insert(nom) //El nombre del sector
              do list.Insert("Tamaño total")
              do list.Insert(volSect)
              do list.Insert("Espacio vacío")
              do list.Insert(volVacio) //Volumen vacío
              do list.Insert("Espacio ocupado")
              do list.Insert(volMed)
              }
        do medi.%Close()
        Else
        {
              do list.Insert(pos) //La posición del sector
              do list.Insert(nom) //El nombre del sector
              do list.Insert("Tamaño total")
```

```
                    do list.Insert(volSect)

                    do list.Insert("Espacio vacío")

                    do list.Insert(volSect) //Volumen vacío

                    do list.Insert("Espacio ocupado")

                    do list.Insert("0")
                    }

        quit list
        }
}
```

5.3.2 Introducción de datos

Otro de los objetivos de la aplicación es que la introducción de información a la base de datos fuese sencilla. Por ello, la aplicación consta de una parte que se encarga de esto, con una interfaz sencilla y agradable para el usuario que hace más amena la tarea de introducción de datos.

Así, los datos se introducen como si de un formulario se tratase. Se ha definido una página CSP, en la que existe una lista con toda la posible información a introducir: medicamento, principio activo, cajón… etc. Seleccionando el nuevo elemento a introducir, ésta página llama a otra CSP (cada clase tiene asociada una página CSP para la introducción de datos) donde se introducen los datos. Dentro de ésta página, primero se controlan que los datos que se introducen sean correctos y si esto se cumple, se realiza una llamada al método *'nuevo´* que toda clase contiene, creando así un objeto nuevo.

Por tanto, cada clase tiene método propio para crear nuevos objetos de dicha clase. A su vez, cada clase tiene una página CSP, en la que se realiza la introducción de datos como si de un formulario se tratase, y que internamente realiza una llamada al método 'nuevo', para almacenar la información

introducida en la página CSP. Por último existe una página CSP general, que enlaza y une a todas las páginas, facilitando así la introducción de los datos.

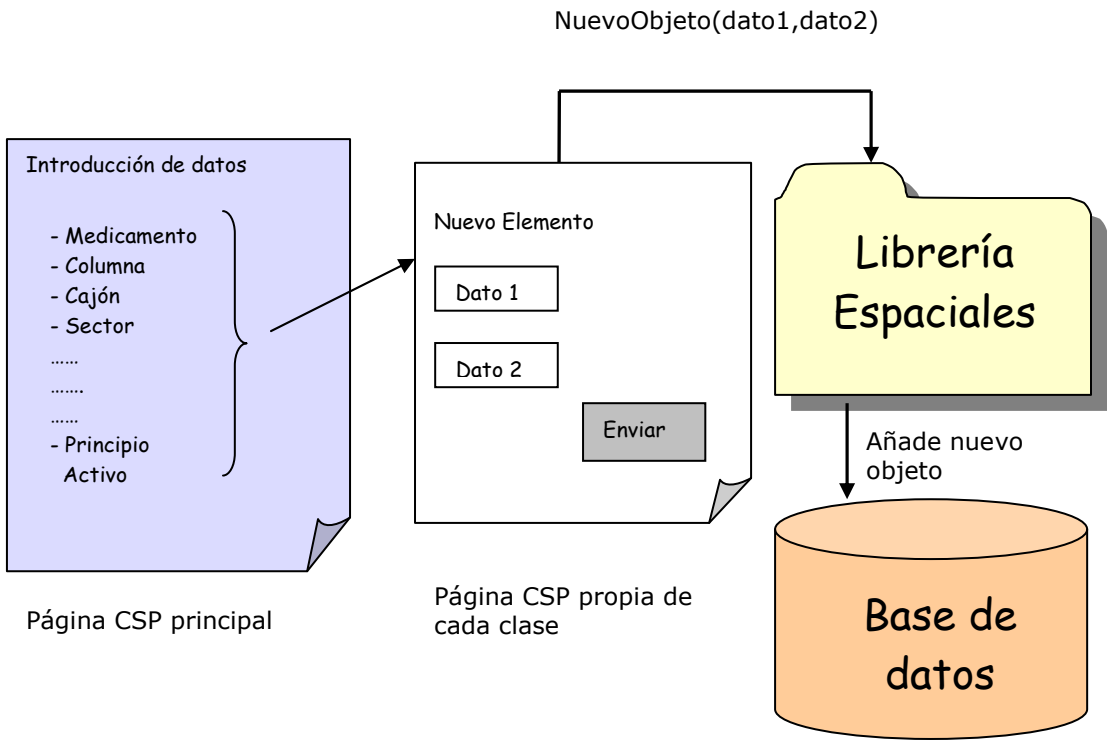


Figura 33: Tratamiento de la introducción de datos

Para los objetos que contienen información de tipo espacial, primero se almacena la información espacial y después la no espacial. Este proceso es transparente al usuario. Lo que permite esto es que al almacenar la información del nuevo objeto, no se introduzcan gran cantidad de números, sino que se pasa al método el identificador del objeto espacial que lo representa.

Algunos ejemplos son:

Nuevo Sector: Se crea un nuevo sector con la información que recibe el método como parámetro. Comprueba primero que no exista un sector con la misma posición y que con el fondo, dado por parámetro, no invade otro sector

existente ni se sale de la calle en la que está contenido. Si lo anterior no se cumple, crea un nuevo sector. La situación geográfica de un sector, es un punto, definido por tanto por unas coordenadas. En lugar de éstas, se introduce el identificador del punto, en el que se almacenan las coordenadas del sector a introducir (sit). La columna, el cajón y la calle a la que pertenece deben estar introducidos con anterioridad, si no es así habrá que hacerlo antes. Por tanto se pasa la posición del sector, y el método busca el identificador de la calle para relacionar así calle y sector.

```
ClassMethod nuevoSector(fondo, calle, sit As %Integer) As %ObjectIdentity
{
	//Se comprueba que no exista un sector con la misma posicion
	set x = ##class(Almacen.Sector).compruebaSectorUnico(calle,sit)

	if (x'= "")
	{
		&js<alert("Existe un sector en el mismo sitio")>
		quit ""

	}
	else
	{
	   //Se comprueba que todo el sector pertenece a la calle
	   set x = ##class(Almacen.Sector).compruebaFondoSector(fondo, calle,sit)

	   if (x'= "")
	   {
			&js<alert("El sector no cabe en la calle")>
			quit ""

	   }
	   else

		set c = ##class(Almacen.Sector).%New()
		set c.fondo = fondo

		set idCalle = ##class(Almacen.calle).buscaCalle(calle)
		set p = ##class(Almacen.calle).%OpenId(idCalle)
		set c.calle = p
		set c.situacion = sit
		do c.%Save()
		set id = c.%Id()
		do c.%Close()
		do p.%Close()
		quit id
	   }
	}
}
```

Para asignar las posiciones a los medicamentos:

Nuevo SeGuardaSector: método para indicar la posición de un medicamento. Primero comprueba que la posición esté vacía y sea lo suficientemente grande para albergar todas las existencias del medicamento. Si es así se "guarda" el medicamento en esa posición.

```
ClassMethod nuevoSeGuardaSector(med, sit As %Integer) As %ObjectIdentity
{

	//Se comprueba que la posición este vacia
	set idPos = ##class(Almacen.Sector).compruebaSector(sit)
	set x = ##class(Almacen.Guardar).posicionVacia(idPos)

	If (x '= "")
	{
		&js<alert("La posción está ocupada")>

	}

	else
	{
	   set idMed = ##class(Almacen.Med).compruebaMed(med)
	   set x = ##class(Almacen.Guardar).compruebaCapacidad(idPos,idMed)

	   If (x '= "")
	   {
		&js<alert("La posción es pequeña")>

	   }

	   else
	   {

		set sg = ##class(Almacen.seGuarda).%New()
		do sg.%Save()

		set id = sg.%Id()

		set medicamento = ##class(Almacen.Med).%OpenId(idMed)
		set posicion = ##class(Almacen.Sector).%OpenId(idPos)

		set sg.medicamento = medicamento
		set sg.posicion = posicion
		do sg.%Save()
		do sg.%Close()
		quit id
	   }
	}
}
```

Objetos sin información de tipo espacial:

Nuevo PpioAct: Introduce un nuevo principio activo. Comprueba antes, que no exista un principio activo con el mismo nombre y con la misma dosis (compruebaPpioAct) y si no se da el caso almacena el nuevo objeto.

```
ClassMethod nuevoPpioAct(nom, dosis As %String) As %ObjectIdentity
{
	//Se comprueba que no exista un principio activo igual
	set x = ##class(Almacen.PioAct).compruebaPpioAct(nom, dosis)

	if (x'= "")
	{
		&js<alert("Ya existe el principio activo en la base de datos")>
		quit ""

	}
	else
	{

		set ppio = ##class(Ocio.PpioAct).%New()

		set ppio.nombre = nom

		set ppio.dosis = dosis

		do ppio.%Save()


		set id = ppio.%Id()

		do ppio.%Close()


		quit id



	}
}
```

Nota: no se muestran todos los métodos 'nuevo' de los distintos objetos, porque son todos muy similares. El comportamiento y la forma son muy parecidos en todos los casos.

5.4 Manual de Usuario

Esta aplicación tiene dos partes: una que permite consultar los medicamentos que tenemos y donde se guardan y otra que permite la introducción de datos en la misma. A continuación se describe como usar ambas.

5.4.1 Guardamed

Ésta es la parte de la aplicación en la que se pueden realizar diferentes consultas sobre qué medicamentos hay en el almacén y dónde.

Para acceder a la página principal, se abre el navegador y en la barra de dirección se escribe:

http://localhost:8972/csp/user/medicamentos.csp

A continuación se muestra la página principal. En ella se observan dos elementos

- Menú
- Consultas

El menú consta de dos opciones: Medicamentos y Almacén. Cuando se selecciona una opción del menú, aparecen nuevas consultas relacionadas con la opción elegida.

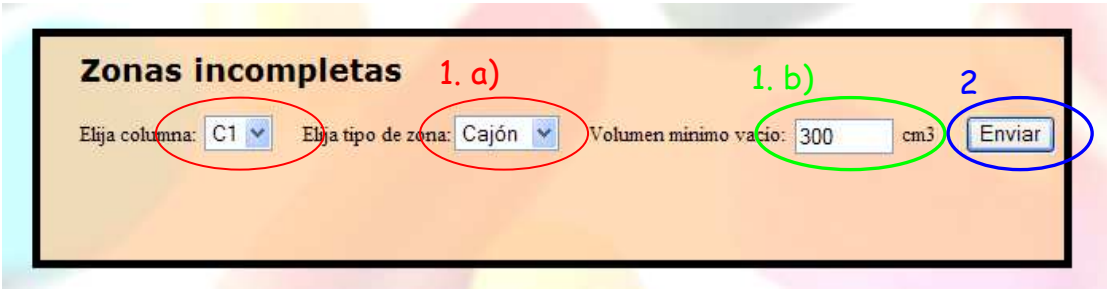
Las distintas consultas, aparecen en recuadros. Separando así las diferentes consultas de una forma visual.

Las consultas, tienen un comportamiento similar:

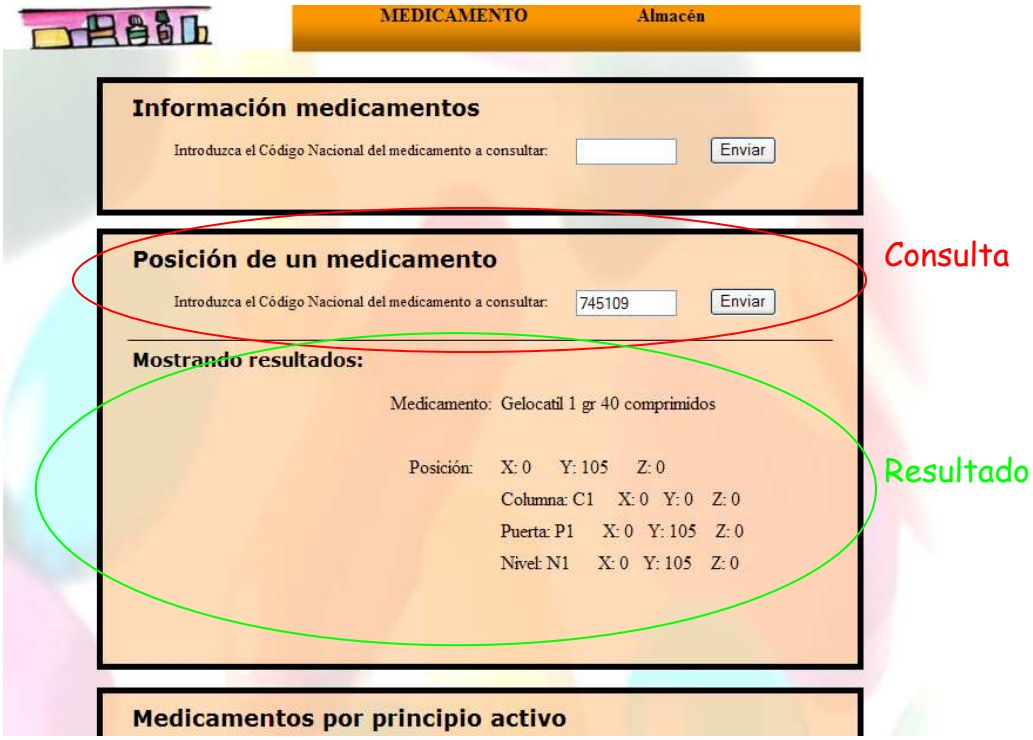
1. Introducción de datos. Puede ser:

   a) Seleccionar un elemento de un desplegable
   b) Introducir datos manualmente

2. Pulsar Enviar



y a continuación, debajo de la consulta, se muestra el resultado de la misma:



En todas las consultas, si se pulsa al botón Enviar y faltan datos por introducir, aparecerá un mensaje de error, que nos alerta del mismo. El error en cuestión es el siguiente:

5.4.1.1 Medicamentos

Al pinchar sobre la opción del menú Medicamentos, aparecen cinco consultas distintas. Nótese que sobre la opción del menú que se ha seleccionado aparece en mayúsculas, mientras que la otra lo hace en minúsculas.

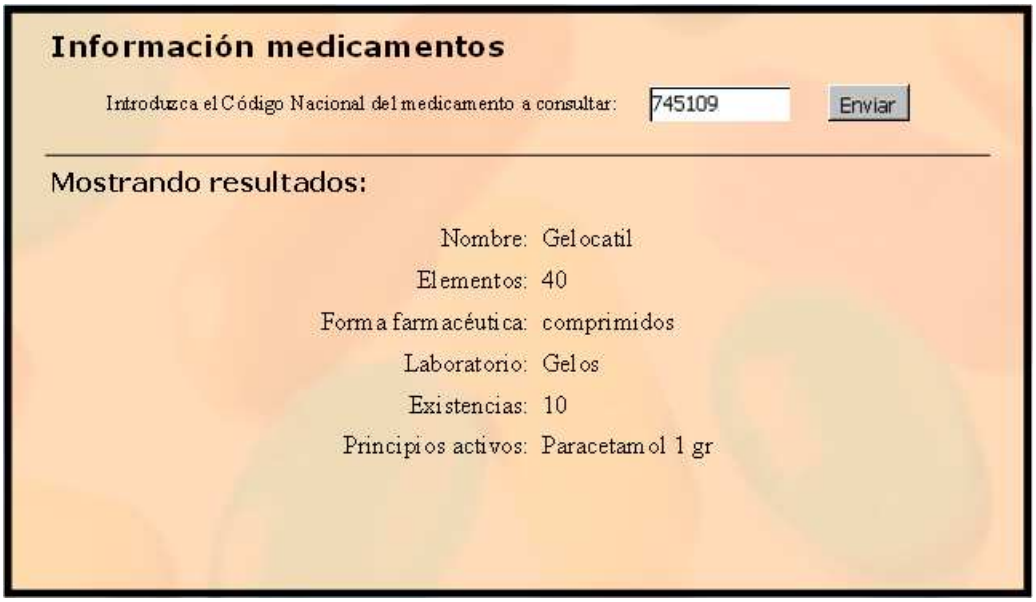
Información Medicamento

Muestra la información de un medicamento seleccionado. Los pasos a seguir para esto son los siguientes:

1. Introducir el código nacional de la especialidad farmacéutica sobre la que queremos información. El código nacional debe ser un número de seis cifras, sino aparecerá un mensaje de error.

2. Pulsamos enviar.

Si el código introducido no existe nos dará un mensaje de error. Si existe aparecerá en la pantalla la información acerca del medicamento.



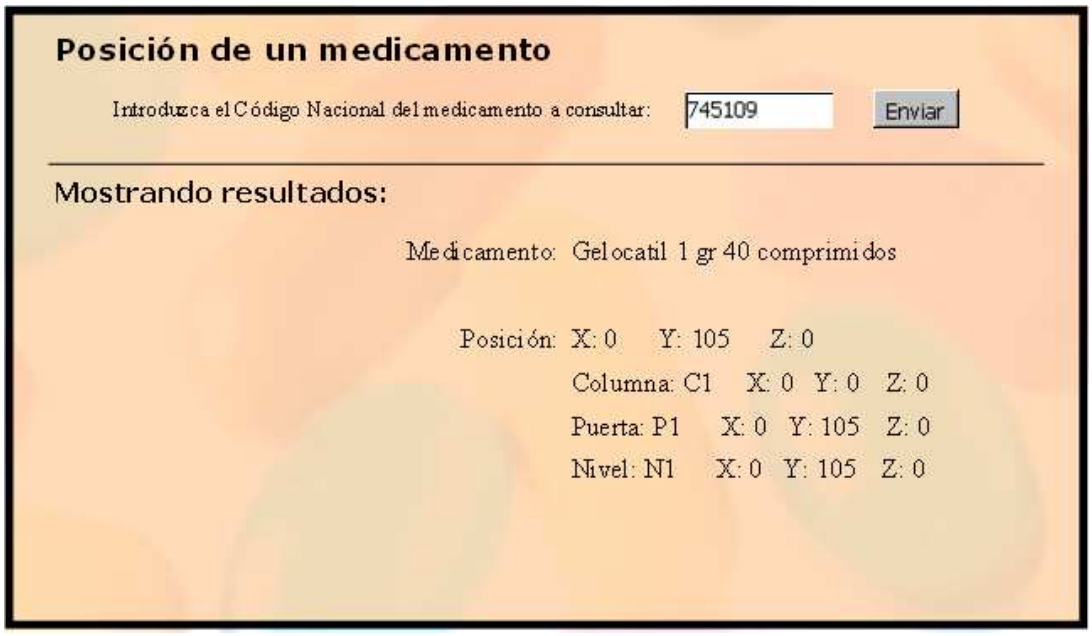
Posición de un medicamento

Muestra los medicamentos guardados en una zona dada. Los pasos a seguir para esto son los siguientes:

1. Introducir el código nacional de la especialidad farmacéutica sobre la que queremos información. El código nacional debe ser un número de seis cifras, sino aparecerá un mensaje de error.

2. Pulsamos enviar.

Si el código introducido no existe nos dará un mensaje de error. Si existe aparecerá en la pantalla la posición en el almacén.



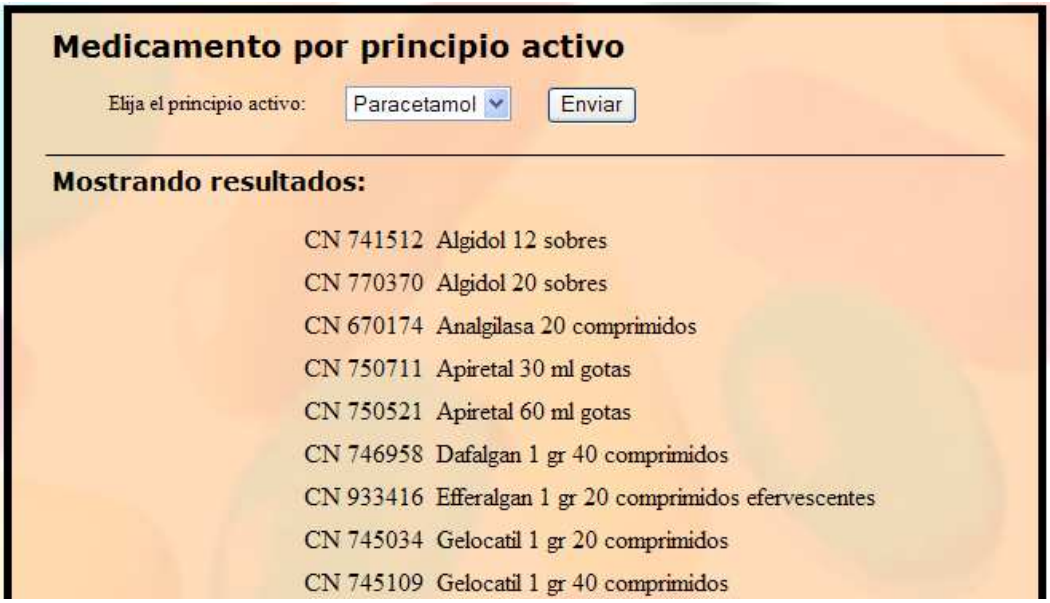
Medicamentos por principio activo

Muestra los medicamentos que contengan el principio activo seleccionado. Los pasos a seguir para esto son los siguientes:

1. Elegir de un desplegable el prinipio activo sobre el que consultar.

2. Pulsamos enviar.

Una vez elegido el principio activo nos muestra los medicamentos que lo contienen.



● ● ●

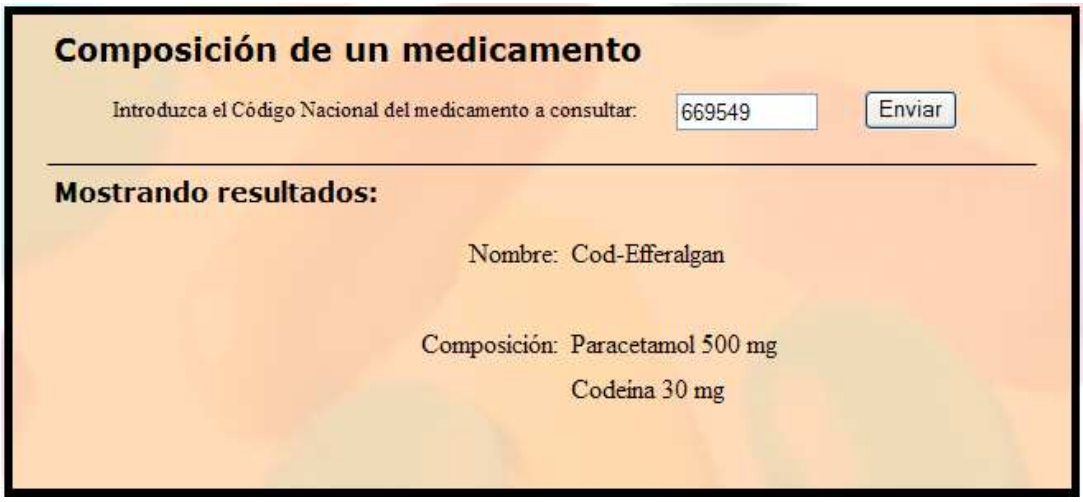
Composición Medicamento

Muestra los principios activos, con las dosis de cada uno, que componen el medicamento consultado. Los pasos a seguir para esto son los siguientes:

1. Introducir el código nacional de la especialidad farmacéutica sobre la que queremos información. El código nacional debe ser un número de seis cifras, sino aparecerá un mensaje de error.

2. Pulsamos enviar.

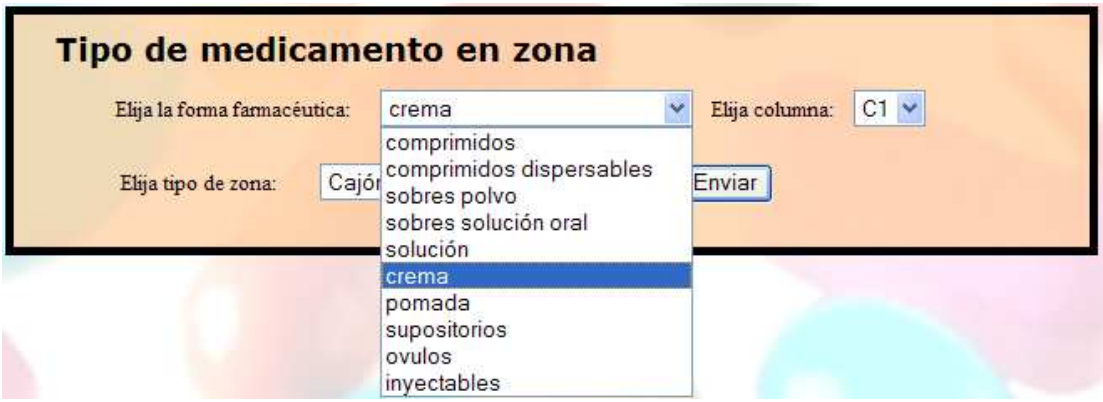
Si el código introducido no existe nos dará un mensaje de error. Si existe aparecerá en la pantalla los principios activos de los que está compuesto el medcamento.



Tipo de medicamento en zona

Muestra los medicamentos de una determinada forma farmacéutica en una zona seleccionada. Los pasos a seguir para esto son los siguientes:

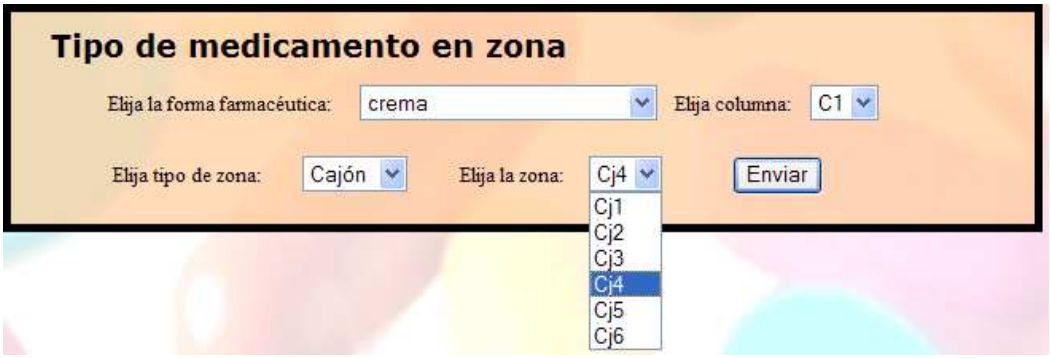
1. Seleccionar la forma farmacéutica.



2. Elegir la columna sobre la que hacer la consulta.

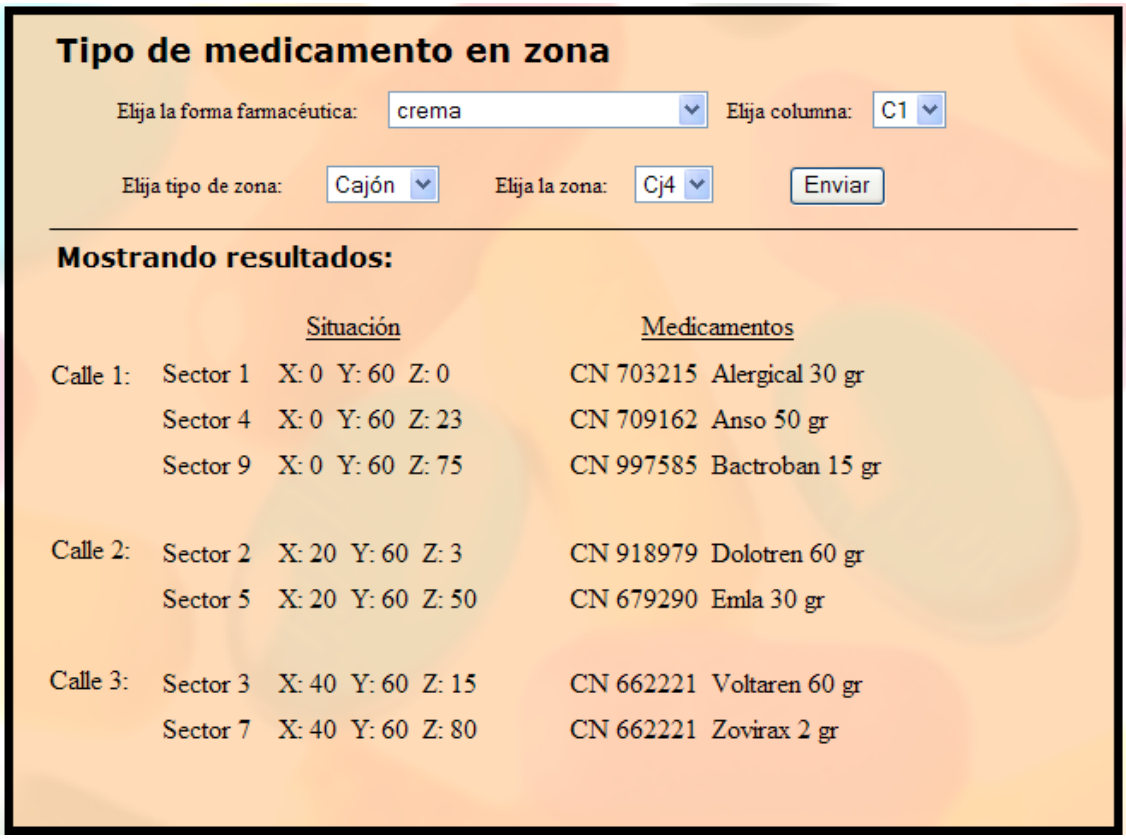
3. Elegir el tipo de zona en la que consultar (cajón o puerta)

4. Al seleccionar el tipo de zona aparece un nuevo desplegable con todas las zonas, de donde se debe seleccionar la zona sobre la que se quiere obtener información.



5. Pulsar enviar.

Entonces aparecerá en pantalla los medicamentos con la forma farmaceutica elegida en la zona seleccionada.



5.4.1.2 Almacén

Al seleccionar la opción del menú Almacén, se abre una nueva página con nuevas consultas. En la parte del menú, ahora la opción que aparece en mayúsculas ALMACÉN.

Información de una zona

Muestra la información de una zona dada. Los pasos a seguir para esto son los siguientes:

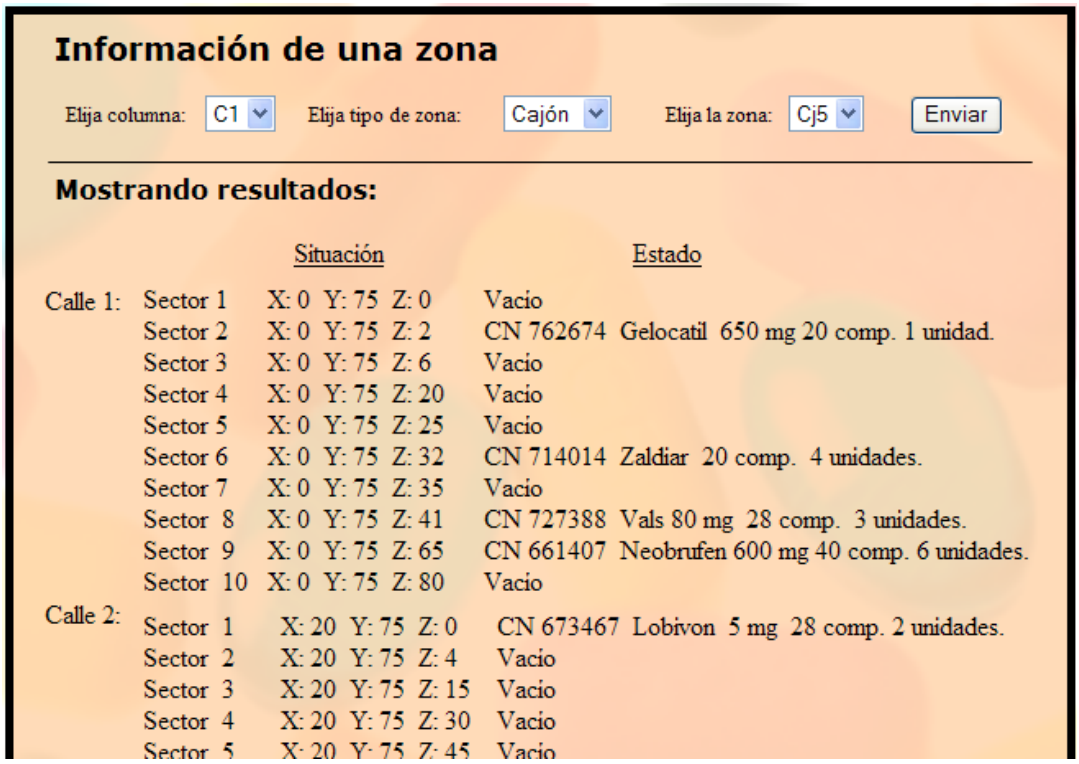
1. Elegir la columna de un desplegable donde aparecen todas las que hay.

2. Elegir, de otro desplegable, si se quiere buscar en una puerta o en un cajón.

3. Elegir, del último desplegable que aparece una vez seleccionada el tipo de zona, la zona concreta sobre la que hacer la consulta.

4. Pulsar enviar.

Entonces aparecerá en pantalla la información relativa a la zona seleccionada.



• • •

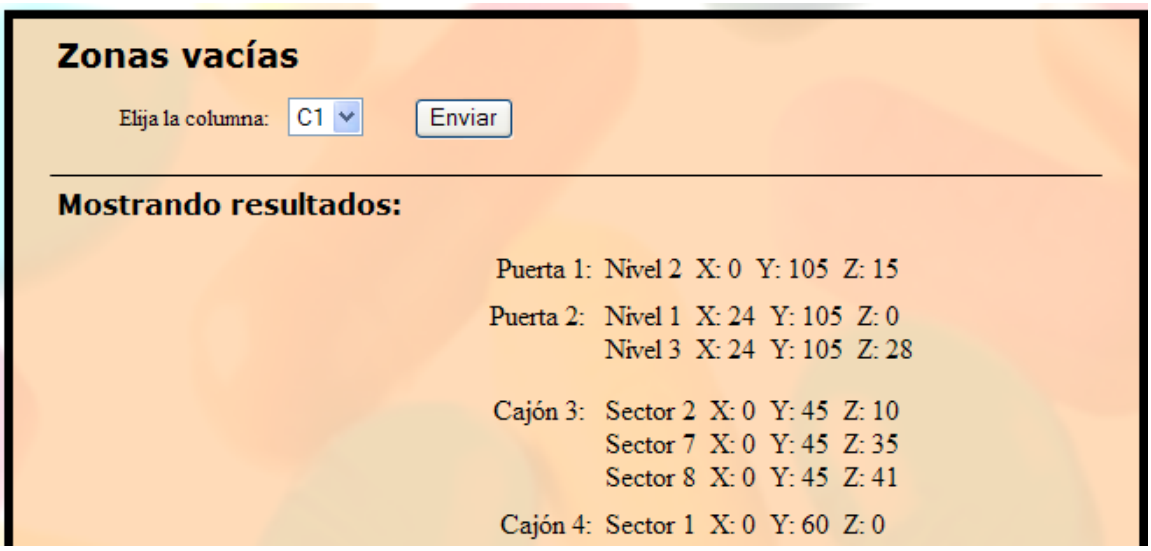
Zonas vacias

Muestra las zonas vacías de una columna dada. Los pasos a seguir para esto son los siguientes:

1. Elegir la columna de un desplegable donde aparecen todas las que hay.

2. Pulsar enviar.

Entonces aparecerá en pantalla las zonas que aún están vacías.

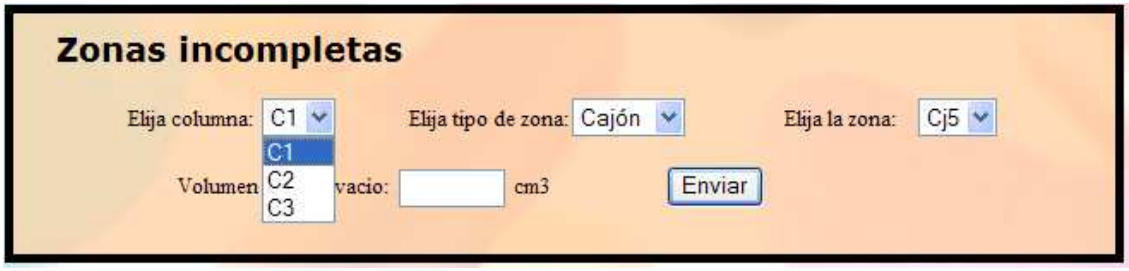


• • •

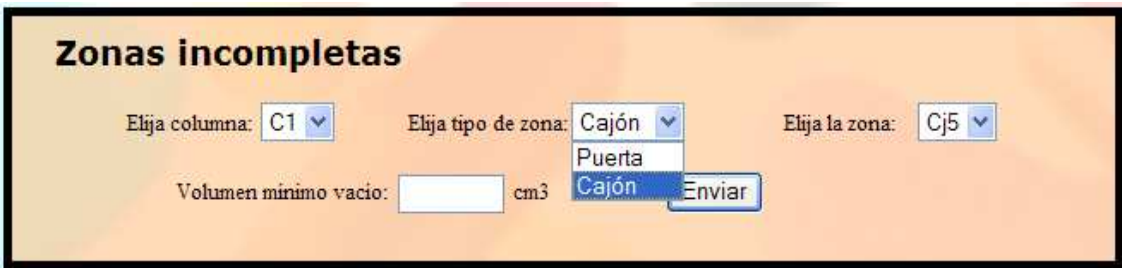
Zonas incompletas

Muestra las zonas incompletas de una columna dada, en las que hay medicamentos guardados, pero queda hueco vacío. El hueco vacío debe ser igual o mayor al que se introduce en la consulta. Los pasos a seguir para esto son los siguientes:

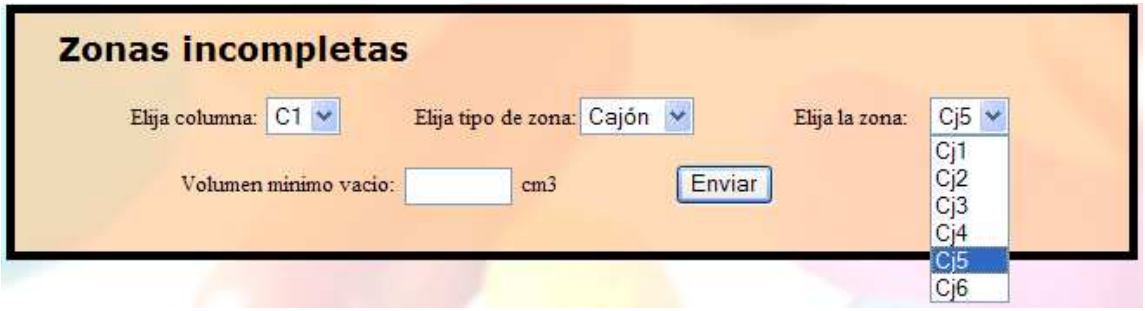
1. Elegir la columna de un desplegable donde aparecen todas las que hay.



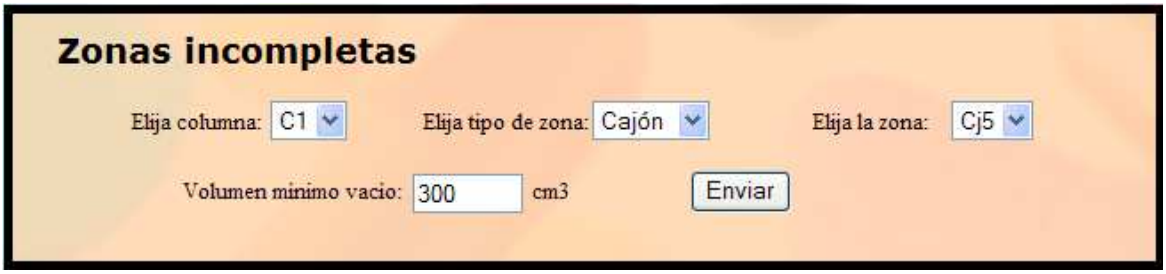
2. Elegir el tipo de zona en la que se quiere hacer la consulta.



3. Elegir, del último desplegable la zona concreta sobre la que hacer la consulta.

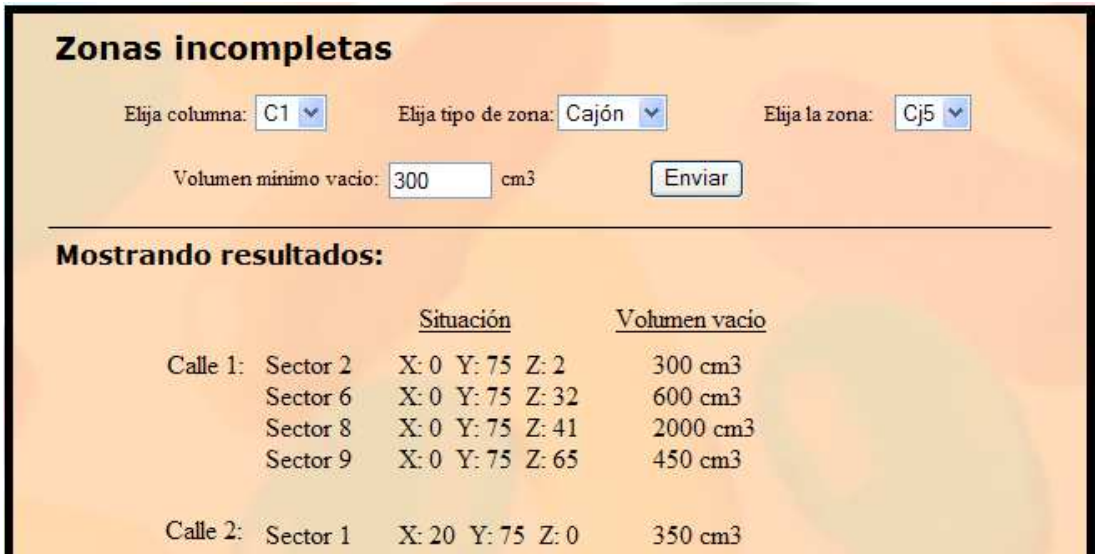


4. Introducir el volumen mínimo que debe quedar vacío aún estando guardado un medicamento.



5. Pulsar enviar.

Entonces aparecerá en pantalla las zonas incompletas con un espacio vacío mayor al volumen dado.

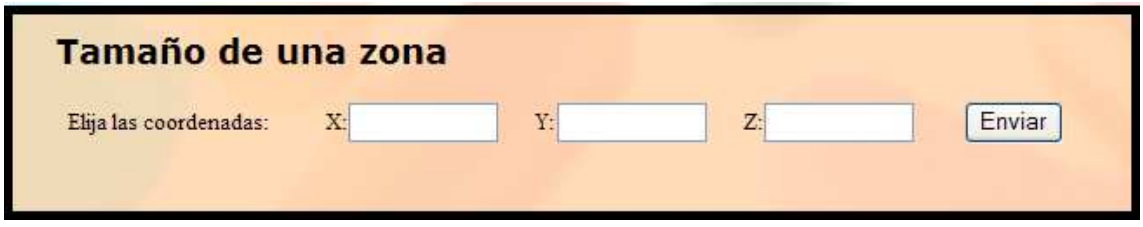


● ● ●

Tamaño de una zona

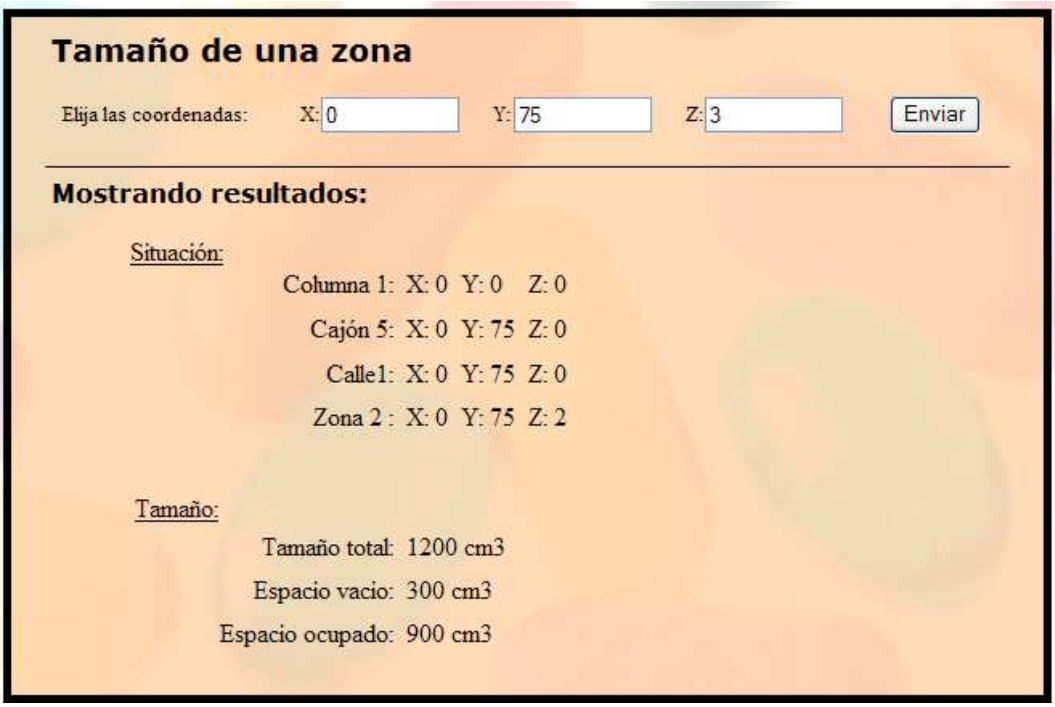
Muestra el volumen de una zona seleccionada. Si la zona no está vacía muestra el espacio ocupado y el vacío. Se introducen las coordenadas en las que se quiere hacer la consulta y el programa nos devueve la zona o sector a la que corresponden las mismas y el resultado. Los pasos a seguir son los siguientes:

1. Introducir las coordenadas en las que se quiere hacer la consulta.



2. Pulsar enviar.

Entonces aparecerá en pantalla la zona concreta a la que pertenecen las coordenadas y el espacio total, el libre y el ocupado de la misma.



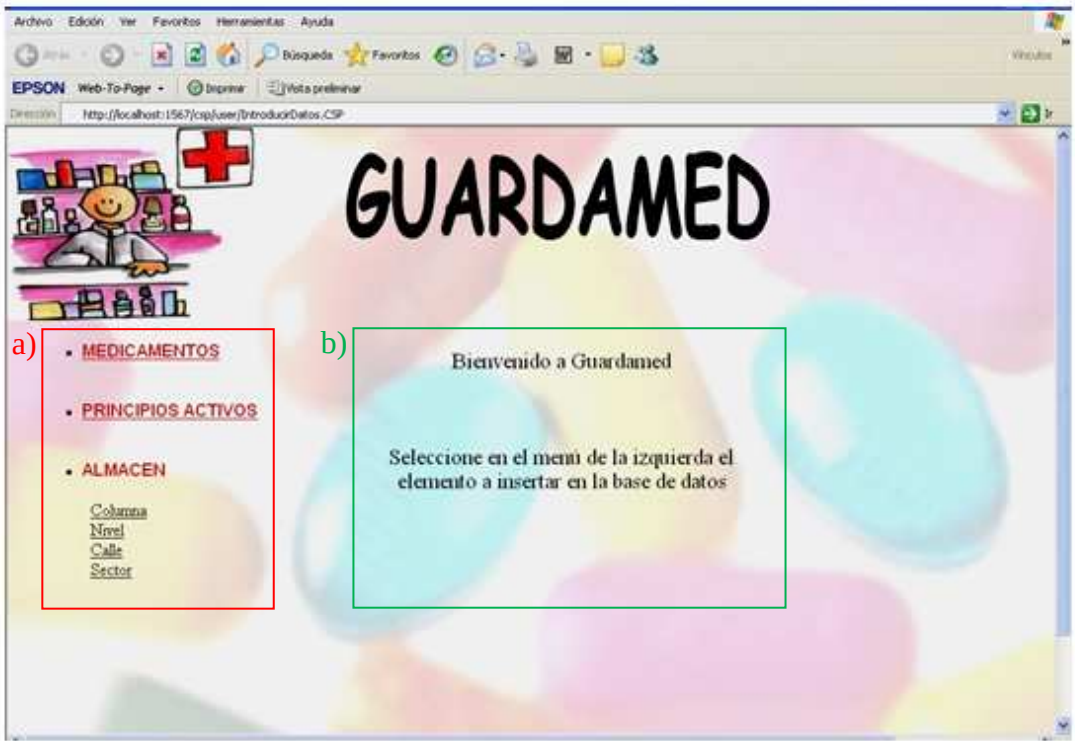
5.4.2 Introducción de datos

Para la introducción de datos, la aplicación dispone de una página dedicada a ello.

Para empezar se debe abrir el navegador y en la barra de direcciones escribir:

http://localhost:1567/csp/user/IntroducirDatos.CSP

Se abrirá la página que se muestra a continuación, donde se explica seguidamente sus partes principales.



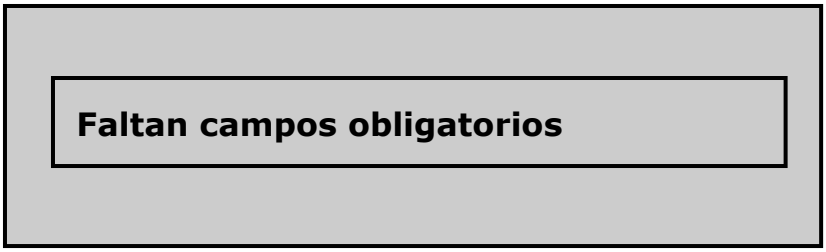
En la parte izquierda de la pantalla se observa una lista (a) con todos los posibles objetos, organizados por categorías, que se pueden insertar en la base de datos. Pinchando sobre cualquiera de ellos, aparecerá en el recuadro verde (b), un formulario, distinto para cada elemento, en el cual se introducirán los datos del mismo. Por tanto, para introducir datos, se deben seleccionar primero de la lista que aparece a al izquierda el nuevo objeto a insertar.

Todos los formularios, tienen características comunes:

1. Botón Guardar: Se pulsa dicho botón, cuando se hayan introducido los datos y se desee almacenarlos.

2. Botón Borrar: borra los datos que se han introducido, dejando los campos del formulario vacíos, cuando se pulsa el botón.

3. Los campos que son obligatorios, están marcados con un asterisco. Si se intenta guardar datos, faltando los obligatorios, la aplicación lanzará un mensaje como el de la figura y no se realizará el almacenamiento de la información.

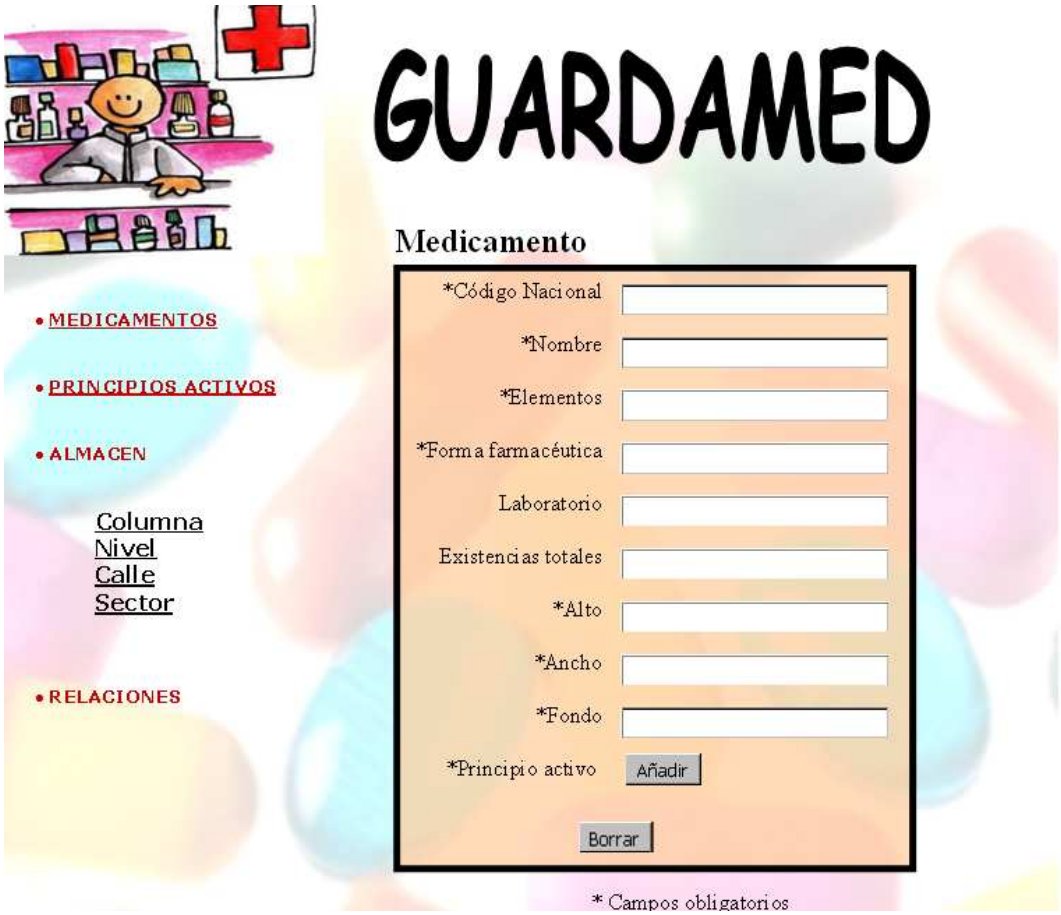


Estas características comunes, arriba descritas pueden observarse en la siguiente figura:

A continuación se describe cómo introducir datos, clasificados por categorías:

5.4.2.1 Medicamentos

Una vez pulsado sobre la opción Medicamento debe aparecer la siguiente pantalla:



Aparecen todos los campos que tiene un medicamento:

- Código nacional: identificador único de seis cifras que el ministerio de sanidad da a cada especialidad farmacéutica.

- nombre: denominación del medicamento

- elementos: número de elementos que contiene (24 sobres, 28 comprimidos, 10 jeringas…)

- forma farmacéutica: forma que se le ha dado al medicamento para su consumo (comprimidos, cápsulas, sobres, óvulos, supositorios, jeringas precargadas, inyectables, solución, aerosol…)

- laboratorio: fabricante de la especialidad farmacéutica.

- existencias totales: existencias de la especialidad, tanto en su lugar en la cajonera como en la reposición.

- alto: altura de la caja que contiene el medicamento.

- ancho: anchura de la caja que contiene el medicamento.

- fondo: fondo de la caja que contiene el medicamento.

Los campos que aparecen con un asterisco, son obligatorios. Sin éstos el medicamento no se guardará.

El "alto", "ancho" y "fondo" de cada medicamento nos da las dimensiones con las que calcular el volumen que van a ocupar. Por lo tanto si no son datos numéricos el programa nos dará un mensaje de error.

Una vez con todos los datos introducidos se pulsa guardar.

5.4.2.2 Principios activos

Para introducir un principio activo se pulsa en el menú de la izquierda sobre "Principio activo" y aparecerá la siguiente pantalla:

Una vez introducidos los datos se pulsa Guardar. Al hacerlo si el principio activo no existía en la base de datos se guarda y los campos se quedan vacíos, preparados para introducir más principios activos.

Si el principio activo con la dosis correspondiente ya existía en la base de datos el programa nos avisa con el siguiente cuadro de texto:

Al pulsar "volver" nos devuelve a la página en la que se introducen los principios activos con los campos en blanco para poder usarla de nuevo.

5.4.2.3 Almacén

Dentro de esta categoría se encuentran: columna, nivel, calle y sector. Puerta y cajón no aparecen porque se introducen cuando se añade una columna, ya que la disposición de la misma no va a cambiar, es fija.

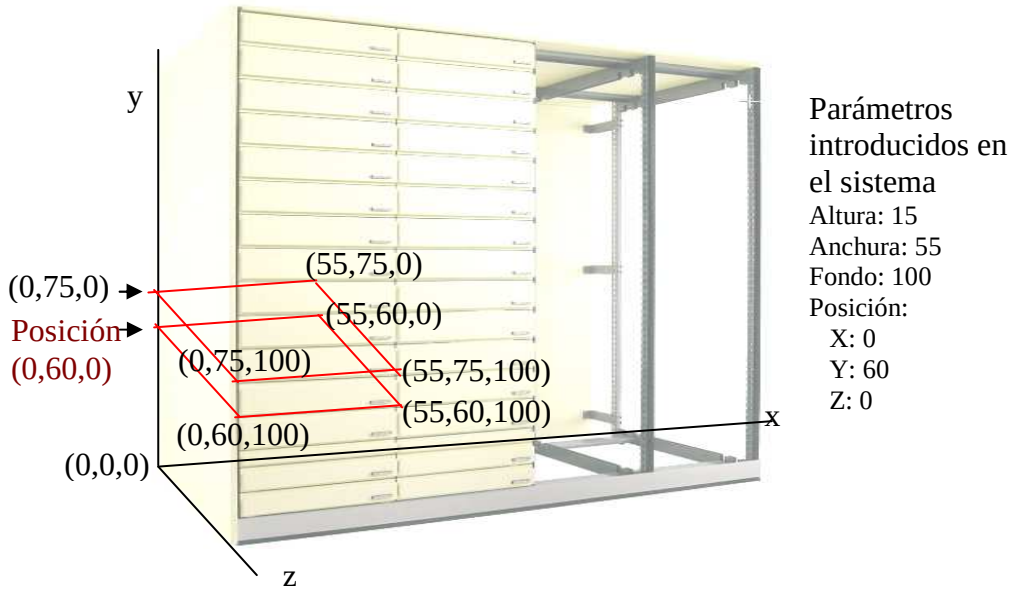
Todas ellas tienen campos comunes:

- Identificador: corresponde al identificador de cada elemento. Es un campo obligatorio.

- Alto, ancho y fondo: corresponden a las medidas que tiene el elemento.

- Posición: Dado que hemos supuesto que todos los elementos son cubos, nos basta para situarlos en el espacio una única posición, ya que a partir de esta y con los parámetros anteriormente introducidos se pueden conocer todas sus coordenadas. Se ha considerado tomar la esquina inferior izquierda del fondo para todos los elementos.

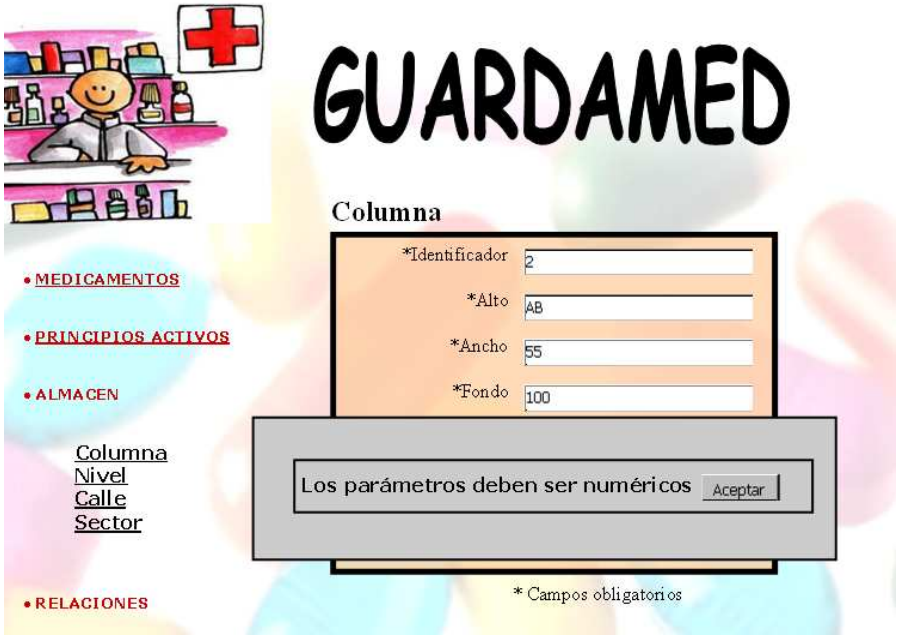
En la siguiente figura se ve mas claro:



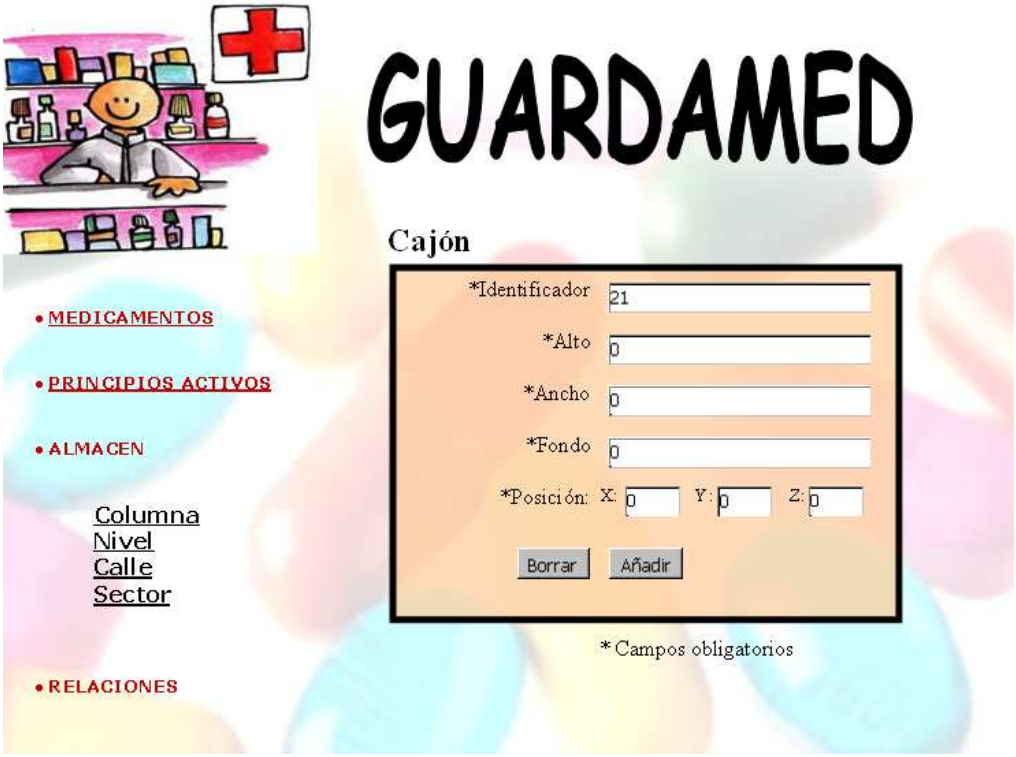
Columna

Una vez pulsado sobre la opción columna debe aparecer la siguiente pantalla:

Se consideran las medidas tomadas en centímetros. Si al introducir los datos alguno de ellos no fuese numérico el programa nos daría un mensaje de error:



Una vez introducidos los datos se deben añadir las puertas y los cajones correspondientes a la columna. Al pulsar el botón añadir cajón aprece la pantalla siguiente:



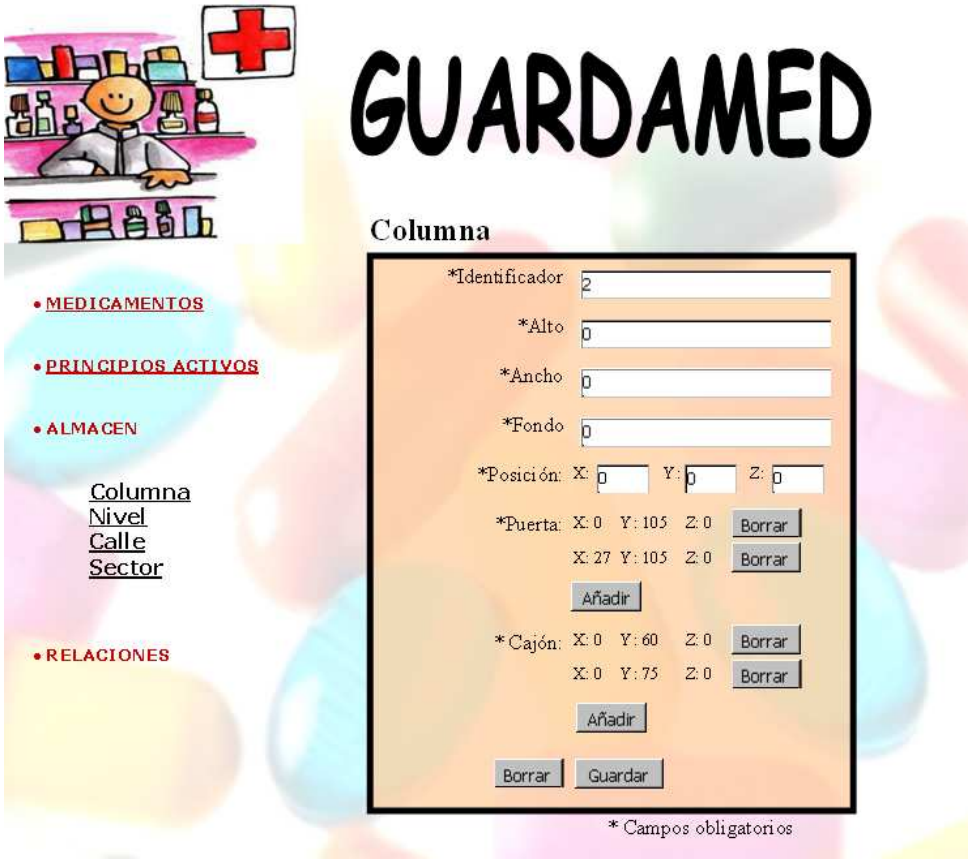
Si al introducir los parámetros del cajón alguno no fuese numérico nos aparecería un mensaje de error como el anterior.

Además, si alguno de ellos fuera mayor que los de la columna o mayor que el rango que quedase libre en la columna teniendo en cuenta los elementos ya introducidos en la misma (como otros cajones o puertas), nos daría un mensaje de error como el siguiente:

Las puertas se introducen de igual modo que los cajones y tienen las mismas restricciones.

Cada vez que se añade una puerta o cajón se vuelve a la pantalla en la que estamos introduciendo los datos de la columna y estos van apareciendo asociados a la misma.

Junto a cada cajón o puerta asociados tenemos la opción de borrarlos por si se ha producido un error al introducir los datos. En la parte inferior tenemos los botones "Guardar" y "Borrar" que afectan a la totalidad de la columna (esto borraría tanto la columna como los cajones y puertas ya asociados a la misma).



Nivel

Un nivel pertenece únicamente a una puerta, por lo tanto, lo primero que debemos hacer es buscar la columna y la puerta correspondiente.

Su funcionamiento es el siguiente:

1. Pulsar botón Buscar que aparece al lado del campo columna.

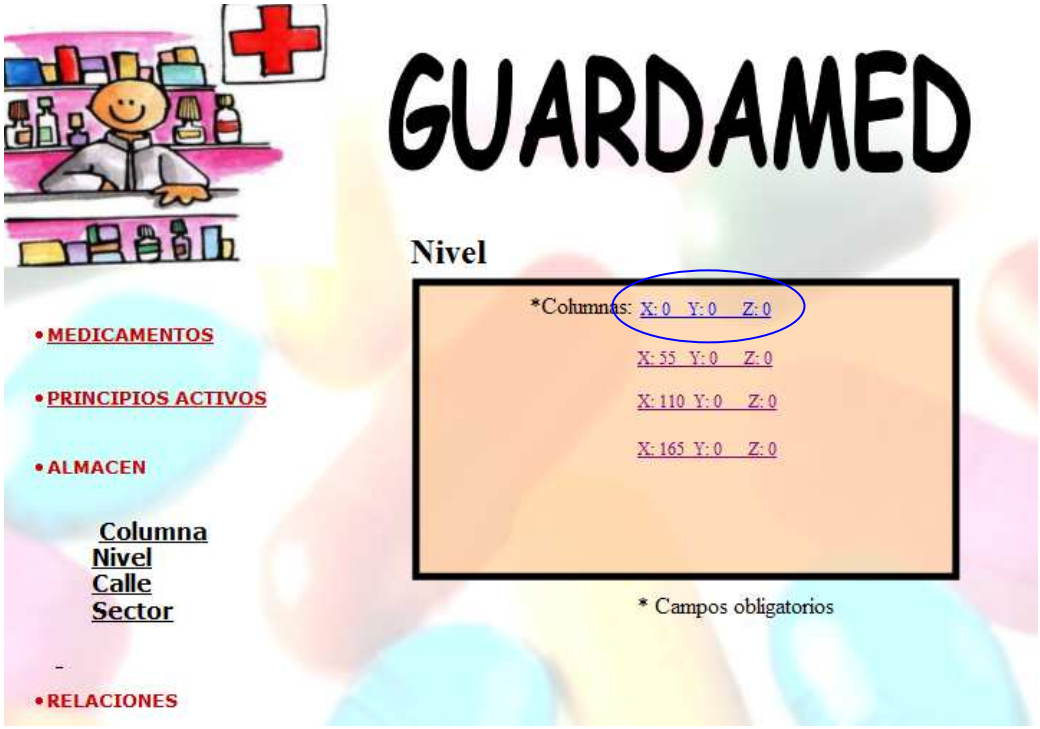
2. Se abre una nueva ventana en la que se muestran todas las columnas que tiene el sistema. Pulsar sobre la que nos interesa.

3. Automáticamente aparecerá dicha columna en el campo del formulario correspondiente.

En las siguientes figuras se ve más claro:

1)

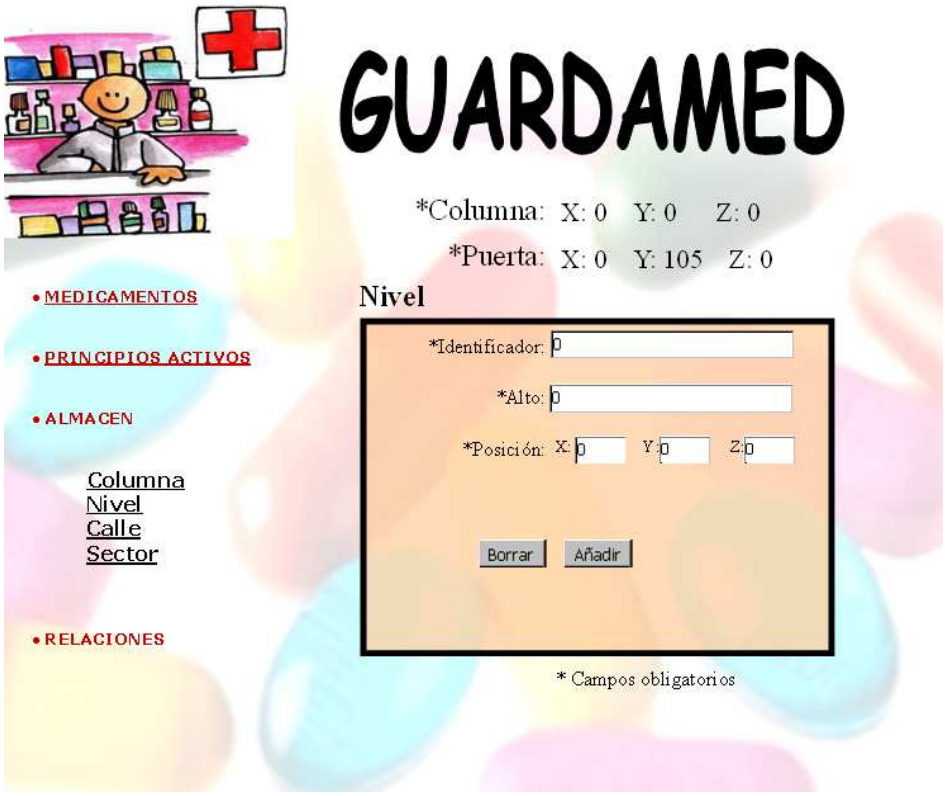
2)



3)

De igual modo se elige la puerta. Al pulsar "buscar" solo aparecerán en la lista las puertas asociadas a la columna elegida.

Una vez que tenemos la columna y la puerta podemos comenzar a asociarle niveles. Al pulsar añadir nivel nos aparece la siguiente pantalla:

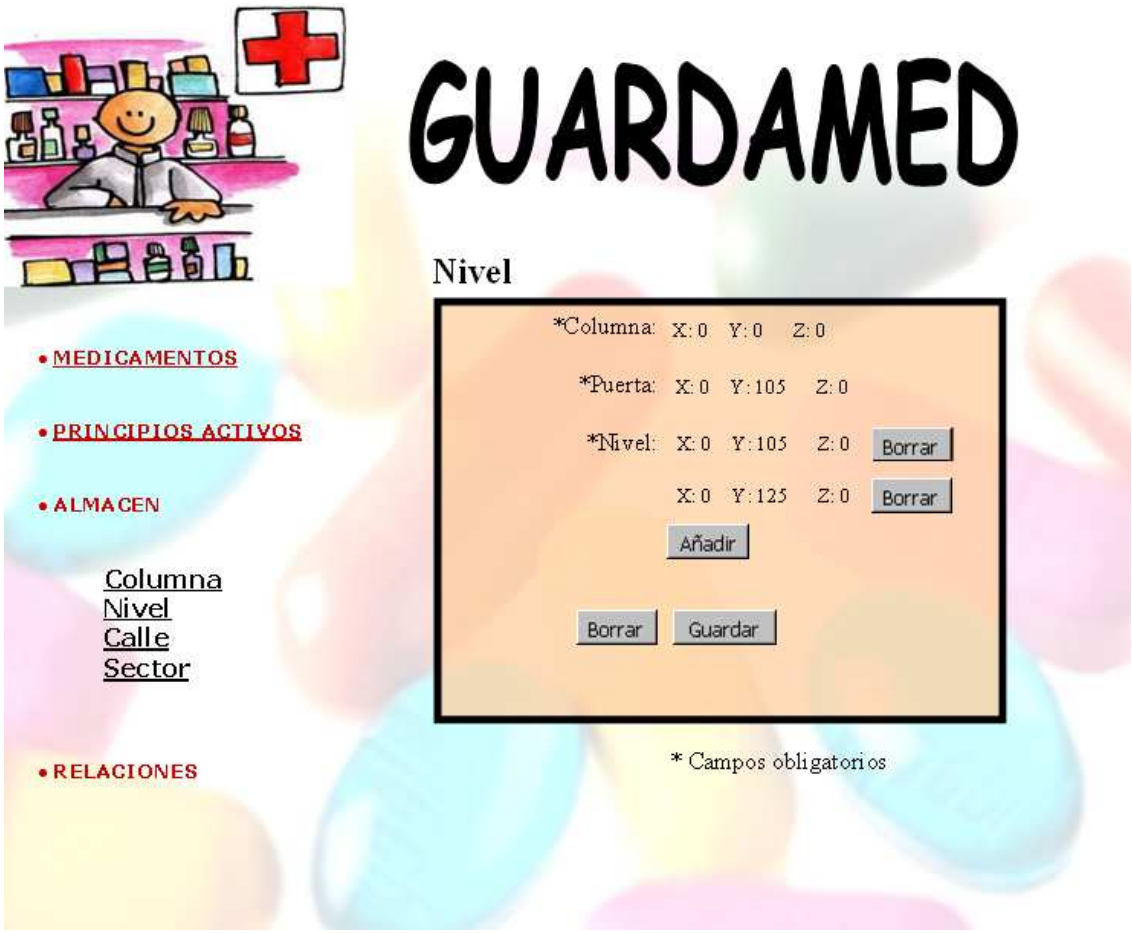


No hay que introducir el ancho y el fondo del nivel ya que corresponde al de la puerta a la que pertenece.

Al igual que en antweriores ocasiones el sistema nos dará un fallo si introducimos parámetros que no sean numéricos o si la altura del nivel es superior al rango que queda libre en la puerta.

Al pulsar "añadir" se vuelve a la pantalla anterior en la que se muestran los niveles asociados a la puerta seleccionada. Se puede borrar cada nivel por separado en el boton que aparece a su derecha o todos a la vez pulsándo el botón borrar situado en la parte inferior del cuadro de diálogo.

Al guardar quedan asociados los niveles a la puerta.

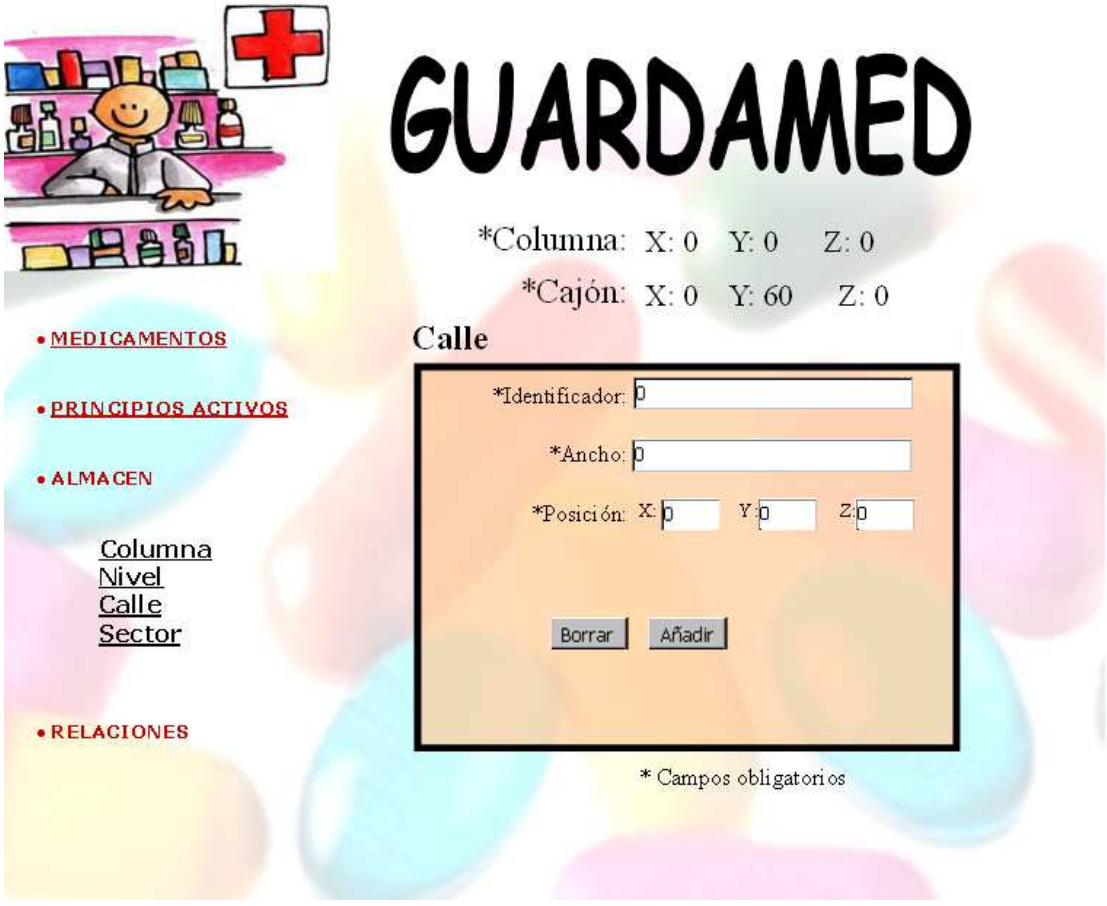


Calle

Una calle solo puede estar asociada a un cajon, por lo tanto, al igual que ocurría al introducir los niveles, primero debemos buscar el cajón y la columna a la que pertenece para asociarle las calles.

El mecanismo es similar al anteriormente explicado. Al pulsar sobre "calle" en elmenú de la izquierda se abre un cuadro de diálogo en el que debemos buscar la columna y el cajón al que le vamos a asociar la calle.

Los datos de la calle que debemos introducir son sus coordenadas y el ancho que ocupa, ya que el alto y el fondo será el correspondiente al cajón en el que está contenida. Si el ancho es mayor al rango que queda libre nos devolverá un error, al igual que si alguno de los parámetros no son numéricos.



Sector

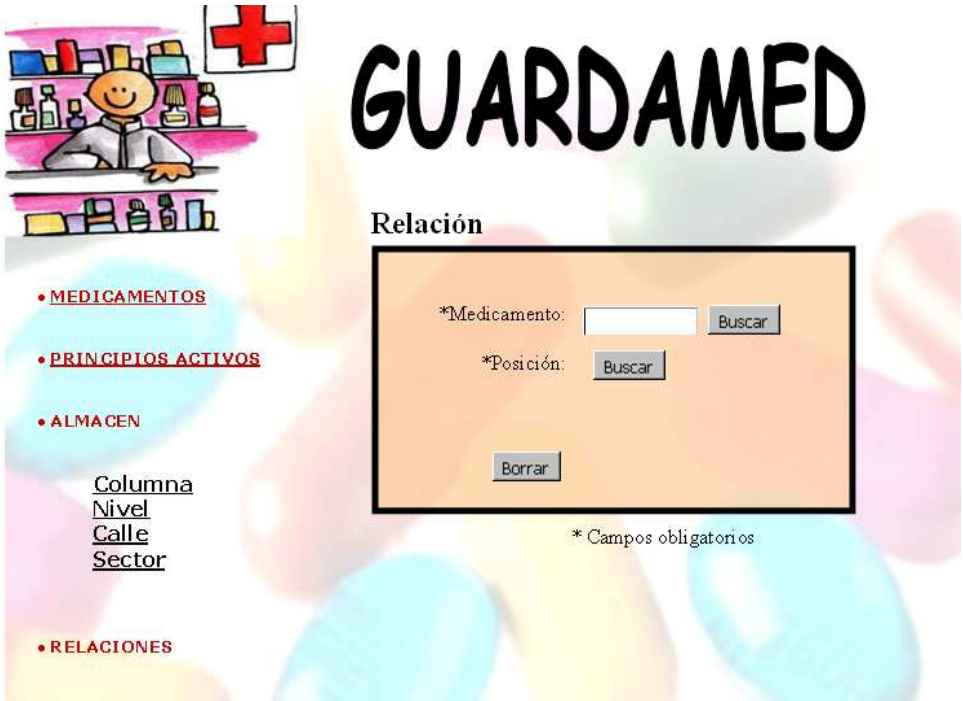
Para introducir un sector deberemos seleccionar la columna, cajón y calle a la que pertenece, al igual que ocurría con nivel y calle. En este caso deberemos introducir el fondo del sector ya que el alto es el del cajón y el ancho el de la calle.

No se explica más, ya que funciona de la misma forma que la introducción de los demás datos.

5.4.2.4 Relaciones

Dentro de esta categoría se encuentran las relaciones entre los medicamentos y la cajonera; dicho de otra forma, en esta categoría se le dice al sistema dónde se guarda cada medicamento.

Para crear una relación hay que elegir un medicamento y una posición en la cajonera.



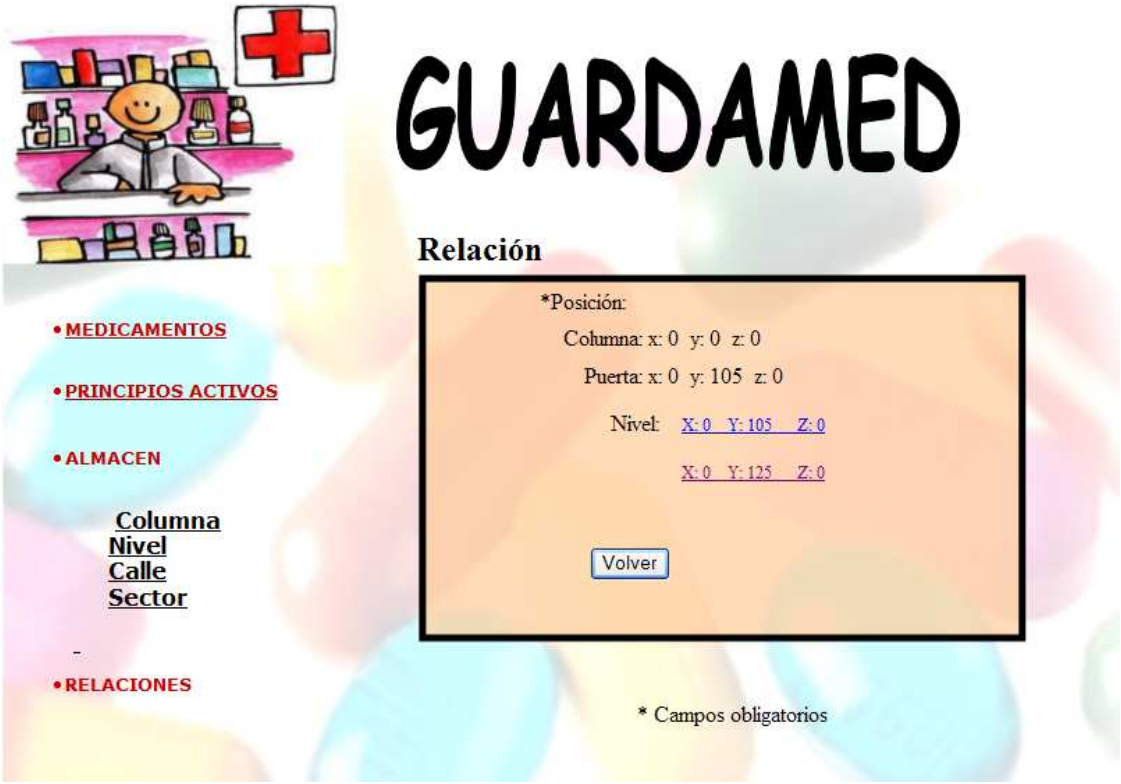
Para el medicamento se tiene un cuadro de diálogo en el que introducir el nombre del medicamento. Una vez introducido se pulsa "buscar" y aparece un listado con los medicamentos que hay en la base de datos con ese nombre, (con el mismo nombre hay distintas dosis y distintos tamaños).

Al seleccionar el medicamento se vuelve a la pantalla anterior en la que hay que buscar una posición vacía en la que poder guardar todas sus existencias.

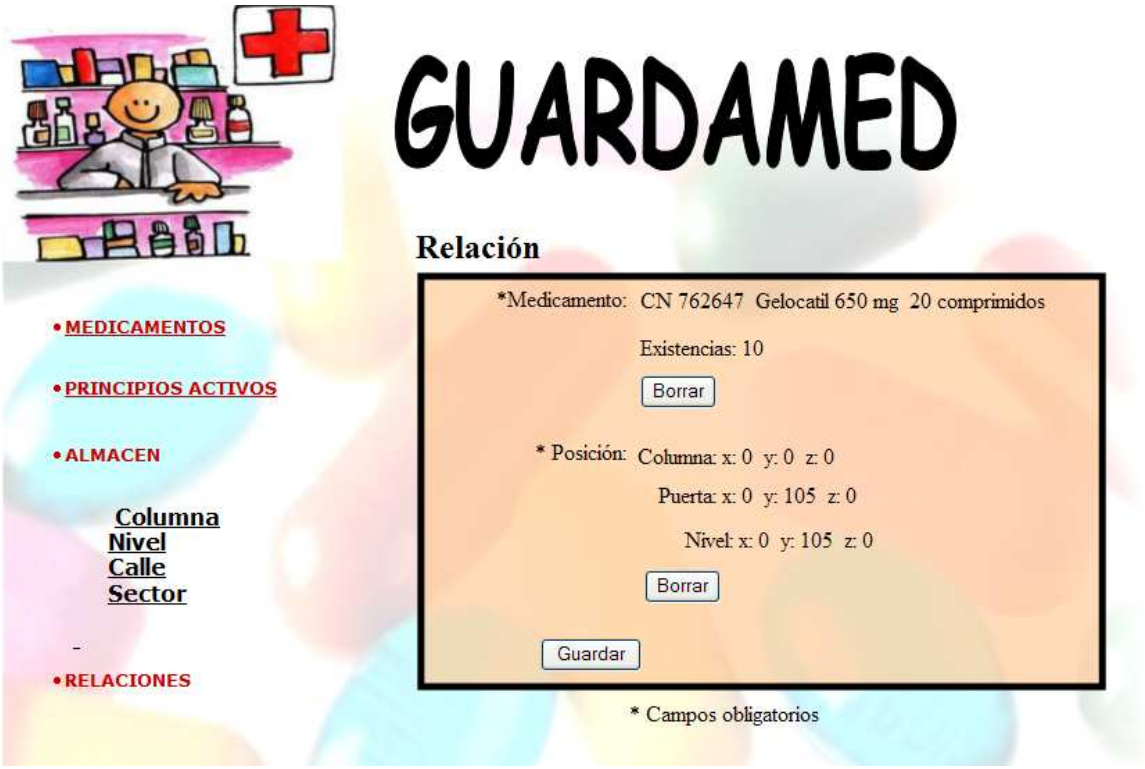
Al pulsar "buscar" nos aparecen las columnas con posiciones vacías. Al elegir una nos aparecen las puertas y/o cajones con posiciones vacías de la misma. Si se elige un cajón habría que elegir entre sus calles con posiciones libres y después una de las posiciones. Si fuese una puerta se elegiría entre los niveles que no estuviesen completos.

Todas estas posiciones vacías serán iguales o mayores que el espacio que ocupan todas las existencias del medicamento. Así nos aseguramos de que se guardan todas juntas.

Todas estas pantallas son similares. A continuación se muestra un ejemplo en el que ya se ha elegido columna y puerta y se está eligiendo nivel.



Una vez elegido el nivel aparecen en pantalla todos los datos de la relación que se va a efectuar.



Se puede "Borrar" el medicamento o la posición elegida para buscar otro. Si estamos de acuerdo con la selección se pulsa "guardar" y la relación queda establecida y la posición "ocupada".

6. EXPERIMENTACIÓN

En este apartado se muestran algunos ejemplos significativos de la aplicación realizada.

6.1 Introducción de un medicamento

En este apartado se va a realizar una inserción en la base de datos de una especialidad farmacéutica.

El primer paso es abrir el explorador y en la barra de dirección escribir:

http://localhost:1567/csp/user/IntroducirDatos.CSP

y debe abrirse la siguiente página:

Una vez abierta la página se selecciona la opción Medicamento.

Una vez pulsado sobre la opción Medicamento debe aparecer la siguiente pantalla:

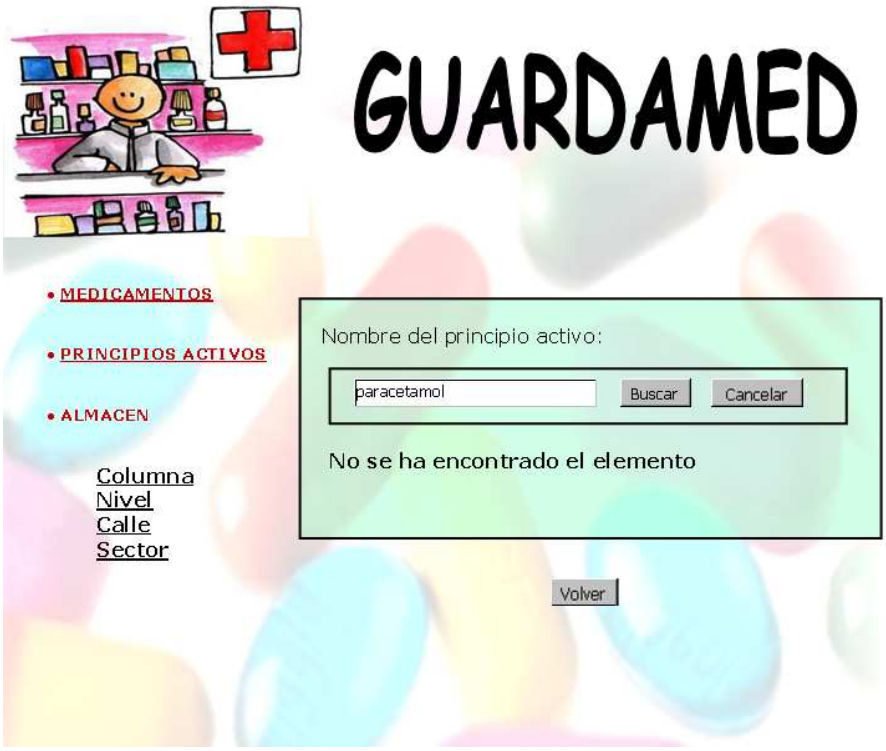
Aparecen todos los campos que tiene un medicamento:

- Código nacional: identificador único de seis cifras que el ministerio de sanidad da a cada especialidad farmacéutica.
- nombre: denominación del medicamento
- elementos: número de elementos que contiene (24 sobres, 28 comprimidos, 10 jeringas…)
- forma farmacéutica: forma que se le ha dado al medicamento para su consumo (comprimidos, cápsulas, sobres, óvulos, supositorios, jeringas precargadas, inyectables, solución, aerosol…)
- laboratorio: fabricante de la especialidad farmacéutica.

- existencias totales: existencias de la especialidad, tanto en su lugar en la cajonera como en la reposición.
- alto: altura de la caja que contiene el medicamento.
- ancho: anchura de la caja que contiene el medicamento.
- fondo: fondo de la caja que contiene el medicamento.

Los campos que aparecen con un asterisco, son obligatorios. Sin éstos el medicamento no se guardará.

Al medicamento hay que asociarle los principios activos que contiene. Para esto hay que pulsar el botón "Añadir". Los principios activos deberán haberse introducido por adelantado para poder asociarlos con un medicamento. Cuando se pulsa el botón "Añadir" aparece el siguiente cuadro de texto:



Se escribe en el cuadro de texto el principio activo y se pulsa "Buscar". Si aparece el mensaje que se muestra arriba "No se ha encontrado el elemento" es porque el principio activo no se ha introducido con anterioridad y hay que hacerlo.

Si al buscar el principio activo se encuentra en la base de datos el programa nos pedirá la dosis:

Se escribe en el cuadro de texto la dosis del principio activo y se pulsa "Buscar". Si aparece el mensaje "No se ha encontrado el elemento" es porque el principio activo con esa dosis no se ha introducido con anterioridad y hay que hacerlo.

En caso contrario se asocia el principio activo al medicamento y el programa vuelve a la página en la que se estaba introduciendo el medicamento. Desde ahí se introducen, con el mismo sistema, todos los principios activos que tenga asociados el medicamento.

Si se pulsa, en cualquier momento, el botón volver que aparece debajo del cuadro de texto se abandonará la introducción del principio activo para el medicamento y se volverá a la página en la que se estaba introduciendo el medicamento.

Es necesario añadir, al menos, un principio activo por medicamento.

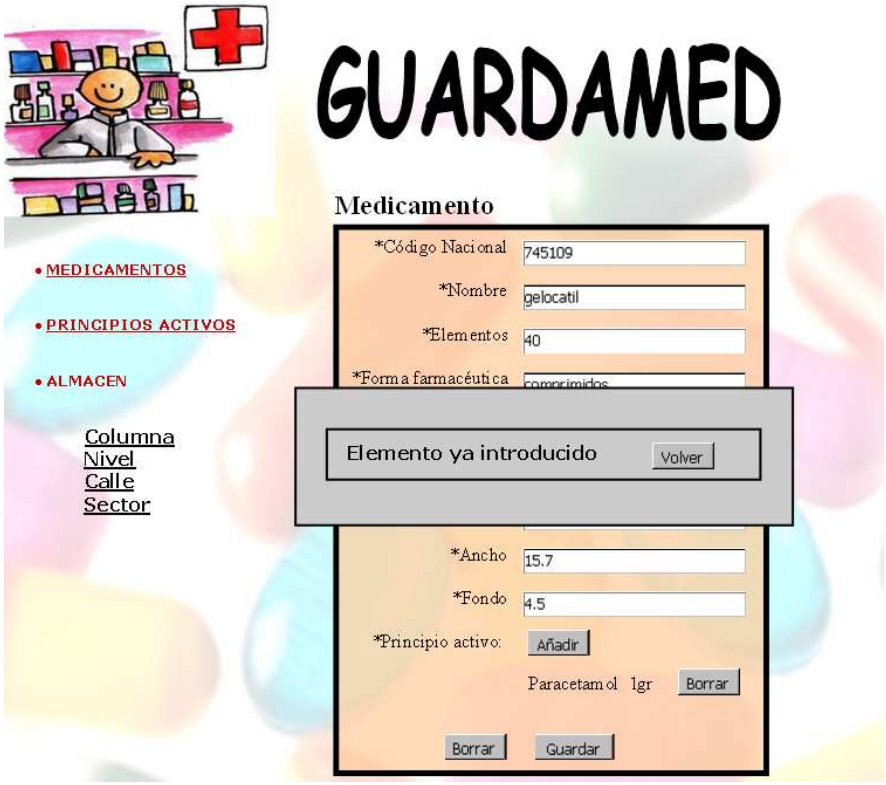
Cada vez que se asocie un principio activo a un medicamento éste aparecerá en la pantalla:

Junto a cada principio activo aparecerá un botón para borrarlo. Con esta acción se eliminará la asociación del principio activo con el medicamento.

Cuando se haya asociado al menos un principio activo aparecerá el botón inferior "Guardar" junto al de "Borrar". Estas acciones son para la totalidad del medicamento.

Con el botón "Borrar" se borrarán todos los datos introducidos hasta el momento de pulsarlo.

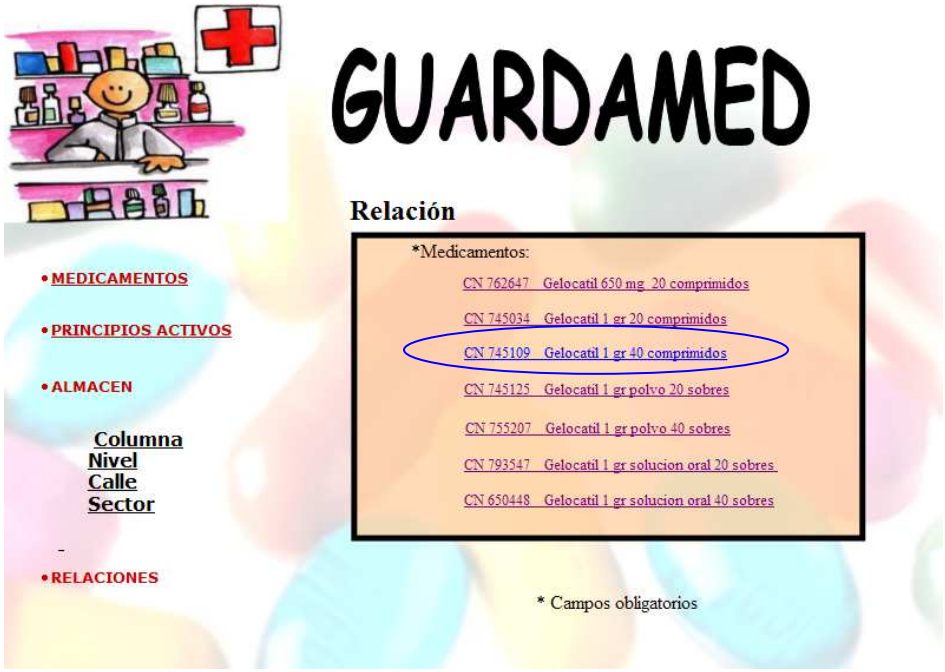
Con el botón "Guardar" se guardará el medicamento si éste no estuviese ya en la base de datos. En caso contrario aparecerá el siguiente mensaje:



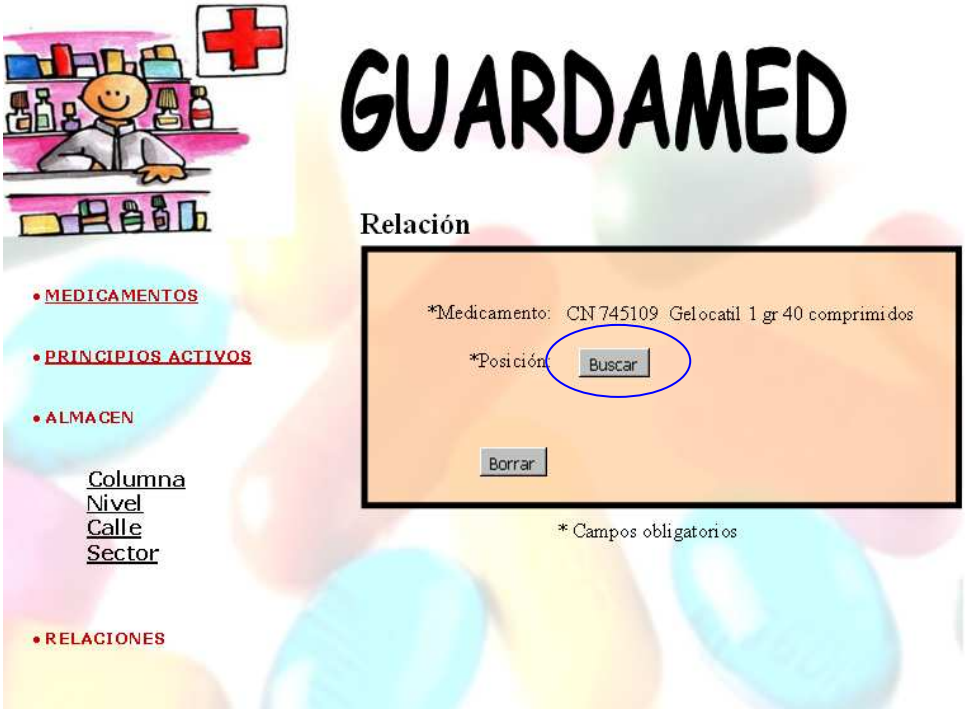
Pulsando el botón "Volver" desaparecerá el cuadro del mensaje y se quedará la pantalla del medicamento.

Suponiendo que no existiese en la base de datos se pulsaría "guardar" y se volvería a la página principal. El medicamento ya estaría introducido en la base de datos. Ahora tendríamos que guardarlo en el almacén. Para esto se pulsa en la opción Relaciones.

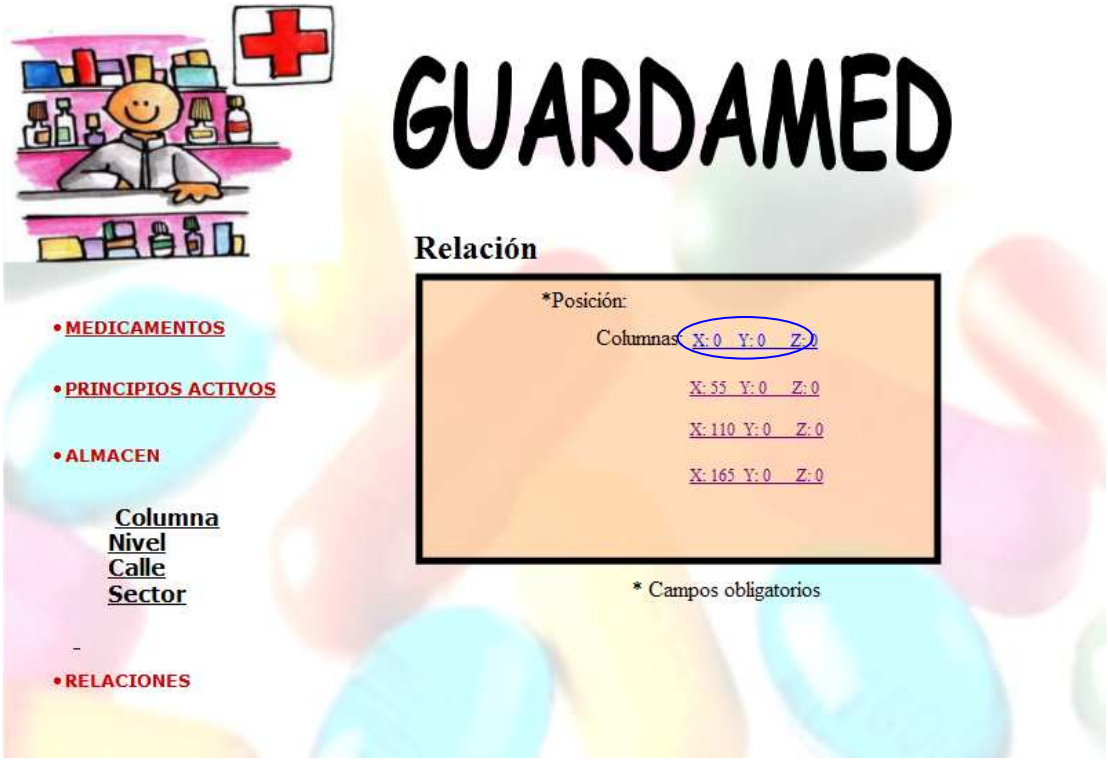
Introducimos el nombre del medicamento que queremos guardar y de la lista que aparece elegimos el que acabamos de introducir.



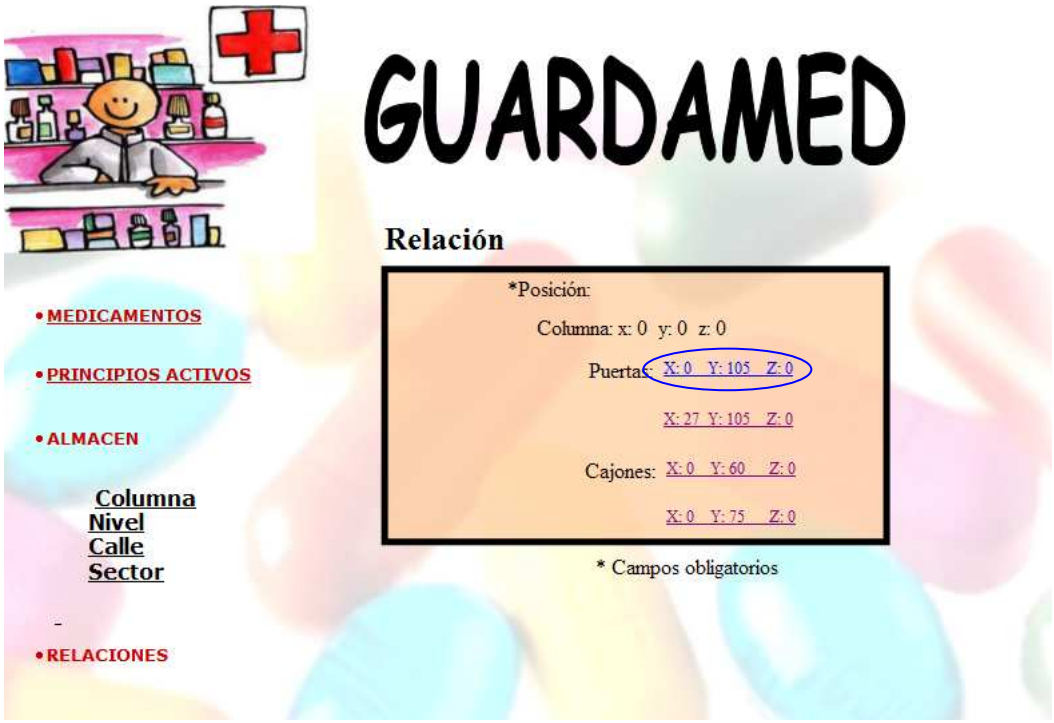
Una vez introducido el medicamento hay que buscar una posición donde guardarlo.



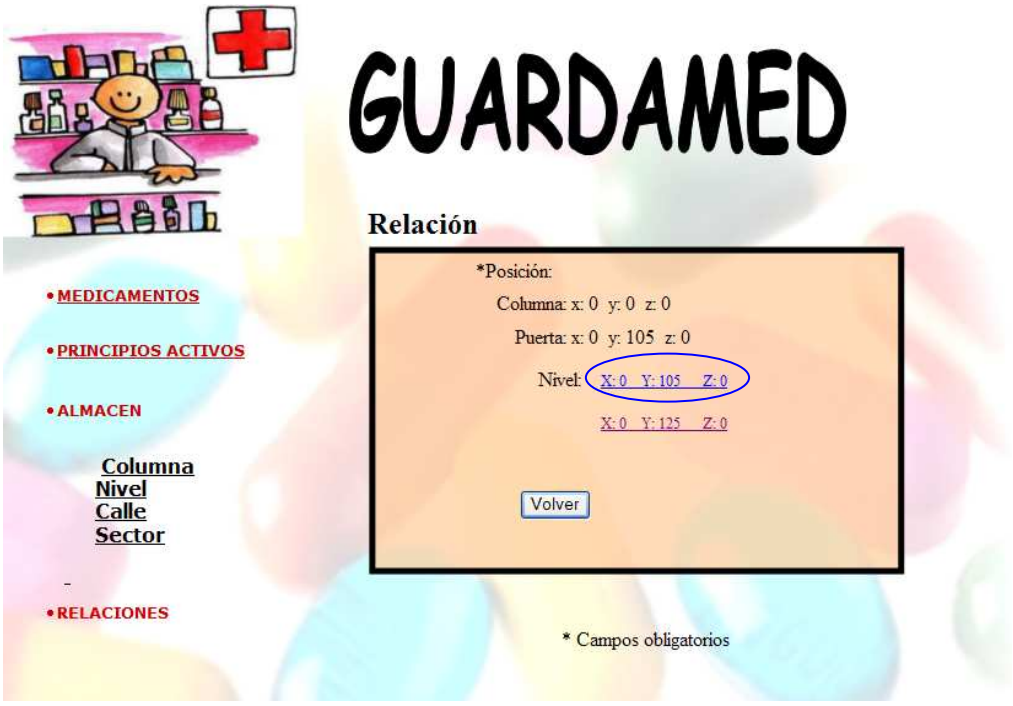
Al pulsar "Buscar" nos aparecen las columnas con posiciones lo suficientemente grandes como para guardar las 10 existencias que tenemos.



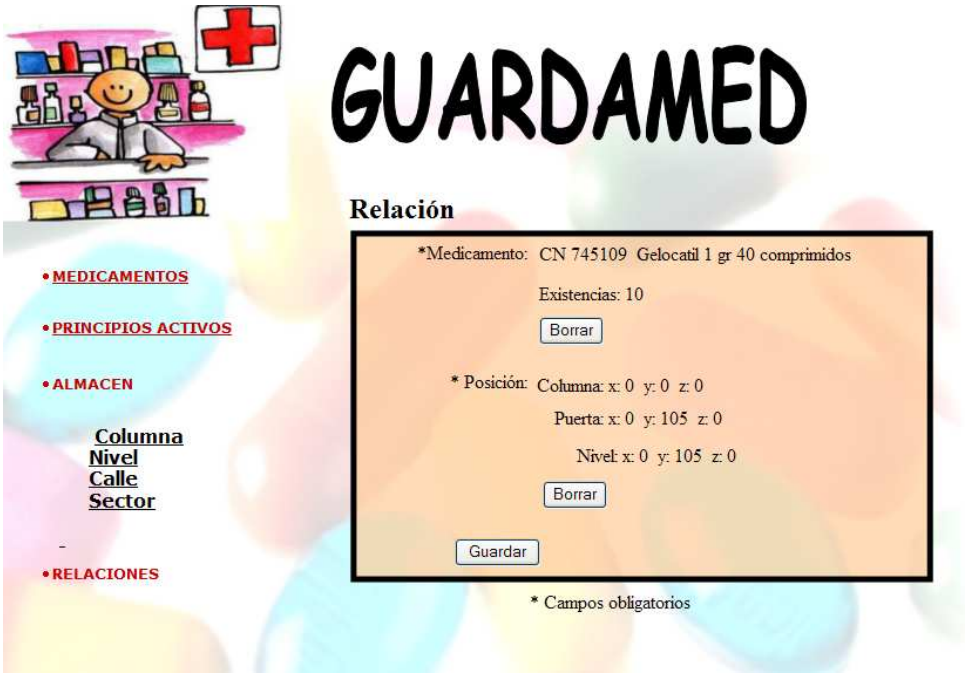
Una vez elegida la columna debemos elegir una puerta o un cajón de los que nos aparecen. Estos son los que pertenecen a la misma y tienen posiciones con suficiente espacio.



Trashaber elegido la puerta, tendremos que elegir de entre los niveles que tengan hueco:



Una vez elegido el nivel aparecen en pantalla todos los datos de la relación que se va a efectuar.



Una vez introducidos todos los datos, se pulsa Guardar. Se tiene por fin el medicamento introducido y guardado en el almacén.

6.2 Consultas

En este apartado se van a realizar algunas consultas como ejemplo. Para todas ellas el primer paso será abrir el explorador y escribir la siguiente dirección:

http://localhost:8972/csp/user/medicamentos.csp

A continuación se detalla cada una de las consultas.

6.2.1 Tipo de medicamento en zona

Al pulsar la dirección anterior en el explorador, se debe abrir una página con consultas. La consulta que se muestra en este apartado es la quinta.

Cómo se puede observar en la figura, la consulta tiene cuatro campos:

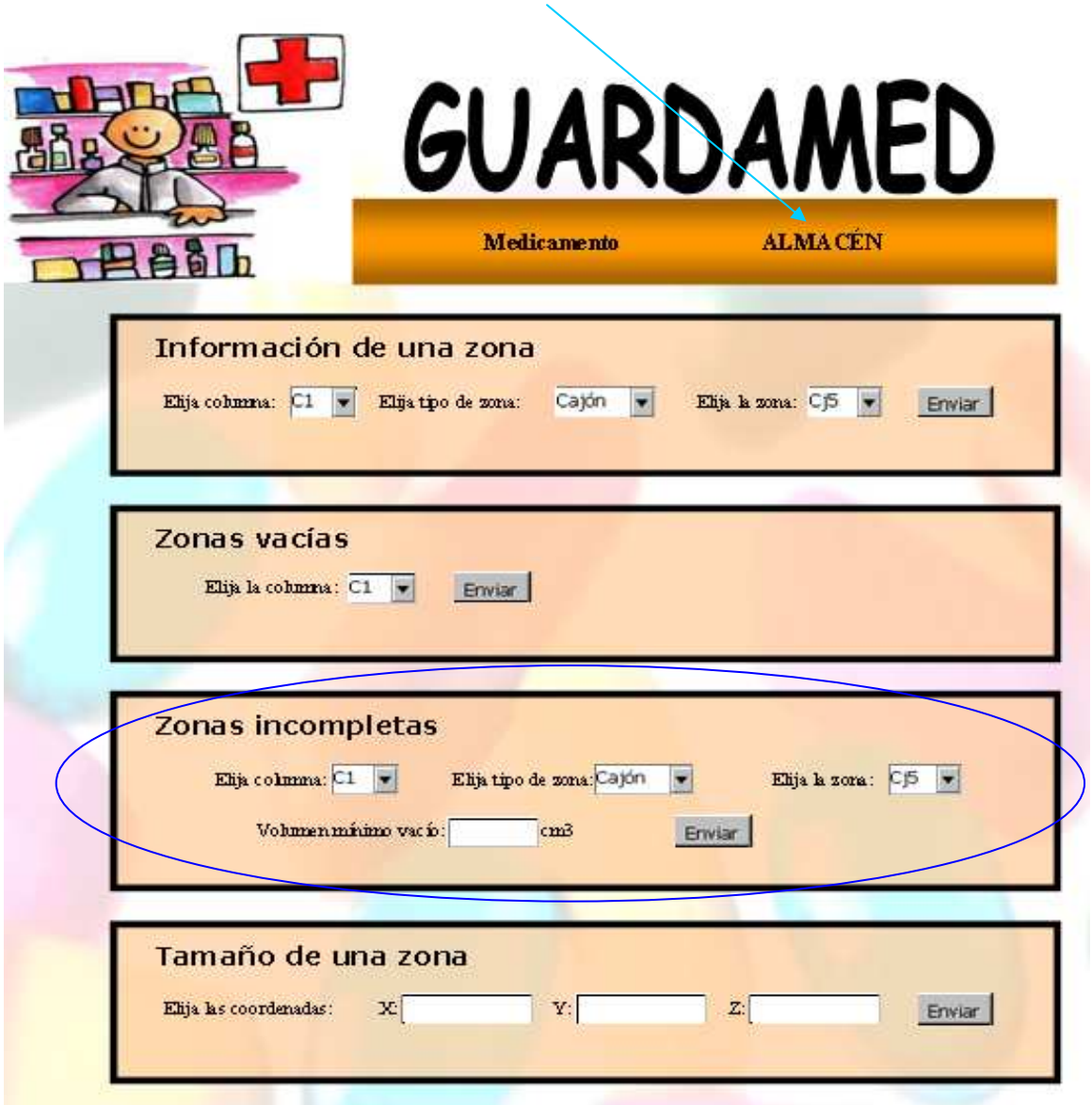
- Forma farmacéutica: es la forma en la que se presentan los medicamentos sobre los que se quiere consultar.
- Columna en la que buscar.
- Tipo de zona en la que buscar: Puerta o cajón.
- Zona concreta en la que buscar. Si se ha elegido una puerta, en el apartado anterior, aparecerán las puertas de la columna elegida, si no, aparecerán los cajones.

Con estos datos introducidos, se pulsa Enviar y éste es el resultado:

Se muestran las cremas que están guardadas en el cuarto cajón de la primera columna. De cada una de ellas se muestra la calle y el sector en el que se encuentra y la posición exacta de los mismos, cumpliendo así con los requisitos exigidos.

6.2.2 Zonas incompletas

Para acceder a esta consulta es necesario pinchar sobre la opción Almacén, en el menú de la parte de arriba. Cuando se acceda a este menú la palabra Almacén aparecerá en mayúsculas.

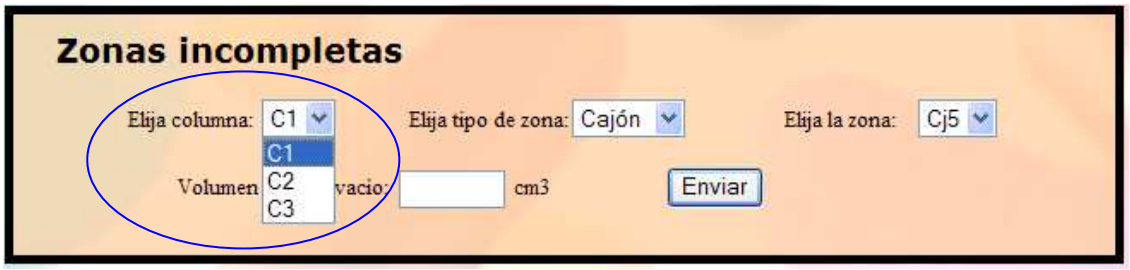


La consulta que se describe es la tercera que aparece.

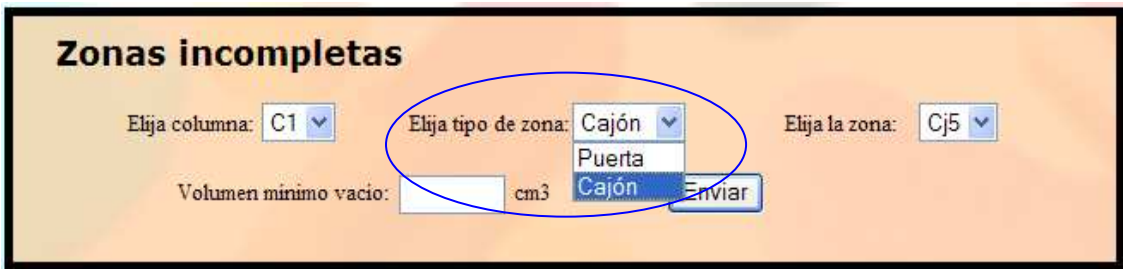
Muestra los sectores o niveles de la zona seleccionada que, teniendo guardado un medicamento, les queda espacio libre sin usar. Este espacio debe ser igual o mayor al introducido por pantalla.

Cómo se puede observar en la figura, la consulta tiene cuatro campos:

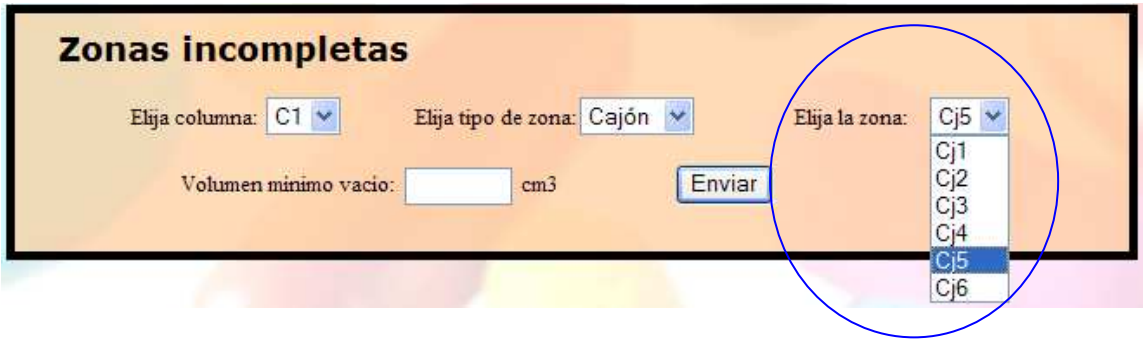
- Columna en la que buscar. Para elegirla se abre el desplegable y se elige la que se quiere.



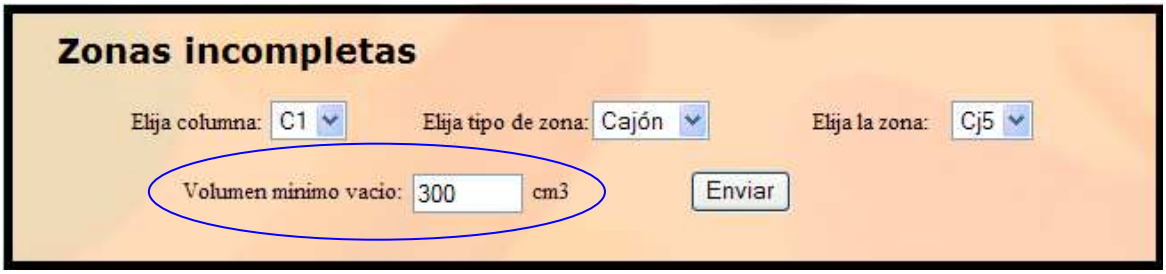
- El tipo de zona en la que se quiere hacer la consulta. Puerta o cajón.



- Zona concreta en la que consultar. Si en el paso anterior se ha elegido una puerta aparecerán las correspondientes a la columna elegida. Si se ha elegido un cajón aparecerán todos los existentres en la columna. Debe elegirse uno.

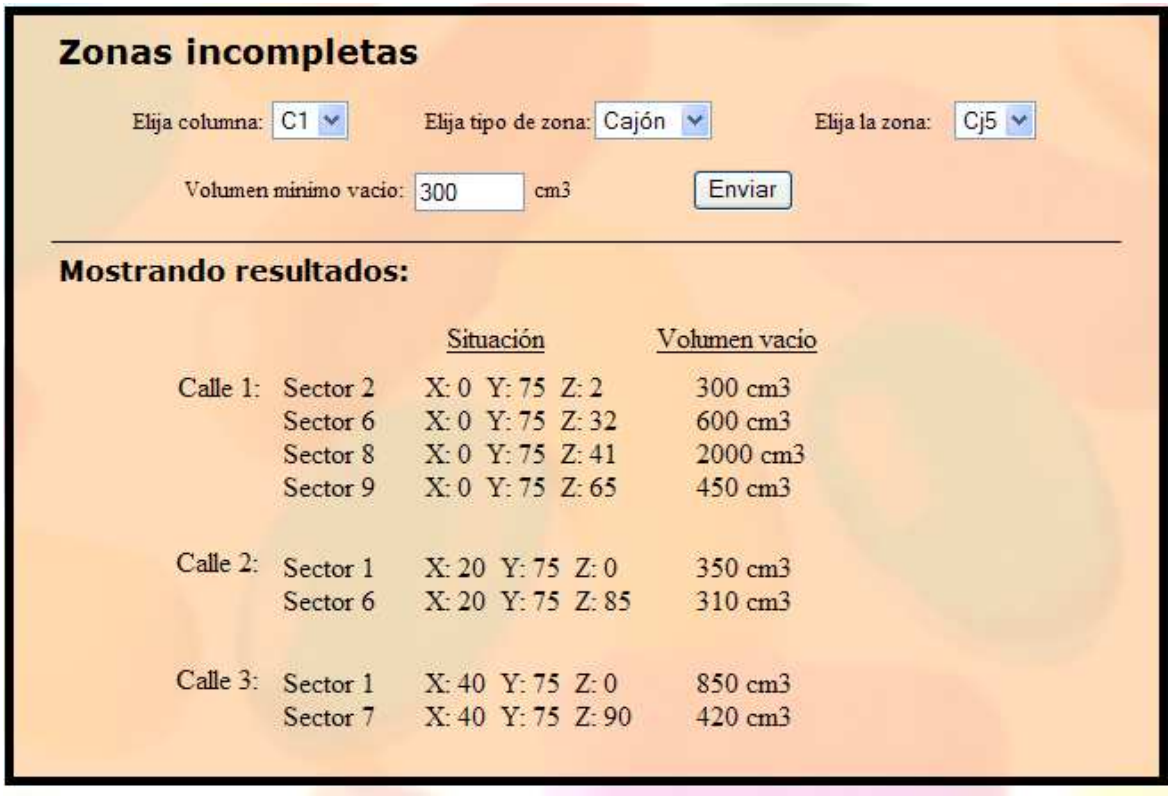


- Por último introduciremos el volumen mínimo que debe quedar vacío aún estando guardado un medicamento.



- Pulsar enviar.

Entonces aparecerá en pantalla las zonas incompletas con un espacio vacío mayor al volumen dado.



Se muestra de cada sector incompleto: la calle a la que pertenece, la posición exacta del sector y el volumen que le queda vacío a cada uno; cumpliendo así con los requisitos iniciales de la consulta.

7. CONCLUSIONES

Los objetivos iniciales del proyecto han sido alcanzados: la librería creada tiene un carácter general, la aplicación muestra el almacenamiento en una cajonera de medicamentos y se ha utilizado la herramienta Caché. Por tanto, con los objetivos iniciales cumplidos y desde un punto de vista retrospectivo, se pueden afirmar las siguientes conclusiones:

- Una base de datos modelable elimina la necesidad de usar software específico. En este caso Caché permite incorporar librerías personalizadas, por lo que se ha conseguido que Caché trate información espacial de una manera natural. Además, el modelo desarrollado en el proyecto, sin ser una herramienta comercial ni estar optimizado al cien por cien, puede ponerse a la altura de software específico, como puede ser por ejemplo Spatial Oracle, ya que ambos tratan información espacial de la misma manera. Incluso este modelo presenta una ventaja sobre Oracle, y es que Caché es más rápido.

- La elección de la herramienta es la adecuada. No sólo porque Caché permita incorporar librerías personalizadas, sino porque esta herramienta está diseñada desde el punto de vista de la orientación a objetos, lo que le permite trabajar en conjunción con las nuevas tecnologías. Asimismo, Caché es una herramienta que ofrece una gran variedad de servicios y posibilidades.

- La solución al objetivo de permitir que la introducción de información a la base de datos se realizase de una forma sencilla e intuitiva, se ha solucionado con la creación de formularios. Tanto los formularios como el resto de la aplicación han sido implementados en CSP, herramienta

que ofrece Caché. CSP en el fondo es una herramienta Web, y como tal, ésta trae implícita una ventaja: en la parte cliente las restricciones hardware son mínimas. En las arquitecturas Web, las restricciones de la parte cliente son muy reducidas.

7.1 Conclusiones personales

La creación de la librería para información de tipo espacial, ha sido la que más me ha gustado, desde el análisis hasta la implementación. El reto de tener que hacerlo en tres dimensiones en lugar de en dos fue muy estimulante. Tuve que repasar algunos conceptos matemáticos, como la ecuación de una recta en tres dimensiones, para poder programar las operaciones entre los distintos tipos de datos.

Lo que más me ha costado ha sido crear la interfaz de la aplicación, es decir las paginas CSP (páginas web con acceso a Caché), tanto su diseño como la navegabilidad entre ellas. Y sobre todo cómo mandar los datos introducidos por el usuario de un lugar a otro de la aplicación.

El trabajo con Caché ha sido agradable. Tiene una interfaz sencilla, visual y muy intuitiva que hace que el trabajo con ella sea fácil.

Finalmente mencionar, que este proyecto me ha permitido conocer más a fondo la herramienta Caché (ya la conocía de la asignatura Base de Datos Avanzadas) y ver las múltiples opciones que ésta permite.

El trabajo con el tutor del proyecto ha sido muy cómodo. Por ello, la experiencia en general ha sido muy positiva.

8. PRESUPUESTO

Para este trabajo se ha calculado un presupuesto desglosándolo en las distintas fases en las que se han ido desarrollando.

1. Formación.
2. Análisis y Diseño.
3. Implementación.
4. Experimentación.
5. Documentación.

El coste de cada fase y el total del presupuesto se han considerado como un proyecto real, en el que inicialmente trabajarían un programador junior y un analista programador. El programador junior realizaría las fases 1, 3, 4 y 5 a un coste/hora de 30 €. El analista programador se ocuparía de la 2ª fase a un coste/hora de 60 €.

En base a esto, el presupuesto ha quedado de la siguiente manera:

Fases	*Horas*	*Coste*
Formación	70	2100 €
Análisis y Diseño	200	12000 €
Implementación	270	8100 €
Experimentación	50	1500 €
Documentación	150	4500 €
TOTAL	**740**	**28200 €**

Tal y como se especifica en la tabla, el coste total del proyecto ascendería a 28.200,00 €. Teniendo en cuenta que la jornada será de 8 horas, los días que se estiman son 92,5. En meses, sabiendo que cada mes tiene unos 22 días laborables, la duración sería de 4,2.

9. DESARROLLOS POSTERIORES

A continuación se exponen una serie de ideas que permiten ampliar y mejorar la funcionalidad de la aplicación.

Para esta aplicación se ha supuesto que todos los elementos a almacenar son cubos de distintas dimensiones. En desarrollos posteriores se tendría que tener en cuenta los distintos tipos de formas a almacenar. Como pueden ser cilindros, conos…

Figura: Tipos de envases

La aplicación nos informa sobre el lugar de almacenamiento de los medicamentos y de si caben nuevos elementos o no. En un futuro la aplicación debería ser capaz de optimizar el espacio. Puede que el sector asignado a un medicamento no sea el adecuado (por ser demasiado pequeño o por ser demasiado grande).

¿Cómo podría saber esto la aplicación? Con la introducción de máximos y mínimos para cada especialidad. Cada medicamento deberá tener unas unidades máximas y mínimas asignadas (dependiendo de cómo sea la necesidad de cada medicamento en un tiempo determinado) Las unidades

almacenadas en un momento dado no podrán ser mayores al máximo ni menores al mínimo establecidos. Con esto la aplicación debería ser capaz de calcular el espacio requerido para cada medicamento, teniendo en cuenta sus dimensiones, y proponerlo cada vez que se le consulte.

También podría gestionar un almacenamiento secundario o de "reposición". Esta gestión la haría de igual forma que para el almacenamiento principal. Su finalidad sería tener localizadas todas las unidades que exceden del stock máximo establecido para cada especialidad y avisarnos cuando haga falta pasar unidades del stock secundario al principal.

Lo que se pretende es que la aplicación, con todos los desarrollos posteriores, no solo se utilizase para un almacenamiento manual, como el tenido en cuenta en este proyecto, sino para un almacenamiento robotizado. Esto sería posible ya que el almacenamiento robotizado dispone de columnas, niveles y sectores (como se muestra en la imagen) al igual que el almacenamiento manual.

Con esto se demuestra el carácter general de la aplicación, que se había propuesto como objetivo en este proyecto.

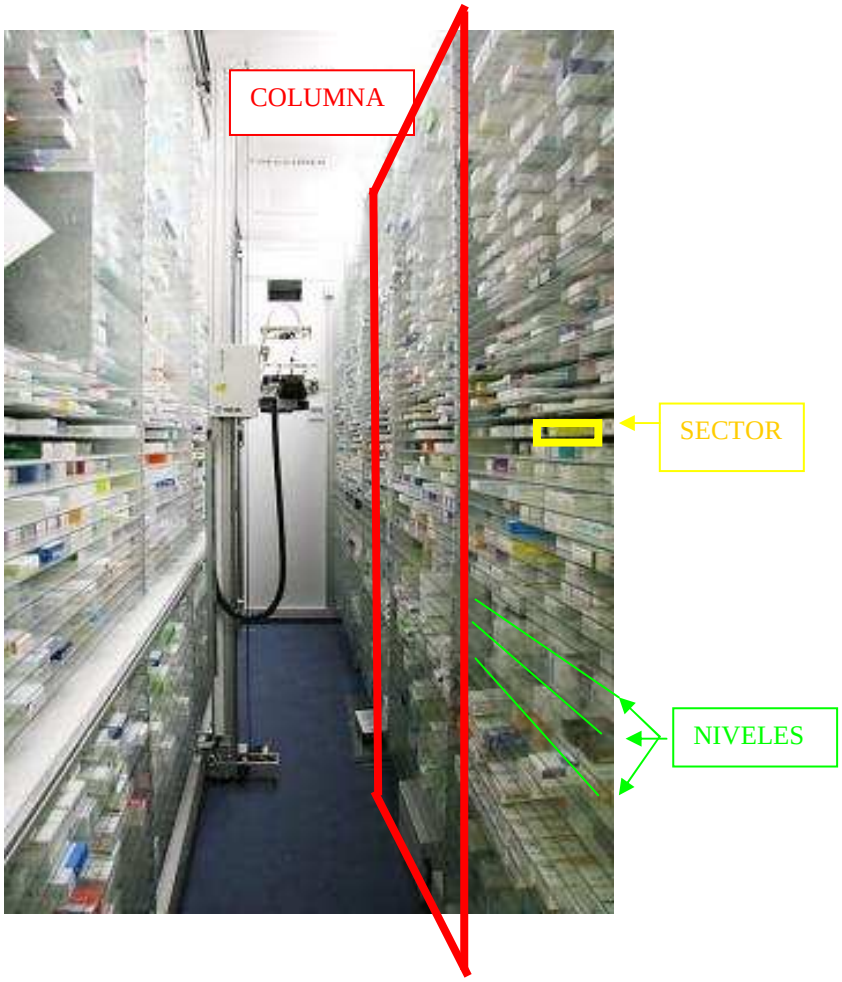


Figura 31: Explicación de un almacén robotizado

10. BIBLIOGRAFÍA

Documentación de Caché incluida en el CD de instalación.

- Documentación on-line de Caché en castellano:
  http://209.61.190.221/spain/cache/cachebasetemplate.csp

- Geometría
  http://www.personal.us.es/almar/docencia/practicas/localizacion/tema1.html#Teorema

  http://wwwdi.ujaen.es/asignaturas/gc/tema4.pdf

  http://wwwdi.ujaen.es/~jmserrano/teaching/geometriacomputacional/pdfs/transpa4.pdf

  Cálculo. James Stewart. Ed: Grupo Editorial Iberoamérica

  Calculus: una y varias variables. Salas, Hille y Etgen. Ed: Reverte

- Definición Datos Espaciales

  http://www.geogra.uah.es/gisweb/1modulosespanyol/SuperposicionVectorial/VOModule/VO_Theory.htm

- Sistemas de cajoneras de medicamentos:

  http://www.tecnyfarma.com/?page_id=812

  http://www.apotheka.com/productos/mobiliario-cajoneras/1cajoneras-diseno-reformas.aspx

- Bases de datos de medicamentos:

  https://botplusweb.portalfarma.com/botplus.asp

  http://www.portalfarma.com

  http://sescam.jccm.es/web1/profHome.do?main=/profesionales/farmacia/indiceFarmacia.html

- Bases de Datos:

  http://basesdatos.uc3m.es/

  Apuntes Diseño Software Avanzado (3º ITIG)

  Apuntes Base de Datos Avanzadas (3º ITIG)

  Diseño y Administración de Bases de Datos. Gary W. Hansen James V. Hansen. Ed: Prentince Hall.

  Fundamentos y modelos de Bases de Datos. Adoración de Miguel, Mario Piattini. Ed: Ra-ma.

Sistemas de Bases de datos: Un enfoque práctico para diseño, implementación y gestión. Thomas M. Connolly, Carolyn E. Begg. Ed: Addison Wesley.

Fundamentos de Bases de Datos. Silberschatz, Korth, Sudarshan. Ed: Mc Graw Hill.

Sistemas de bases de datos orientadas a objetos: Conceptos y arquitecturas. Elisa Bertino, Lorenzo Martino. Ed: Addison-Wesley

- Sistemas de Información Geográfica (SIG)

SIG: Sistemas de Información Geográfica, Javier Gutierrez Puebla, Michael Gould, Ed: Síntesis

http://www.topografiaglobal.com.ar/archivos/teoria/sig.html

- General:

http://www.google.es

http://es.wikipedia.org/

http://www.wordreference.com/