Tema 21. Modelos de dominio

Contenido

1. Introducción 1

2. Elementos de un modelo de Dominio 3

2.1. Entidades 5

2.2. Valores (Value Objects) 6

2.3. Servicios 7

2.4. Módulos 8

2.5. Agregados 8

2.6. Factorías 9

2.7. Repositorios 10

3. Relaciones entre tipos 10

3.1. Relación de uso 10

3.2. Generalización/Especialización 11

3.3. Asociaciones 11

3.4. Ejemplo de Modelo de Dominio 13

3.5. Expresiones sobre un dominio 15

4. Casos de uso 16

4. Introducción 17

4.1. Casos de uso básicos 18

4.2. Casos de uso 20

5. Diseño del interfaz web 21

6. Diseño de la Navegación 24

7. Capas de una aplicación 26

1. Introducción

La Informática está cada vez más presente en nuestra sociedad. La informática penetra y se integra prácticamente en todas las actividades y sectores sociales, y hoy no es posible prescindir de ella, puesto que han transformado la manera de trabajar, de aprender, y de comunicarse. Ha cambiado no solo la forma de actuar de los gobiernos sino también la forma de participar de los ciudadanos en política y de gestionar los servicios públicos.

Dentro de la informática el software se refiere a las instrucciones que se incorporan a un sistema informático para que este lleve a cabo una determinada función. El término software abarca desde pequeñas aplicaciones para llevar a cabo tareas muy específicas, a conocidos sistemas operativos con capacidad para realizar miles de funciones.

El software tiene que ser práctico y útil. Si no lo fuera no sería necesario invertir tanto tiempo y recursos en su diseño e implementación. Tiene que ser útil para un usuario fina, para cubrir una necesidad de la sociedad.

El diseño de software es un arte, y como cualquier arte que no se puede enseñar y no puede ser aprendido como una ciencia exacta, por medio de fórmulas y teoremas.

Podemos descubrir los principios y técnicas útiles para ser aplicadas durante todo el proceso de creación de software, pero probablemente nunca seremos capaces de proporcionar una ruta exacta que partiendo de una necesidad del mundo real alcance como objetivo una aplicación informática, un software, que la satisfaga.

Al igual que una imagen o un edificio, un producto de software incluirá el toque personal de quien lo diseñó y desarrolló, algo del carisma y el estilo (o la falta de ella) de los que han contribuido a su creación, diseño e implementación.

Hay diferentes maneras de abordar el diseño de software. La industria del software ha conocido y utilizado varios métodos para crear sus productos, cada uno con sus ventajas y deficiencias. Aquí vamos a seguir una metodología denominada Diseño guiado por el Dominio. El dominio es el contexto dónde se ubican la necesidad o necesidades que la creación del la herramienta informática quiere cubrir. Debemos entender desde el principio que el software se origina a partir de y profundamente relacionada con un dominio.

El software se compone de código. Podríamos ver el software sólo como código, sólo como una simple colección de objetos y métodos.

Pensemos en la fabricación de automóviles. Los trabajadores involucrados en la fabricación pueden especializarse en la producción de partes del coche, pero al hacerlo a menudo tienen una visión limitada del proceso completo de fabricación. Ven el coche como una gran colección de piezas que deben encajar, pero un coche es mucho más que eso. Un buen coche comienza con una visión. Se inicia con las especificaciones escritas cuidadosamente. Y continúa con el diseño. Meses, tal vez años de tiempo gastado en el diseño, el cambiándolo y refinándolo hasta que llega la perfección desde el punto de vista de los diseñadores. El procesamiento diseño s gran parte en papel, peo no es todo en el papel. Gran parte de este incluye hacer modelos del coche, y pruebas bajo ciertas condiciones para ver si funcionan. El diseño se

modifica sobre la base de los resultados de las pruebas. Posteriormente el coche se envía a producción, y las partes son creadas y ensambladas juntas.

El desarrollo de software es similar. No podemos simplemente sentarnos y empezar a escribir código. Podemos hacer eso, y puede funciona bien para los casos triviales, pero no podemos crear software complejo de esa forma.

Con el fin de crear un buen software, hay que conocer las necesidades que el software tiene que cubrir. No se puede crear un software bancario a menos que tenga una buena comprensión de lo que es la banca, hay que entender, por tanto el dominio de la banca. No es posible crear software bancario complejo sin un buen conocimiento del dominio bancario. ¿Quién conoce el dominio? El sistema bancario está muy bien entendido por las personas de su interior, por sus especialistas. Saben todos los detalles, todos los problemas posibles, todas las reglas.

Para construir un buen software aquí es donde siempre debemos comenzar: en la comprensión del dominio. Para ello el diseñador de software tiene que empezar extrayendo el conocimiento sobre el dominio que tienen los expertos en el mismo. Con ese conocimiento el diseñador de software debe hacer un modelo del dominio. Un modelo de dominio es una abstracción rigurosamente organizada y seleccionada del conocimiento que tienen los expertos.

Un modelo del dominio podría considerarse como un diccionario visual de las abstracciones relevantes, un vocabulario de los conceptos del dominio e información del mismo, sus restricciones, reglas de negocio, etc. Es, por lo tanto, una representación de las cosas del mundo real del dominio de interés y está formado por objetos que representan conceptos del mundo real y relaciones entre los mismos. El modelo de domino ofrece una descripción estática de la situación a modelar en un determinado instante del tiempo. En definitiva una descripción del estado del sistema.

Junto con el modelo de dominio hay capturar del usuario la forma en la que va a usar el software construido. Los escenarios de uso posibles desde el punto de vista del usuario. Estos escenarios de uso los denominaremos Casos de Uso y, también, tenemos que documentarlos. A partir de esos escenarios de uso es posible diseñar de forma abstracta le interfaz que la aplicación a la ofrecer a los posibles usuarios de la misma. Veamos con un poco más de detalle cada uno de estos aspectos. Los Casos de Uso describen los mecanismos disponibles para que los usuarios consulten el estado del sistema o lo modifiquen.

2. Elementos de un modelo de Dominio

Un modelo de dominio, como hemos dicho, está formado por objetos que representan conceptos del mundo real. El modelo de dominio tiene como objetivo describir, construir un modelo, del estado de un sistema en un instante dado del tiempo. Es, por lo tanto, una visión estática de un sistema.

El estado de un sistema se compone de un conjunto de objetos. Los objetos tienen propiedades y operaciones adecuadas para cambiar el valor de las propiedades.

El conjunto de objetos con las mismas propiedades y operaciones puede ser descrito mediante un **tipo**. Un tipo tiene una *extensión*, un conjunto de *propiedades*, un conjunto de *restricciones* sobre las mismas y un conjunto de *operaciones*. La extensión del tipo es el conjunto, posiblemente infinito, de instancias (objetos) que pueden ser construidos con propiedades que cumplan las restricciones dadas. Un objeto puede ser creado y eliminado y tiene un *estado* formado por los valores de sus propiedades. En un contexto determinado un tipo tiene asociado, también, una *población* formada por el conjunto de instancias del tipo creadas y no eliminadas. Las *operaciones* son mecanismos para cambiar el estado interno de u objeto o para crear objetos nuevos a partir de otros objetos. Una secuencia de operaciones sobre instancias de objetos forma una expresión.

Un ejemplo de un tipo es el tipo *Entero*. Su extensión está formada por todos los enteros posibles. Una propiedad es si es par o no. Una operación es la suma de enteros que toma dos enteros y produce como resultado otro entero. Igualmente tenemos el tipo *Real*.

Un segundo ejemplo de tipo es el *Boolean*. Su extensión es $\{false, true\}$ y algunas de sus operaciones son las operaciones lógicas *and, or* que denotaremos por los operadores $\&\&\ y\ ||$.

Una propiedad de un tipo viene definida por su nombre, el tipo de los valores que puede tomar y posiblemente algunas restricciones que debe cumplir. Una propiedad puede ser modificable o no según que se permita modificar su valor o no. Individual o compartida según que su valor sea individual para cada objeto o compartido por todos los objetos de la población del tipo. Derivada o básica según que su valor esté ligado o no al valor de otras propiedades.

La modificabilidad de una propiedad de un tipo o una asociación entre tipos indica que sus valores pueden ser cambiados.

Una propiedad de un tipo o una asociación entre tipos es derivada si su valor puede calcularse a partir de otras propiedades.

Un tercer ejemplo es el tipo *Círculo* con propiedades:

- *Radio*, Real, Modificable, Mayor o igual que cero, Individual, Básica
- *Centro*, Punto2D, Modificable, Individual, Básica
- *Área*, Double, Individual, Derivada ($= \pi\mathrm{Radio\^{}2}$)
- *Origen,* Punto2D, Compartida, Básica

Un tipo que tiene todas sus propiedades no modificables se denomina **tipo inmutable**.

Un tipo puede tener restricciones sobre los valores que pueden tener sus propiedades o las asociaciones con otros tipos. El conjunto de restricciones del tipo lo llamamos su *invariante*. El concepto de invariante puede ser extendido a un conjunto de tipos, un módulo o un modelo completo.

En el caso anterior el invariante está compuesto por las expresiones

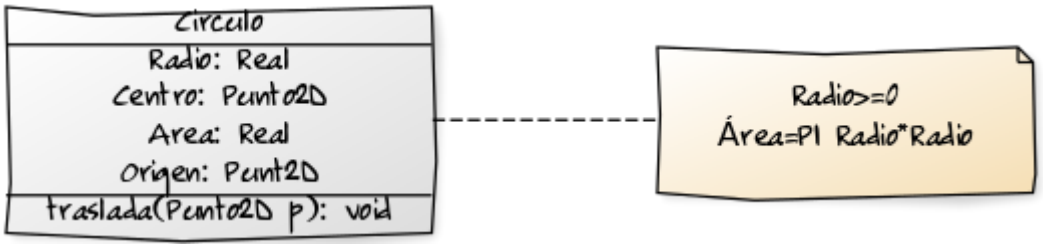
```
Radio >= 0;
Area = 2*PI*Radio^2;
```

Un tipo puede tener también operaciones que son mecanismos para cambiar los valores de las propiedades. En el caso anterior el tipo Círculo podría tener la operación traslada capaz de trasladar el centro del círculo a un punto *p.*

```
traslada(Punt2D p): void
```

Cada operación tiene a un objeto determinado como argumento implícito y el comportamiento de la operación depende del tipo de este objeto.

Gráficamente un tipo lo representaremos de la forma:



A un tipo le podemos asociar una nota dónde podemos explicitar el invariante o algún comentario adicional. El tipo *Círculo* usa los tipos *Real* y *Punto2D* que se suponen ya definidos.

Dado un tipo podemos declarar variables y construir expresiones a partir de las variables declaradas, las constantes del tipo, las propiedades y operaciones del mismo. Un ejemplo sería

```
Entero x = 3*5+2;
```

Con los elementos anteriores podemos decir que un modelo de dominio se compone de un conjunto de tipos, un conjunto de relaciones entre los mismos, formas de agregarlos y mecanismos para crear instancias o agregados de instancias.

Los tipos, a su vez, se pueden clasificar en *Entidades* y *Valores*. Las relaciones entre los tipos pueden ser *Asociaciones*, *Dependencias* o *Relaciones de Uso* y *Generalizaciones-Especializaciones* también denominadas *Relaciones de Herencia*. En lo siguiente iremos viendo cada uno de esos conceptos.

2.1. Entidades

Hay una categoría de objetos que parecen tener una identidad, que sigue siendo el mismo a lo largo de los diferentes estados por los que va pasando. Tales objetos se denominan *Entidades*.

Si tuviéramos que modelar el concepto de persona usando un programa de software, probablemente crearíamos el tipo Persona con una serie de propiedades: nombre, fecha de nacimiento, lugar de nacimiento, etc. ¿Pero definen estas propiedades la identidad de la persona? El nombre no porque puede haber más personas con el mismo nombre. No podíamos distinguir entre las personas con el mismo nombre, si tuviéramos que tener en cuenta sólo su nombre. Igualmente ocurre con la fecha de nacimiento porque hay muchas personas que nacieron en el mismo día. Lo mismo se aplica al lugar de nacimiento. Una entidad debe ser distinguida de otras a pesar de las mismas propiedades. Identidad equivocada puede conducir a la corrupción de datos.

Veamos el caso de una cuenta bancaria. Cada cuenta tiene su propio número. Una cuenta se puede identificar precisamente por su número. Este número se mantiene sin cambios durante toda la vida del sistema, y asegura la continuidad. Por lo tanto, la implementación de las entidades en el software significa definir y crear su identidad. Para una persona ser una combinación de propiedades: nombre, fecha de nacimiento, lugar de nacimiento, nombre de los padres, la corriente dirección. El DNI se podría utilizar para crear su identidad. Para una cuenta bancaria al número de cuenta parece ser suficiente para su identidad. Por lo general, la identidad es o bien una propiedad del objeto, una combinación de las mismas, una propiedad especialmente creada para preservar y expresar la identidad, o incluso un comportamiento.

Hay diferentes maneras de crear una identidad única para cada objeto. Una posibilidad es crear la propiedad ID y generar automáticamente los valores para la misma. El ID puede ser creado específicamente como ocurre con los códigos asociados a los aeropuertos. Cada aeropuerto tiene una cadena única ID que es reconocida y utilizada internacionalmente para identificar los aeropuertos.

Las entidades son objetos importantes de un modelo de dominio, y deben ser considerados desde el comienzo de la modelización proceso.

2.2. Valores (Value Objects)

Hay otros tipos de objetos dónde no estamos interesados en qué objeto se trata, solamente qué atributos que tiene. Un objeto que se utiliza para describir ciertos aspectos de un dominio, y que no tiene identidad, es un tipo *Valor*.

Al no tener identidad, los objetos pueden ser fácilmente creados y descartados. Esto simplifica el diseño mucho.

Es altamente recomendable que los objetos valores sean de tipos inmutables. Si son inmutables se crean con un constructor, y nunca son modificados durante su tiempo de la vida. Cuando se desea un valor diferente para el objeto, simplemente se crea otro nuevo. Esto tiene consecuencias importantes para el diseño. Un objeto inmutable, y sin identidad, es decir un valor inmutable, pueden ser compartidos. Eso puede ser imprescindible para algunos diseños.

Una regla de oro es: si objetos de tipo valor son compartibles, entonces deberían ser de tipo inmutables. Cuando un nuevo valor se necesita por otra parte, una copia del mismo o un nuevo valor pueden ser creados. Realizar una copia de un objeto valor es simple, y por lo general sin ningún tipo de consecuencias. Si no hay identidad, se pueden hacer la cantidad de copias que se desee, y destruirlas cuando sea necesario.

Los objetos valor pueden contener otros objetos valor, y pueden incluso contener referencias a entidades.

2.3. Servicios

Cuando analizamos el dominio y tratamos de definir los principales tipo de objetos que conforman el modelo, descubrimos que algunos aspectos del dominio no son fácilmente asignables a los objetos. Normalmente cada objeto tiene propiedades, un estado interno que es gestionado por el objeto, y exhibe un comportamiento. Usualmente los conceptos clave del dominio se introducen en la lengua, y los nombres de la lengua son fácilmente asignados a los tipos de objetos. Los verbos de la lengua son asociados con sus correspondientes nombres y se convierten en la parte del comportamiento de esos objetos. Pero hay algunas acciones en el dominio, algunos verbos, que no parecen pertenecer a ningún objeto. Representan un comportamiento importante del dominio, por lo que no puede ser descuidado o simplemente incorporado en algunas de las *Entidades* o *Valores* del dominio. La adición de este comportamiento a un tipo del dominio podría confundir el concepto que queremos captar con el mismo. Cuando un comportamiento de esas características es reconocido en el dominio, la mejor práctica es declararlo como un *Servicio*. Un servicio, en la mayoría de los casos, es un objeto que no tiene un estado interno, y su propósito es proporcionar simplemente funcionalidad para el dominio como un todo o para otros tipos del dominio. La asistencia prestada por un servicio puede ser muy importante y es mucho mejor declarar el *Servicio* de forma explícita, ya que se encapsula un concepto.

Los *Servicios* actúan como interfaces que proporcionan operaciones que involucran a otros objetos. De esta manera, un servicio, por lo general, se convierte en un punto de conexión para muchos objetos. Si tal funcionalidad se incluye en objetos de dominio, se crea una densa red de asociaciones entre ellos. Un alto grado de acoplamiento entre muchos objetos es un signo de un mal diseño, ya que hace que el código resultante sea difícil de leer y entender, y lo más importante, que hace que sea difícil cambiar.

Un servicio no debe sustituir a la operación que normalmente pertenece a los objetos del dominio. No debemos crear un servicio para cada operación necesaria. Pero cuando tal operación se destaca como un concepto importante en el dominio, un servicio debe ser creado para ella.

Hay tres características de un servicio:

1. La operación realizada por el Servicio se refiere a un concepto del dominio que no es naturalmente asignable a una Entidad o a un Valor.

2. La operación realizada por el servicio usa o modifica otros objetos del dominio.

3. La operación no tiene estado.

2.4. Módulos

Para una aplicación grande y compleja, el modelo tiende a crecer más. El modelo alcanza un punto donde es difícil visualizarlo como un todo, y la comprensión de las relaciones y las interacciones entre las diferentes partes se hace difícil. Por esa razón, es necesario organizar el modelo en *módulos*.

Los módulos se utilizan como un método de organizar conceptos relacionados con el fin de reducir la complejidad.

Los módulos son ampliamente utilizados en la mayoría de los proyectos. Es más fácil conseguir la imagen global de un modelo grande si nos fijamos en los módulos que contiene, a continuación, en las relaciones entre esos módulos. Es una simple y forma eficaz de gestionar la complejidad.

Hay dos conceptos relacionados con los módulos y su diseño: la *cohesión* y el *acoplamiento* El grado de cohesión mide la coherencia del módulo, esto es, lo coherente que es la información que almacena, lo coherente del conjunto de entidades y relaciones que contiene y la claridad de la responsabilidades del módulo. Un módulo tiene alta cohesión si todos sus elementos se relacionan fuertemente y todos ellos se utilizan para lograr un objetivo común, que es la función del módulo.

El grado de acoplamiento entre módulos indica lo vinculados que están unos a otros, es decir, lo que afecta un cambio en uno de ellos a los demás y por tanto lo dependientes que son unos de otros. Decimos que hay dependencia entre módulos cuando hay entidades o valores de un módulo que dependen o usan entidades o valores de otro, sea por herencia, asociación o simple dependencia débil. Un módulo debe tener vinculaciones mínimas con otros módulos.

Como se ve claramente, los conceptos de cohesión y acoplamiento están íntimamente relacionados. Un mayor grado de cohesión implica uno menor de acoplamiento. Maximizar el nivel de cohesión intramodular en todo el sistema resulta en una minimización del acoplamiento intermodular.

La *cohesión* tiene que ver con que cada unidad del modelo (módulo, entidad, servicio, valor,…). A mayor cohesión, mejor: la unidad en cuestión será más sencilla de diseñar, programar, probar y mantener. En los servicios, se logra alta cohesión cuando cada servicio realiza una única tarea.

Un bajo acoplamiento reduce la complejidad y aumenta la mantenibilidad. Es más fácil entender cómo funciona un sistema cuando hay pocas conexiones entre los módulos que realizan tareas bien definidas, que cuando cada módulo tiene muchas conexiones con los demás módulos.

2.5. Agregados

Un agregado es un grupo de objetos asociados que se consideran como una unidad. Un agregado está demarcado por un límite que separa los objetos dentro de los que están fuera. Cada agregado tiene una raíz. La raíz es una entidad, y es el único objeto accesible desde el exterior del agregado. La raíz puede contener referencias a cualquiera de los objetos agregados, y los otros objetos pueden contener referencias entre sí, pero un objeto exterior puede sostener referencias sólo al objeto raíz. Si hay otras Entidades dentro de los límites, la identidad de esas entidades es local, tener sentido sólo dentro del agregado.

¿Cómo se asegura la integridad de los datos agregados y la aplicación de los invariantes o restricciones entre las propiedades de los objetos? Desde otros objetos pueden contener referencias únicamente a la raíz, lo que significa que no pueden cambiar directamente los otros objetos del agregado. Todo lo que pueden hacer es cambiar la raíz, o pedir a la raíz para realizar algunas acciones. Y la raíz será capaz de cambiar los otros objetos, pero que es una operación que figura en el interior del agregado, y es controlable. Si se elimina la raíz, todos los demás objetos del agregado se eliminarán también.

2.6. Factorías

Es necesario introducir un concepto y un mecanismo para encapsular el proceso de creación de objetos complejos. Este se llama *Factoría*. Las factorías se utilizan para encapsular el conocimiento necesario para la creación de objetos, y son especialmente útiles para crear agregados. Cuando la raíz del agregado se crea también todos los objetos contenidos en el agregado respetando todos los invariantes se deben cumplir.

Es importante que el proceso de creación para ser *atómico*. Es decir que se haga completamente o no se haga. Si no lo atómico, hay posibilidades para que el proceso de creación quede a medio terminar, dejando el agregado en un estado indefinido.

Para inmutable objetivo de tipo valor, todos los atributos se deben inicializar a su estado válido. Si un objeto no se puede crear correctamente, una excepción debe ser levantada, asegurándose de que un valor no válido no es devuelto.

Por lo tanto, trasladar la responsabilidad de crear las instancias de objetos y agregados complejos a un objeto separado (la factoría) es una buena estrategia. La factoría debe proporcionar una interfaz que encapsule todo montaje de objetos u agregados complejos y que no requiera que el cliente conozca los detalles de los objetos que se crean.

Hay momentos en que no se necesita una factoría, y un simple constructor es suficiente. Utilice un constructor cuando:

- La construcción no es complicada.
- La creación de un objeto no implica la creación de otros, y todos los atributos necesarios son pasado a través del constructor.
- El cliente está interesado en los detalles de la construcción y tal vez quiera elegir la estrategia utilizada.
- La entidad o valor tiene un tipo que viene definido por una clase.

Las factorías tienen que crear objetos nuevos a partir de cero, reconstituir objetos que han existido previamente, pero que han sido probablemente guardados en una base de datos. Traer Entidades de nuevo a la memoria desde una base de datos implica un proceso completamente diferente de crear uno nuevo. Una diferencia obvia es que el nuevo objeto no necesita una nueva identidad. El objeto ya tiene una.

Violaciones de los invariantes son tratadas de manera diferente. Cuando un nuevo objeto es creado a partir de cero, cualquier violación de los invariantes debe disparar una excepción.

2.7. Repositorios

El propósito de un repositorio es encapsular toda la lógica necesaria para obtener referencias a objetos. Los objetos de dominio no tendrán que lidiar con la infraestructura necesaria para conseguir las referencias necesarias a otros objetos del dominio.

El repositorio puede almacenar referencias a algunos de los objetos. Cuando se crea un objeto, éste se puede guardar en el repositorio, y recuperarlo de allí para ser utilizado más tarde. Si el cliente solicitó un objeto del repositorio, y el repositorio no lo tiene, puede obtenerlo del almacenamiento. De esta manera podemos considerar el repositorio como un lugar de almacenamiento para objetos globalmente accesibles. El efecto general es que el modelo de dominio se desacopla de la necesidad de almacenar los objetos o sus referencias, y el acceso a la infraestructura de persistencia subyacente.

Para cada tipo de objeto que necesita acceso global es conveniente crear un objeto que puede proporcionar la ilusión de una colección en memoria de todos objetos de ese tipo. Proporcionar métodos para añadir y eliminar objetos, que encapsulará la inserción real o eliminación de los datos en el almacén de datos. Proporcionar métodos que seleccionan objetos basado en algunos criterios y devolver objetos o colecciones de objetos completamente instanciados y cuyas propiedades cumplan algunos criterios. Por lo tanto encapsular el almacenamiento real y la tecnología de la consulta.

El cliente llama a un método tal y pasa uno o más parámetros que representan los criterios de selección utilizados para seleccionar un objeto o un conjunto de objetos coincidentes. Una entidad puede ser fácilmente especificada por su identidad. Otros criterios de selección pueden ser formados por un conjunto de propiedades de los objetos.

3. Relaciones entre tipos

Entre los tipos de un modelo de dominio puede haber relaciones de uso, de generalización/especialización y asociaciones. Veamos cada una de ellas

3.1. Relación de uso

En general si un tipo *T1* tiene una propiedad de tipo *T2* decimos que el tipo *T1* usa el tipo *T2*. Igualmente *T1* usa *T2* si *T2* aparee en los parámetros de alguna de sus propiedades. El conjunto de relaciones de uso entre tipos define un grafo donde se explicita que tipo usa a quien.

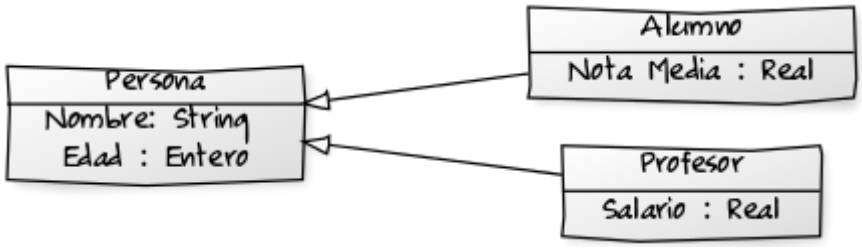
Hay un conjunto de tipos que no usan a otros. Se les suele denominar tipos básicos. Cada lenguaje de programación aporta un conjunto de tipos básicos. Los demás tipos los tenemos que definir a partir de estos o de otros que hayamos definido previamente.

Los tipos básicos más usuales son *Entero* (en sus distintas variantes), *Real* (en sus distintas variantes), *Carácter*, *Hilera de Caracteres (String)*, *Boolean*, etc. Otros tipos usuales que suelen venir proporcionados por los lenguajes de programación son el tipo *Fecha* y distintas modalidades del tipo numérico como *Racional*, *Complejo*, etc.

3.2. Generalización/Especialización

Esta relación entre tipo permite agrupar, en un tipo *T0*, propiedades y operación es comunes a varios tipos *T1, T2*. A *T1* y *T2* se les denomina subtipos de *T0* y a *T0* supertipo. Todos los subtipos comparten las propiedades, operaciones e invariante del supertipo. Esta relación mirada desde el punto de vista de los subtipos se denomina generalización y desde el punto de vista del supertipo especialización.

La forma de representar gráficamente esta relación es de la forma:



Para precisar con más detalle el carácter de la relación de generalización en algunos casos se suele añadir un *discriminador* que indica el nombre de la generalización y la forma en la que dividen las poblaciones de los tipos involucrados que pude ser:

- *Disjunta (disjoint):* Un objeto sólo pertenece a la población de uno de los subtipos
- *Solapada (overlapping):* Un objeto puede pertenecer a varias poblaciones de los subtipos
- *Completa (complete):* La población del supertipo es la unión de a la población de los subtipos
- *Incompleta (incomplete):* La población del supertipo es incluye estrictamente a la unión de las poblaciones de los subtipos. En este caso puede haber más subtipos.

3.3. Asociaciones

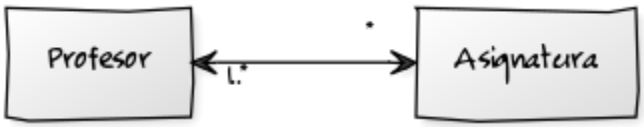
Una asociación entre tipos, permite asociar objetos que colaboran entre sí. Una asociación es una relación entre tipos (o más concretamente, instancias de estos tipos) que indica alguna conexión significativa e interesante.

Las asociaciones que merece la pena tener en cuenta, normalmente, implican conocimiento de una relación que es necesario conservar durante algún tiempo.

Una asociación puede ser reflexiva en el sentido de relacionar un tipo consigo mismo. Es decir instancias de un tipo con otras del mismo tipo.

Cada asociación tiene un nombre y unos elementos adicionales que son los siguientes:

- *Rol:* Cada asociación tiene dos roles (uno para cada extremo de la asociación). Cada rol se identifica con un nombre que aparece en un extremo de la línea que denota la asociación. Dicho nombre describe la semántica que tiene la relación en el sentido indicado.
- *Navegabilidad:* La navegabilidad es una propiedad de un rol. Indica la posibilidad de ir desde el objeto fuente al objeto destino. En un extremo de una asociación se puede indicar la navegabilidad mediante una flecha. Significa que es posible navegar desde el objeto de la clase origen hasta el objeto de la clase destino y por tanto poder llamar a alguna de sus operaciones o consultar sus propiedades. Una asociación navegable en un solo sentido se denomina unidireccional y si lo es en los dos bidireccional. En un primer momento las asociaciones no tendrán navegabilidades. Posteriormente, se agregarán las navegabilidades necesarias en los refinamientos progresivos del modelo.
- *Multiplicidad o Cardinalidad:* La multiplicidad de un rol determina cuantos objetos de ese rol pueden intervenir en la relación. Usualmente la multiplicidad se define mediante un par de enteros que establecen el valor mínimo y máximo de instancias de ese rol que pueden relacionarse con una instancia del rol opuesto. Cuando la multiplicidad mínima es 0, la relación es opcional. Una multiplicidad mínima es mayor o igual a 1 se establece una asociación obligatoria. Muy a menudo se usa una notación simplificada para indicar el rango de valores permitido:
    - Cero o más es representado por *
    - Uno o más por 1..*
    - Exactamente uno por 1

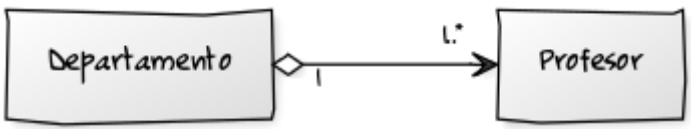


En el ejemplo se muestra que un profesor tiene asignaturas, aunque puede que no tenga ninguna, y cada asignatura tiene al menos un profesor, aunque puede tener más.

Agregación

La agregación es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye (el objeto base utiliza al incluido para su funcionamiento).

- Tipo de asociación que permite modelar relaciones parte-todo
- Las cadenas de agregaciones no pueden formar ciclos
- La multiplicidad del extremo del compuesto puede ser más de una
- Puede ser reflexiva para los tipos, pero no para las instancias.



En el ejemplo un departamento se compone de un conjunto de profesores y a su vez cada profesor está en un único departamento.

Composición

La composición es un tipo de relación estática, en donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye (el objeto base se construye a partir del objeto incluido, es decir, es parte/todo).

Es un tipo especial de agregación no compartida donde:

- La multiplicidad del extremo del compuesto no puede ser mayor a uno
- Ninguna parte puede existir sin el todo
- Las operaciones de copiado y borrado se propagan del agregado a las partes

En el ejemplo anterior el expediente de un alumno se compone de un conjunto de notas.

3.4. Ejemplo de Modelo de Dominio

Veamos como ejemplo el modelo de dominio de un centro universitario que incluye los siguientes conceptos.

Las *asignaturas* son uno de los participantes fundamentales en el problema. Cada asignatura se identifica mediante un nombre (por ejemplo Fundamentos de Programación), un acrónimo (que se construye a partir de las mayúsculas que aparecen en el nombre de la asignatura, por ejemplo FP) y un código numérico de 7 dígitos (por ejemplo, 0000230). Cada asignatura tiene un número determinado de créditos (se admiten asignaturas con número no entero de créditos). Las asignaturas pueden ser de tres tipos, según el periodo del curso en el que se imparten: anual, de primer cuatrimestre y de segundo cuatrimestre. Cada asignatura es impartida en un curso, y su docencia es asumida por un departamento concreto. Ninguna de las propiedades que definen a una asignatura, salvo el departamento, varía una vez creada la misma.

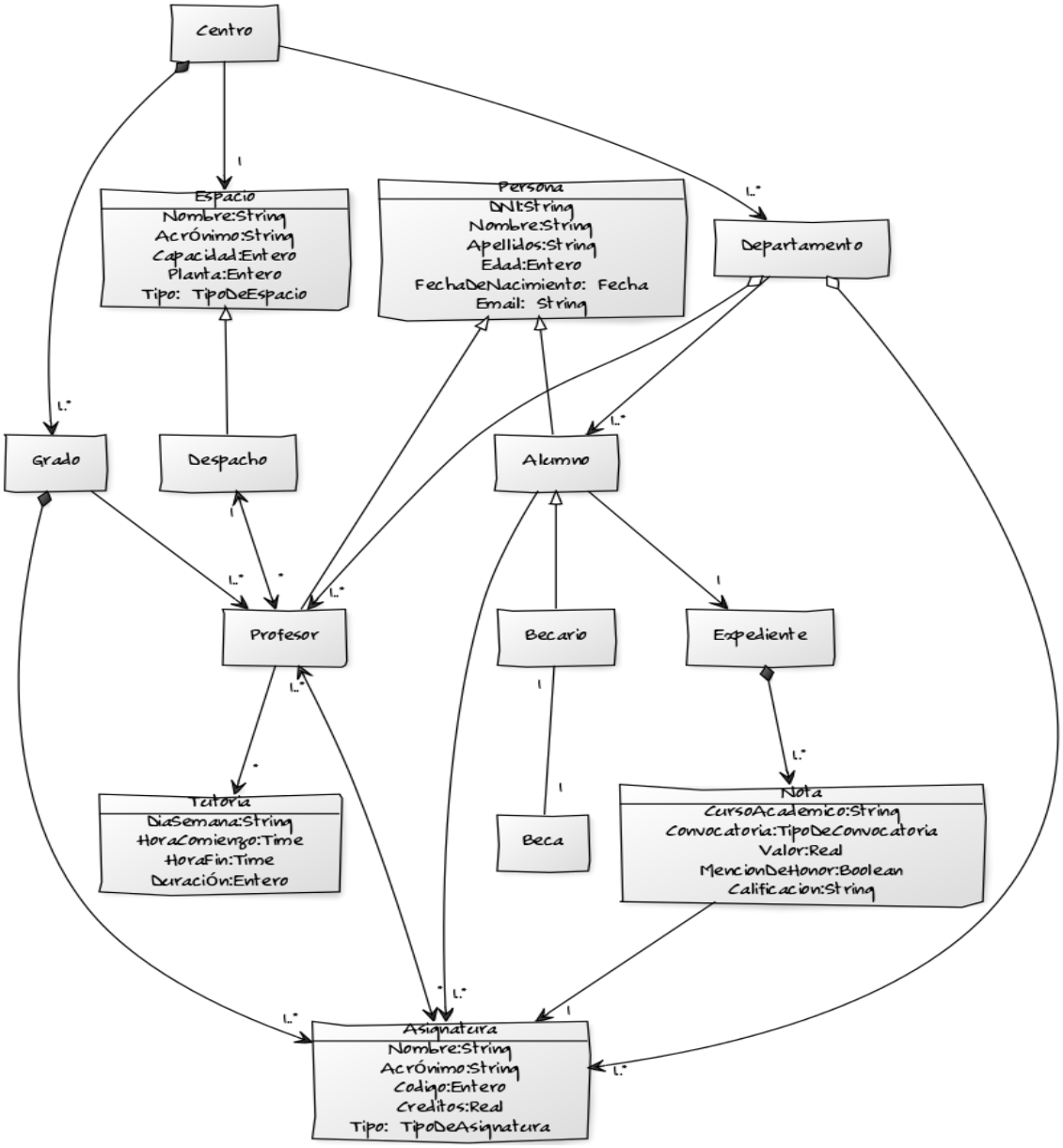
Otro elemento participante en el problema son las *becas* que en ocasiones son concedidas a los *alumnos*. Las becas se identifican mediante un código alfanumérico único. Cada beca tiene asignada una cuantía total en euros (puede contener decimales), y una duración en meses (las becas siempre duran un número determinado de meses completos). Existen becas de tres tipos: ordinaria, de movilidad y de empresa. Para cada beca es necesario conocer su cuantía mensual, que se calcula dividiendo la cuantía total entre el número de meses que dura la beca. Una vez concedida una beca su código y su tipo nunca cambian, no así la cuantía y la duración, que pueden variar debido a alegaciones por parte de los alumnos o por cambios legislativos.

El tipo *persona* se utiliza para modelar a todas las personas que participan en la aplicación. Cada persona tiene un DNI (formado siempre por ocho dígitos y una letra), un nombre, unos apellidos, una fecha de nacimiento, una dirección de email y una edad. Todas las propiedades de la persona pueden variar una vez creada.

Un *espacio* representa un lugar físico ubicado en un centro y en el cual se realizan tareas docentes. Un espacio puede ser de varios tipos: un aula de teoría, un laboratorio, un seminario, un aula de examen o de otro tipo. Cada espacio tiene un nombre (por ejemplo, A3.10) y una capacidad dada por el número máximo de personas que admite. Además, un espacio está ubicado en una determinada planta. Todas las propiedades de un espacio pueden variar una vez creado, con excepción de la planta.

Una *nota* representa la calificación obtenida por un alumno en una asignatura de un curso académico concreto, dado por el año de comienzo del mismo (por ejemplo, 2014 para el curso 2014/15). La nota corresponde a una convocatoria, que puede ser la primera, la segunda, o la tercera, y tiene un valor numérico comprendido entre 0 y 10. Si el valor numérico es mayor o igual a 9, la nota puede ser distinguida con una mención de honor. La nota también tiene una calificación, que se calcula a partir del valor numérico y de la mención de honor, y que puede ser suspenso (si el valor numérico es menor que 5), aprobado (si el valor numérico es mayor o igual que 5 y menor que 7), notable (si el valor numérico es mayor o igual que 7 y menor que 9), sobresaliente (si el valor numérico es mayor o igual que 9 y la nota no tiene mención de honor) o matrícula de honor (si el valor numérico es mayor o igual que 9 y la nota tiene mención de honor). Ninguna de las propiedades que definen a una nota varía una vez creada la misma.

La *tutoría* representa un intervalo de tiempo que todo profesor tiene reservado para atender a sus alumnos. Una tutoría tiene lugar un día de la semana (de lunes a viernes1) y tiene una hora de comienzo, una hora de fin y una duración en minutos, que es la diferencia entre la hora de fin y la hora de comienzo. Ninguna de las propiedades que definen a una tutoría varía una vez creada la misma.



3.5. Expresiones sobre un dominio

Cuando tenemos un dominio modelado disponemos de un conjunto de tipos y unas relaciones entre los mismos. Usualmente disponemos también de un conjunto de tipos básicos que ya tienen definidos un conjunto de propiedades y operaciones (que en muchos casos vienen definidos como operadores). Los tipos básicos usuales, sus valores y algunos operadores son:

- Entero:
  - 1,5,-45,…
  - +,-,*,/, <,<=,…
- Real:
  - 1.,5.3,-4.5,…
  - +,-,*,/,<,<=,…
  -
- Boolean:

    - true, false
    - &&,||,!, …
- String:
    - “Antonio”, “Viernes”, …
    - +,<,<=,…
- Carácter:
    - ‘h’,’8’,…
    - <,<=,…
- Fecha:
    - 2/10/2014,…
    - <,<=,…

Usualmente los lenguajes de programación también ofrecen tipo para modelar agregados de objetos:

- Conjuntos:
    - {1,7,-1}, {6-7,8.9,-5.6}, {“Antonio”,”Juan”,”Pedro”,”Andrés”},…
    - Unión, Intersección, Diferencia, Inclusión, Contiene, Tamaño, …
- Secuencias:
    - [1,7,-1,1], [6-7,8.9,-5.6,8.9], [“Antonio”,”Juan”,”Pedro”,”Andrés”,”Andrés”],…
    - Concatenar, Contiene, Tamaño, …

La diferencia más importante entre estos dos tipos de agregados es que en los conjuntos los elementos no se pueden repetir y están esencialmente desordenados. En las secuencias los elementos se pueden repetir y entre ellos existe un orden definido por la posición que ocupan en la secuencia.

Los tipos que hemos definido en el modelo de dominio tienen unas propiedades y unas relaciones (que podemos pensar como un tipo especial de propiedades). A partir de una instancia de un tipo, las propiedades y las relaciones podemos construir expresiones. Para ello el operador punto.

Si la instancia *x* ha sido declarada de tipo Persona la expresión *x.Nombre* es válida y representa un valor de tipo *String* con el nombre de la instancia x. Las relaciones pueden usarse de forma similar para construir expresiones. Si la instancia *y* ha sido declarada de tipo Departamento la expresión *x.Profesor* es válida y representa un valor de tipo *Conjunto de Profesores* que contiene los profesores del departamento y. Si existe un nombre de rol *R* cuando navegamos de un tipo *T1* a otro *T2* a otro entonces la expresión que podemos formar es *x.R* siendo *x* una instancia de *T1*, *R* el rol en el lado de *T2* y siendo la asociación navegable en ese sentido. Si no existe nombre de rol entonces usamos la expresión *x.T2*. Esta expresión será un conjunto o un elemento según la multiplicidad asociada.

4. Casos de uso

4. Introducción

Un caso de uso es una descripción de los pasos o las actividades que se deberán realizar para conseguir algún objetivo de un sistema dado. Los personajes o entidades que participarán en un caso de uso se denominan actores. Un caso de uso es una secuencia de interacciones que se desarrollarán entre un sistema y algunos actores. Cada caso de uso se centra en describir cómo alcanzar una única meta o finalizar una tarea. Un caso de uso se inicia por un actor principal sobre el propio sistema. Los casos de uso no describen ninguna funcionalidad interna del sistema, ni explican cómo se implementará. Simplemente muestran los pasos que el actor sigue para realizar una operación.

Los casos son adecuados para describir la dinámica del sistema. Es decir cómo cambia el estado de un sistema, descrito por su modelo de dominio, cuando algunos actores interactúan con el mismo.

Se le llama actor a toda entidad externa al sistema que guarda una relación con éste y que le demanda una funcionalidad. Esto incluye a los operadores humanos pero también incluye a todos los sistemas externos.

En el caso de los seres humanos se pueden ver a los actores como definiciones de roles por lo que un mismo individuo puede corresponder a uno o más actores.

Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema.

La técnica de caso de uso tiene éxito en sistemas interactivos, ya que expresa la intención que tiene el actor (su usuario) al hacer uso del sistema.

Entre los casos de uso y entres estos y los actores hay diversos tipos de relaciones. Un actor se *comunica* con un caso de uso (si se trata de un actor primario la comunicación la iniciará el actor, en cambio si es secundario, el sistema será el que inicie la comunicación). Un caso de uso *extiende* otro caso de uso. Un caso de uso *incluye* otro caso de uso.

Se utiliza una relación de tipo *extiende* entre casos de uso cuando nos encontramos con un caso de uso similar a otro pero que hace algo más que éste. Por contra, utilizaremos una relación tipo *incluye* cuando nos encontramos con una parte de comportamiento similar en dos casos de uso y no queremos repetir la descripción de dicho comportamiento común.

La técnica de casos de uso extracción permite centrarse en las necesidades del usuario, qué espera éste lograr al utilizar el sistema, evitando que los especialistas en informática dirijan la funcionalidad del nuevo sistema basándose solamente en criterios tecnológicos.

Para describir un caso de uso necesitamos los siguinetes elementos:

Nombre	nombre del caso de uso
Autor	nombre del autor (o autores) del caso de uso
Fecha	fecha de creación del caso de uso

Descripción	breve descripción del caso de uso
Actores	actores participantes en el caso de uso
Precondiciones	condiciones que deben cumplirse para poder ejecutar el caso de uso
Poscondiciones	condiciones que deben cumplirse al finalizar la ejecución del caso de uso
Flujo Normal	flujo normal de ejecución del caso de uso
Flujo Alternativo	flujos alternativos de ejecución del caso de uso

4.1. Casos de uso básicos

Dado un modelo de dominio se deducen un cojunto de casos de uso que vamos a denominar básicos. Son casos de uso que siempre aparecen y que vienen determinados por los tipos que aparecen en el modelo de dominio.

Estos casos de uso son los denominados CRUD (*Create, Read, Update y Delete*). Es decir asociado a cualquier tipo de los que aparecen en el modelo de dominio aparecen casos de uso adecuados para crear una instancia del tipo, leerla del almacén donde se encuentre, actualizar los valores de sus propiedades y eliminar una instancia. A estos casos de uso añadiremos uno que liste todas las instancias del tipo que llamaremos *Listar*.

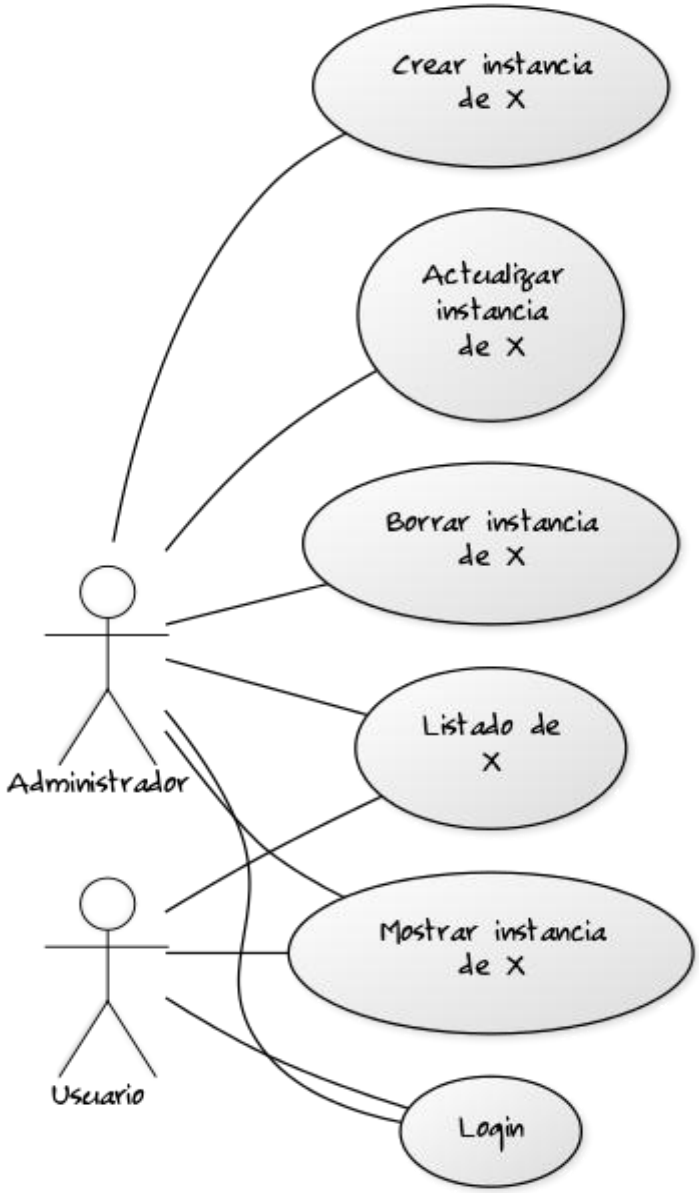
Normalmente hay un par de casos de uso más que llamaremos *Login* y *Asigna Rol*. El primero es un caso de uso dónde un usuario se autentica ante el sistema para poder usarlo. El segundo es un caso de uso dónde a un usuario se le asignan un o o varios roles.

Para cada caso de uso anterior y cada tipo debemos decidir el tipo de Actor participante. O dicho de otra forma el rol que debe tener el usuario del sistema para poder participar en ese caso de uso. Es decir para que se le permita llevar a cabo las operaciones del caso de uso.

Adicionalmente los casos de uso anteriores definen un conjunto de servicios que debe ofrecer el sistema. Recordamos que un servicio tiene como objetivo proporcionar una funcionalidad para el dominio como un todo. Es muy importante declarar el *Servicio* de forma explícita e implementarlo como tal.

Por lo tanto por cada tipo debemos definir los servicios CRUD y el servicio Listar y por la misma razón los servicios *Autenticar* y *Asignar Rol*.

Gráficamente los casos de uso anteriores podemos representarlos de la forma:



Los detalles del caso de uso Listado de X podrían ser:

Nombre	Listado de X
Autor	Miguel Toro
Fecha	26/11/2014
Descripción	Listado de todas las instancias de la entidad X
Actores	Usuario, Administrador
Precondiciones	El actor está identificado en el sistema
Poscondiciones	El estado del sistema no cambia
Flujo Normal	1. Se elige una entidad X 2. Se muestran todas las instancias de la misma
Flujo Alternativo	

Los detalles del caso de uso Actualizar podrían ser:

Nombre	Actualizar instacia de X
Autor	Miguel Toro
Fecha	26/11/2014
Descripción	Actualizar los valores de las propiedades de una instancia de la entidad X
Actores	Administrador
Precondiciones	El actor está identificado en el sistema
Poscondiciones	La instancia cambia los valores de sus propiedades a los nuevos valores
Flujo Normal	1. Se elige una entidad X 2. Se muestran los valores de sus propiedades 3. Se modifican los valores de las propiedades 4. El sistema comprueba la validez de los datos 5. Se guardan los nuevos valores
Flujo Alternativo	4a Si los datos no son válidos 4b El sistema muestra los errores 4c Ir a 2

En este caso de uso el que participa es el Administrador. Es decir un usuario que tenga asignado el rol de Administrador. El sistema debe ofrecer dos servicios para responder a las peticiones del actor en este caso de uso: leer una instancia y actualizarla.

4.2. Casos de uso

Antes hemos visto los casos de uso básicos. Estos se deducen del modelo de dominio y por lo tanto dependen de los tipos que hayamos incluido en el mismo. Pero una pregunta que nos podemos hacer es ¿hasta dónde ampliar el modelo de dominio?, ¿Qué tipos y que relaciones entre os mismos incluir? La respuesta a esta pregunta viene dada por los casos de uso (excluyendo a los casos de uso básicos). Los casos de uso vienen definidos por los intereses de los usuarios. Estos casos de uso definen implícitamente servicios que le sistema debe proporcionar para interaccionar con el Actor. Estos servicios, a su vez, se definen en base a un conjunto de tipos y sus relaciones.

A partir de los casos de uso encontramos un conjunto de servicios que el sistema debe ofrecer. Estos servicios necesitan de un conjunto de tipos y relaciones entre ellos que están incluidos en el modelo de dominio. Un tipo, o una relación, que no sea usado en algún servicio requerido en un caso de uso no es interesante es irrelevante en un modelo de dominio y puede ser eliminado del mismo. A su vez una vez definido el modelo de dominio podemos añadir de forma automática los casos y uso y servicios que hemos denominado básicos: *CRUD, Login,* etc.

Vemos, por tanto, que los detalles del modelo de dominio vienen definidos por los intereses de los usuarios recogidos en los casos de uso.

Veamos un par de casos de uso relacionados con el sistema anterior:

- Un alumno pide su expediente que incluirá las asignaturas cursadas con su calificación, el grado alcanzado, la nota media del expediente y la fecha de finalización de la última asignatura.
- Un alumno pide el listado de profesores que le han impartido docencia con los detalles de asignatura y curso académico dónde los hicieron

5. Diseño del interfaz web

Una vez escogido los casos de uso, los servicios del sistema y realizado un modelo de dominio debemos diseñar la interfaz web apropiada. La interfaz web es el mecanismo de comunicación entre el usuario y el sistema y tiene que estar diseñado para ello. Desde este punto de vista la intefaz web es un elemento que está entre el usuario y el sistema para cursar peticiones del primero al segundo. Tiene que ser pensada para incluir toda la información relevante y facilite al usuario conseguir su objetivo.

Los elementos de una interfaz web son:

- Un conjunto de pantallas adecuadas para mostrar información o recoger la información.
- Un conjunto de botones que al ser pulsados permitan una acción sobre le sistema y posteriormente pasar a ottra pantalla. Lamaremos a estos botones de acción.
- Un conjunto de botones o enlaces en una pantalla que al ser pulsados permitan navegar a otra de la pantallas permitidas posiblemente tras hacer una acción sobre le sistema. Llamaremos a estos botones de navegación.

Cada página tiene un determinado diseño. Es decir una distribución de los elementos en la página y un conjuntos de estilos (colores, tipos de letra, etc.) para aplicar a cada uno de los elementos.

Hay muños diseños posibles. Aquí veremos uno dónde página está estructurada en varias partes:

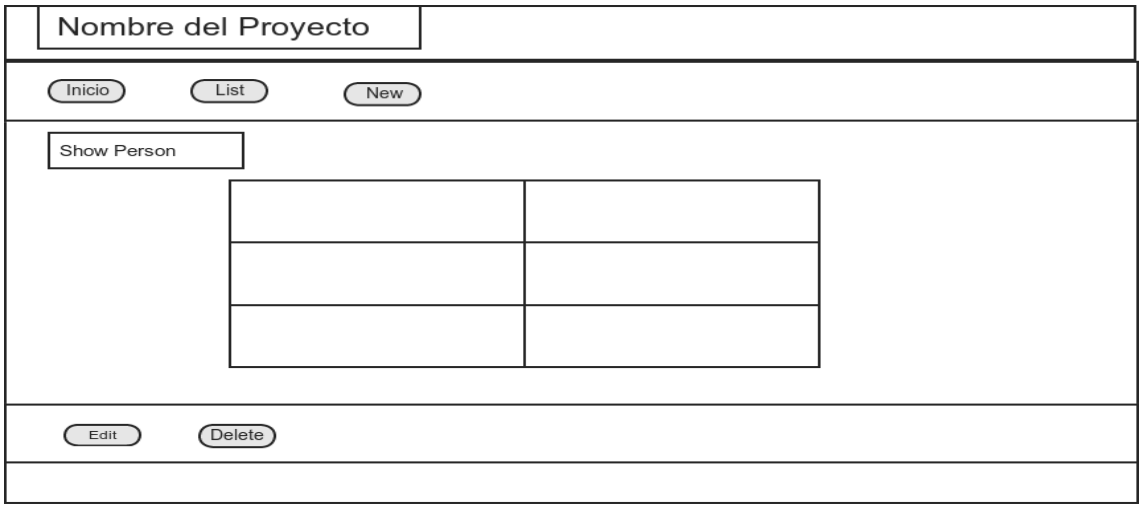
- Title (Titulo). Es la sección de arriba de la página y contiene texto con el título de la página
- Header (Cabecera). Esta sección está colocada arriba en la pantalla y puede estar dividida en dos partes: la primera contiene el logo del proyecto e información sobre el mismo, la segunda una serie de botones de navegación posiblemente en forma de menú o botones individuales.
- Body (Cuerpo). Contiene el área de trabajo
- Footer (Pie de página). Es la sección usualmente colocada abajo y puede estar divida a su vez en dos partes: la primera puede contener un conjutnod e botones de acción y la segunda información como fecha, el copyright, etc.

Normalmente todos estos elementos de diseño están en una página maestra dónde se especifica la distribución de los elementos (*layout*) y los diversos *estilos* a aplicar (estilo del título de los enlaces, …)

Algunos ejemplos de diseño de páginas relacionados con los casos de uso básicos asociados a u tipo dado son:

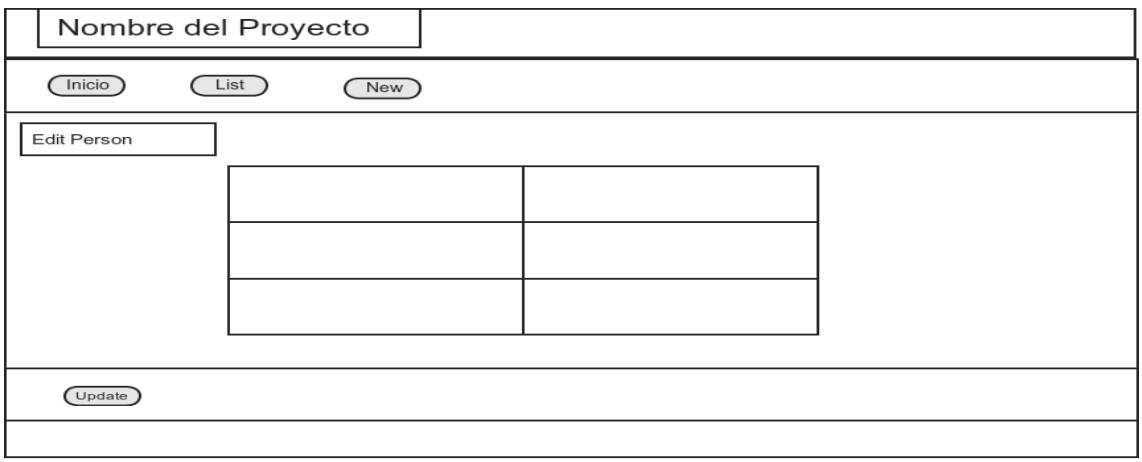
Mostrar Instancia

Pretende mostrar las propiedades, y sus valores, de una instancia de una persona. Los valores de las propiedades no son editables. Esta pantalla puede ser alcanzada, según los casos de uso visto arriba, por usuario o un administrador. Dependiendo del tipo de actor aparecen o no los botones que permiten saltar a las pantallas de *Nueva Instancia*, *Editar* o *Borrar*.



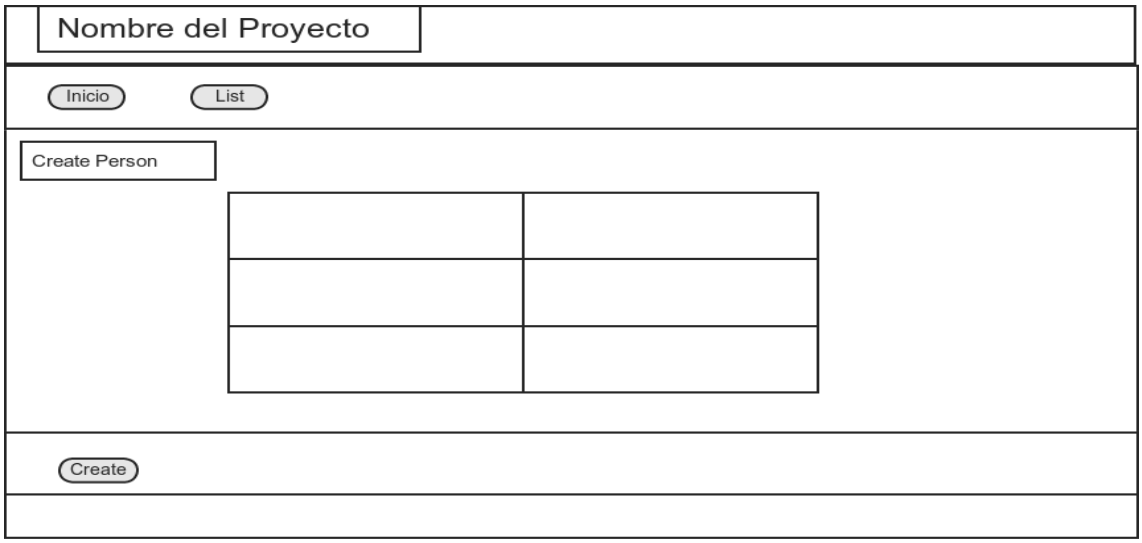
Editar Instancia

Pretende mostrar las propiedades, y sus valores, de una instancia de una persona. Los valores de las propiedades son editables. Esta pantalla puede ser alcanzada solamente por , según los un administrador.



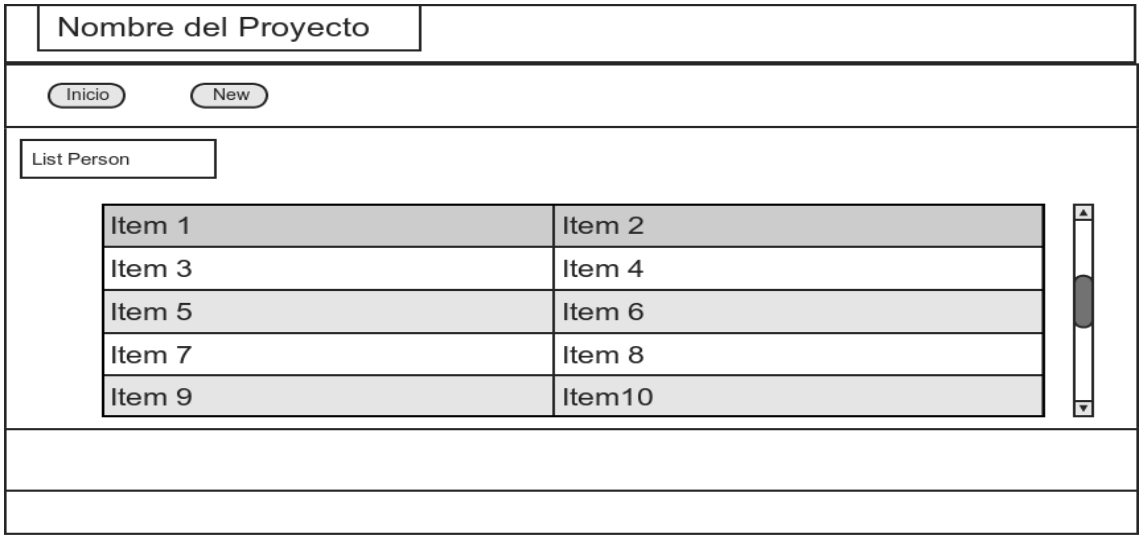
Crear Instancia

Pretende crear una nueva instancia. Para ello se muestan las propiedades y sus valores por defecto. Los valores de las propiedades son editables. Cuando se intenta actualizar se comprobará que los valores de las propiedades cumplen las restricciones. Si no se cumplen se vuelve a la misma pantalla dónde en la parte superior del cuerpo aparecen los errores que se han cometido.



Listar todas las instancias

En esta pantalla aparece un listado de todas las instancias de una tipo dado. Podemos movernos por la lista y existir la posibilidad de elegir una instancia. Si se elige se pasará a la pantalla de mostrar los detalles de esa instancia.



6. Diseño de la Navegación

El diseño de la navegación consiste en pensar en los caminos posibles de una página a otra. Estos caminos serán explícitos a solicitud del usuario o implícitos tras llevar a cabo una acción determinada. Los primeros se concretarán en botones de navegación y los segundos en navegaciones asociadas a los botones de acción. En definitiva diseñar la navegación es pensar en un grafo cuyos vértices son las páginas y cuyas aristas son las posibles transiciones de una página a otra. Es decir diseñar la navegación es escoger el conjunto de páginas posibles y los posibles saltos de una a otra.

Una navegación eficaz es quizás el aspecto más importante para asegurar que un sitio Web sea usable. La navegación consistente e informativa ayuda a asegurar que los usuarios son capaces de identificar dónde están, dónde está el contenido que necesitan, y cuál es la forma más fácil de llegar a él.

El diseño de la navegación hay que hacerlo pensando la aplicación como un todo y, por lo tanto, teniendo en cuenta todos los casos de uso. Junto a ello hay que considerar un conjunto de recomendaciones, por una parte, y adicionalmente seguir algunos patrones que ya han sido depurados por la práctica.

El diseño de la navegación en una aplicación web trata facilitar que los usuarios sepan donde están en cada momento, dónde ir y cómo llegar allí desde el punto donde están.

La navegación por un sitio web es similar a los desplazamientos por una ciudad. Tienes que hacerlos para llegar a donde necesitas ir, pero en muchos casos es un gasto de energía o es aburrido o exasperante. De forma similar al moverse por una ciudad lo mejor es tener lo que se necesita al alcance y por lo tanto haciendo los mínimos desplazamientos. Como en una ciudad el concepto de distancia es clave para el diseño de la navegación.

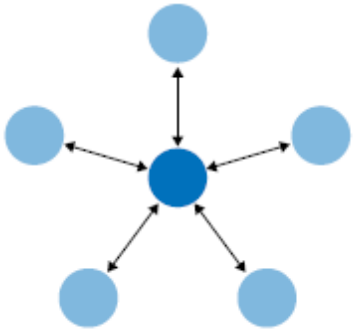
Veamos algunos patrones para conseguir los objetivos anteriores:

Mantener al usuario localizado: Es decir que el usuario sepa dónde se encuentra en cada momento y donde puede ir en uno o más saltos.

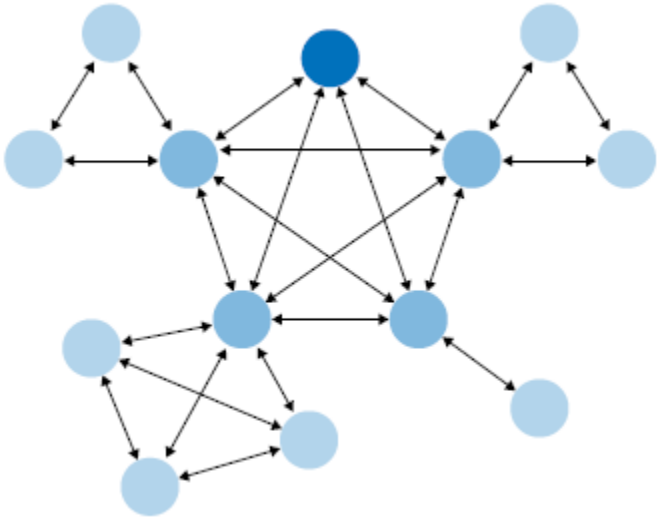
Mantener las distancias cortas: Se sabe que hay un costo asociado con el salto de una página a otra. Por eso es importante mantener bajo el número de esos saltos para conseguir un objetivo.

Modelos de Navegación: Las diferentes pantallas (o páginas), los enlaces entre sí, y cómo se mueven los usuarios entre ellas. Algunos esquemas usuales son:

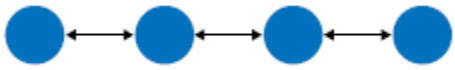
- Núcleo y rayos: Se enumera todos las principales partes del sitio en la pantalla principal o núcleo de la aplicación. El usuario navega a alguna de la partes, hace lo que tiene que hacer, y regresa al núcleo para ir a posteriormente otro lugar.



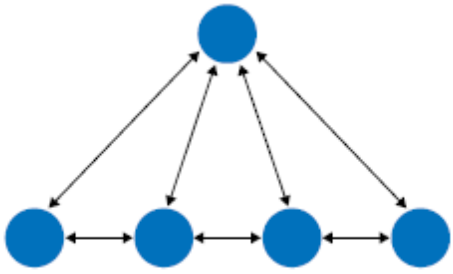
- Multinivel: Hay un conjunto de páginas principales están conectadas entre sí y otro de subpáginas están sólo conectadas entre ellas y con una de las páginas principales.



- Paso a paso: Conducen al usuario paso a paso a través de las pantallas en una secuencia prescrita. Cada página tiene una anterior y otra siguiente.



- Piramidal: Una variante del esquema anterior que utiliza una página principal. El usuario que está en un punto puede pasar al siguiente al anterior o ir a la página principal.



7. Capas de una aplicación

La aplicación que desarrollemos es conveniente que se estructure en capas. Desarrollar un diseño cohesivo dentro de cada capa y donde cada una de ellas dependa sólo de las capas inferiores. Concentrar todo el código relacionado con el modelo de dominio en una capa y aislarlo de la interfaz de usuario.

Una solución arquitectónica común suele contener cuatro capas conceptuales:

- Interfaz de usuario (capa de presentación). Responsables de la presentación de la información para el usuario y la interpretación de comandos de usuario.
- Capa de aplicación. Esta es una capa delgada que coordina la aplicación actividad. No contiene lógica de negocio. No mantiene el estado de los objetos, puede puede contener el estado de progreso de la tarea de la aplicación.
- Capa de Dominio. Esta capa contiene información sobre el dominio. Éste es el corazón del software de negocios. El estado de objetos de negocio se mantiene aquí. La persistencia y, posiblemente, su estado se delega a la capa de infraestructura.
- Capa de Infraestructura. Esta capa actúa como soporte para todas las demás capas. Proporciona comunicación entre capas, persistencia para los objetos, etc.