



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

**INGENIERÍA
EN INFORMÁTICA**

PROYECTO FIN DE CARRERA

**GCAD: Sistema Cliente/Servidor para la Gestión de
Decisiones en Desarrollo Distribuido de Software.**

Juan Andrada Romero

Febrero, 2012



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

**DEPTO. DE TECNOLOGÍAS,
Y SISTEMAS DE INFORMACIÓN**

PROYECTO FIN DE CARRERA

**GCAD: Sistema Cliente/Servidor para la Gestión de
Decisiones en Desarrollo Distribuido de Software.**

Autor: Juan Andrada Romero

Director: Aurora Vizcaíno Barceló

Febrero, 2012

GCAD: Sistema Cliente/Servidor para la Gestión de Decisiones en Desarrollo Distribuido de Software.
© Juan Andrada Romero, 2012

TRIBUNAL:

Presidente: _____

Vocal 1: _____

Vocal 2: _____

Secretario: _____

FECHA DE DEFENSA: _____

CALIFICACIÓN: _____

PRESIDENTE

VOCAL 1

VOCAL 2

SECRETARIO

Fdo.: _____

Fdo.: _____

Fdo.: _____

Fdo.: _____

Resumen

Abstract

... english version for the abstract ...

Agradecimientos

Escribir agredecimientos

Índice general

1	Resultados	1
1.1	Fase de Inicio	2
1.1.1	Identificación de requisitos	2
1.1.1.1	Requisitos funcionales	3
1.1.1.2	Requisitos no funcionales	7
1.1.2	Modelo de casos de uso	7
1.1.2.1	Modelo de casos de uso para el subsistema cliente	7
1.1.2.2	Modelo de casos de uso para el subsistema servidor	8
1.1.3	Análisis de riesgos	16
1.1.4	Plan de iteraciones	16
1.2	Fase de Elaboración	18
1.2.1	Primera iteración	19
1.2.1.1	Identificación de requisitos	19
1.2.1.2	Modelo de casos de uso	20
1.2.1.3	Plan de iteraciones	25
1.2.1.4	Análisis de casos de uso	30
1.2.1.5	Arquitectura del sistema	44
1.2.2	Segunda iteración	48
1.2.2.1	Diagrama de clases de dominio	49

1.2.2.2	Diseño de la base de datos	51
1.2.2.3	Funcionalidad de <i>Acceso al sistema</i>	53
1.3	Fase de construcción	68
1.3.1	Primera iteración	69
1.3.1.1	Funcionalidad de <i>Visualizar información</i>	69
1.3.2	Segunda iteración	82
1.3.2.1	Funcionalidad de <i>Gestión de decisiones</i>	82
1.3.2.2	Funcionalidad de <i>Gestión de notificaciones</i>	91
1.3.3	Tercera iteración	95
1.3.3.1	Funcionalidad de <i>Gestión de proyectos</i>	95
1.3.4	Cuarta iteración	106
1.3.4.1	Funcionalidad de <i>Generación de informes</i>	106
1.3.4.2	Funcionalidad de <i>Generación de estadísticas</i>	115
1.3.5	Quinta iteración	120
1.3.5.1	Funcionalidad de <i>Exportar conocimiento</i>	120
1.3.5.2	Funcionalidad de <i>Gestión de idiomas</i>	122
Bibliografía		124

Índice de figuras

1.1 Fase de inicio en el PUD	2
1.2 Diagrama de casos de uso - Cliente - v1.0	8
1.3 Diagrama de casos de uso - Servidor - v1.0	9
1.4 Diagrama de casos de uso - Cliente - Acceso al sistema - v1.0	10
1.5 Diagrama de casos de uso - Servidor - Acceso al sistema - v1.0	10
1.6 Diagrama de casos de uso - Cliente - Visualizar información - v1.0	10
1.7 Diagrama de casos de uso - Servidor - Visualizar información - v1.0	11
1.8 Diagrama de casos de uso - Cliente - Gestión decisiones - v1.0	11
1.9 Diagrama de casos de uso - Servidor - Gestión decisiones - v1.0	12
1.10 Diagrama de casos de uso - Cliente - Gestión notificaciones - v1.0	12
1.11 Diagrama de casos de uso - Servidor - Gestión notificaciones - v1.0	13
1.12 Diagrama de casos de uso - Cliente - Gestión Proyectos - v1.0	13
1.13 Diagrama de casos de uso - Servidor - Gestión Proyectos - v1.0	14
1.14 Diagrama de casos de uso - Cliente - Gestión Idiomas - v1.0	14
1.15 Diagrama de casos de uso - Servidor - Gestión Idiomas - v1.0	14
1.16 Diagrama de casos de uso - Cliente - Exportar información - v1.0	15
1.17 Diagrama de casos de uso - Servidor - Exportar información - v1.0	15
1.18 Fase de elaboración en el PUD	18
1.19 Diagrama de casos de uso - Cliente - v2.0	23

1.20 Diagrama de casos de uso - Servidor - v2.0	24
1.21 Diagrama de casos de uso - Cliente - Visualizar información - v2.0	24
1.22 Diagrama de casos de uso - Servidor - Visualizar información - v2.0	25
1.23 Diagrama de casos de uso - Cliente - Gestión decisiones - v2.0	26
1.24 Diagrama de casos de uso - Servidor - Gestión decisiones - v2.0	27
1.25 Diagrama de casos de uso - Cliente - Generación Estadísticas - v2.0	27
1.26 Diagrama de casos de uso - Cliente - Generación Informes - v2.0	28
1.27 Diagrama de casos de uso - Servidor - Generación Informes - v2.0	28
1.28 Diagrama de clases de análisis - Cliente - Login	31
1.29 Diagrama de clases de análisis - Servidor - Login	31
1.30 Diagrama de clases de análisis - Cliente - Logout	33
1.31 Diagrama de clases de análisis - Servidor - Logout	33
1.32 Diagrama de clases de análisis - Cliente - Consultar Decisiones	35
1.33 Diagrama de clases de análisis - Servidor - Consultar Decisiones	35
1.34 Diagrama de clases de análisis - Cliente - Consultar compañía	35
1.35 Diagrama de clases de análisis - Servidor - Consultar compañía	36
1.36 Diagrama de clases de análisis - Cliente - Crear Tema	38
1.37 Diagrama de clases de análisis - Servidor - Crear Tema	38
1.38 Diagrama de clases de análisis - Cliente - Crear Propuesta	40
1.39 Diagrama de clases de análisis - Servidor - Crear Propuesta	40
1.40 Diagrama de clases de análisis - Cliente - Crear Respuesta	40
1.41 Diagrama de clases de análisis - Servidor - Crear Respuesta	42
1.42 Arquitectura cliente-servidor	45
1.43 Arquitectura multicapa	46
1.44 Arquitectura multicapa	47

1.45 Diagrama de clases de dominio	52
1.46 Diagrama EER de la base de datos	54
1.47 Diagrama de secuencia - Cliente - Login	55
1.48 Diagrama de secuencia - Servidor - Login	56
1.49 Diagrama de secuencia - Cliente - Logout	57
1.50 Diagrama de secuencia - Servidor - Logout	57
1.51 Diagrama de clases - Capa de comunicación cliente-servidor	59
1.52 Diagrama de clases - Capa de comunicación para bases de datos	62
1.53 Diagrama de clases - Capa de comunicación para gestionar el log	63
1.54 Diagrama de clases - Gestor de sesiones	65
1.55 Ejemplo de <i>spinner</i> de carga	66
1.56 Fase de construcción en el PUD	68
1.57 Diagrama de secuencia - Cliente - Consultar decisiones	70
1.58 Diagrama de secuencia - Servidor - Consultar decisiones	70
1.59 Diagrama de secuencia - Cliente - Consultar compañía	71
1.60 Diagrama de secuencia - Servidor - Consultar compañía	71
1.61 Diagrama de secuencia - Cliente - Descargar ficheros adjuntos	72
1.62 Diagrama de secuencia - Servidor - Descargar ficheros adjuntos	72
1.63 Diagrama de clases - Jerarquía de decisiones	74
1.64 Ejemplo de jerarquía de decisiones en foros de debates	74
1.65 Diagrama de clases - Composición de clases de interfaz gráfica	79
1.66 Diagrama de secuencia - Cliente - Crear decisión (<i>Topic</i>)	83
1.67 Diagrama de secuencia - Cliente - Crear decisión (<i>Topic</i>)	83
1.68 Diagrama de secuencia - Cliente - Crear decisión (<i>Topic</i>)	83
1.69 Diagrama de secuencia - Servidor - Crear decisión	83

1.70 Diagrama de secuencia - Cliente - Adjuntar ficheros	84
1.71 Diagrama de secuencia - Servidor - Adjuntar ficheros	84
1.72 Diagrama de clases - Gestión de decisiones	85
1.73 Diagrama de clases - Observador para actualizar clientes conectados	86
1.74 Diagrama de secuencia - Cliente - Consultar notificaciones	91
1.75 Diagrama de secuencia - Servidor - Consultar notificaciones	91
1.76 Diagrama de secuencia - Cliente - Modificar notificaciones	92
1.77 Diagrama de secuencia - Servidor - Modificar notificaciones	92
1.78 Diagrama de secuencia - Cliente - Eliminar notificaciones	92
1.79 Diagrama de secuencia - Servidor - Eliminar notificaciones	93
1.80 Diagrama de secuencia - Cliente - Consultar proyectos	95
1.81 Diagrama de secuencia - Servidor - Consultar proyectos	96
1.82 Diagrama de secuencia - Cliente - Consultar usuarios	96
1.83 Diagrama de secuencia - Servidor - Consultar usuarios	96
1.84 Diagrama de secuencia - Cliente - Crear proyecto	97
1.85 Diagrama de secuencia - Servidor - Crear proyecto	97
1.86 Diagrama de secuencia - Cliente - Modificar proyecto	98
1.87 Diagrama de secuencia - Servidor - Modificar proyecto	98
1.88 Diagrama de secuencia - Cliente - Seleccionar proyecto activo	99
1.89 Diagrama de secuencia - Servidor - Seleccionar proyecto activo	99
1.90 Diagrama de clases - Servidor - Razonamiento Basado en Casos	103
1.91 Diagrama de secuencia - Cliente - Generar informe (comportamiento normal)	107
1.92 Diagrama de secuencia - Cliente - Generar informe (comportamiento normal)	107
1.93 Diagrama de secuencia - Cliente - Generar informe (comportamiento extendido)	107
1.94 Diagrama de secuencia - Servidor - Generar informe (comportamiento extendido)	108

1.95 Diagrama de clases - Generación de documentos PDF - Servidor	109
1.96 Diagrama de clases - Generación de documentos PDF - Cliente	113
1.97 Proceso simplificado de <i>Drag & Drop</i>	115
1.98 Diagrama de secuencia - Cliente - Generar estadísticas	115
1.99 Diagrama de clases - Cliente - Generar estadísticas	117
1.100 Diagrama de secuencia - Cliente - Gestión de idiomas	122
1.101 Diagrama de secuencia - Servidor - Gestión de idiomas	122

Índice de tablas

1.1	Requisitos funcionales - v1.0	5
1.2	Roles identificados en el sistema	5
1.3	Acciones que un usuario puede realizar en el sistema - v1.0	6
1.4	Plan de iteraciones	17
1.5	Requisitos funcionales - v2.0	21
1.6	Acciones que un usuario puede realizar en el sistema - v2.0	22
1.7	Plan de iteraciones actualizado	30
1.8	Especificación del caso de uso <i>Login</i>	32
1.9	Especificación del caso de uso <i>Logout</i>	33
1.10	Especificación del caso de uso <i>Consultar decisiones</i>	34
1.11	Especificación del caso de uso <i>Consultar compañía</i>	36
1.12	Especificación del caso de uso <i>Crear decisión - Crear Tema</i>	37
1.13	Especificación del caso de uso <i>Crear decisión - Crear Propuesta</i>	39
1.14	Especificación del caso de uso <i>Crear decisión - Crear Respuesta</i>	41

Índice de listados

1.1	Proceso para exportar un objeto utilizando RMI	60
1.2	Proceso para localizar un objeto remoto utilizando RMI	61
1.3	Fragmento de código para acceder al sistema en el cliente	66
1.4	Respuesta del servicio Web Yahoo! PlaceFinder	76
1.5	Invocación del servicio Web Yahoo! PlaceFinder	77
1.6	Fragmento de código para mostrar mapas geoposicionados	80
1.7	Fragmento de código del controlador de clientes	86
1.8	Soporte multi-hilo para actualizar el estado de clientes	87
1.9	Fragmento de código utilizando <i>reflection</i>	90
1.10	Trigger de base de datos para gestionar la eliminación de alertas	93
1.11	Fragmento de código para el algoritmo <i>NN</i> del CBR	104
1.12	Fragmento de código para la generación de documentos PDF	109
1.13	Fragmento de código para la generación de <i>datasets</i>	117
1.14	Fragmento de código para la generación de gráficos	118
1.15	Anotaciones de JAXB sobre la clase <i>TopicWrapper</i>	121
1.16	Anotaciones de JAXB sobre la clase <i>Knowledge</i>	121
1.17	Soporte multi-idioma	123

Índice de algoritmos

Capítulo 1

Introducción

1.1 Estructura del documento

Capítulo 2

Motivación y Objetivos del Proyecto

2.1 Motivación

La globalización y competitividad de los mercados ha obligado, y está obligando, a que las empresas vayan evolucionando y adaptando sus procesos de negocio y sus procesos de gestión de proyectos para adaptarse a esta nueva tendencia de mercados globales. Estas nuevas estrategias requieren de cooperación intensiva entre las compañías, de nuevos procesos de gestión de proyectos, de mejoras en las comunicaciones entre organizaciones que se encuentran en diferentes husos horarios, etc. Todo ello ha provocado que aparezcan nuevas organizaciones, llamadas *organizaciones virtuales*, que son aquellas que se encuentran deslocalizadas, con proyectos en los que participan empleados de cualquier parte del mundo y que utilizan las tecnologías de la información para sus procesos de comunicación, coordinación y control [?].

Anteriormente a este marco de globalización, las empresas gestionaban sus proyectos de manera co-localizada, es decir, en el mismo lugar y al mismo tiempo. Sin embargo, ahora se ven obligadas a desarrollar procesos de gestión que permitan la gestión de *proyectos virtuales*, es decir, proyectos que ya no están centralizados en un único lugar y se realizan de manera distribuida. En la Figura ?? se muestran las dimensiones de los proyectos que las compañías deben gestionar [?].

Sin embargo, este marco de globalización, de *proyectos virtuales* y, por tanto, del desarrollo de los proyectos de manera distribuida, acarrea una serie de dificultades, siendo las más destacables los problemas de comunicación, coordinación y control entre las diferentes localizaciones de los centro de desarrollo de las organizaciones.

Otro de los problemas encontrados, muy relacionado con el anterior, es como las empresas,

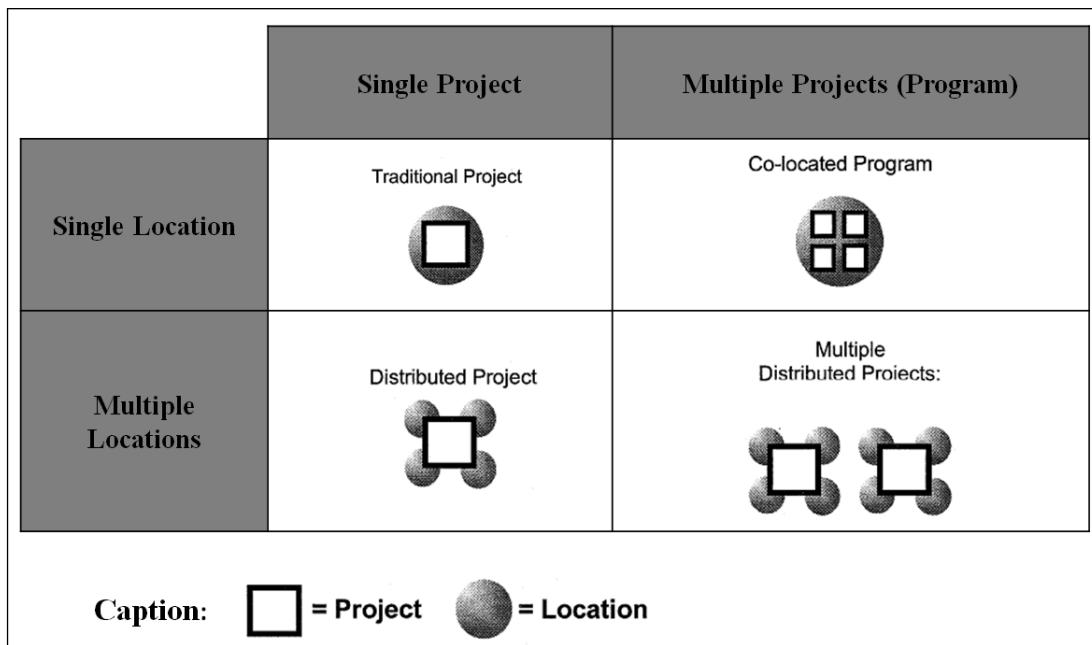


Figura 2.1: Proyectos según su localización

que ahora se encuentran en localizaciones diferentes y que tienen mayor problema para la comunicación y control, pueden crear, organizar y gestionar su conocimiento organizacional, recurso intangible imprescindible para que la empresa pueda innovar y ser competitiva.

Por tanto, la motivación de este PFC viene dada por la necesidad de resolver o minimizar estos problemas que aparecen en las compañías que desarrollan de manera global. Particularmente, la herramienta estará enfocada a la gestión de conocimiento en desarrollo global, intentando minimizar todo lo posible los problemas de comunicación y control que aparecen en esta nueva forma de desarrollo global (ver Capítulo ?? para una revisión de la literatura acerca del Desarrollo Global, Gestión de Conocimiento, así como sus beneficios y problemas).

2.2 Objetivos

El objetivo principal del PFC consistirá en el diseño y construcción de una herramienta visual que permita dar soporte a la gestión de conocimiento en el paradigma de Desarrollo Global de Software. Concretamente, el conocimiento que se desea gestionar son las decisiones y alternativas planteadas en las fases del ciclo de vida de un producto software, desarrollado de manera global.

Como se ha comentado anteriormente, con esta herramienta visual se intenta reducir o eliminar una de las problemáticas que aparece en el desarrollo global, como es la falta de comunicación y control entre los equipos de desarrollo, así como los malentendidos que pueden surgir debido a las diferencias socio-culturales de los participantes en el desarrollo de los proyectos.

A continuación, se desglosan los objetivos que se deben alcanzar para atenuar estos desafíos del desarrollo global de software.

2.2.1 Objetivos a alcanzar con el desafío de la comunicación

Para poder minimizar los desafíos de comunicación que aparecen en el desarrollo global de software, la herramienta debe permitir representar el conocimiento almacenado por las organizaciones de una manera visual, clara e intuitiva, evitando malentendidos. De este modo, se representa el conocimiento de la misma manera para todos los equipos de desarrollo y empleados, facilitando su transmisión y evitando confusiones.

A la hora de crear y almacenar nuevo conocimiento, la herramienta debe proveer formularios para crear dicho conocimiento, evitando por tanto formatos e información diferentes, estandarizando el conocimiento y facilitando su posterior comunicación.

La herramienta también debe poseer una comunicación implícita síncrona, es decir, que si un empleado se encuentra visualizando el conocimiento en la aplicación y, en ese momento, otro empleado de otra localización diferente realiza un cambio sobre el mismo conocimiento, éste debe ser notificado al primer empleado en tiempo real, actualizando su vista. Además, deberá crearse una notificación automática, favoreciendo también de este modo la comunicación asíncrona.

Otro punto a tener en cuenta para facilitar la comunicación, es que la herramienta pueda configurarse en diferentes idiomas. Es por ello que debe tenerse en cuenta la internacionalización de la aplicación.

2.2.2 Objetivos a alcanzar con el desafío del control

Uno de los grandes problemas de control en desarrollo global, es que los jefes o encargados de proyectos muchas veces no controlan correctamente la situación del proyecto, a los miembros

de los equipos que trabajan en él, cuál es la situación del proyecto, etc. Por ello, la herramienta debe permitir la generación de gráficos estadísticos a partir del conocimiento almacenado, informando de la situación de los proyectos.

Relacionado con lo anterior, es necesario que la aplicación pueda generar informes en PDF, resumiendo el conocimiento almacenado en un determinado proyecto. Además, se debe poder controlar cuales de las decisiones y alternativas tomadas durante el ciclo de vida del proyecto están aprobadas por el jefe de proyecto y cuáles no.

Para favorecer una estructura de proyectos común y facilitar así un posterior control entre los diferentes equipos de desarrollo, la herramienta será capaz de proveer formularios para la creación de nuevos proyectos, así como de su posterior modificación.

Para terminar, otro de los puntos principales a tener en cuenta es que el conocimiento almacenado pueda controlarse y utilizarse en proyectos futuros que sean similares a proyectos ya pasados. Para ello, la herramienta debe permitir la búsqueda de esta información en base a proyectos pasados ya finalizados, utilizando técnicas de inteligencia artificial.

2.2.3 Resumen de objetivos

En modo de síntesis, la herramienta debe cumplir con los siguientes objetivos y requisitos funcionales:

- Crear y almacenar conocimiento en forma de alternativas y decisiones planteadas en las fases del ciclo de vida de un proyecto software.
- Mostrar de una forma visual e intuitiva las diferentes alternativas y decisiones que han sido almacenadas.
- Aceptar o rechazar alternativas y decisiones.
- Notificar a los empleados de las organizaciones nuevas contribuciones y conocimiento disponible.
- Proporcionar información acerca de quién aporta una alternativa o decisión. Dicha información puede ser el rol de esa persona, el país donde se encuentra trabajando, experiencia acumulada en desarrollo de software, antigüedad en la empresa, etcétera.

- Aconsejar decisiones y alternativas que hayan sido exitosas en otros proyectos con características similares, basándose en proyectos ya finalizados.
- Generar estadísticas que ayuden a llevar control sobre los proyectos: proyectos con mayor conocimiento almacenado, qué empleado participa más, histórico de decisiones planteadas, etc.
- Exportar el conocimiento disponible utilizando un estándar para el intercambio de información estructurada, como es XML. También se permitirá exportar imágenes de los gráficos generados en formato PNG.
- Crear y modificar proyectos.
- Generar documentos PDF a partir del conocimiento almacenado en los proyectos, creando automáticamente tablas con las decisiones y alternativas planteadas para ese proyecto.
- Internacionalización de la herramienta.
- La herramienta debe poder ser accedida desde diferentes localizaciones.
- Soporte para diferentes perfiles y roles de usuarios.

No se debe olvidar que esta herramienta, aunque enfocada y con aplicación en el desarrollo global de software, constituye también un sistema para la gestión de conocimiento, ya que como se verá en el Capítulo ??, permite la creación, almacenamiento, recuperación, transmisión y aplicación del conocimiento, procesos que se llevan a cabo en la gestión de conocimiento. En nuestro caso concreto, el conocimiento son las decisiones y alternativas planteadas durante las fases del ciclo de vida de los proyectos software.

Capítulo 3

Estado del Arte

A lo largo de este capítulo se presenta el estado del arte existente en la literatura sobre Desarrollo Global de Software, exponiendo lo que este concepto implica en el desarrollo de software, así como sus beneficios y desafíos existentes.

Se expone también el concepto de Gestión de Conocimiento, así como un marco de trabajo que las empresas pueden adoptar para gestionar su conocimiento interno.

Para terminar, se introduce el Razonamiento Basado en Casos, una técnica de inteligencia artificial que puede resolver problemas en base a experiencias pasadas.

3.1 Desarrollo Global de Software

En los últimos años, la globalización económica está provocando que la industria y el comercio se adapten a nuevos modelos de negocios, en búsqueda de aumentar la competitividad y minimizar los costes, por lo que las empresas se han visto obligadas a evolucionar en su modo de trabajo. Así, se ha cambiado el concepto de *mercado nacional* por el de *mercado global* [?] [?].

Centrándose en el desarrollo de software, hasta hace unos años el desarrollo era llevado a cabo por Centros de Desarrollo Software (en adelante CDS) que estaban co-localizados en un mismo edificio al mismo tiempo. Sin embargo, debido a este nuevo modelo de economía global, la manera de desarrollar software está evolucionando en los últimos años, tendiendo a un desarrollo distribuido del mismo. Los CDS comenzaron a distribuirse en diferentes ciudades e incluso diferentes provincias de un mismo país, hasta llegar al punto en que los CDS se encuentran distribuidos en diferentes países y continentes. Esto es lo que se conoce

como **Desarrollo Global de Software**, denominado a partir de este punto como GSD, por sus siglas en inglés Global Software Development. En la Tabla ?? se muestran los diferentes modelos de desarrollo de software existentes, en base a la dispersión geográfica.

EL GSD puede definirse como: “*el desarrollo de software que se realiza en localizaciones separadas geográficamente más allá de fronteras nacionales, de manera coordinada e involucrando participación en tiempo real (síncrona) e interacción asíncrona*” [?].

Localización	Tipo de desarrollo
Mismo lugar	Desarrollo tradicional (Co-localizado)
Mismo país	Desarrollo Distribuido de Software (DSD)
Distintos países	Desarrollo Global de Software (GSD)

Tabla 3.1: Modelos de desarrollo de software según dispersión geográfica

En este nuevo modelo de GSD, además de tener en cuenta que los *stakeholders*¹ se encuentran distribuidos geográficamente, hay que contar con el número de compañías que colaboran en un mismo proyecto. Un mismo proyecto bien puede ser desarrollado por una misma compañía, pero con CDS distribuidos en diferentes países, o bien puede llevarse a cabo por un conjunto de compañías que colaboran o están subcontratadas por la primera [?]. Así, surgen los conceptos de outsourcing, offshoring y nearshoring.

Por **outsourcing** se entiende la externalización o subcontratación de uno o varios servicios a terceros, ya que resulta más rentable, en términos de disminución de costes, que la compañía subcontratada realice estos servicios, ya que la mano de obra puedes ser más barata en un país diferente al país de la compañía contratante [?].

Relacionado con el *outsourcing*, está el concepto de **offshoring**, entendido como la relocalización de actividades de negocio, mediante empresas filiales de otros países [?]. Si estos países implicados en el proceso son relativamente cercanos al país de origen de la compañía que realiza el *outsourcing*, se habla de **nearshoring** [?].

En GSD, estas prácticas son cada vez más utilizadas, debido a la rentabilidad que proporcionan a la compañía. En la Figura ?? se puede observar el crecimiento del *offshoring* de los últimos años y la reducción de costes que conlleva.

¹Stakeholders: *aquellos que pueden afectar o son afectados por las actividades de una empresa* [?].

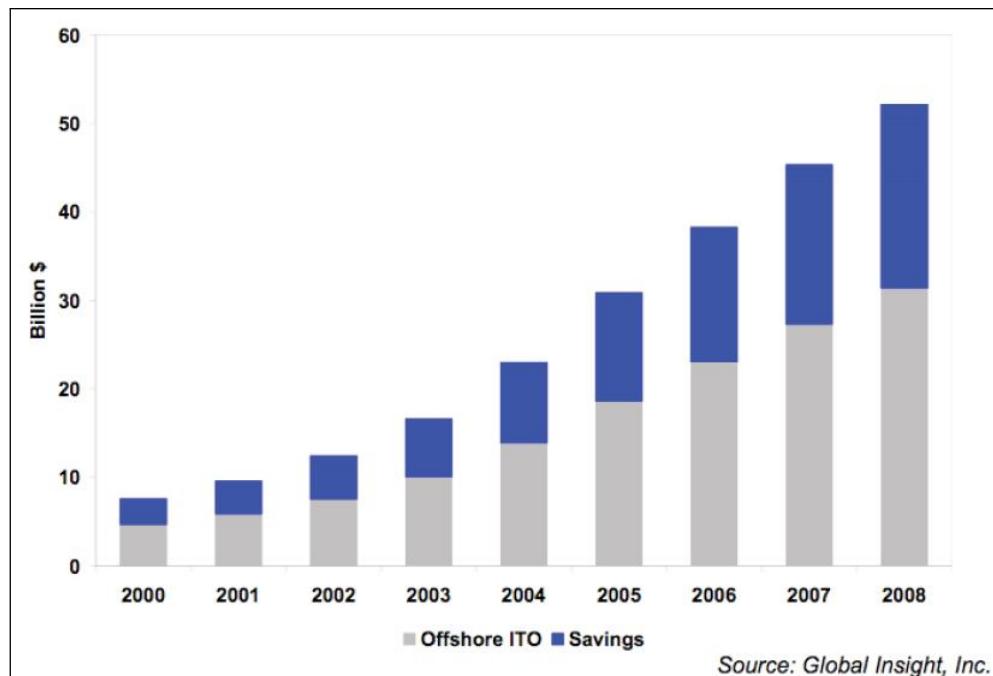


Figura 3.1: Crecimiento del *offshoring* en los últimos años

Como se puede apreciar, el GSD y sus prácticas introducen una serie de beneficios o ventajas, pero también de dificultades o inconvenientes a tener en cuenta. En el siguiente apartado se desarrollan los más importantes.

3.1.1 Beneficios del Desarrollo Global de Software

A continuación se desarrollan los beneficios más destacables del Desarrollo Global de Software, según [?] y [?].

3.1.1.0.1 Reducción de costes

La diferencia de salarios entre países puede ser enorme. Por ejemplo, el salario de un ingeniero de software en Estados Unidos puede llegar a ser varias veces el salario de un ingeniero similar en Asia o Sudamérica, por lo que las compañías buscan localizaciones alternativas que permitan recortar el coste de la mano de obra.

Gracias al GSD, las compañías tienen acceso a mano de obra más barata, debido a su deslocalización en diferentes países, pudiendo encontrar salarios mucho menores que en el país de origen. Ya que las compañías siempre buscan reducir el coste, éste ha sido el principal

motivo que ha impulsado el GSD.

3.1.1.0.2 Aumento de la competitividad

El GSD provee la posibilidad de poder encontrar mano de obra más cualificada o más experimentada en diferentes países, de modo que las compañías pueden contratar esos recursos personales y expandir su desarrollo a zonas más cualificadas y competitivas.

3.1.1.0.3 Disminución de tiempo de lanzamiento al mercado

En GSD, existe el modelo de desarrollo “*follow-the-sun*”. Esto significa que, gracias a que existen varios centros de desarrollo en diferentes países, una compañía puede aprovechar las diferencias horarias entre sus diferentes centros de desarrollo para estar desarrollando software las 24 horas del día (ver Figura ??). De este modo, se consigue maximizar la productividad y disminuir el tiempo de salida al mercado (*time-to-market*) de un producto software.

De este modo, cada uno de los CDS localizados en cada país, pueden ir realizando diferentes actividades, como, por ejemplo, diseño, implementación, pruebas, etc. Para que este modelo de trabajo sea efectivo, debe existir una coordinación entre todos los CDS, que es una de las dificultades del GSD, como se comentará posteriormente.



Figura 3.2: Modelo de desarrollo *Follow-the-sun*

3.1.1.0.4 Proximidad al mercado y al cliente

Al establecer filiales en aquellos países donde se localizan potenciales clientes, el GSD permite el desarrollo de software de una manera más cercana a sus clientes, aprovechando el conocimiento del mercado local y, por tanto, conociendo mejor sus necesidades.

3.1.1.0.5 Mejoras en la innovación

Al disponer de CDS en diferentes países, se puede tomar ventaja de las diferentes nacionalidades, culturas, experiencias y habilidades de cada uno de ellos, pudiendo así evolucionar, innovar y enriquecer la compartición de conocimiento.

Como se ha visto, el GSD acarrea una serie de ventajas y beneficios que lo han convertido en un nuevo modelo de desarrollo utilizado por muchas empresas del sector. Sin embargo, existen también una serie de dificultades y problemas derivados de la deslocalización de los recursos, que se detallan en el siguiente apartado.

3.1.2 Dificultades del Desarrollo Global de Software

El GSD provee a las compañías y organizaciones unos beneficios muy prometedores. Sin embargo, existen dificultades a las que las compañías tienen que hacer frente, todas ellas debidas a la distancia y a la deslocalización (ver Figura ??) [?] [?].

Según [?], existen tres tipos de distancia que dificultan el GSD:

- **Distancia geográfica.** En GSD, se define como “*la medida de esfuerzo que un individuo necesita realizar para visitar otro punto, alejado del primero*”. Por ejemplo, dos lugares dentro del mismo país con un enlace aéreo directo y vuelos regulares, se pueden considerar relativamente cercanos, aunque estén separados por grandes distancias kilométricas. Sin embargo, no se puede decir lo mismo de dos lugares que están cerca geográficamente (separación de pocos kilómetros) pero con poca infraestructura de transporte. Este último caso tendría una elevada distancia geográfica.
- **Distancia temporal.** En GSD, se define como “*la medida de la deslocalización en tiempo, experimentada por dos individuos que desean interactuar*”. Esta distancia

normalmente va unida a la anterior, provocando que existan husos horarios diferentes entre dos puntos debidos a la distancia geográfica existente entre dichos puntos.

- **Distancia socio-cultural.** Se define como “*la medida en que un individuo comprende las costumbres (símbolos, normas y valores sociales) y cultura de otro individuo*”. Aunque esta distancia es muy frecuente en el GSD, también aparece en equipos de desarrollo co-localizados, ya que cada miembro del equipo puede tener nacionalidades y culturas diferentes. Este tipo de distancia es el mayor desafío a superar en el GSD, ya que con mucha frecuencia se producen conflictos y malentendidos entre los diferentes equipos de desarrollo, debido al carácter multicultural y multinacional del GSD.

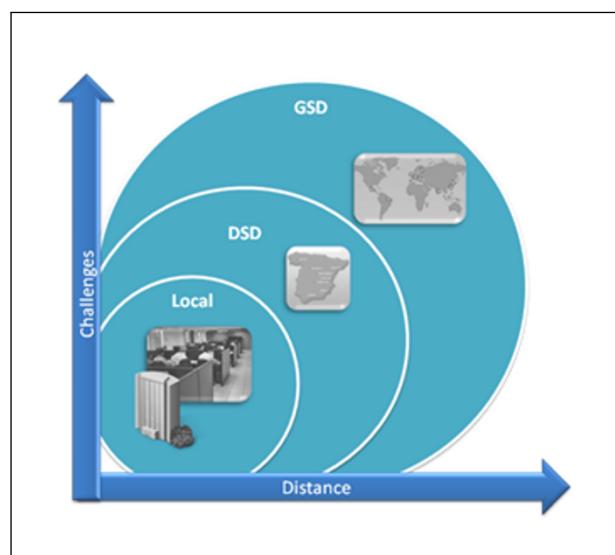


Figura 3.3: Influencia de la distancia en el desarrollo de software

Estas distancias afectan a los tres grandes procesos que intervienen en el proceso de desarrollo de software: **comunicación, coordinación y control**, siendo los de comunicación y control los más problemáticos en GSD, según [?].

Por tanto, los desafíos y dificultades del GSD se agrupan en tres categorías, lo que se conoce como las tres “C” [?]:

- Desafíos en la **Comunicación**, es decir, en el proceso de intercambio de conocimiento e información entre individuos.
- Desafíos en la **Coordinación**, relacionados con objetivos e intereses comunes.

- Desafíos en el **Control**, relacionados con la gestión del proyecto, informes, progreso, etc.

En la Tabla ?? se observa como afecta cada una de las distancias a los procesos de desarrollo software.

Proceso	Distancia		
	Geográfica	Temporal	Socio-Cultural
Comunicación	Dependiente de la tecnología	Uso de comunicación asíncrona	Malentendidos
Coordinación	Falta de conciencia de equipo	Modificación de calendarios laborales	Falta de confianza
Control	Incrementa la dificultad de gestión y control	Control asíncrono sobre recursos remotos	Diferencias culturales en mecanismos de control

Tabla 3.2: Procesos de desarrollo software afectados por las distancias en GSD

En los siguientes apartados se resumen los desafíos más destacables en cada uno de los grupos de las tres “C” y el tipo de distancia que afecta.

3.1.2.1 Desafíos en la comunicación y la distancia geográfica

Debido a la distancia geográfica entre los distintos CDS, aparecen las siguientes dificultades en la comunicación [?] [?].

3.1.2.1.1 Falta de comunicación “face-to-face”

La posibilidad de un intercambio de información *face-to-face* (cara a cara, en español) desaparece o es prácticamente nula debido a la deslocalización de los equipos de desarrollo. Por ello, se depende casi en la totalidad de herramientas de comunicación, que son muy variadas y pueden no seguir estándares de comunicación, por lo que se provocan malentendidos.

La distancia temporal también influye en este tipo de comunicación, ya que para que un intercambio de información cara a cara se produzca, es necesario estar “en el mismo lugar, al mismo tiempo”.

3.1.2.1.2 Aumento del esfuerzo en la comunicación

Debido al uso de herramientas de comunicación, los tiempos de respuesta son altos, no se conoce personalmente en muchos casos a otros miembros que intervienen en la comunicación ni su disponibilidad, por lo que la comunicación cada vez se vuelve más escasa y de menor calidad.

3.1.2.1.3 Coste de los viajes

En alguno casos, la necesidad de comunicarse cara a cara es indispensable, especialmente en fases tempranas del proyecto. Esto supone gastos de dinero y tiempo para realizar los viajes, además de ser necesario la ruptura de las barreras culturales y lingüísticas.

3.1.2.1.4 Dependencia en las tecnologías y herramientas de comunicación

En GSD, la dependencia de este tipo de tecnología y herramientas es muy elevada ya que, como se ha comentado, la posibilidad de una comunicación cara a cara es escasa. Por lo tanto, la comunicación y su efectividad va a depender en gran medida de las funcionalidades que la herramienta permita a los equipos de desarrollo para comunicarse.

3.1.2.2 Desafíos en la comunicación y la distancia temporal

Debido a la distancia temporal entre los distintos CDS, aparecen las siguientes dificultades en la comunicación [?] [?].

3.1.2.2.1 Interrupciones

Al encontrarse muchas veces los CDS en diferentes husos horarios, resulta muy complicado el poder contactar con individuos de otros equipos de trabajo y, en multitud de ocasiones, se

produce la comunicación en horarios no adecuados o se producen interrupciones en el trabajo de un equipo, lo que aumenta el malestar en el trabajo.

3.1.2.2.2 Cambio de horario laboral

Si los diferentes CDS siguen horarios muy diferentes, debido a las grandes distancias geográficas, a veces es necesario ajustar el horario laboral de cada equipo, para poder cumplir el modelo “follow-the-sun” y favorecer la comunicación.

Además de estas dificultades, al igual que en el caso anterior, se depende de herramientas de comunicación (asíncronas en su mayoría) y falta comunicación “face-to-face”.

3.1.2.3 Desafíos en la comunicación y la distancia socio-cultural

Debido a la distancia socio-cultural entre los distintos CDS, aparecen las siguientes dificultades en la comunicación [?].

3.1.2.3.1 Malentendidos debidos a diferencias lingüísticas

En multitud de ocasiones se producen malentendidos al comunicarse, debido a que los equipos de desarrollo hablan idiomas nativos diferentes. Estos fallos en la comunicación repercutirán posteriormente en el desarrollo, sobre todo si son malentendidos producidos durante la fase de análisis de requisitos del proyecto.

3.1.2.3.2 Malentendidos culturales

Al establecer una comunicación entre personas que no conocen totalmente las costumbres y cultura de la otra parte, puede provocar situaciones incómodas de manera involuntaria. Por ejemplo, en USA, prefieren especificar cada detalle en un documento, utilizar el teléfono y el e-mail de manera informal, mientras que en Japón prefieren el uso de medios más formales y prefieren la comunicación verbal a documentos escritos [?].

Por otro lado, en GSD el marco socio-cultural puede influir en opiniones diferentes sobre la propia naturaleza de llevar a cabo los procesos de desarrollo.

En la Tabla ?? se resumen los desafíos encontrados en el proceso de comunicación en GSD.

Comunicación	
Geográfica	Falta de comunicación <i>face-to-face</i> Aumento del esfuerzo en comunicación Coste de los viajes Dependencia en tecnologías de comunicación
Temporal	Interrupciones Cambio de horario laboral
Socio-cultural	Malentendidos por diferencias lingüísticas Malentendidos culturales

Tabla 3.3: Desafíos en el proceso de comunicación en GSD

3.1.2.4 Desafíos en la coordinación y la distancia geográfica

Debido a la distancia geográfica entre los distintos CDS, aparecen las siguientes dificultades en la coordinación [?] [?].

3.1.2.4.1 Reducción de la confianza

La confianza mutua en los equipos de desarrollo es crucial para una colaboración y coordinación. Sin embargo, para llegar a tener confianza, es necesario una interacción cercana, para poder empatizar con la otra parte, lo que es muy difícil en GSD, debido a la distancia. Debido a esta falta de confianza, empeora la coordinación entre diferentes CDS.

3.1.2.4.2 Falta de espíritu y conciencia de equipo

Los miembros de equipos virtuales, es decir, miembros de equipos separados por la distancia, tienden a ser menos productivos debido a los sentimientos de lejanía, indiferencia y aislamiento y falta de contacto informal. Por ello, se tiende menos a la colaboración, coordinación, resolución de tareas comunes, etc.

3.1.2.4.3 Modularización del trabajo

La naturaleza del GSD obliga a los distintos CDS a dividir su desarrollo en módulos bien definidos, de modo que puedan ser desarrollados de manera separada, lo que aumenta el coste de coordinación, para su posterior integración.

Además, relacionada con esta modularización del trabajo, aumenta también el esfuerzo para definir y estructurar todos los módulos, convirtiéndose en un gran reto para los jefes de proyecto.

3.1.2.4.4 Aumento del esfuerzo en acuerdos de externalización

Debido al *outsourcing*, aumenta el esfuerzo que debe hacerse al contratar recursos externos, contactar con ellos, coordinarlos, etc.

3.1.2.4.5 Riesgo de alternativas de bajo coste

Debido a la gran variedad de mando de obra y recursos, los empleados con mayor coste se sienten amenazados por compañeros que resultan ser mano de obra más barata, creando una sensación de “nosotros contra ellos” (“we versus they”).

3.1.2.5 Desafíos en la coordinación y la distancia temporal

Debido a la distancia temporal entre los distintos CDS, aparecen las siguientes dificultades en la coordinación [?].

3.1.2.5.1 Reducción de horas de colaboración

El número de horas se solapamiento en una jornada laboral entre diferentes CDS deslocalizados, varía mucho según el país donde se localice. Esto afecta a la realimentación (o “feedback” en inglés) entre equipos y complica su coordinación.

Del mismo modo, se reducen las horas efectivas de coordinación y comunicación síncrona entre equipos, lo que dificulta seguir el modelo de “follow-the-sun”, que necesita una fuerte coordinación para poder aprovechar al máximo las diferencias horarias.

3.1.2.5.2 Disponibilidad de infraestructura técnica

La tecnología es utilizada para la comunicación síncrona y asíncrona a través de los diferentes países. Sin embargo, la tecnología utilizada para la coordinación no es suficiente, lo que hace necesario un tipo de tecnología adaptada a procesos de coordinación y colaboración [?].

Esto acarrea el aumento de esfuerzo que los equipos tienen que realizar para poder coordinarse correctamente, lo que puede provocar retrasos en el proyecto.

3.1.2.6 Desafíos en la coordinación y la distancia socio-cultural

Debido a la distancia socio-cultural entre los distintos CDS, aparecen las siguientes dificultades en la coordinación [?] [?].

3.1.2.6.1 Formación cultural y lingüística

Aumenta el coste dedicado a la formación del idioma y la cultura de los diferentes miembros de los CDS, necesario para mejorar la comunicación y coordinación en los proyectos.

3.1.2.6.2 Falta de dominio de conocimiento

Las tareas de coordinación y desarrollo del proyecto requiere conocer el dominio de dicho proyecto, lo que algunos desarrolladores no conocen al provenir de diferentes culturas y países, teniendo cada uno de ellos diferentes puntos de vista sobre un mismo asunto.

En la Tabla ?? se resumen los desafíos encontrados en el proceso de coordinación en GSD.

3.1.2.7 Desafíos en el control y la distancia geográfica

Debido a la distancia geográfica entre los distintos CDS, aparecen las siguientes dificultades en el control [?].

Coordinación	
Geográfica	Reducción de la confianza Falta de espíritu y conciencia de equipo Modularización del trabajo Aumento del esfuerzo en acuerdos de externalización Riesgo de alternativas de bajo coste
Temporal	Reducción de horas de colaboración Disponibilidad de infraestructura técnica
Socio-cultural	Formación cultural y lingüística Falta de dominio de conocimiento

Tabla 3.4: Desafíos en el proceso de coordinación en GSD

3.1.2.7.1 Asignación de roles y estructuras de equipo

La deslocalización de los equipos de desarrollo complica y aumenta los costes en la asignación de roles y estructuras de equipos, debido a la multitud de habilidades de todas las personas involucradas en diferentes países.

3.1.2.7.2 Aplicación de principios de ingeniería del software concurrente

La ingeniería de software concurrente (en adelante CSE, por sus siglas en inglés Concurrent Software Engineering) es una técnica utilizada para disminuir el tiempo de lanzamiento al mercado (“time-to-market”), realizando varias actividades al mismo tiempo [?].

Implementar principios de CSE en GSD es muy complicado, debido a recursos no disponibles y falta de coordinación y comunicación.

3.1.2.7.3 Riesgos organizacionales

En el GSD, parte del control y gestión de proyectos puede recaer sobre países emergentes, los cuales, históricamente, han sido menos estables, predecibles y transparente. Por tanto pueden existir riesgos de terrorismo, guerra, expropiación, problemas de propiedades intelectuales, pérdida de datos, etc.

3.1.2.8 Desafíos en el control y la distancia temporal

Debido a la distancia temporal entre los distintos CDS, aparecen las siguientes dificultades en el control [?] [?].

3.1.2.8.1 Gestión de los artefactos del proyecto

Cuando un proyecto involucra miembros de diferentes compañías o de diferentes CDS de la misma organización, hacer cumplir los procesos y normas del artefacto software es particularmente importante para mantener la consistencia e interoperabilidad entre los diferentes artefactos del proyecto.

3.1.2.9 Desafíos en el control y la distancia socio-cultural

Debido a la distancia socio-cultural entre los distintos CDS, aparecen las siguientes dificultades en el control [?].

3.1.2.9.1 Adaptación a normas y estructuras locales

Se deben aprender las normas y estructuras locales de cada país (tradiciones, regulaciones, leyes de exportación e importación, leyes aplicables, etc.), lo que puede afectar a los métodos de desarrollo de los proyectos.

3.1.2.9.2 Diferentes percepciones de la autoridad

La naturaleza de la autoridad dentro de un proyecto puede variar entre las diferentes culturas, por lo que pueden surgir problemas en el control de un proyecto si los equipos en diferentes lugares esperan que se maneje de manera diferente, y si estas expectativas diferentes no se identifican desde el principio.

En la Tabla ?? se resumen los desafíos encontrados en el proceso de control en GSD.

Coordinación	
Geográfica	Asignación de roles y estructuras de equipo Aplicación de principios de ingeniería del software concurrente Modularización del trabajo Riesgos organizacionales
Temporal	Gestión de los artefactos del proyecto
Socio-cultural	Adaptación a normas y estructuras locales Diferentes percepciones de la autoridad

Tabla 3.5: Desafíos en el proceso de control en GSD

3.2 Gestión de conocimiento

En la actualidad, nos encontramos sumidos en la era digital, en la era de la información, es decir, vivimos en la sociedad de la información y del conocimiento, que puede definirse como “*aquella sociedad en la que las tecnologías facilitan la creación, distribución, manipulación y reutilización de la información y el conocimiento*” [?] [?].

Debido a que vivimos en esta era digital, la información y el conocimiento se han convertido en el recurso más importante para las empresas, más incluso que los bienes materiales tangibles, dependiendo su economía, productividad y competitividad directamente de su capacidad para adquirir y utilizar la información y el conocimiento [?] [?] [?].

La **información** se puede definir como “*un conjunto de datos, organizados, que aportan un significado para el receptor que los recibe*” [?]. Por otra parte, el **conocimiento** puede definirse como ”*una mezcla fluida de experiencias enmarcadas, valores, información contextual, y pericia que provee de un marco para evaluar e incorporar nuevas experiencias e información. Esto se origina y es aplicado en la mente de quienes conocen. En las organizaciones, con frecuencia viene embebido no solo en documentos o repositorios, sino también en rutinas, procesos, prácticas o normas de la organización*” [?]. En la Figura ?? se puede observar la relación entre datos, información y conocimiento [?].

Por todo ello, las empresas están centrando sus esfuerzos en nueva infraestructura tecnológica, procesos y recursos necesarios para poder gestionar y utilizar el conocimiento y la

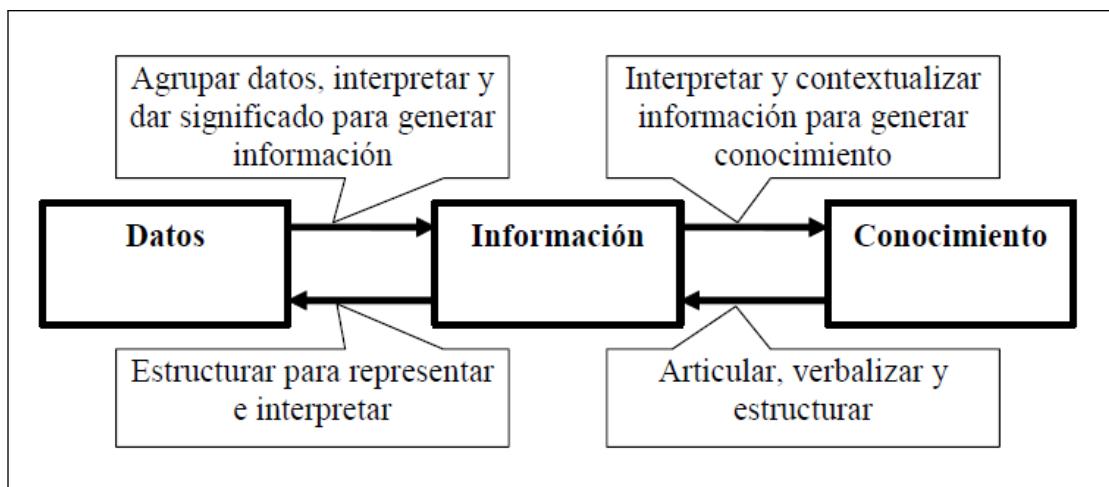


Figura 3.4: Relaciones entre datos, información y conocimiento

información, vitales para que la empresa siga siendo competitiva en el mercado. Surge así lo que se conoce como *economía informacional* [?] y **Gestión del Conocimiento** (en adelante KM, por sus siglas en inglés Knowledge Management) [?].

En la actualidad, no existe un consenso acerca de la definición de gestión de conocimiento, ya que KM es un campo multidisciplinar con diferentes perspectivas y definiciones, según donde se aplique. En [?] se presentan tres perspectivas de KM:

1. Desde la perspectiva de *negocio*, donde el KM se considera parte de la estrategia, políticas y prácticas del negocio. En este sentido, KM es un enfoque colaborativo e integrado para la captura, creación y acceso de activos de negocio [?].
2. Desde la perspectiva de la *ciencia cognitiva*, donde el conocimiento es el recurso fundamental para la inteligencia.
3. Desde la perspectiva *tecnológica y de procesos*, donde KM es el concepto por el cual la información se convierte en conocimiento potencialmente utilizable y útil para las personas.

Desde este último punto de vista, en el que nos centraremos en este documento, la gestión de conocimiento se podría definir como “*un proceso que permite crear, capturar, identificar, almacenar, buscar, recuperar, compartir, transferir y diseminar el conocimiento existente dentro de una organización, con el fin de incrementarlo y evitar su pérdida y sub-utilización*” [?] [?] [?]. En la Figura ??, se puede apreciar un modelo básico de KM.

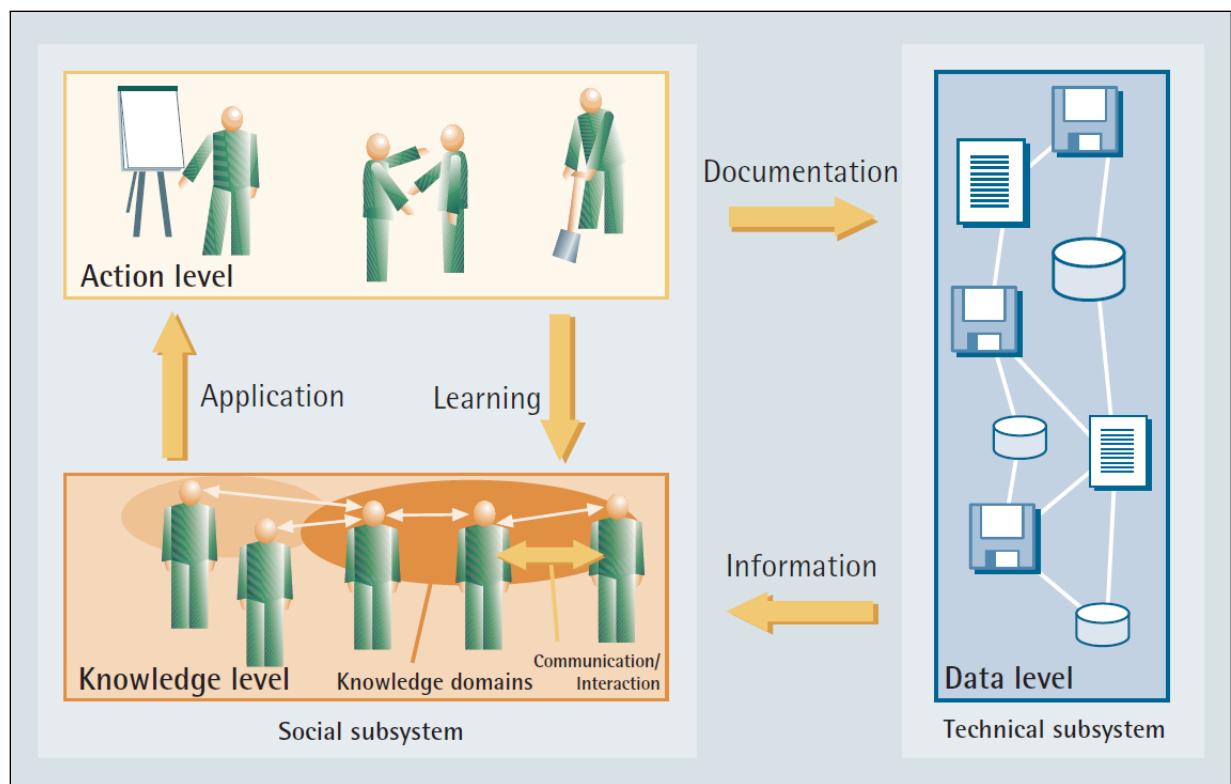


Figura 3.5: Modelo básico de gestión del conocimiento

En el caso concreto de una organización dedicada al desarrollo software, el conocimiento que más interesa gestionar son las alternativas y decisiones planteadas en las fases de desarrollo de un producto software, los diferentes diagramas, la documentación, las pruebas del software, etc.

Para que la adquisición y manipulación de conocimiento sea efectiva, las organizaciones y compañías deben integrar entre sus procesos de negocio un marco de trabajo para la gestión de conocimiento, que involucra una serie de pasos a seguir [?], desarrollado en el siguiente apartado.

3.2.1 Marco de trabajo para la gestión de conocimiento

Atendiendo a la definición de KM en la perspectiva tecnológica y de procesos, en [?] y [?] se definen cuatro tareas que componen un marco de trabajo organizacional para la gestión de conocimiento. Éstas son:

- Creación (o construcción).

- Almacenamiento/Recuperación.
- Transmisión.
- Aplicación.

Se detallan a continuación cada uno de estos procesos.

3.2.1.1 Creación de conocimiento

La creación de conocimiento organizacional involucra crear nuevo conocimiento, o reemplazar ciertos contenidos con nuevo conocimiento tácito y explícito, obtenido tanto a través de colaboraciones e interacciones sociales, como a partir del propio proceso cognitivo de cada empleado [?].

Por conocimiento *tácito* se entiende como aquel conocimiento personal de cada individuo que necesita para realizar sus tareas. Por otra parte, el conocimiento *explícito* es aquel expresado de manera formal (palabras, números, etc.) por lo que se puede comunicar y transferir de manera sencilla [?].

Existen cuatro modos identificados para la creación de conocimiento [?] [?] [?]:

- **Socialización:** proceso por el cual se convierte el conocimiento tácito en nuevo conocimiento tácito a partir de interacciones sociales y experiencias compartidas. De este modo, los individuos comparten sus conocimiento tácito.
- **Exteriorización:** proceso por el cual el conocimiento tácito se convierte en conocimiento explícito, convirtiéndolo a medios formales como documentos, gráficos, imágenes, etc.
- **Interiorización:** proceso por el cual se convierte el conocimiento explícito en tácito, consultando reiteradamente conocimiento explícito en sus actividades diarias.
- **Combinación:** proceso por el cual se crea nuevo conocimiento explícito mezclando, clasificando y sintetizando otro conocimiento explícito.

En la Figura ?? se muestra un resumen del ciclo de creación de conocimiento [?].

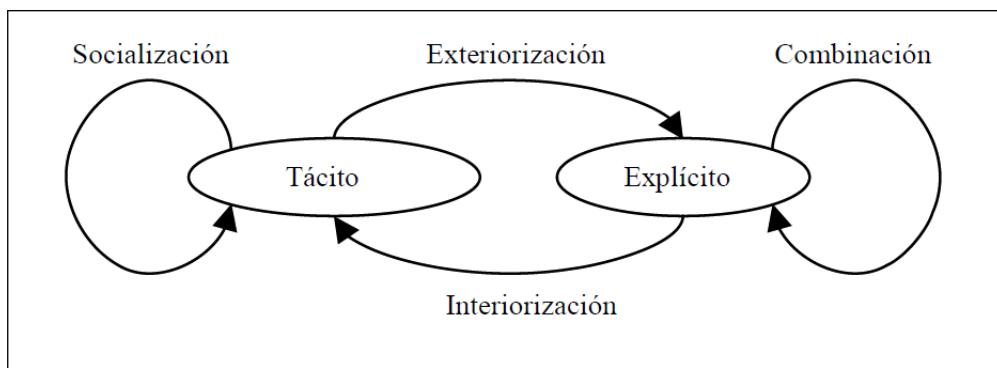


Figura 3.6: Mecanismos de creación de conocimiento

3.2.1.2 Almacenamiento y recuperación del conocimiento

Del mismo modo que las organizaciones generan y aprenden nuevo conocimiento, también pueden olvidarlo. Por ello, para una gestión de conocimiento eficaz, es necesario que las empresas almacenen y utilicen su conocimiento generado [?]. La memoria organizacional de conocimiento suele formarse a partir de documentos, bases de datos estructuradas, procesos y procedimientos de negocio, vídeos, etc. Por lo general, a la hora de almacenar el conocimiento, hay que tener en cuenta el tipo de almacenamiento, el alcance (personal, grupal, organizacional o inter-organizacional) y el tipo de conocimiento a almacenar.

Una vez almacenado el conocimiento de la empresa, éste puede ser utilizado en cualquier momento (e incluso en cualquier lugar, dependiendo del tipo de almacenamiento) por los individuos, utilizando herramientas como consultas de bases de datos, una intranet para procesos y documentos internos, wikis, etc.

3.2.1.3 Transmisión del conocimiento

El objetivo primordial de KM es que el conocimiento pueda ser utilizado y conocido por las empresas, para que puedan ser competitivas en el marco de globalización y sociedad de la información en el que nos encontramos, por lo que los mecanismos de transferencia de conocimiento, tanto a nivel interno como inter-organizacional, son de vital importancia en KM.

La transmisión de conocimiento principalmente se produce por procesos de socialización, donde un grupo de individuos, normalmente con intereses comunes, comparten el

conocimiento e intentan aprender y resolver problemas juntos [?].

Un aspecto a tener en cuenta en este proceso de transmisión, es que el conocimiento debe poder ser accesible desde cualquier momento y lugar, por lo que las tecnologías de comunicación son cruciales, así como los métodos utilizados para esta transmisión (e-mail, wikis, conferencias, etc.).

3.2.1.4 Aplicación del conocimiento

Las organizaciones no llegan a ser competitivas sólo por poder crear, almacenar, recuperar y transferir conocimiento, sino por la aplicación de dicho conocimiento. Así, una empresa puede utilizarlo para la toma de decisiones, para mejorar sus procesos de negocio, para crear productos mejores, para conocer el mercado y adaptarse más rápidamente a sus necesidades, etc.

Aplicar el conocimiento supone ejecutar todos los procesos anteriores ya que, por ejemplo, en el caso de tomar decisiones, primero se debe obtener todo el conocimiento necesario para llevar a cabo esa decisión. Además, este proceso de aplicación, genera nuevo conocimiento, haciendo que el conocimiento organizacional crezca y evolucione para futuras aplicaciones, formando un ciclo (ver Figura ??).

En el caso particular del desarrollo software, ya sea localizado, distribuido o global, la gestión y aplicación de conocimiento provee beneficios como reducción de costes, reducción de errores, mejora en la calidad del producto, etc. Esto se debe a que se puede aprender y reutilizar conocimiento (código, documentos, pruebas, diagramas, etc.) de otros productos software.

3.2.2 Beneficios y desafíos de la gestión de conocimiento

En esta sección se resaltan algunos de los beneficios y dificultades de KM al utilizarse en las organizaciones [?] [?] [?] [?].

3.2.2.0.1 Beneficios

- Incremento de la competitividad.

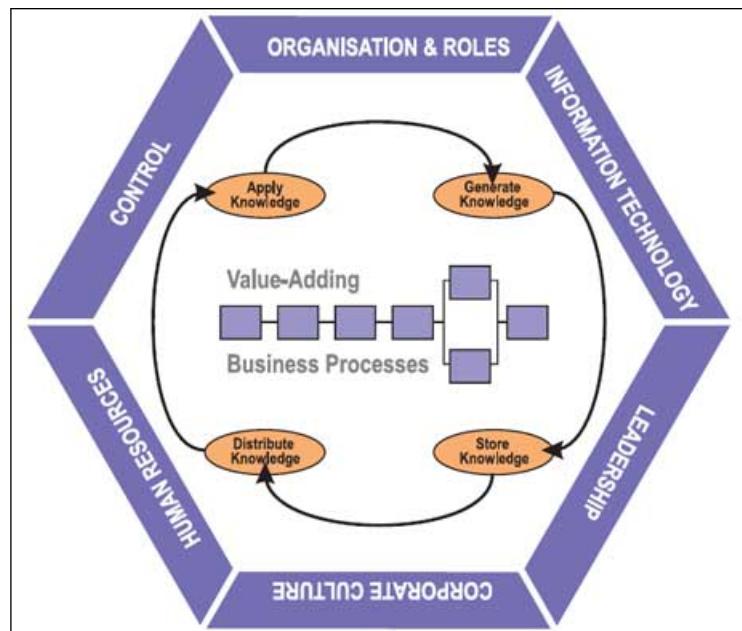


Figura 3.7: Ciclo de gestión de conocimiento

- EN GSD, suple la carencia de interacción “face-to-face” y provee mecanismos de coordinación y aprendizaje mutuo.
- Mejora la toma de decisiones de la empresa.
- Mayor motivación del equipo, al estar involucrados en la creación y aprendizaje del conocimiento.
- La organización puede disponer del conocimiento adecuado en el momento adecuado.
- Reduce el tiempo para resolver problemas y del ciclo de desarrollo de productos.
- Mayor cercanía a las necesidades del mercado.
- Incremento de ventas.

3.2.2.0.2 Problemas

- Mantener el conocimiento en constante evolución y actualización.
- Integrar tecnologías en la empresa para la gestión de conocimiento.
- Barreras y diferencias culturales y de trabajo entre diferentes miembros de los equipos, especialmente en GSD.

- Falta de estandarización para el conocimiento almacenado.
- Inversión de tiempo para la generación de conocimiento.

3.3 Razonamiento basado en casos

Estrechamente relacionado con la gestión de conocimiento desarrollada en la sección ??, está el concepto de **Razonamiento Basado en Casos** (en adelante CBR, por sus siglas en inglés Case-Based Reasoning).

El CBR puede definirse como “*un proceso en el que experiencias específicas son recuperadas, reutilizadas, revisadas y almacenadas para utilizarse en la solución de problemas similares*” [?]. De este modo, se pueden encontrar soluciones a nuevos problemas basándose en soluciones de problemas similares anteriores. Dichas soluciones puede que sea necesario adaptarlas para resolver el nuevo problema y se almacenarán en forma de nuevas experiencias (casos) para poder reutilizarlas en un futuro. Es por esto que un sistema CBR puede “aprender” a partir de nuevas soluciones [?].

A diferencia de otras técnicas y algoritmos de Inteligencia Artificial, como pueden ser Razonamiento Basado en Reglas o Algoritmos Genéticos, CBR no se considera como una tecnología, sino más bien como una metodología, que indica cómo resolver problemas a partir de soluciones previas almacenadas en el sistema, pero sin detallar una tecnología concreta [?]. Por tanto, un sistema CBR se define por lo que hace, pero no cómo se hace o se implementa.

Conceptualmente, el CBR se describe como un ciclo (ver Figura ??) con cuatro grandes etapas [?] [?]:

- **Recuperación (Retrieve)** de casos similares al problema planteado.
- **Reutilización (Reuse)** de una solución propuesta por un caso similar.
- **Revisión (Revise)** de la solución propuesta, para adaptarla mejor a las condiciones del nuevo problema.
- **Retención (Retain)** de la nueva solución, pasando a formar un nuevo caso.

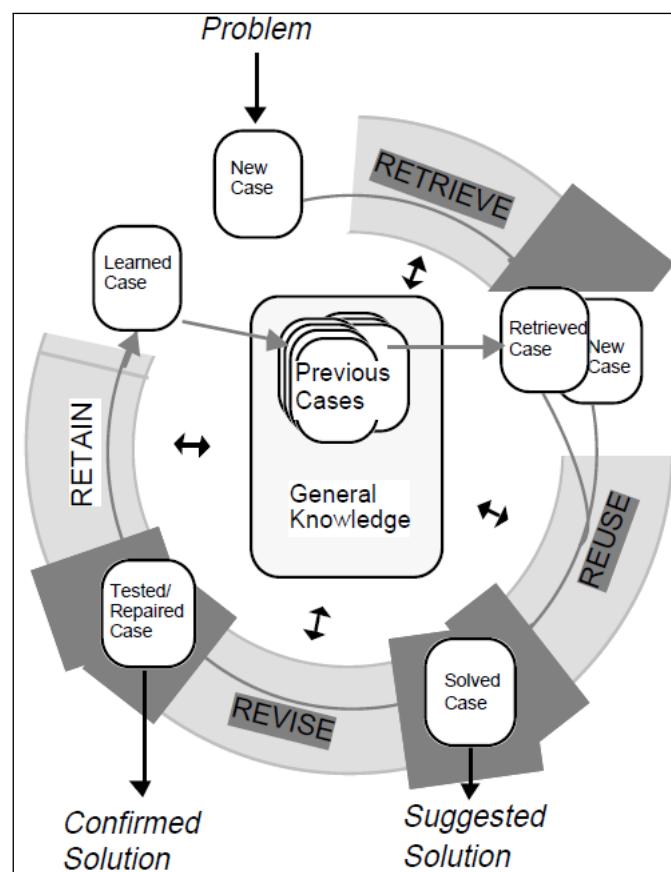


Figura 3.8: Ciclo del CBR

Estas cuatro etapas son las que constituyen la metodología del CBR. Así, para solucionar un nuevo problema, primero se debe obtener una descripción de éste, midiendo la similitud del nuevo problema con otros problemas previos almacenados en el sistema. A continuación, se recuperan las soluciones de estos problemas similares y se reutiliza la solución de uno de estos casos, adaptándola si es necesario. Para terminar, este nuevo problema, junto a la solución encontrada, se almacena en el sistema, formando un nuevo caso.

Uno de los puntos críticos en el CBR es la función de similitud que se va a utilizar para buscar casos similares a uno dado. Por ejemplo, un problema se podría describir como una serie de atributos (o características) que pueden ser cuantificadas con un “peso” numérico, obteniendo un vector numérico, y utilizar como función de similitud la distancia Euclídea entre los vectores que representan cada problema.

Como se puede apreciar, el ciclo que se ha visto anteriormente encaja perfectamente con un sistema de gestión del conocimiento (KMS, por sus siglas en inglés Knowledge Management System), pues en este tipo de sistemas también existen fases de creación, recuperación, reutilización y aplicación, en este caso de conocimiento, como se ha visto en la sección anterior. El CBR se basa en conocimiento previo (casos) para buscar soluciones a un nuevo problema, generando nuevo conocimiento, que son los nuevos casos producidos en la fase de retención. Por lo tanto, el CBR y KM están estrechamente relacionados.

Capítulo 4

Método de Trabajo

Para el desarrollo de este producto software se ha optado por utilizar la metodología genérica descrita por el **Proceso Unificado de Desarrollo** (en adelante PUD), propuesta por Rumbaugh, Booch y Jacobson [?].

Para el modelado de los diagramas del producto software se utilizará el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés Unified Modelling Language).

4.1 Proceso Unificado de Desarrollo

El Proceso Unificado de Desarrollo es una evolución del Proceso Unificado de Rational (RUP), que define un “*conjunto de actividades necesarias para transformar los requisitos de usuario en un sistema software*” [?]. Más concretamente se puede definir como “*un marco de trabajo marco genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos.*” [?].

Las principales características del PUD son:

- **Dirigido por casos de uso.** Un caso de uso representa un requisito funcional al cuál el sistema debe dar soporte para proporcionar un resultado de valor al usuario. Los casos de uso guían el proceso de desarrollo, ya que basándose en los casos de uso, los desarrolladores crean una serie de modelos para poder llevarlos a cabo. Todos los casos de uso juntos constituyen el **modelo de casos de uso**.
- **Centrado en la arquitectura.** Un sistema software puede contemplarse desde varios

puntos de vista. Por tanto, la arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema y debe estar profundamente relacionada con los casos de uso, ya que debe permitir el desarrollo de los mismos.

- **Iterativo e incremental.** El esfuerzo de desarrollar un proyecto de software se divide en partes más pequeñas, llamadas **mini-proyectos**. Cada mini-proyecto es una **iteración**, compuesta por una serie de requisitos funcionales representados por casos de uso, y que abarcan entre dos y seis semanas de duración. Las iteraciones deben estar controladas y deben seleccionarse y ejecutarse de una forma planificada, siguiendo el esquema *requisitos, análisis, diseño, implementación y pruebas*, que es conocido como **flujo de trabajo**. En cada iteración, los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño mediante componentes y verifican que los componentes satisfacen los casos de uso. Finalmente, cada iteración aporta un **incremento** en la funcionalidad del sistema, por eso que el PUD se considere un proceso incremental (ver Figura ??) [?].

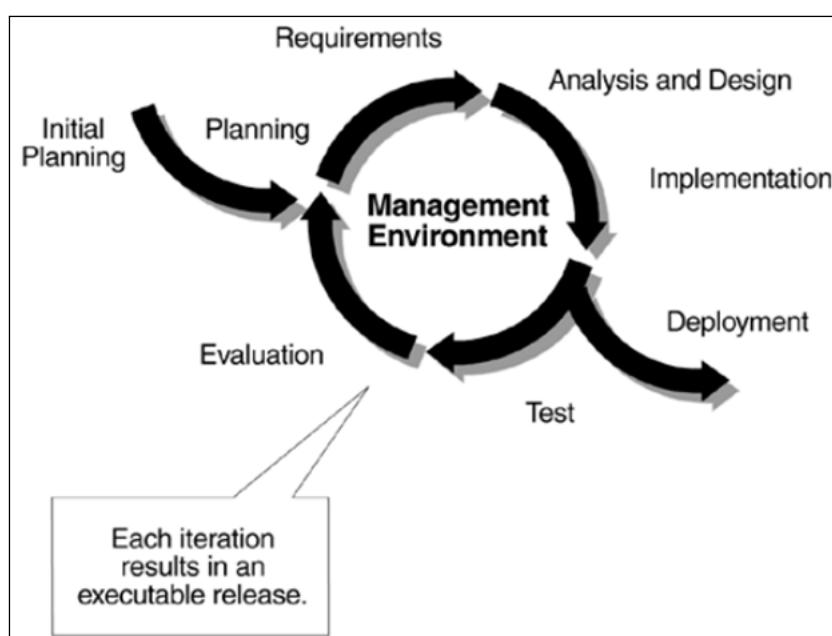


Figura 4.1: Proceso iterativo e incremental

4.1.1 Fases de desarrollo basado en PUD

En PUD, el ciclo de vida de un producto software se divide en ciclos, donde cada uno de estos ciclos compone una versión del producto, totalmente operativa y preparada para su implantación. Cada ciclo se compone de cuatro fases (Inicio, Elaboración, Construcción y Transición) y éstas, a su vez, se dividen en iteraciones que siguen el flujo de trabajo *requisitos, análisis, diseño, implementación y pruebas* (ver Figura ??).

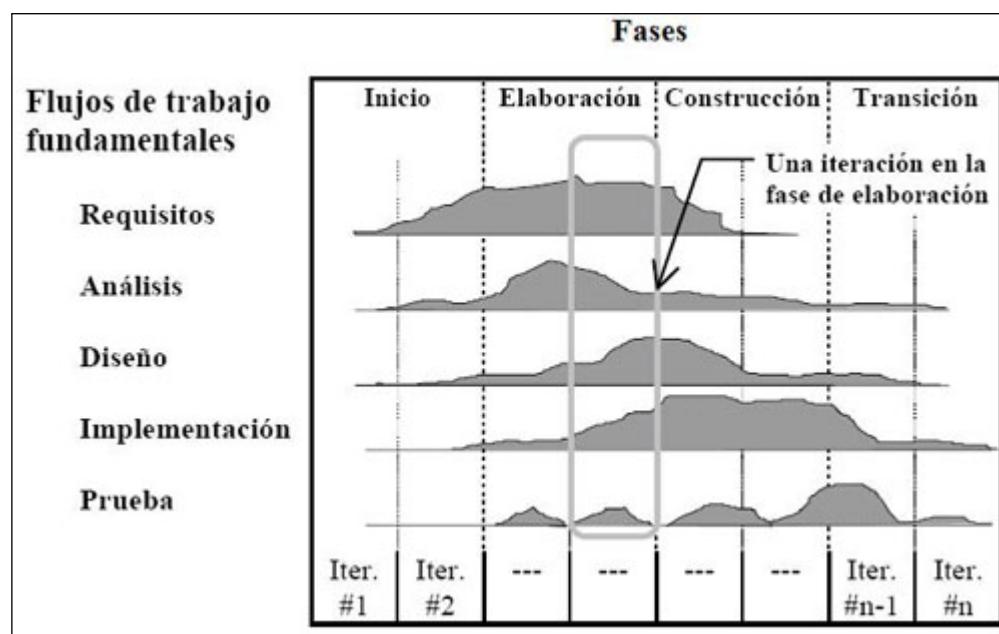


Figura 4.2: Fases y flujo de trabajo del PUD

A continuación se detallan cada una de las cuatro fases de cada ciclo del PUD [?] [?].

4.1.1.1 Fase de Inicio

En esta fase es donde se determina el alcance del proyecto, su viabilidad, riesgos potenciales y donde se realiza una planificación del proyecto. Normalmente, esta fase abarca una única iteración y los artefactos más relevantes utilizados son:

- **Modelo de Casos de Uso:** se obtiene un primer modelo de casos de uso simplificado, representando los requisitos funcionales del sistema.
- **Descripción de Riesgos:** se obtiene un documento que recoge posibles riesgos que pueden aparecer y afectar al ciclo de vida del producto.

- **Glosario:** se elabora un glosario de términos del dominio de aplicación.
- **Plan de Proyecto:** se elabora un plan de iteraciones a seguir para el desarrollo del producto software.

4.1.1.2 Fase de Elaboración

En esta fase, compuesta de una una o más iteraciones, se especifican en detalle la mayoría de los casos de uso identificados en la fase de inicio y se diseña la arquitectura del sistema, obteniendo la línea base de la arquitectura. Por arquitectura se entiende el “*conjunto de decisiones significativas acerca de la organización de un sistema software, la selección de los elementos estructurales a partir de los cuales se compone el sistema, las interfaces entre ellos, su comportamiento, sus colaboraciones, y su composición*” [?].

En esta fase, se construyen los siguientes artefactos:

- **Arquitectura:** se obtiene el diseño de la arquitectura del sistema.
- **Modelo de Casos de Uso:** se obtiene un modelo de casos más detallados y con todos los requisitos funcionales.
- **Modelo de Análisis:** es un modelo compuesto de diagramas de clases de análisis y de secuencia, más detallado que el de casos de usos pero menos que el de diseño.
- **Modelo de Diseño:** es un modelo compuesto de diagramas de clases de diseño que describen el funcionamiento del sistema.

4.1.1.3 Fase de Construcción

En la fase de construcción es donde se lleva a cabo la implementación de cada una de las iteraciones en las que se ha dividido el desarrollo del producto software, a partir de los artefactos generados en la fase de elaboración. Aunque esta fase se centra en la implementación, puede haber iteraciones donde también haya tareas de requisitos, análisis y diseño, refinando modelos de etapas anteriores, pero éstas serán prácticamente nulas en iteraciones avanzadas de esta fase. Cada iteración se cierra con las pruebas realizadas al código implementado.

Esta fase abarca un número muy variable de iteraciones y suele ser la fase de más larga duración en el ciclo de vida del producto software. En esta fase se obtienen los siguientes artefactos:

- **Modelo de Diseño:** a partir del modelo de diseño de la fase anterior, se obtiene otro modelo refinado durante iteraciones de esta fase.
- **Modelo de Implementación:** es un modelo compuesto por diagramas de componentes, junto a sus relaciones y dependencias.
- **Modelo de Pruebas:** es el conjunto de casos de pruebas unitarias que cierran cada iteración de esta fase.
- **Modelo de Despliegue:** modelo que refleja los aspectos físicos y como se ejecutarán los componentes identificados en el modelo de implementación.

4.1.1.4 Fase de Transición

La fase de transición es el período en el cual el producto se convierte en una versión beta, es decir, se procede a su implantación pero se seguirá probando y, quizás, incrementando su funcionalidad. En esta fase se obtienen los siguientes artefactos:

- **Modelo de Despliegue:** se obtiene el modelo final a partir del de la fase anterior.
- **Modelo de Distribución:** es un modelo que muestra el funcionamiento físico del sistema.
- **Manuales:** el sistema se documenta y se generan los manuales de usuario para explicar su funcionamiento.

4.1.2 Flujo de trabajo en el PUD

Como se ha comentado en la sección ??, en cada una de las fases desarrolladas anteriormente se sigue un flujo de trabajo, compuesto por las disciplinas de captura de requisitos, análisis, diseño, implementación y pruebas.

Según en la fase e iteración en la que se encuentre el desarrollo, estas disciplinas serán más o menos relevantes en dicha iteración dentro de esa fase (ver Figura ??).

En los siguientes apartados se comentan las actividades a realizar en cada una de las disciplinas que forman el flujo de trabajo de cada fase del PUD [?] [?].

4.1.2.1 Captura de requisitos

Esta disciplina tiene una mayor relevancia en las fases de Inicio y Elaboración y tiene como objetivo el identificar los requisitos del producto para guiar su posterior desarrollo. Se realizan las siguientes actividades:

- Se identifican los requisitos funcionales del sistema a desarrollar.
- Se identifica el contexto o dominio de aplicación, para comprender los requisitos.
- Se identifican los requisitos no funcionales del sistema, como comunicación con otros sistemas, entorno de funcionamiento, etc.
- Los requisitos identificados se modelan utilizando un diagrama de casos de uso, que será más o menos detallado según la fase del PUD en la que nos encontramos.

4.1.2.2 Análisis

Esta disciplina tiene una mayor relevancia en la fase de Elaboración y tiene como objetivo la especificación detallada de los casos de uso obtenidos anteriormente, refinándolos con posibles nuevos requisitos. Se realizan las siguientes actividades:

- Se detallan los casos de uso obtenidos en fases anteriores.
- Se realiza el diagrama de clases de análisis.
- Se identifican nuevos requisitos que puedan surgir al refinar los casos de uso y generar los diagramas de análisis.

4.1.2.3 Diseño

Esta disciplina tiene una mayor relevancia en las fases de Elaboración y Construcción y tiene como objetivo el modelado del sistema y de su arquitectura para soportar los requisitos, tanto funcionales como no funcionales. Se realizan las siguientes actividades:

- Se realiza el modelo de diseño, con diagramas de clases detallados y las relaciones entre clases.
- Se identifica la arquitectura del sistema.
- Se realiza el modelo de despliegue.

4.1.2.4 Implementación

Esta disciplina tiene una mayor relevancia en la fase de Construcción y tiene como objetivo generar la herramienta software. Se realizan las siguientes actividades:

- Se realiza el modelo de implementación.
- Se detalla y refina la arquitectura del sistema.
- Se detalla y refina el modelo de despliegue.
- Se codifican todos los requisitos del sistema, obteniendo código ejecutable que le da soporte a dichos requisitos.

4.1.2.5 Pruebas

Esta disciplina afecta a todas las iteraciones y fases donde se haya producido una implementación, ya que es necesario probar que dicha codificación funciona correctamente. Su objetivo es crear los casos de prueba para cada iteración y crear un plan de pruebas del sistema global. Se realizan las siguientes actividades:

- Se realiza el modelo de pruebas, creando casos de prueba unitarios, de integración y de sistema.

4.1.3 Resumen del Proceso Unificado de Desarrollo

El PUD está compuesto de ciclos, donde cada ciclo consta de cuatro fases y cada fase, a su vez, se divide en una o varias iteraciones, donde se aplica un flujo de trabajo compuesto por cinco disciplinas. En la Tabla ?? se resumen las cuatro fases de un ciclo presentes en el PUD.

Fase	Resumen	Artefactos generados
Inicio	Se identifican los requisitos Se crea un glosario de términos Se analiza la viabilidad del proyecto Se realiza el plan de iteraciones	Modelo de casos de uso Glosario de términos Descripción de riesgos Plan de proyecto
Elaboración	Se crea un modelo de casos de uso detallado Se diseña la arquitectura Se crea un modelo de clases de análisis Se realiza el modelo de diseño	Modelo de casos de uso detallado Modelo de Diseño Arquitectura del sistema Modelo de Análisis
Construcción	Se implementan los requisitos del sistema Se definen pruebas Se crea un modelo de despliegue	Modelo de implementación Modelo de pruebas Modelo de despliegue Ejecutables del sistema
Transición	Se escribe la documentación y manuales de usuario Se explica como instalar e implantar el sistema	Documentación Manuales de usuario Modelo de distribución

Tabla 4.1: Resumen de las fases de un ciclo del PUD

En el capítulo 5 se expondrán los resultados obtenidos al aplicar PUD en el desarrollo del presente PFC, desarrollando cada una de las fases y qué artefactos se han ido obteniendo en cada una de ellas hasta llegar al sistema final.

4.2 Marco tecnológico de trabajo

En esta sección se expondrán las herramientas, tecnologías y librerías utilizadas durante todo el ciclo de vida del proyecto.

4.2.1 Herramientas para la gestión del proyecto

4.2.1.1 Subversion

Subversion es un sistema de control de versiones donde, a diferencia de CVS (Concurrent Versions System), los archivos versionados del repositorio no tienen cada uno un número de versión distinto, sino que todo el repositorio tiene un único numero de revisión. Además, Subversion permite acceder a repositorios de red.

En el proyecto, se utilizará un repositorio Subversión (o SVN) para gestionar las diferentes versiones del mismo. El repositorio SVN estará alojado en *Google Code*, de manera que será accesible en red. Para acceder a dicho repositorio, se utilizará la herramienta **TortoiseSVN**, que es un cliente de Subversion implementado como una extensión del shell de sistemas Windows y que permite gestionar el repositorio. En la Figura ?? se puede observar esta herramienta.

4.2.1.2 Maven

Maven es una herramienta de gestión y construcción de proyectos, creada por Jason van Zyl en 2002. Maven permite describir el proyecto a construir, sus dependencias con otros módulos y el orden de construcción utilizando un Modelo de Objetos del Proyecto (POM, por sus siglas del inglés Project Object Model), que es un fichero XML donde se describe el proyecto, sus dependencias, etc.

Maven tiene una arquitectura dinámica basada en *plugins* que son descargables desde su repositorio central, de tal modo que permite la reutilización para diferentes proyectos, cambiando los plugins a utilizar y su configuración y manteniendo patrones similares en diferentes proyectos.

Se utilizará esta herramienta, en su versión 2.2.1, para la gestión del proyecto y sus

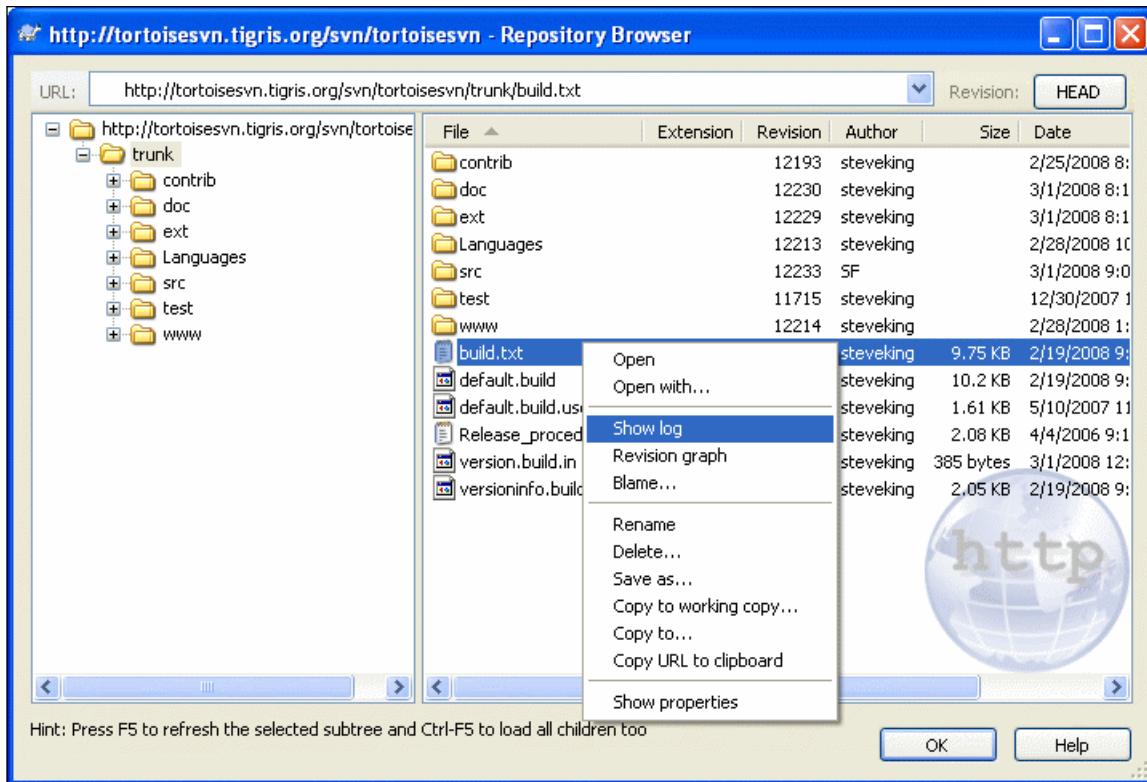


Figura 4.3: Herramienta *TortoiseSVN*

dependencias con otras librerías y módulos, usando el plugin de Maven para Eclipse llamado **m2eclipse**. En la Figura ?? se puede observar la interfaz de este plugin sobre Eclipse.

4.2.2 Herramienta para el modelado

4.2.2.1 Visual Paradigm

Visual Paradigm es una herramienta CASE (Computer Aided Software Engineering o Ingeniería de Software Asistida por Computador) profesional para el modelado UML (Unified Modeling Language o Lenguaje Unificado de Modelado) que soporta el ciclo de vida del desarrollo de un producto software. Permite crear diferentes tipos de diagramas y también permite la ingeniería inversa, es decir, a partir de código, generar diferentes diagramas.

Esta herramienta, en su versión 8.0, se utilizará para realizar los diagramas de casos de uso, de secuencia, de clases, etc. En la Figura ?? se puede observar la interfaz de esta herramienta.

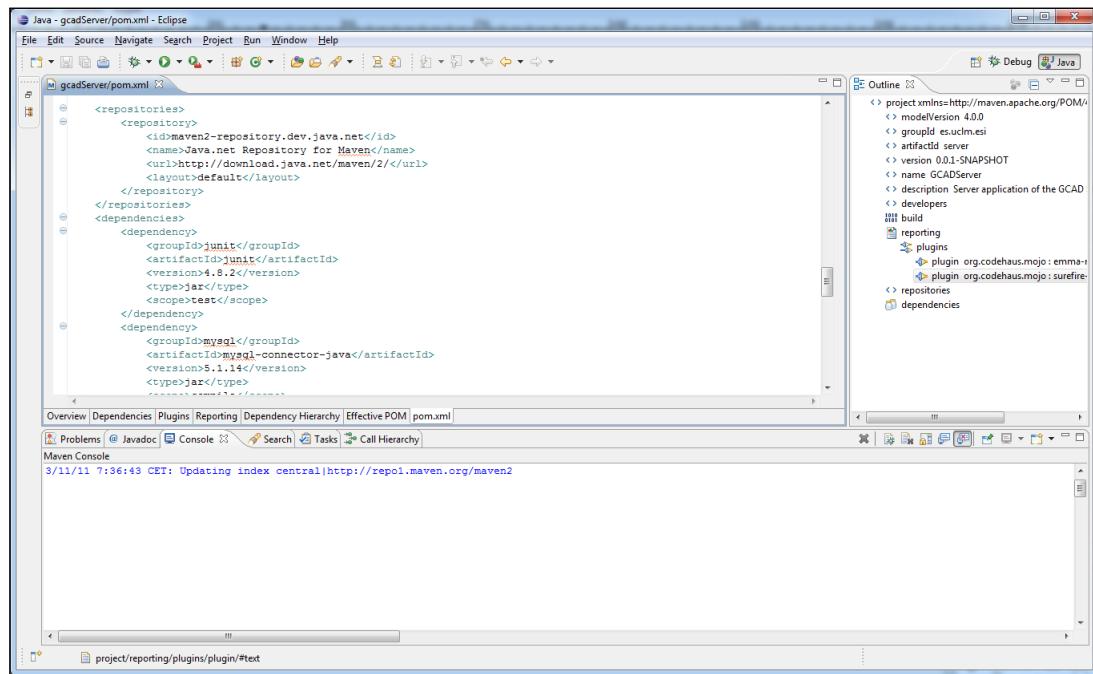


Figura 4.4: Plugin de Maven sobre Eclipse Helios

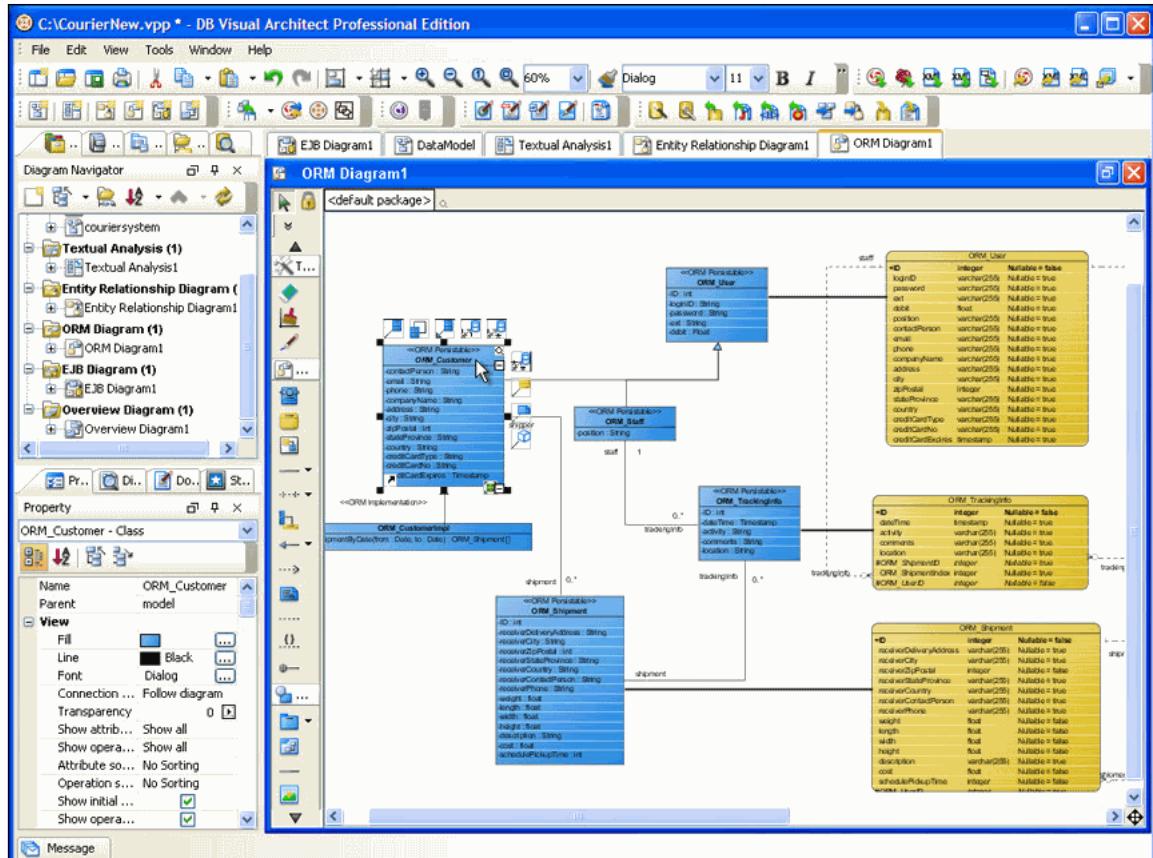


Figura 4.5: Herramienta Visual Paradigm

4.2.3 Herramientas y tecnologías para el desarrollo del proyecto

4.2.3.1 Eclipse Helios

Eclipse es un IDE (Integrated Development Environment o Entorno Integrado de Desarrollo) multiplataforma, de código abierto y basado en plugins para aumentar su funcionalidad.

Es el entorno de desarrollo utilizado para la codificación de proyecto utilizando el lenguaje de programación Java, junto a plugins utilizados para la gestión de proyectos con Maven (plugin **m2eclipse**) y diseño de interfaces gráficas con Swing (plugin **Jigloo**). En la Figura ?? se muestra una captura de pantalla de este IDE.

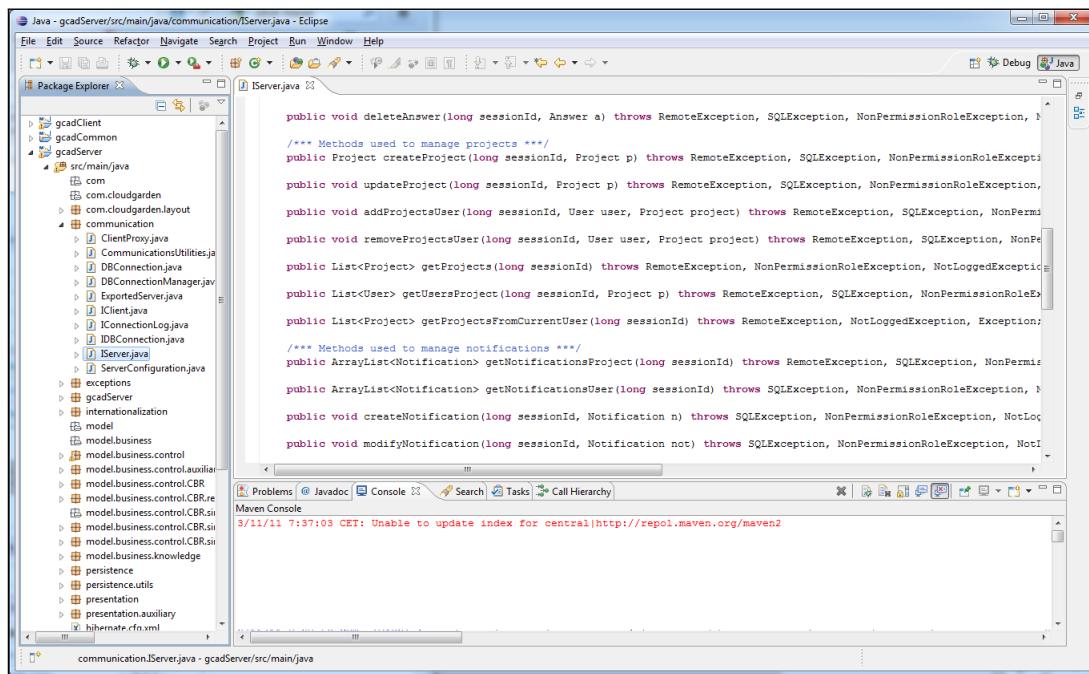


Figura 4.6: Eclipse Helios

4.2.3.2 Yahoo! PlaceFinder

Yahoo! PlaceFinder es un servicio web de geocodificación que permite la conversión de direcciones de calles o nombres de lugares en coordenadas geográficas, y viceversa. La respuesta a una petición de este servicio Web viene dada por un fichero XML, como el mostrado en la Figura ??.

Se utilizará para obtener y localizar en mapas las coordenadas geográficas a partir de

direcciones postales.

```
- <ResultSet version="1.0">
  <Error>0</Error>
  <ErrorMessage>No error</ErrorMessage>
  <Locale>us_US</Locale>
  <Quality>87</Quality>
  <Found>1</Found>
  - <Result>
    <quality>85</quality>
    <latitude>38.898717</latitude>
    <longitude>-77.035974</longitude>
    <offsetlat>38.898590</offsetlat>
    <offsetlon>-77.035971</offsetlon>
    <radius>500</radius>
    <name/>
    <line1>1600 Pennsylvania Ave NW</line1>
    <line2>Washington, DC 20006</line2>
    <line3/>
    <line4>United States</line4>
    <house>1600</house>
    <street>Pennsylvania Ave NW</street>
    <xstreet/>
    <unittype/>
    <unit/>
    <postal>20006</postal>
    <neighborhood/>
    <city>Washington</city>
    <county>District of Columbia</county>
    <state>District of Columbia</state>
    <country>United States</country>
    <countrycode>US</countrycode>
    <statecode>DC</statecode>
    <countycode>DC</countycode>
    <uzip>20006</uzip>
    <hash>B42121631CCA2B89</hash>
    <woeid>12765843</woeid>
    <woetype>11</woetype>
  </Result>
</ResultSet>
```

Figura 4.7: Respuesta XML del servicio Web Yahoo! PlaceFinder

4.2.3.3 OpenStreetMap

OpenStreetMap, conocido también como OSM, es un proyecto colaborativo para crear mapas del mundo libres y editables. Los mapas se crean utilizando GPS para capturar información geográfica, a través de ortofotografías y a través de otras fuentes libres. En la Figura ?? se puede apreciar un ejemplo de mapa obtenido con OpenStreetMap.

Se utilizará para mostrar mapas y poder localizar posiciones geográficas en ellos, usando el servicio Web comentado en el apartado ??.

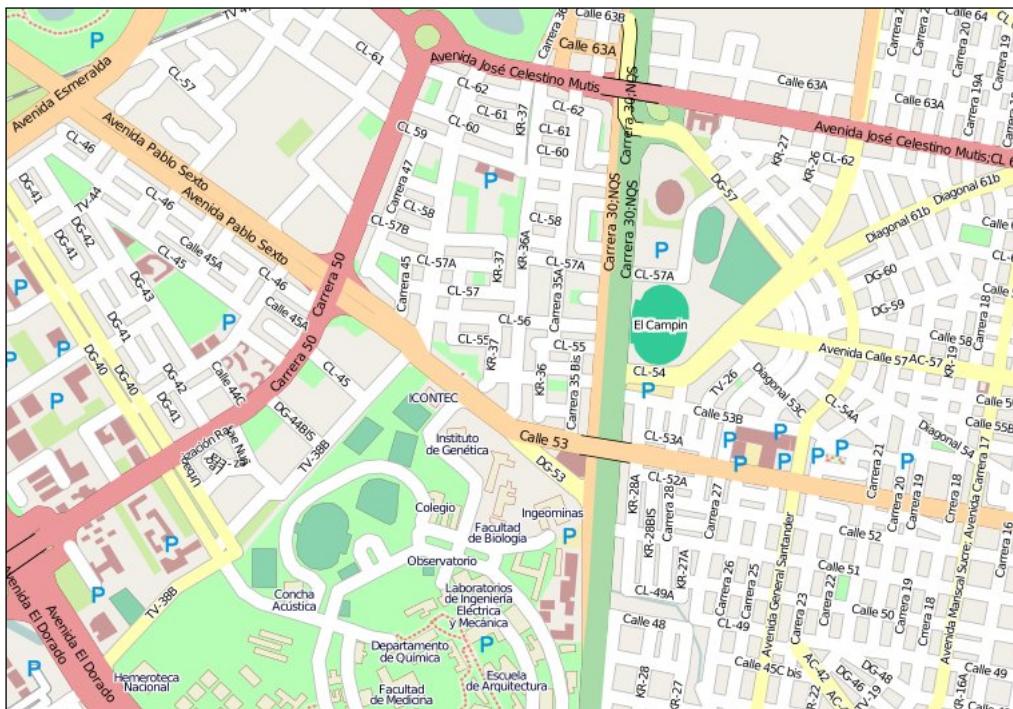


Figura 4.8: Ejemplo de mapa de OpenStreetMap

4.2.3.4 JAXB

Java Architecture for XML Binding es una tecnología de Java que permite la transformación de una jerarquía de objetos Java a un fichero XML, y viceversa. El proceso de serialización recibe el nombre de *marshalling*, mientras que el proceso inverso recibe el nombre de *unmarshalling* (ver Figura ??).

Se utilizará, en su versión 2.2.4, para poder convertir objetos del dominio de aplicación en ficheros XML, y viceversa.

4.2.3.5 JDOM

JDOM es una librería de Java de código abierto para el acceso y manipulación de ficheros XML optimizada para Java. Se creó para ser específicamente utilizado con Java, por lo que incluye sobrecarga de métodos, colecciones, etc, a diferencia del DOM creado por el

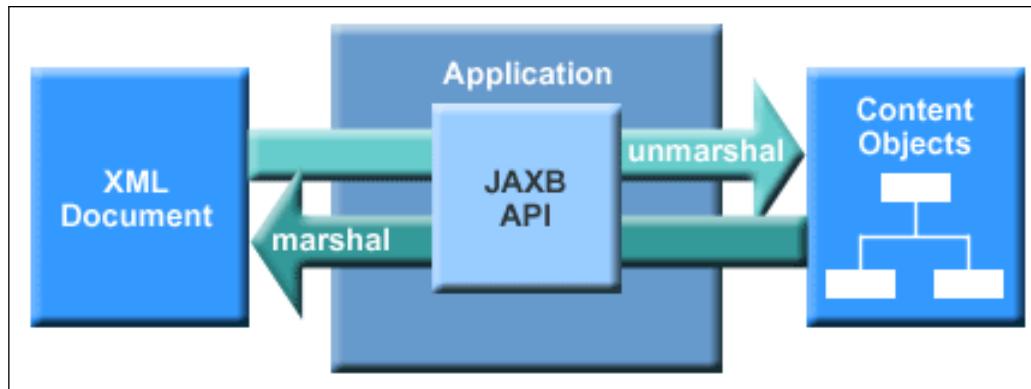


Figura 4.9: JAXB

consorcio W3C (World Wide Web), creado inicialmente para manipulación de páginas HTML con JavaScript. Señalar que JDOM no es un acrónimo de *Java Document Object Model*.

Dicha librería, en su versión 1.1, se utilizará para gestionar ficheros XML presentes en la aplicación, como, por ejemplo, los ficheros de perfiles.

4.2.3.6 Jaxen

Jaxen es una librería de Java de código abierto escrita en Java, adaptable a diferentes modelos de objetos como DOM, XOM, dom4j y JDOM, y que permite trabajar con expresiones *XPath* (XML Path Language).

Se utilizará en su versión 1.1.1 para la búsqueda de nodos en ficheros XML utilizando expresiones *XPath*.

4.2.3.7 Hibernate

Hibernate es un framework para el mapeo objeto-relacional (ORM, por sus siglas en inglés Object Relational-Mapping) utilizado en plataformas Java y que permite realizar el mapeo de atributos entre una base de datos relacional y un modelo de objetos, utilizando archivos XML o anotaciones sobre los objetos para realizar dicho mapeo. Proporciona también un lenguaje de consultas de bases de datos de alto nivel, llamado HQL (Hibernate Query Language).

Las ventajas de utilizar este framework es que permite al desarrollador abstraerse del código que hay que escribir para dar soporte a la persistencia, facilita la mantenibilidad,

proporciona independencia del proveedor del sistema gestor de base de datos y flexibilidad para adaptarse al esquema de tablas utilizado.

Se utilizará Hibernate, en su versión 3.5.4, para gestionar la persistencia de los objetos de dominio que deban ser persistentes en el sistema.

4.2.3.8 Swing

Swing es una librería gráfica para Java, la cual incluye *widgets* para generar interfaces gráficas de usuario, como son cajas de texto, botones, paneles, tablas, etc. Swing fue desarrollado para proveer un conjunto de elementos gráficos más sofisticado que su antecesor, AWT (Abstract Window Toolkit), de tal modo que los componentes Swing están escritos en Java y son independientes de la plataforma, a diferencia de AWT.

Swing sigue el modelo vista controlador (MVC), es independiente de la plataforma y extensible, ya que los componentes puede extenderse y sobrescribir las implementaciones por defecto.

Existe una extensión a Swing, llamada SwingX, que incluye nuevos componentes gráficos que normalmente se demandan en aplicaciones avanzadas (Rich Client Applications), como son selectores de fechas, framework para la autenticación de usuarios, auto-completado, paneles expandibles, etc.

Por otra parte, existe un framework para Swing, llamado Swing Application Framework, que forma parte de la JSR 296 (Java Specification Request) y que permite gestionar la arquitectura, ciclo de vida, hilos, manejo de eventos y almacenamiento del estado de una aplicación Swing.

Estos tres componentes, Swing, SwingX y Swing Application Framework se utilizarán para el desarrollo de una interfaz gráfica de usuario extensible, personalizable y adaptable. Además, para realizar el diseño de la interfaz, se utilizará el plugin para eclipse **Jigloo**.

4.2.3.9 JUNG

Jung (*Java Universal Network/Graph Framework*) es una librería que provee mecanismos para el modelado, análisis y visualización de datos que pueden ser representados en una red o en un grafo. Su arquitectura está diseñada para soportar variedad de representaciones

de entidades y sus relaciones, como grafos dirigidos, no dirigidos, hipergrafos, grafos con arcos paralelos, entidades con metadatos, etc. Además, incluye numerosos algoritmos para la visualización de grafos, para su optimización, cálculo de distancias, métricas, etc. En la Figura ?? se puede observar un grafo generado con este framework.

Se utilizará, en su versión 2.0, para la creación y visualización de grafos.

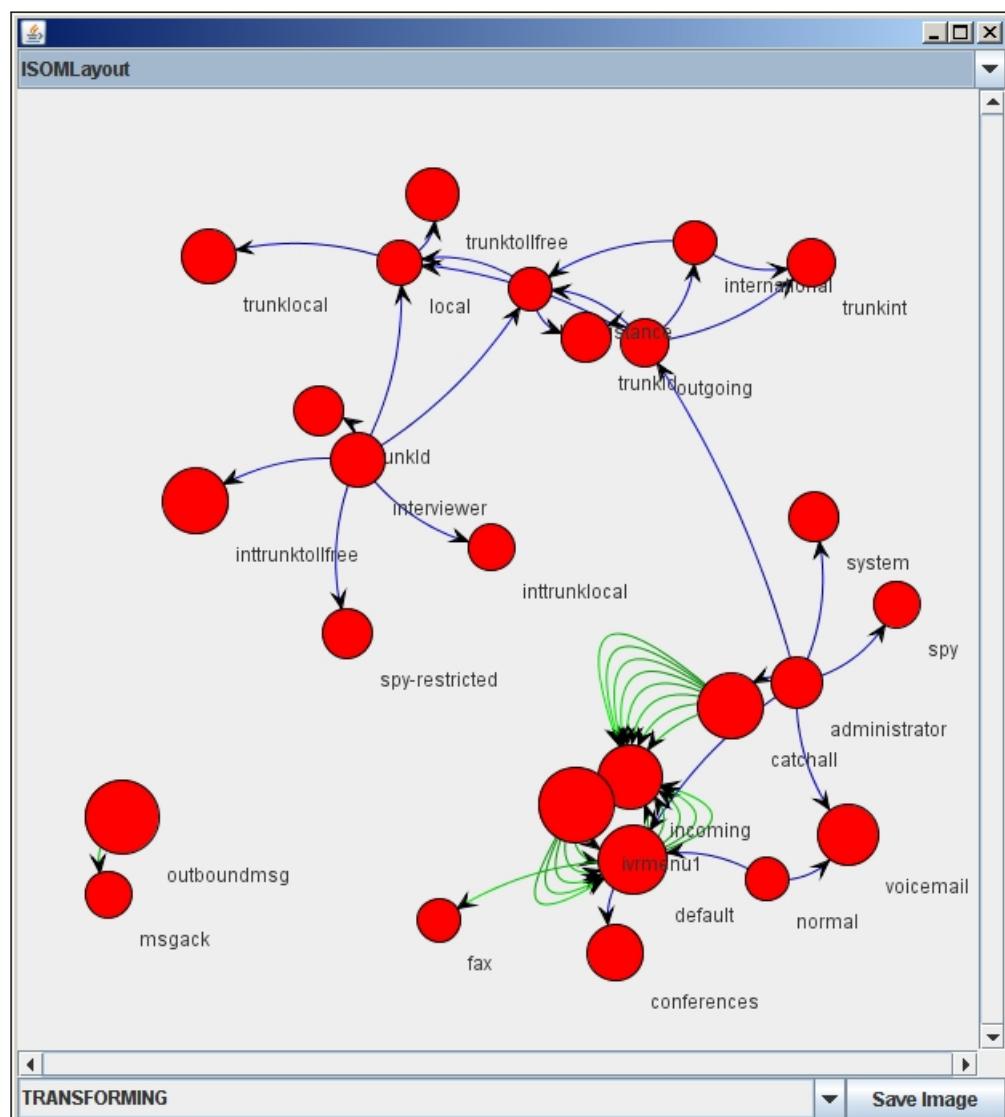


Figura 4.10: Ejemplo de grafo generado con JUNG

4.2.3.10 iText

iText es una librería que permite la creación y manipulación de documentos PDF. Dicha herramienta permite:

- Servir documentos PDF a un navegador.
- Generar documentos PDF a través de archivos XML o bases de datos.
- Añadir marcadores, número de páginas, marcas de agua, etc.
- Dividir, concatenar y gestionar páginas PDF

Se utilizará, en su versión 5.1.0 para Java (disponible también para .NET) para la generación de documentos PDF.

4.2.3.11 JFreeChart

JFreeChart es una librería escrita en Java que permite la creación y visualización de gráficos. En la Figura ?? se puede observar ejemplos de gráficos generados con esta librería. Las características de esta librería son:

- Proporciona un API para generar una amplia variedad de gráficos.
- Diseño flexible y extensible.
- Soporte para diferentes tipos de salida, como componentes de Swing, imágenes (PNG y JPEG) y gráficos vectoriales (EPS y SVG)
- Es *OpenSource*

Se ha utilizado en su versión 1.0.13 para la generación y visualización de gráficos.

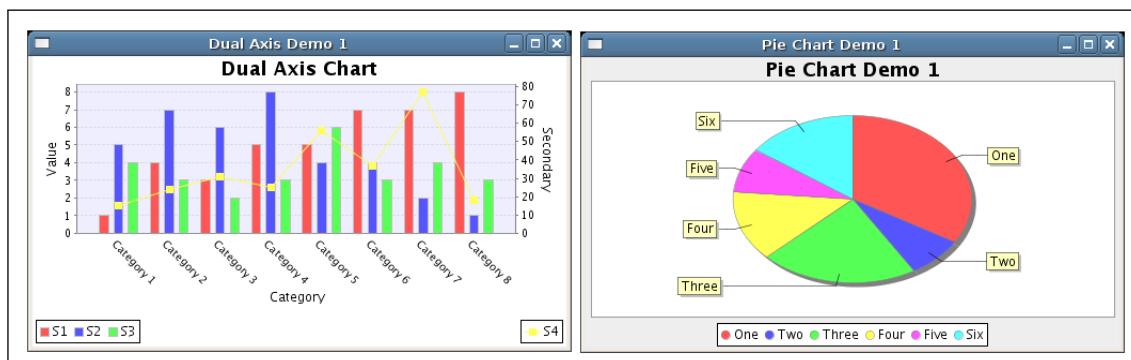


Figura 4.11: Ejemplo de gráficos generados con JFreeChart

4.2.3.12 RMI

RMI (*Remote Method Invocation*) es un mecanismo ofrecido por Java que permite la creación de aplicaciones Java distribuidas, en las cuales los métodos de objetos remotos pueden ser invocados desde otras máquinas virtuales de Java, posiblemente distribuidas en diferentes máquinas. RMI hace uso de la serialización de objetos para poder invocar sus métodos remotamente. Se caracteriza por su facilidad de uso, al estar integrado con Java y permite el paso de objetos por referencia, recolección de basura distribuida y paso de tipos arbitrarios.

Normalmente, una aplicación RMI se compone de un servidor, que crea objetos remotos y crea sus referencias para hacerlos accesibles remotamente, y un cliente, que invoca métodos de esos objetos remotos, obteniendo las referencias que el servidor facilita. La invocación remota de métodos se compone de los siguientes pasos:

- Se encapsulan los parámetros, utilizando la serialización de Java.
- Se invoca el método, quedando el invocador o cliente a la espera.
- Se ejecuta el método remoto y el servidor devuelve la respuesta al invocador, si la hay.
- El cliente recibe la respuesta y continua su ejecución, como se hubiese utilizado el método de un objeto local.

RMI sigue una arquitectura compuesta de cuatro capas:

1. **Capa de aplicación:** es la capa de implementación de las aplicaciones cliente y servidor, donde se crean y exportan los objetos remotos, haciendo uso de la interfaz *Remote*.
2. **Capa de proxy:** también llamada *capa stub-skeleton*. Es la capa que interactúa con la anterior y donde se producen las llamadas a métodos remotos.
3. **Capa de referencia remota:** responsable de manejar las llamadas remotas, mantener las referencias a los objetos exportados, etc.
4. **Capa de transporte:** responsable del transporte de datos de una máquina a otra, utilizando el protocolo JRMP (Java Remote Method Protocol).

RMI se utilizará para la comunicación e invocación de métodos entre la aplicación cliente y el servidor del sistema.

4.2.3.13 Java Reflection API

La reflexión es una técnica utilizada por los programas para examinar o modificar su estructura (introspección) y comportamiento en tiempo de ejecución.

La reflexión se utiliza cuando la clase de un objeto a crear se conoce en tiempo de ejecución, para enumerar los miembros de una clase, para depurar programas, conociendo los miembros privados de las clases, etc. Sin embargo, no se debe abusar de esta técnica, ya que introduce problemas de rendimiento, al resolver dinámicamente los tipos de objetos, y problemas de seguridad, al exponer los miembros privados de una clase.

Se utilizará para crear algunos objetos en tiempo de ejecución, ya que sólo en ese punto se conocerá su tipo.

4.2.4 Herramientas y tecnologías para bases de datos

4.2.4.1 MySQL

MySQL es un sistema de gestión de bases de datos (SGBD) relacional, multiusuario y multihilo, de software libre. MySQL está escrito en C y C++ y el *parser* de SQL está escrito en el lenguaje yacc. Es una herramienta multiplataforma y con soporte para diferentes lenguajes de programación utilizando los adaptadores adecuados para cada lenguaje.

Entre sus características, destacan las siguientes:

- Soporte para el lenguaje ANSI SQL 99, así como para sus extensiones.
- Soporte multiplataforma, multiusuario y multihilo.
- Procedimientos almacenados.
- Triggers.
- Cursos.
- Vistas actualizables.
- Sistema de procesamiento de transacciones distribuidas, con dos fases de *commit*.

- Soporte de conexiones seguras (SSL).
- Soporte de subconsultas.
- Soporte unicode.
- Soporte para replicación de las bases de datos.

Se utilizará la versión MySQL Community Server 5.5.17 como sistema gestor de bases de datos del sistema.

4.2.4.2 MySQL WorkBench

MySQL Workbench es una herramienta CASE que permite el modelado de bases de datos MySQL en un entorno visual (ver Figura ??). Sus características principales son:

- **Modelado:** proporciona las herramientas necesarias para el modelado Entidad-Interrelación (modelo ER) de la base de datos: además, soporta ingeniería directa, obteniendo el código SQL a partir del modelo, e ingeniería inversa.
- **Desarrollo:** proporciona herramientas visuales para crear, ejecutar y optimizar sentencias SQL.
- **Administración:** proporciona una consola visual para administrar fácilmente el servidor MySQL, cuentas de usuario, permisos, etc.

Se utilizará MySQL Workbench 5.2 CE para el modelado de la base de datos, para el desarrollo SQL y para administrar la base de datos MySQL.

4.2.5 Herramientas para la documentación del proyecto

4.2.5.1 L^AT_EX

L^AT_EX es un lenguaje de marcado utilizado para la creación de documentos, especialmente libros y documentos científico-técnicos. Está formado por un gran conjunto de macros (u órdenes) del lenguaje TeX, de código abierto y que permite crear libros, tesis y artículos técnicos con una elevada calidad tipográfica, comparable a la de una editorial científica.

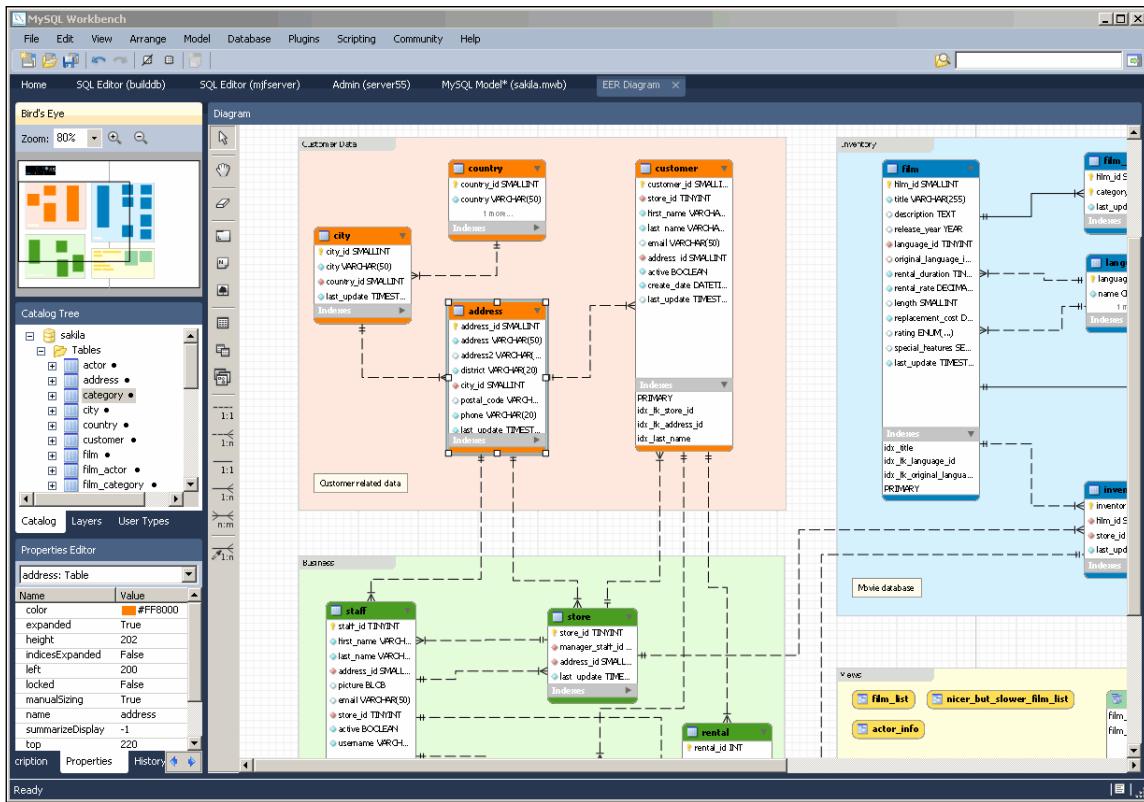


Figura 4.12: MySQL Workbench

A diferencia de los editores de texto habituales, conocidos como *WYSIWYG* (es decir, *What You See Is What You Get*, o *lo que ves es lo que obtienes*), L^AT_EX permite centrarse exclusivamente en el contenido del documento, sin preocuparse de los detalles del formato del texto. También posee capacidades gráficas para representar fórmulas complejas, ecuaciones, notación científica, etc y permite estructurar de una manera muy sencilla el documento, con capítulos, secciones, generación automática de índices, etc. Es por estas razones por las que L^AT_EX se ha extendido rápidamente por todo el sector científico y técnico, llegando a convertirse incluso en un uso obligado para presentar publicaciones en ciertos congresos y revistas.

Para crear un documento con L^AT_EX, hay que utilizar un editor de texto para crear los archivos fuente, con las macros y el contenido adecuado para, posteriormente, procesarlo, compilarlo y generar la salida, normalmente en PDF, aunque se pueden generar diferentes salidas como DVI, PS, etc.

Para la generación del presente documento, se ha utilizado L^AT_EX, usando como editor de textos **TexnicCenter**, el cual es un editor de software libre para sistemas Windows que

proporciona las herramientas necesarias para la composición y compilación de textos escritos en L^AT_EX (ver Figura ??).

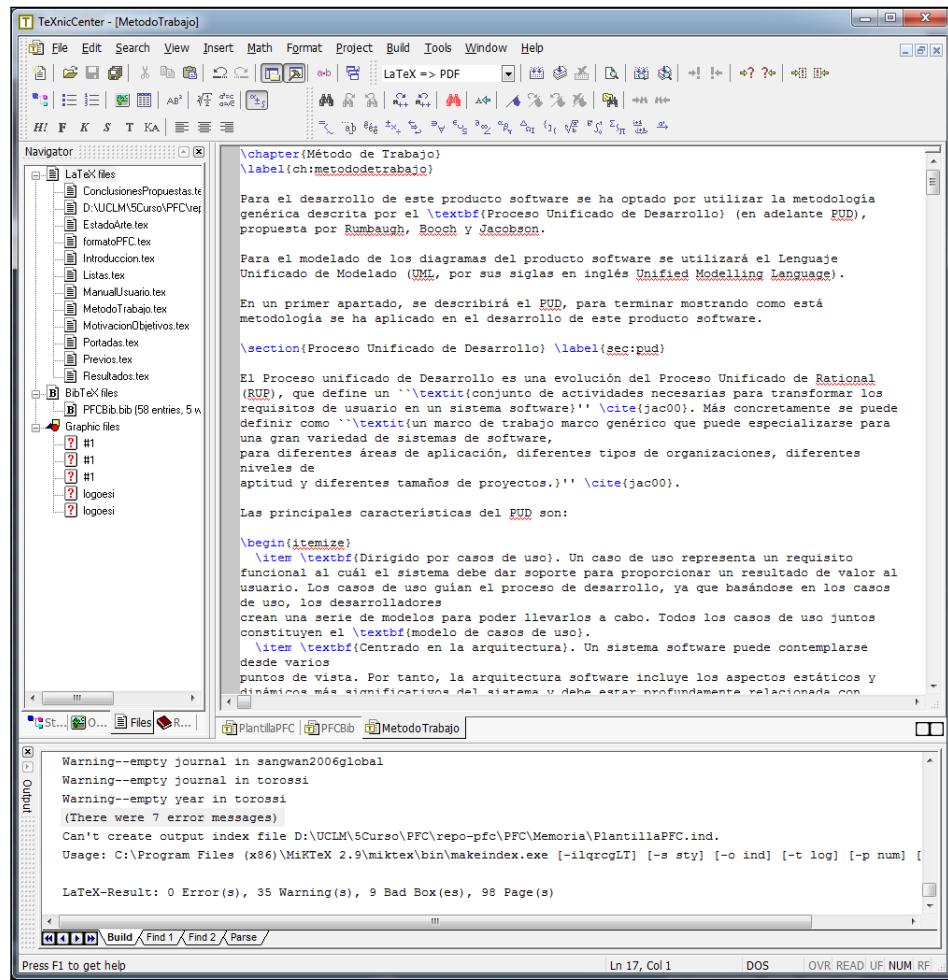


Figura 4.13: Editor TeXnicCenter

4.2.5.2 BIB^AT_EX

BIB^AT_EX es un software de gestión de referencias para dar formato a dichas referencias, usado normalmente en conjunto con L^AT_EX. Facilita la cita de fuentes con un formato consistente, separando la información bibliográfica de la presentación de la misma.

BIB^AT_EX utiliza un fichero con extensión .bib para definir la lista de elementos bibliográficos, que pueden ser artículos, tesis, libros, manuales, parte de libros o artículos, etc. Todos estos tipos de bibliografía deben definirse con una sintaxis concreta, indicando elementos como autor, título, editorial, fecha de publicación, URL, etc.

Se ha utilizado BibTeX para formatear las citas literarias y bibliográficas de este documento. Además, se ha utilizado **JabRef**, que es una herramienta visual basada en Java que permite editar los tipos bibliográficos utilizados en el archivo *.bib* de BibTeX (ver Figura ??).

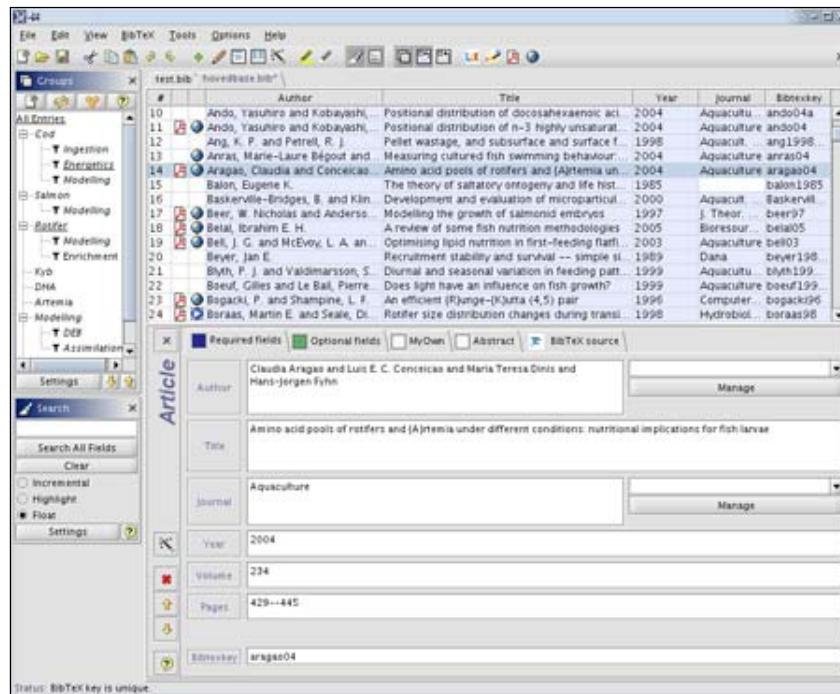


Figura 4.14: Herramienta JabRef

Capítulo 5

Resultados

En este capítulo se expondrán los resultados obtenidos al aplicar la metodología descrita en el capítulo ?? para el desarrollo del presente PFC.

Se muestra todo el proceso de desarrollo real que se ha seguido hasta la obtención del sistema, desde su fase de inicio hasta su finalización, mostrando su evolución de manera incremental y la evolución de requisitos funcionales que ha sufrido.

Siguiendo la metodología propuesta por el Proceso Unificado de Desarrollo, el ciclo de vida de desarrollo del sistema se ha dividido en las cuatro fases que el PUD propone: Inicio, Elaboración, Construcción y Transición. Cada una de esas fases, a su vez, se ha dividido en una o más iteraciones, donde intervienen las disciplinas de Captura de Requisitos, Análisis, Diseño, Implementación y Pruebas. De este modo, el sistema va incrementando su funcionalidad al terminar cada una de estas iteraciones.

En las sucesivas secciones se detallan cada una de las fases del PUD aplicadas al desarrollo del sistema, las iteraciones que abarca y se expondrá la evolución que ha ido sufriendo el proyecto hasta obtener el producto final.

5.1 Fase de Inicio

La fase de inicio abarca una única iteración, en la cuál, como se indica en esta fase del PUD (ver sección ??), se llevarán a cabo las siguientes tareas:

- Identificación de requisitos funcionales y no funcionales del sistema a desarrollar.
- Elaboración de un primer modelo de casos de uso.
- Análisis de viabilidad y riesgos.
- Elaboración del plan de iteraciones (o plan de proyecto) para el resto de fases.

En esta fase, como se muestra en la Figura 1.1, interviene principalmente la disciplina de Captura de Requisitos.

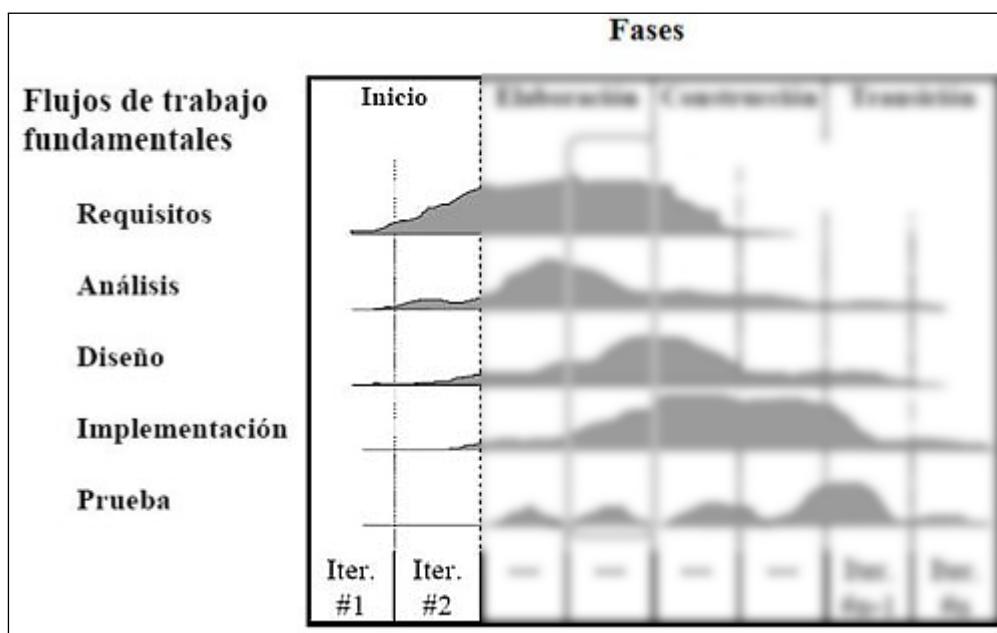


Figura 5.1: Fase de inicio en el PUD

En los siguientes apartados se exponen las tareas realizadas en esta fase.

5.1.1 Identificación de requisitos

Para la identificación de requisitos del sistema, se estudió la literatura existente relacionada con el tema y se fueron extrayendo requisitos funcionales que pudieran resolver o mitigar los

desafíos encontrados en la literatura, desarrollados en el capítulo ??.

Por otra parte, para que el sistema obtenido como resultado del desarrollo de este proyecto pudiera tener una aplicación real, era necesario conocer el funcionamiento de las empresas que siguen el paradigma de GSD y qué información es relevante para los miembros de los equipos de desarrollo. La información que interesaba conocer era cómo se distribuyen los proyectos, los tipos de roles que participan en el desarrollo, experiencia de los desarrolladores, cómo se distribuyen los equipos en los países, cómo se comunican e intercambian información, etc.

Para poder conocer esta información acerca del dominio de aplicación del sistema, se recurrió a un jefe de proyecto de INDRA Software Labs S.L. Además, también se ha utilizado la propia experiencia del autor y desarrollador del presente PFC en una empresa que cuenta con centros de desarrollo distribuidos en diferentes países.

Una vez estudiada la literatura, conocidos los problemas del GSD y con conocimiento del dominio de la aplicación del sistema, se pasó a elaborar una serie de requisitos funcionales que el sistema debe cumplir para alcanzar los objetivos propuestos en el Capítulo ??.

5.1.1.1 Requisitos funcionales

A continuación, se enumera el conjunto de requisitos funcionales que el sistema debe cumplir.

- Se debe permitir que diferentes miembros de los equipos de desarrollo de una compañía puedan acceder al sistema, mediante un nombre de usuario y contraseña, proporcionados por la empresa.
- Como en los equipos de desarrollo existen diferentes roles entre sus miembros, el sistema debe adaptarse a las acciones que cada miembro del equipo (en adelante, usuario) puede realizar, según su rol.
- Se deben poder dar de alta nuevos proyectos en el sistema, así como modificar los existentes, permitiendo seleccionar los usuarios que en dichos proyectos participarán.
- Los miembros del equipo de desarrollo normalmente trabajan en varios proyectos, por lo que el sistema deberá permitir escoger el proyecto en el cuál se van a gestionar las decisiones.

- Los usuarios pueden participar activamente en las decisiones que se van tomando durante el ciclo de vida de los proyectos en los que participan, de tal modo que la herramienta debe permitir gestionar estas decisiones. Así, se podrán crear, modificar y eliminar dichas decisiones, agrupadas en tres categorías: *Temas, Propuestas y Respuestas*.
- La herramienta debe mostrar, de una manera gráfica e intuitiva, el conjunto de decisiones que han sido tomadas en un proyecto software. Además, se deberá proporcionar diferente información acerca de esas decisiones: su autor, centro de desarrollo y compañía donde pertenece, detalles de esa decisión, etc.
- Cuando se presenta un nuevo proyecto, se debe poder aconsejar decisiones tomadas en proyectos ya terminados, que tuviesen características similares.
- La herramienta debe proveer mecanismos de notificación síncrona y asíncrona que faciliten la comunicación de las últimas modificaciones:
 - Se debe crear un sistema de notificaciones o alertas, informando al usuario cuando se producen cambios sobre las decisiones de los proyectos donde trabaja.
 - Se deben mostrar cambios en tiempo real, es decir, si un empleado se encuentra visualizando el conocimiento en la aplicación y, en ese momento, otro empleado de otra localización diferente realiza un cambio sobre el mismo conocimiento, éste debe ser notificado al primer empleado en tiempo real, actualizando su vista.
- La herramienta debe poder configurarse y mostrarse en diferentes idiomas.
- La herramienta debe generar *logs* con las acciones que se van realizando en el sistema.
- Se debe poder exportar todas las decisiones realizadas en un proyecto a un fichero XML, así como toda la información asociada a dichas decisiones.

Una vez identificada esta lista de funciones a las que el sistema debe dar soporte, éstas se agruparon en diferentes categorías de requisitos, como se muestra en la tabla 1.1.

Además, se identificaron los roles de usuario mostrados en la Tabla 1.2, los cuales pueden realizar las acciones mostradas en la Tabla 1.3.

Requisito	Tareas asociadas
Acceso Sistema	Login (acceso) Logout (desconexión)
Visualizar información	Consultar decisiones Visualizar compañía
Gestión Decisiones	Crear nueva decisión Modificar decisión Eliminar decisión
Gestión Notificaciones	Consultar notificaciones Modificar notificación Eliminar notificación
Gestión Proyectos	Dar de alta un proyecto Modificar datos proyecto Consultar usuarios Seleccionar proyecto sobre el que trabajar Aconsejar decisiones de proyectos similares
Gestión Idiomas	Consultar idiomas Cambiar idioma
Exportar información	Exportar información

Tabla 5.1: Requisitos funcionales - v1.0

Rol	Descripción
Empleado	Es un usuario del sistema que representa a un miembro de un equipo de desarrollo de una compañía.
Jefe de Proyecto	Es un usuario del sistema que representa a un jefe de proyecto de un equipo de desarrollo de una compañía.

Tabla 5.2: Roles identificados en el sistema

Requisito	Empleado	Jefe Proyecto
Login	X	X
Logout	X	X
Crear decisión	X (excepto Temas)	X
Modificar decisión	X (excepto Temas)	X
Eliminar decisión	X (excepto Temas)	X
Consultar decisiones	X	X
Visualizar compañía	X	X
Consultar notificaciones	X	X
Modificar notificación	X	X
Eliminar notificación	X	X
Dar de alta proyecto		X
Modificar proyecto		X
Consultar usuarios	X	X
Seleccionar proyecto activo	X	X
Aconsejar decisiones de proyectos similares		X
Consultar idiomas	X	X
Cambiar idioma	X	X
Exportar información		X

Tabla 5.3: Acciones que un usuario puede realizar en el sistema - v1.0

5.1.1.2 Requisitos no funcionales

A continuación, se enumera el conjunto de requisitos no funcionales del sistema, dependientes del entorno tecnológico de la aplicación.

- El sistema debe ser distribuido, para facilitar su uso entre los diferentes miembros de los equipos de desarrollo que se encuentran geográficamente deslocalizados. Por tanto, es un sistema cliente-servidor, formado por dos subsistemas.
- Debe existir una base de datos en el servidor que permita almacenar todas las decisiones tomadas en los proyectos, la información de los proyectos, la información de usuarios, etc.

5.1.2 Modelo de casos de uso

Una vez identificados los requisitos funcionales del sistema, éstos se modelan en un diagrama de casos de uso con un alto nivel de abstracción, solamente mostrando los diferentes grupos funcionales del sistema, comentados en la Tabla 1.1.

Cabe destacar que el sistema está compuesto de dos subsistemas, cliente y servidor, como se ha comentado en los requisitos no funcionales (ver apartado 1.1.1.2), por lo que, aunque los requisitos funcionales se refieren al sistema global, se han modelado casos de uso para ambos subsistemas, ya que los actores que intervienen son diferentes.

5.1.2.1 Modelo de casos de uso para el subsistema cliente

En la Figura 1.2 se muestra el diagrama de casos para el subsistema cliente. Todos los casos de uso en este subsistema, tienen un mecanismo común de funcionamiento: se solicitan al usuario los datos que correspondan a la funcionalidad de ese caso de uso (datos del proyecto, datos de una decisión, etc.) y se envían dichos datos al servidor, esperando su respuesta y mostrando los resultados. Por esta razón, los actores que aparecen en el diagrama de casos de uso de este subsistema son los diferentes roles del sistema, y el servidor.

Para terminar, cabe señalar que se ha utilizado una jerarquía de herencia entre los diferentes tipos de usuarios, pues hay operaciones que pueden realizar varios de ellos. Por ejemplo, el empleado y el jefe de proyecto pueden ambos consultar las decisiones de un proyecto.

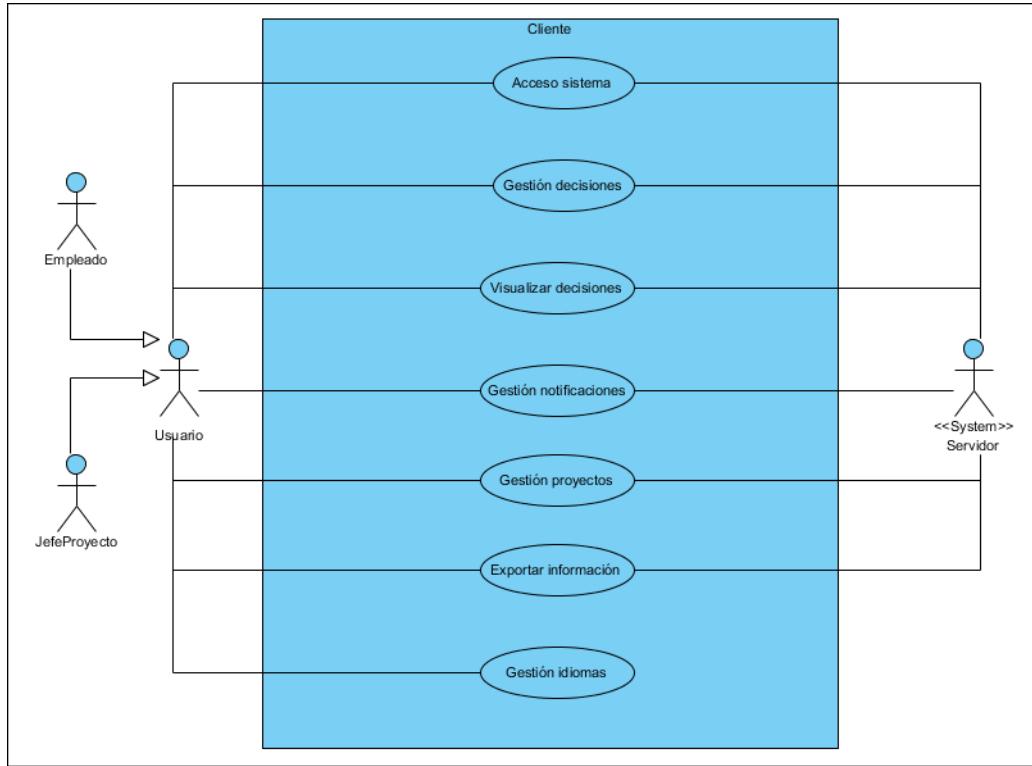


Figura 5.2: Diagrama de casos de uso - Cliente - v1.0

5.1.2.2 Modelo de casos de uso para el subsistema servidor

En la Figura 1.3 se puede observar el diagrama de casos de uso de alto nivel para el servidor. En este subsistema, el modo de proceder es el siguiente: se reciben las peticiones del subsistema cliente, se opera con la lógica de control y de dominio, accediendo a la base de datos y se envía la respuesta al cliente, notificando los posibles cambios. Por esta razón, los actores que aparecen en el diagrama de casos de uso de este subsistema son el subsistema cliente y el SGDB.

En cuanto a los casos de uso, aparecen los mismos que en el subsistema anterior, salvo dos casos de uso adicionales, descritos a continuación. Todos los casos de uso del servidor incluyen la funcionalidad de estos casos de uso, pero no se ha representado en el diagrama para facilitar la legibilidad.

- **Actualizar ventanas:** este caso de uso se utiliza para que el servidor pueda notificar cambios al cliente, producidos por otros clientes.
- **Actualizar log:** este caso de uso se utiliza para almacenar y reflejar en el servidor todas

las acciones realizadas por los clientes, creando *logs*.

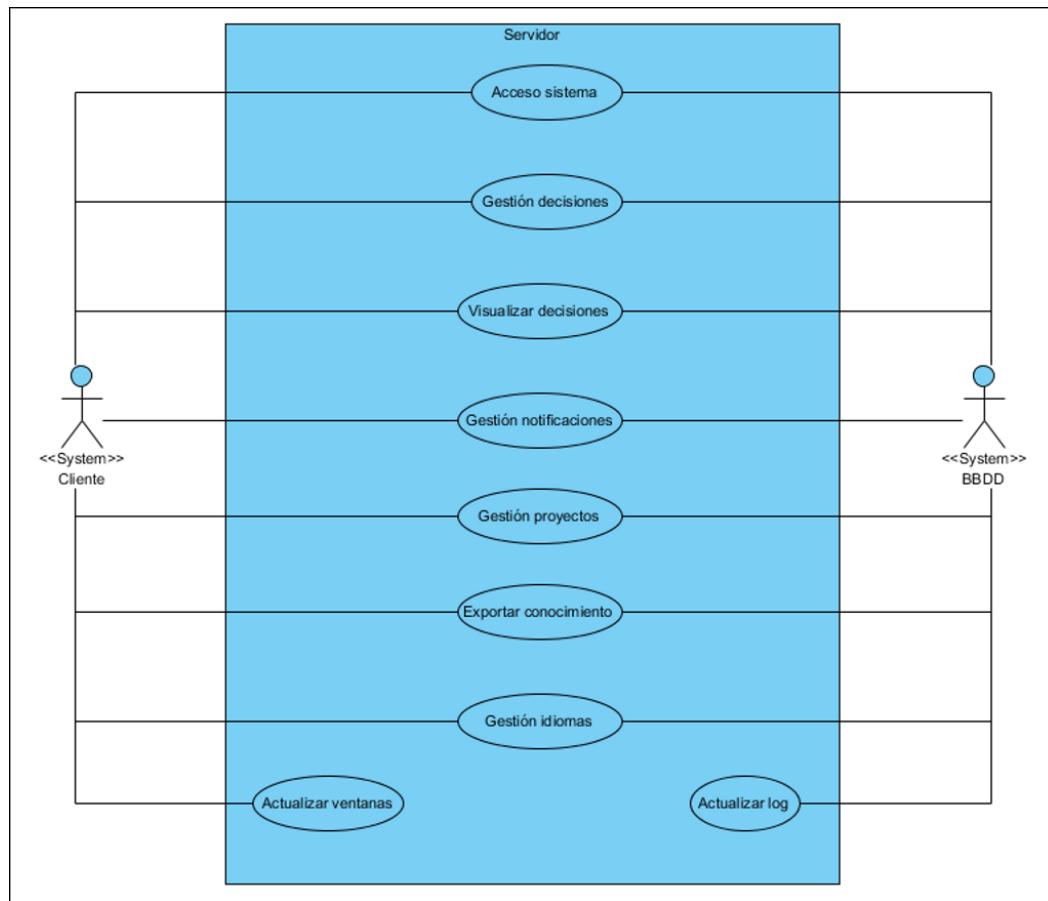


Figura 5.3: Diagrama de casos de uso - Servidor - v1.0

A continuación, se muestran los diagramas de casos de uso cada uno de los grandes grupos funcionales de manera más detallada para cada uno de los subsistemas.

5.1.2.2.1 Acceso al sistema

En la Figura 1.4 se muestra el diagrama de casos de uso para el cliente, mientras que en la Figura 1.5 se muestra el diagrama de casos de uso para el servidor.

5.1.2.2.2 Visualización de decisiones

En la Figura 1.6 se muestra el diagrama de casos de uso para el cliente, mientras que en la Figura 1.7 se muestra el diagrama de casos de uso para el servidor.

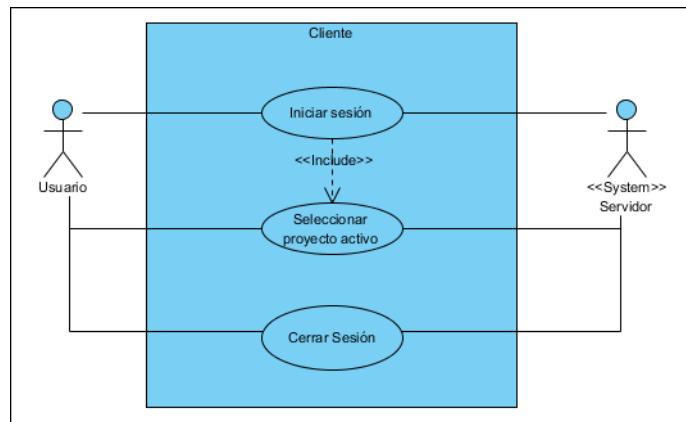


Figura 5.4: Diagrama de casos de uso - Cliente - Acceso al sistema - v1.0

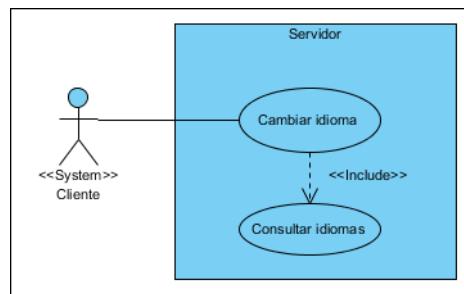


Figura 5.5: Diagrama de casos de uso - Servidor - Acceso al sistema - v1.0

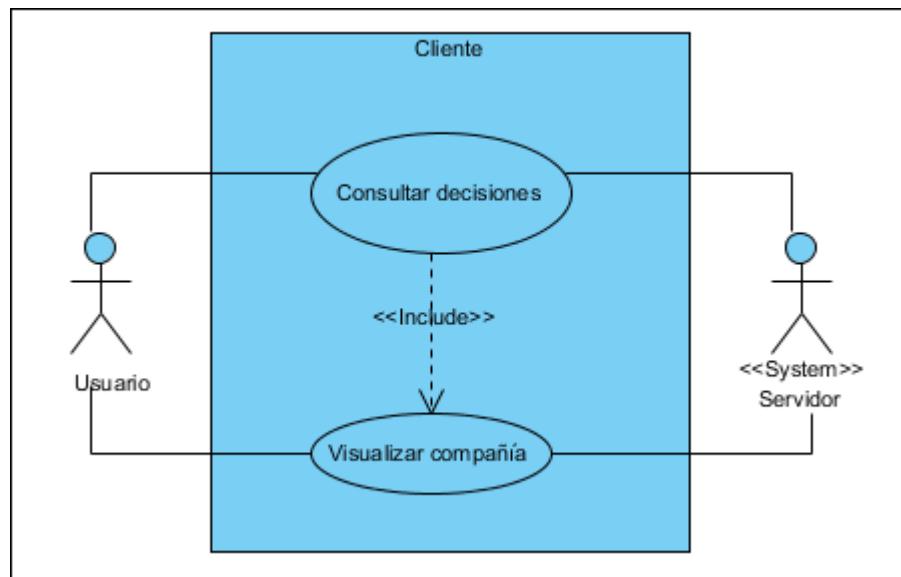


Figura 5.6: Diagrama de casos de uso - Cliente - Visualizar información - v1.0

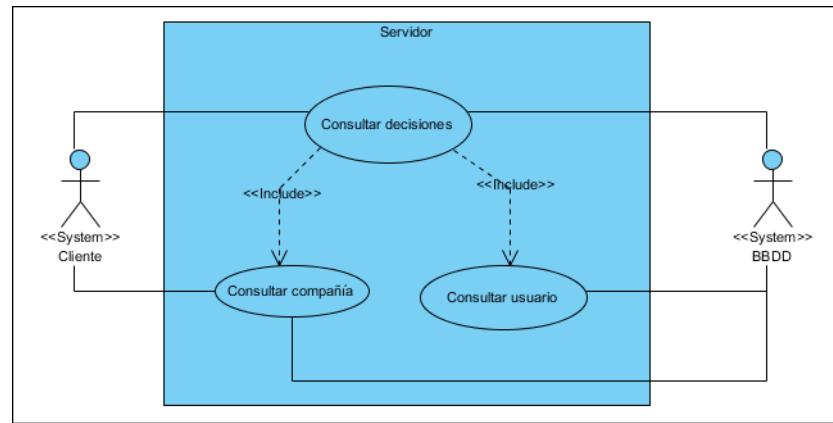


Figura 5.7: Diagrama de casos de uso - Servidor - Visualizar información - v1.0

5.1.2.2.3 Gestión de decisiones

En la Figura 1.8 se muestra el diagrama de casos de uso para el cliente, mientras que en la Figura 1.9 se muestra el diagrama de casos de uso para el servidor.

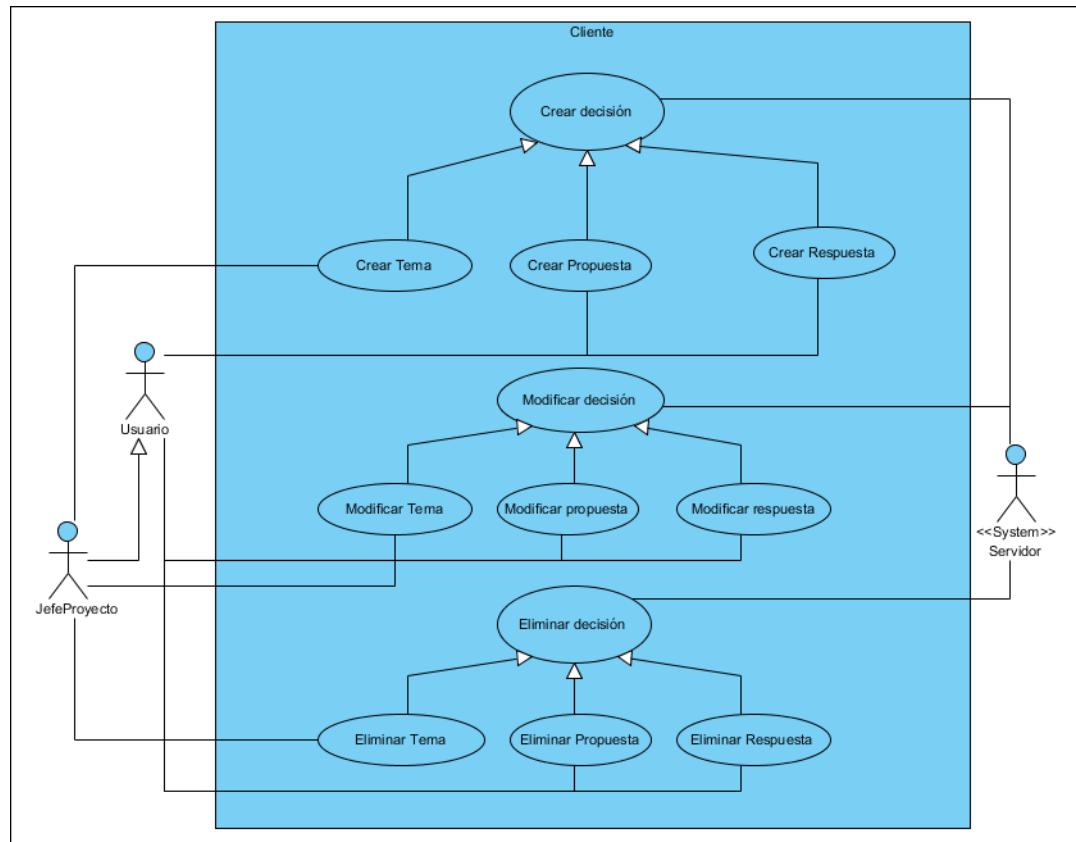


Figura 5.8: Diagrama de casos de uso - Cliente - Gestión decisiones - v1.0

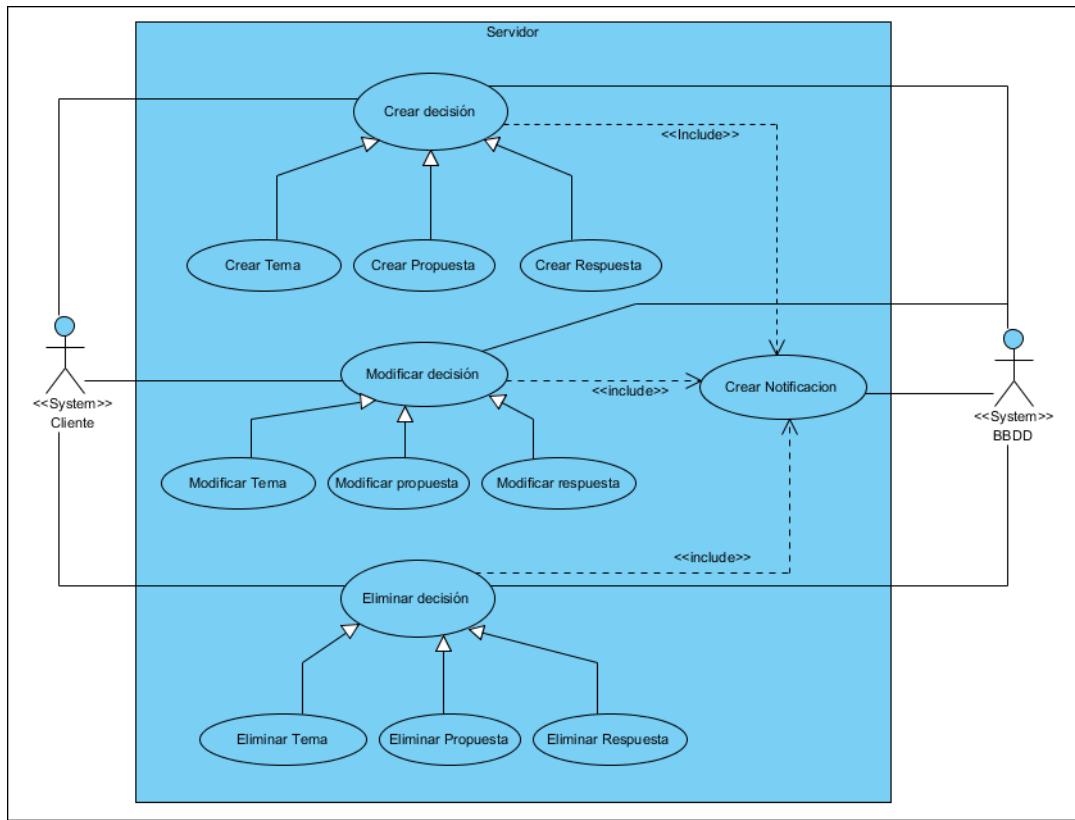


Figura 5.9: Diagrama de casos de uso - Servidor - Gestión decisiones - v1.0

5.1.2.2.4 Gestión de notificaciones

En la Figura 1.10 se muestra el diagrama de casos de uso para el cliente, mientras que en la Figura 1.11 se muestra el diagrama de casos de uso para el servidor.

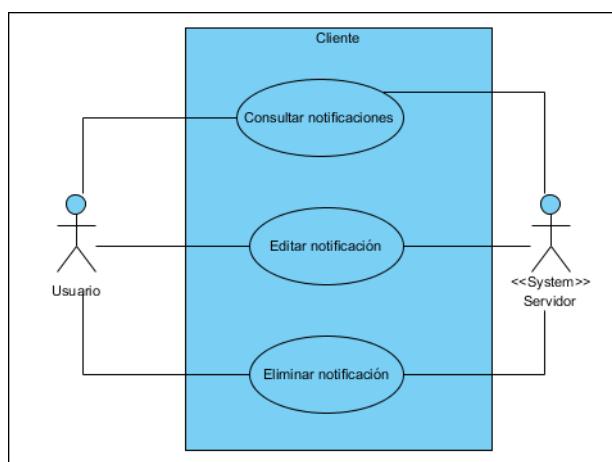


Figura 5.10: Diagrama de casos de uso - Cliente - Gestión notificaciones - v1.0

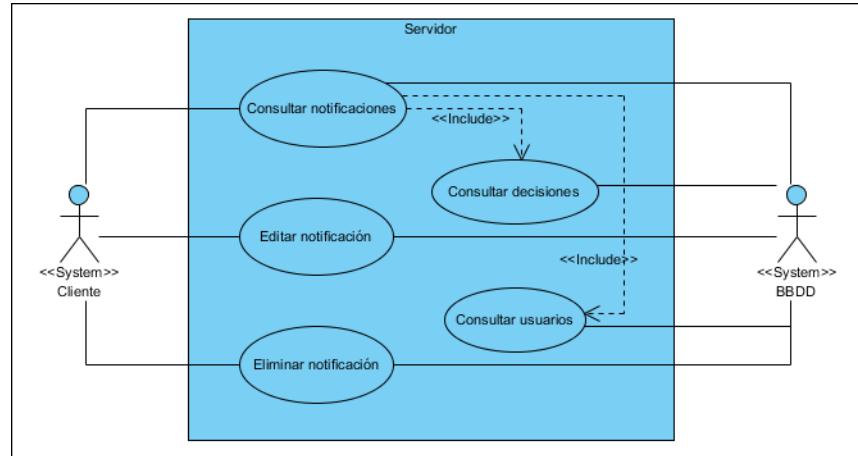


Figura 5.11: Diagrama de casos de uso - Servidor - Gestión notificaciones - v1.0

5.1.2.2.5 Gestión de proyectos

En la Figura 1.12 se muestra el diagrama de casos de uso para el cliente, mientras que en la Figura 1.13 se muestra el diagrama de casos de uso para el servidor.

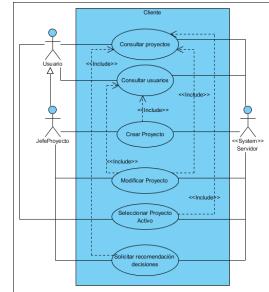


Figura 5.12: Diagrama de casos de uso - Cliente - Gestión Proyectos - v1.0

5.1.2.2.6 Gestión de idiomas

En la Figura 1.14 se muestra el diagrama de casos de uso para el cliente, mientras que en la Figura 1.15 se muestra el diagrama de casos de uso para el servidor.

5.1.2.2.7 Exportar información

En la Figura 1.16 se muestra el diagrama de casos de uso para el cliente, mientras que en la Figura 1.17 se muestra el diagrama de casos de uso para el servidor.

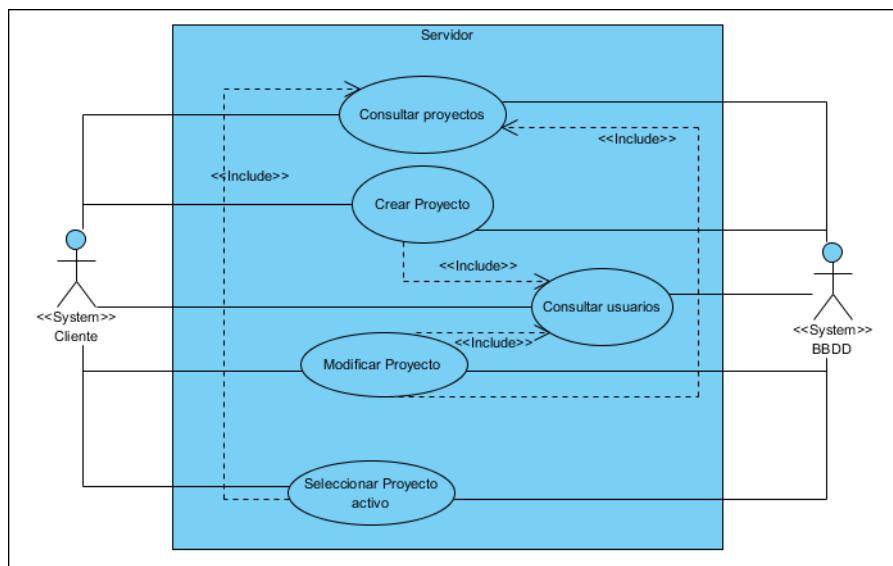


Figura 5.13: Diagrama de casos de uso - Servidor - Gestión Proyectos - v1.0

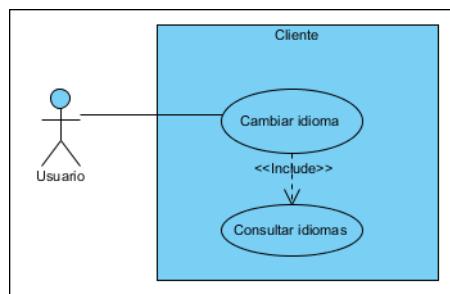


Figura 5.14: Diagrama de casos de uso - Cliente - Gestión Idiomas - v1.0

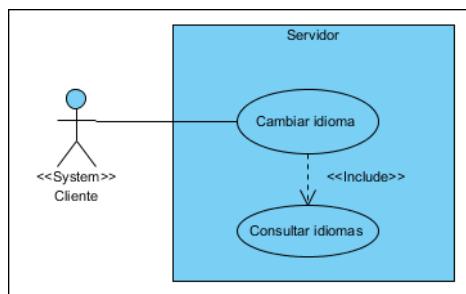


Figura 5.15: Diagrama de casos de uso - Servidor - Gestión Idiomas - v1.0

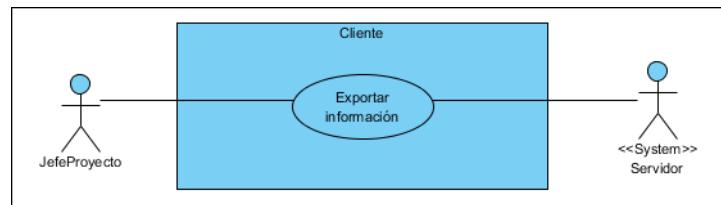


Figura 5.16: Diagrama de casos de uso - Cliente - Exportar información - v1.0

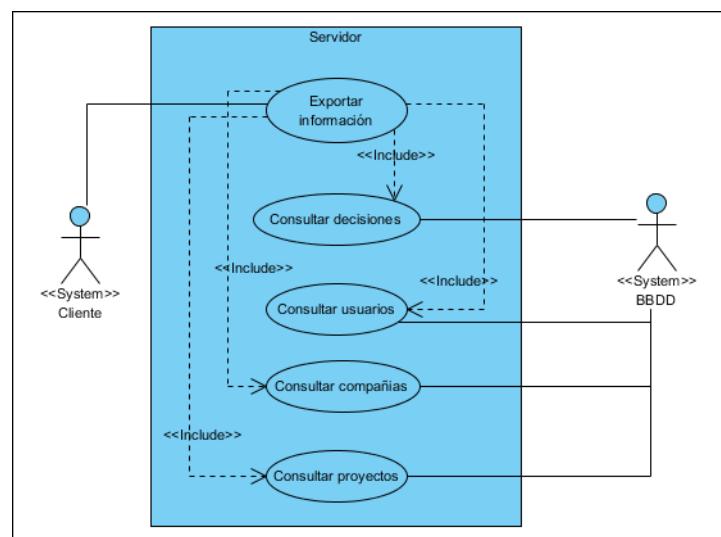


Figura 5.17: Diagrama de casos de uso - Servidor - Exportar información - v1.0

5.1.3 Análisis de riesgos

5.1.4 Plan de iteraciones

Dado que el PUD es centrado y dirigido por casos de uso, se utilizan los casos de uso modelados anteriormente para poder crear el plan de proyecto, o plan de iteraciones, y llevar a cabo el desarrollo de dichos casos de uso en las fases e iteraciones posteriores. De este modo, se obtienen las siguientes iteraciones, mostradas en la Tabla 1.4.

Fase	Iteración	Tareas a realizar
Inicio	1	Identificar requisitos Analizar la viabilidad del proyecto Crear modelo de casos de uso de alto nivel Plan de proyecto
Elaboración	1	Identificar requisitos Crear modelo de casos de uso Diseñar arquitectura Crear modelo de clases de análisis
	2	Diseñar caso de uso Acceso al sistema Implementar caso de uso Acceso al sistema Probar caso de uso Acceso al sistema
Construcción	1	Diseñar caso de uso Visualizar información Implementar caso de uso Visualizar información Probar caso de uso Visualizar información
	2	Diseñar casos de uso Gestión decisiones y Gestión notificaciones Implementar casos de uso Gestión decisiones y Gestión notificaciones Probar casos de uso Gestión decisiones y Gestión notificaciones

Fase	Iteración	Tareas a realizar
	3	Diseñar caso de uso Gestión proyectos Implementar caso de uso Gestión proyectos Probar caso de uso Gestión proyectos
	4	Diseñar casos de uso Gestión idiomas y Exportar información Implementar casos de uso Gestión idiomas y Exportar información Probar casos de uso Gestión idiomas y Exportar información
Transición	1	Preparar entregables Realizar documentación y manuales

Tabla 5.4: Plan de iteraciones

5.2 Fase de Elaboración

En esta fase intervienen la Captura de Requisitos, Análisis, Diseño, Implementación y Pruebas, con mayor o menor influencia, según la iteración donde nos encontremos (ver Figura 1.18).

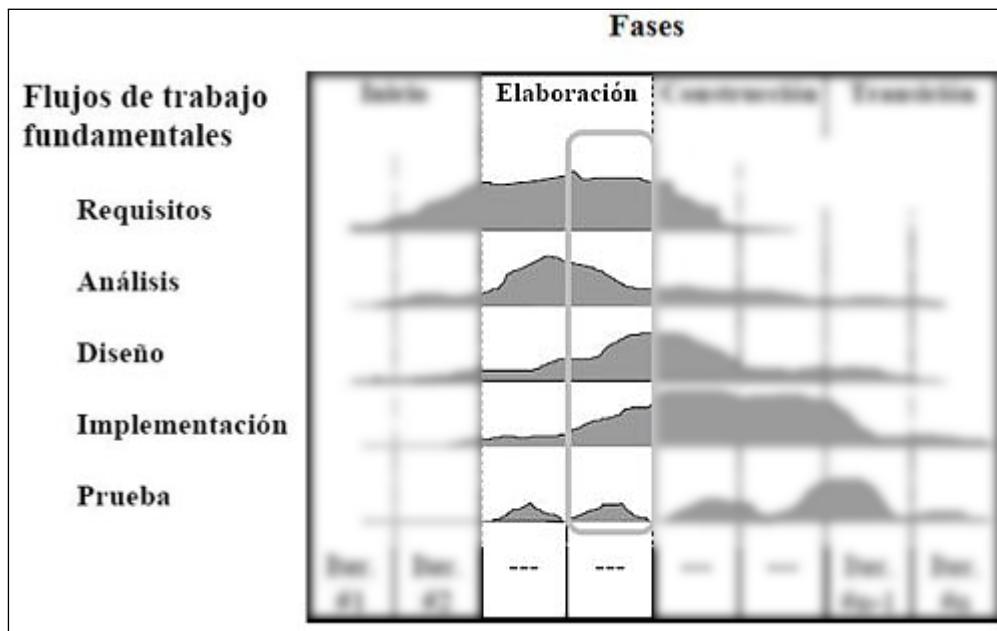


Figura 5.18: Fase de elaboración en el PUD

Esta fase se ha dividido en dos iteraciones: la primera de ellas se centra de nuevo en la captura de requisitos, en el análisis y en algunas tareas de diseño. La segunda iteración se centra en el diseño, la implementación y pruebas de un requisito funcional básico del sistema.

Las tareas que se llevan a cabo en esta fase son:

- **Primera iteración**

- Identificación de nuevos requisitos funcionales del sistema.
- Elaboración del modelo de casos de uso definitivo.
- Elaboración del plan de proyecto definitivo.
- Modelo de clases de análisis de los casos de uso.
- Definición y diseño de la arquitectura del sistema.

- **Segunda iteración**

- Modelo de alto nivel de clases de diseño del dominio.
- Modelado y diseño de la base de datos.
- Definición del diagrama de secuencia correspondiente a la funcionalidad de “acceso al sistema”.
- Implementación de la funcionalidad de “acceso al sistema”.
- Creación de casos de prueba para dicho caso de uso implementado.

5.2.1 Primera iteración

En esta iteración se abordan las siguientes tareas:

- Identificación de nuevos requisitos funcionales.
- Análisis de los casos de uso que componen el sistema.
- Definición y diseño de la arquitectura del sistema.

5.2.1.1 Identificación de requisitos

Al terminar la iteración de la fase anterior, obteniendo un primer modelo de casos de uso con los requisitos identificados, se llevó a cabo una reunión de seguimiento, para revisar la planificación realizada en el plan de proyecto y poder identificar nuevos requisitos funcionales, si los hubiera.

Como resultado de esta reunión, se identificaron nuevos requisitos funcionales para añadir al sistema, y que servirán para mitigar o resolver más problemas que aparecen en GSD. Estos nuevos requisitos fueron:

- Además de poder mostrar las decisiones tomadas en un proyecto, se debe poder cambiar el estado de éstas, aceptando o rechazando dichas decisiones, para posteriormente, cuando el proyecto llegue a su fin, poder conocer qué notificaciones fueron de ayuda y cuáles no.
- Para aumentar la información que se añade al crear nuevas decisiones de un proyecto, se debe poder adjuntar ficheros a dichas decisiones. Del mismo modo, se deben poder descargar esos archivos adjuntos, si los hubiera.

- Se deben poder generar de manera automática estadísticas que ayuden a llevar un control sobre los proyectos que se están desarrollando por los diferentes centros de desarrollo software de una compañía. También, se podrán generar estadísticas acerca de los miembros de los equipos.
- Se deben poder generar informes de las decisiones tomadas en un proyecto, en forma de tabla y de manera automática. Dicho informe debe generarse como un documento PDF.

Con estos nuevos requisitos, se actualizan los grupos funcionales creados en la fase de inicio, las acciones que pueden realizar los diferentes roles de usuarios y el modelo de casos de uso, explicado en el siguiente apartado.

En la Tabla 1.5 se muestran los nuevos grupos de requisitos, incluyendo los nuevos requisitos que se han identificado.

Por otra parte, en la Tabla 1.6 se muestran las acciones que se pueden realizar en el sistema según el rol del usuario.

5.2.1.2 **Modelo de casos de uso**

En la Figura 1.19 se muestra el diagrama de casos de usos global del cliente, incluyendo los nuevos requisitos. Del mismo modo, en la Figura 1.20 se muestra el diagrama de casos de uso actualizado del subsistema servidor.

A continuación, para cada uno de los grupos funcionales del sistema, se representan los diagramas de casos de uso detallados que han sufrido un cambio respecto a los casos de uso descritos en la fase de Inicio.

5.2.1.2.1 **Visualización de decisiones**

En la Figura 1.21 se muestra el diagrama de casos de uso para el cliente, mientras que en la Figura 1.22 se observa el diagrama de casos de uso para el servidor.

5.2.1.2.2 **Gestión de decisiones**

En la Figura 1.23 se muestra el diagrama de casos de uso para el cliente, mientras que en la Figura 1.24 se muestra el diagrama de casos de uso para el servidor.

Requisito	Tareas asociadas
Acceso Sistema	Login (acceso) Logout (desconexión)
Visualizar información	Consultar decisiones Visualizar compañía Descargar ficheros adjuntos
Gestión Decisiones	Crear nueva decisión Modificar decisión Eliminar decisión Adjuntar ficheros Aceptar o rechazar decisiones
Gestión Notificaciones	Consultar notificaciones Modificar notificación Eliminar notificación
Gestión Proyectos	Dar de alta un proyecto Modificar datos proyecto Consultar usuarios Seleccionar proyecto sobre el que trabajar Aconsejar decisiones de proyectos similares
Generación Estadísticas	Generación estadísticas
Generación Informes	Generación informes PDF
Gestión Idiomas	Consultar idiomas Cambiar idioma
Exportar información	Exportar información

Tabla 5.5: Requisitos funcionales - v2.0

Requisito	Empleado	Jefe Proyecto
Login	X	X
Logout	X	X
Crear decisión	X (excepto Temas)	X
Modificar decisión	X (excepto Temas)	X
Eliminar decisión	X (excepto Temas)	X
Adjuntar ficheros	X	X
Aceptar o rechazar decisiones		X
Consultar decisiones	X	X
Visualizar compañía	X	X
Consultar notificaciones	X	X
Modificar notificación	X	X
Eliminar notificación	X	X
Dar de alta proyecto		X
Modificar proyecto		X
Consultar usuarios	X	X
Seleccionar proyecto activo	X	X
Aconsejar decisiones de proyectos similares		X
Generación estadísticas		X
Generación informes		X
Consultar idiomas	X	X
Cambiar idioma	X	X
Exportar información		X

Tabla 5.6: Acciones que un usuario puede realizar en el sistema - v2.0

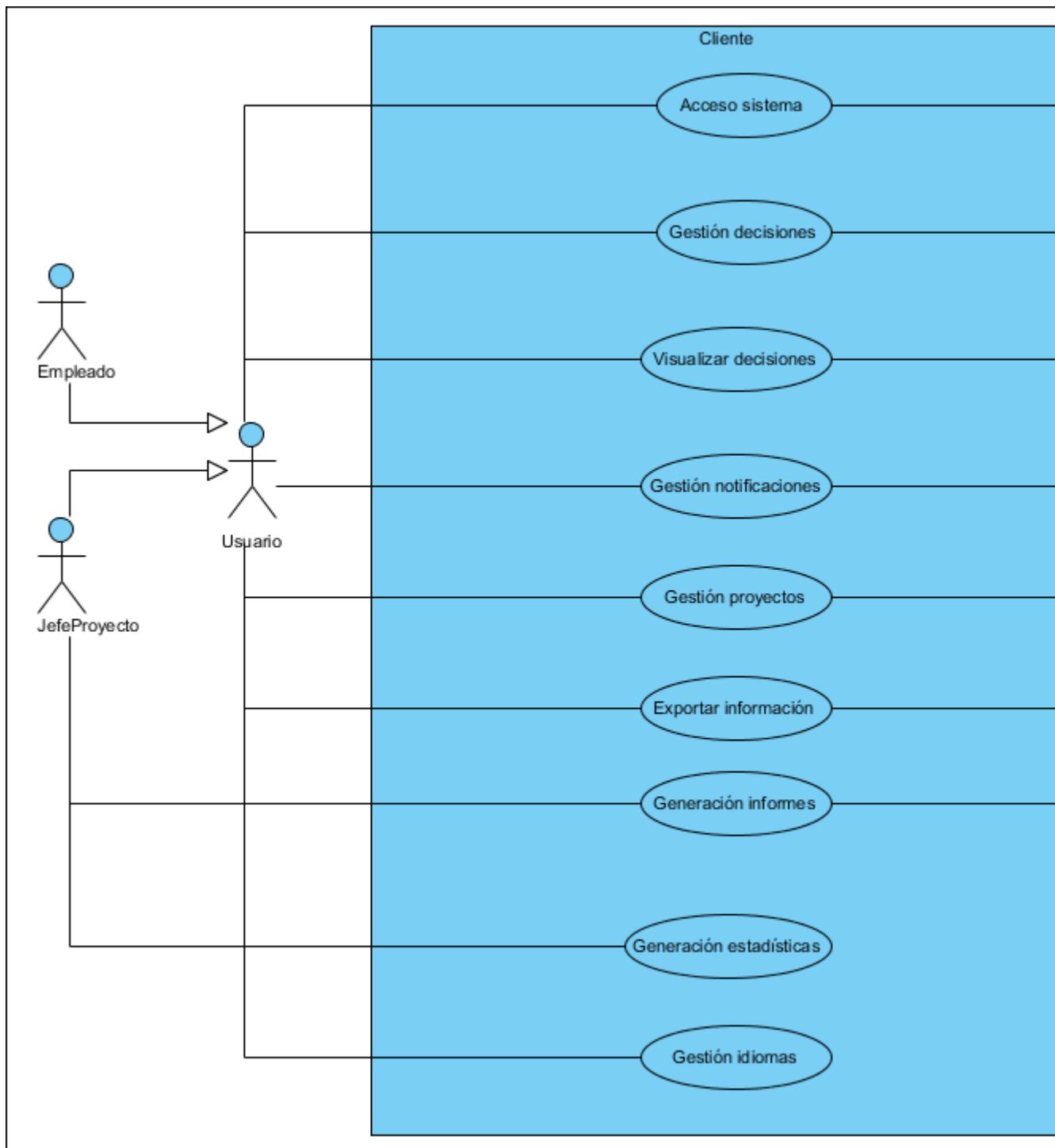


Figura 5.19: Diagrama de casos de uso - Cliente - v2.0

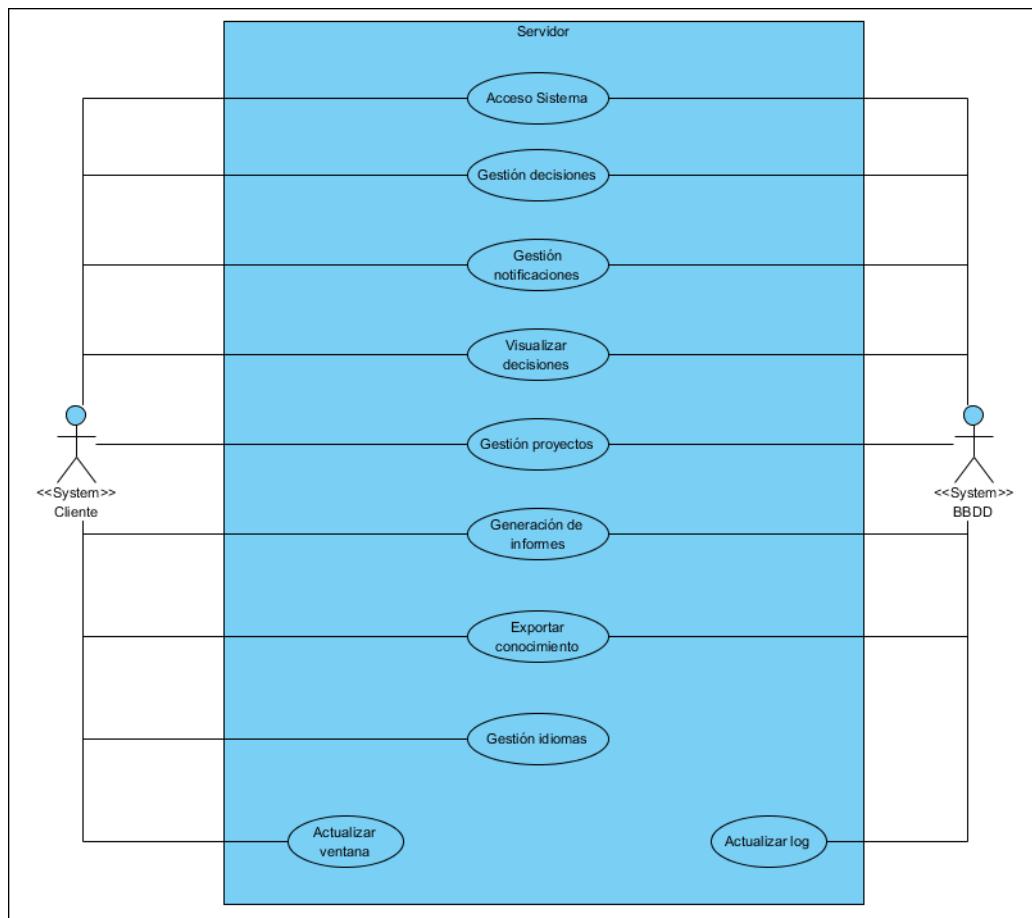


Figura 5.20: Diagrama de casos de uso - Servidor - v2.0

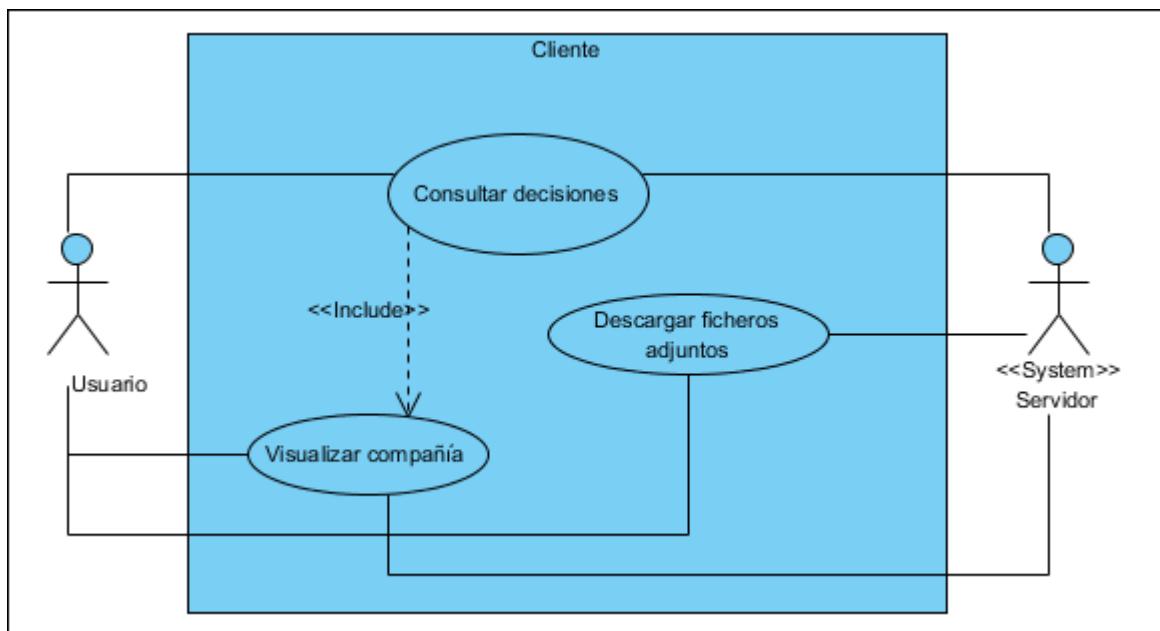


Figura 5.21: Diagrama de casos de uso - Cliente - Visualizar información - v2.0

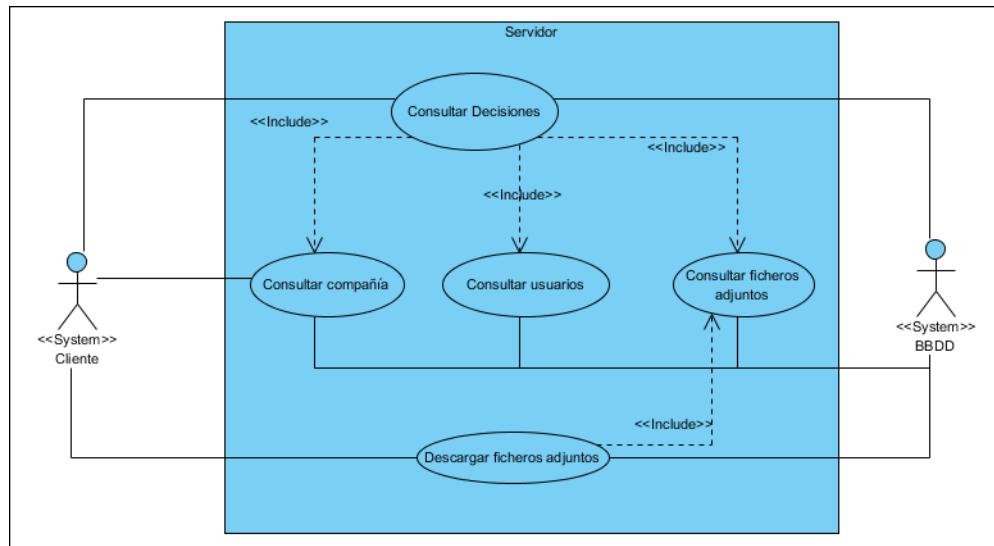


Figura 5.22: Diagrama de casos de uso - Servidor - Visualizar información - v2.0

5.2.1.2.3 Generación de estadísticas

En la Figura 1.25 se muestra el diagrama de casos de uso para el cliente. Esta funcionalidad es exclusiva del cliente, ya que, utilizando los datos que proporciona el servidor, las gráficas se generan y muestran en el cliente, ya que los tipos de gráficas son independientes del servidor y dependen de la tecnología utilizada para representarlas en el cliente.

5.2.1.2.4 Generación de informes

En la Figura 1.26 se muestra el diagrama de casos de uso para el cliente, mientras que en la Figura 1.27 se muestra el diagrama de casos de uso para el servidor.

5.2.1.3 Plan de iteraciones

Una vez identificados nuevos requisitos y modelados los casos de uso del sistema con esos nuevos requisitos, se reorganiza el plan de proyecto con nuevas iteraciones, para poder tener en cuenta el desarrollo de los nuevos casos de uso. Por tanto, a partir de las iteraciones planificadas en la Tabla 1.4, se obtiene el nuevo plan de iteraciones mostrado en la Tabla 1.7 (teniendo en cuenta que la fase de inicio ya se ha completado).

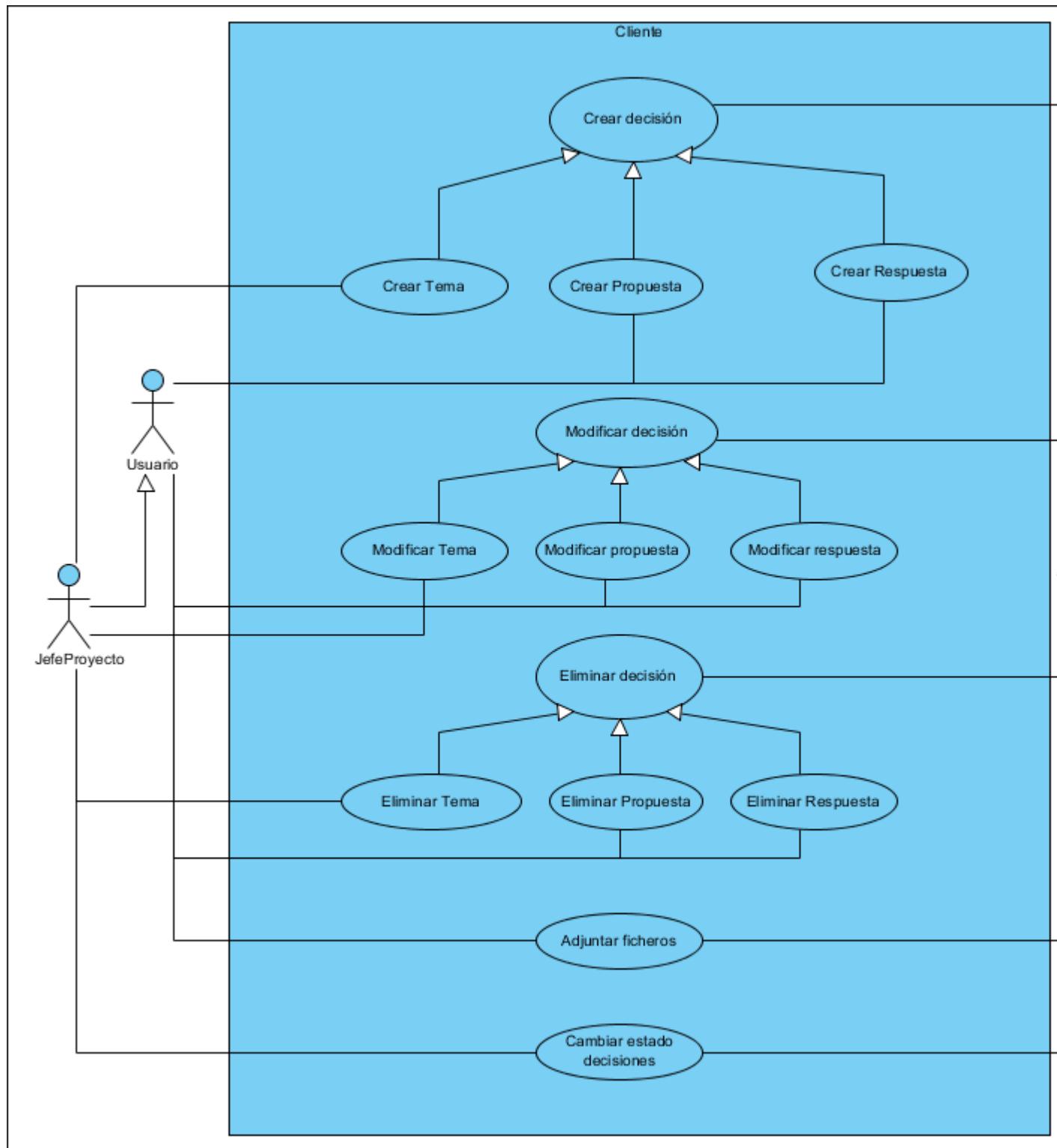


Figura 5.23: Diagrama de casos de uso - Cliente - Gestión decisiones - v2.0

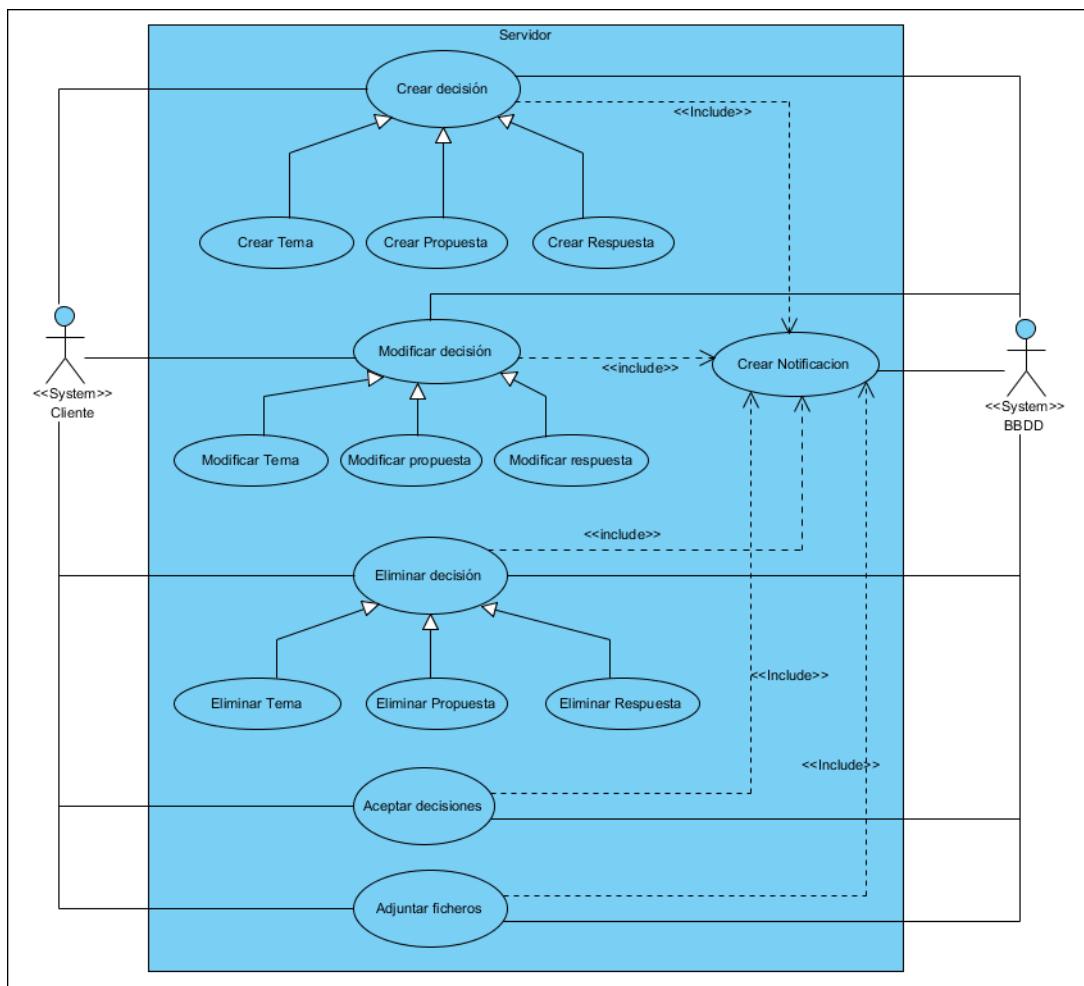


Figura 5.24: Diagrama de casos de uso - Servidor - Gestión decisiones - v2.0

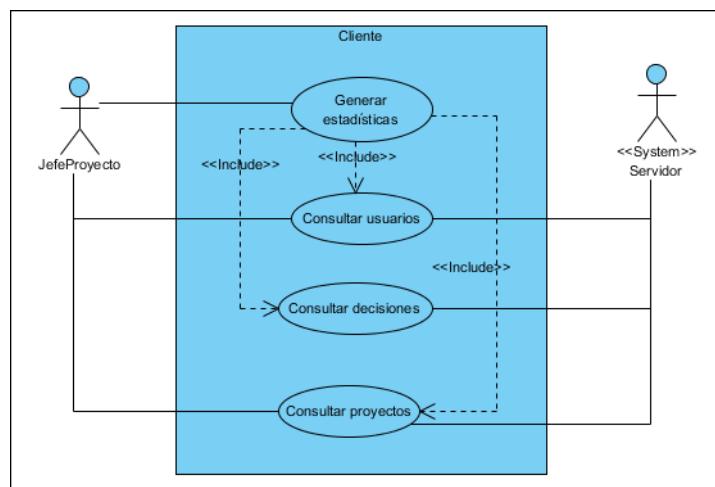


Figura 5.25: Diagrama de casos de uso - Cliente - Generación Estadísticas - v2.0

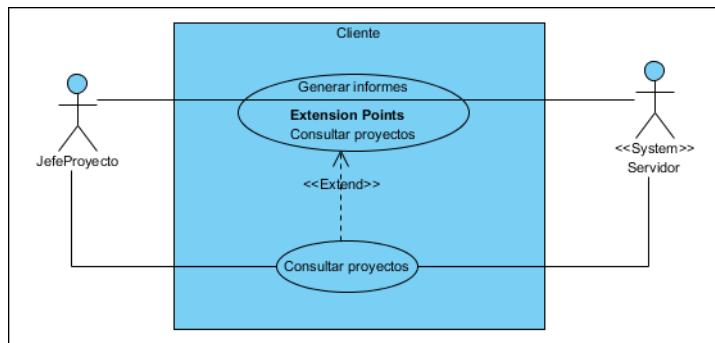


Figura 5.26: Diagrama de casos de uso - Cliente - Generación Informes - v2.0

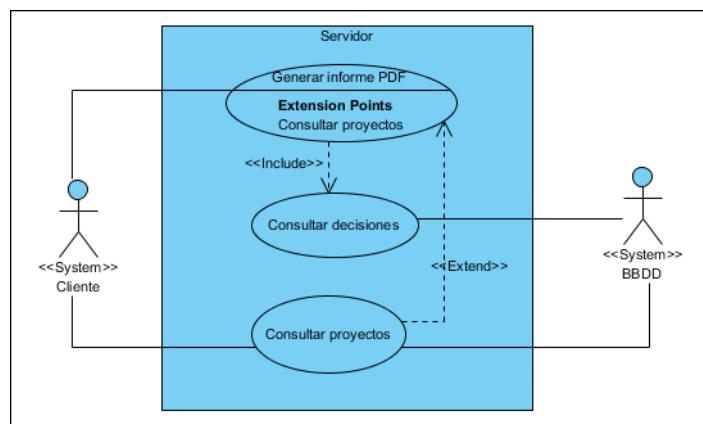


Figura 5.27: Diagrama de casos de uso - Servidor - Generación Informes - v2.0

Fase	Iteración	Tareas a realizar
Inicio	1	Identificar requisitos Analizar la viabilidad del proyecto Crear modelo de casos de uso de alto nivel Plan de proyecto
Elaboración	1	Identificar requisitos Crear modelo de casos de uso Diseñar arquitectura Crear modelo de clases de análisis
	2	Diseñar caso de uso Acceso al sistema Implementar caso de uso Acceso al sistema Probar caso de uso Acceso al sistema
Construcción	1	Diseñar caso de uso Visualizar información Implementar caso de uso Visualizar información Probar caso de uso Visualizar información
	2	Diseñar casos de uso Gestión decisiones y Gestión notificaciones Implementar casos de uso Gestión decisiones y Gestión notificaciones Probar casos de uso Gestión decisiones y Gestión notificaciones
	3	Diseñar caso de uso Gestión proyectos Implementar caso de uso Gestión proyectos Probar caso de uso Gestión proyectos
	4	Diseñar casos de uso Generación de estadísticas y Generación de informes Implementar casos de uso Generación de estadísticas y Generación de informes Probar casos de uso Generación de estadísticas y Generación de informes

Fase	Iteración	Tareas a realizar
	5	Diseñar casos de uso Gestión idiomas y Exportar información Implementar casos de uso Gestión idiomas y Exportar información Probar casos de uso Gestión idiomas y Exportar información
Transición	1	Preparar entregables Realizar documentación y manuales

Tabla 5.7: Plan de iteraciones actualizado

5.2.1.4 Análisis de casos de uso

Los casos de uso obtenidos anteriormente se especifican de manera más formal y en detalle, describiendo el funcionamiento de cada uno de ellos y sus escenarios. Además, para cada uno de ellos, se crea el diagrama de clases de análisis, que representa las acciones que se llevan a cabo entre las clases de dominio identificadas para dar soporte a cada caso de uso. Existen tres tipos de clases:

1. **Interfaz o Boundary:** representa clases utilizadas para la comunicación entre el actor y el sistema.
2. **Control:** representa clases y objetos de control. Es la que controla y coordina el flujo (o escenario) de funcionamiento del caso de uso.
3. **Entidad o Entity:** son clases que representan objetos de información persistentes del sistema.

En los sucesivos apartados se describen los casos de uso que componen el sistema y se muestran los diagramas de clases de análisis para cada uno de ellos.

Señalar que para los casos de uso se describe su funcionamiento desde un punto de vista global al sistema, sin entrar en detalle para cada subsistema, ya que lo que interesa es describir

a alto nivel su funcionamiento. Posteriormente, con los diagramas de clases de análisis, de secuencia y de clases de diseño, se irá profundizando más en detalle en el flujo que se sigue específicamente en cada subsistema.

5.2.1.4.1 Acceso al sistema

Login

En la Tabla 1.8 se describe el caso de uso *Login*.

En la Figura 1.28 se muestra el diagrama de clases de análisis para el subsistema cliente.

En la Figura 1.29 se muestra el diagrama de clases de análisis para el subsistema servidor.

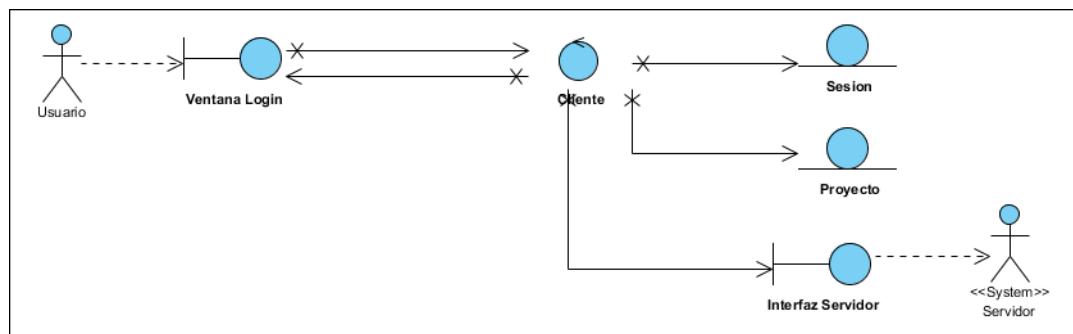


Figura 5.28: Diagrama de clases de análisis - Cliente - Login

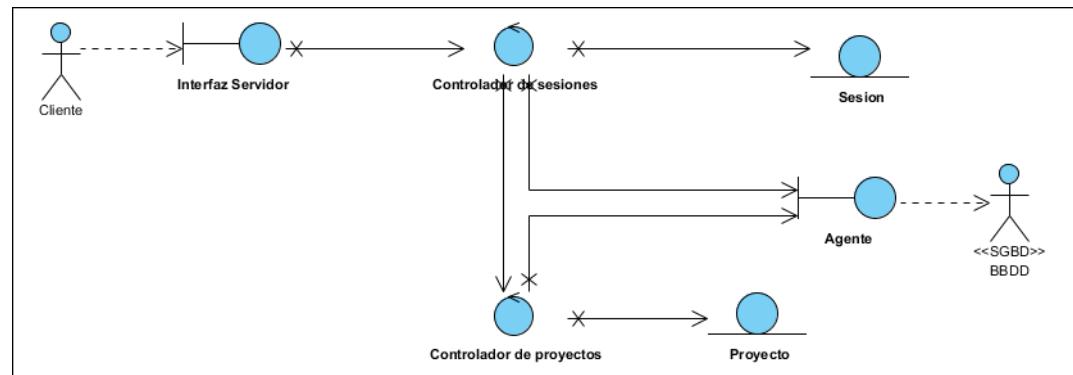


Figura 5.29: Diagrama de clases de análisis - Servidor - Login

Logout

En la Tabla 1.9 se describe el caso de uso *Logout*.

Nombre: Iniciar sesión (Login)
Descripción: Funcionalidad para que un usuario pueda acceder al sistema
Precondiciones: Disponer de nombre de usuario y contraseña y estar dado de alta en al menos un proyecto
Post-condiciones: El usuario accede al sistema
<p>Flujo principal:</p> <ol style="list-style-type: none"> 1. El usuario introduce usuario y contraseña. 2. Se valida el usuario y contraseña. 3. Se crea una sesión para el usuario. 4. Se muestran los proyectos en los que el usuario está dado de alta. 5. El usuario elige uno de los proyectos. 6. El usuario accede al sistema para trabajar sobre ese proyecto.
<p>Flujo alternativo 1: error de Login:</p> <ol style="list-style-type: none"> 1. El usuario introduce usuario y contraseña. 2. Se valida el usuario y contraseña. 3. El usuario no es válido. Se informa del error y se vuelve al comienzo.
<p>Flujo alternativo 2: usuario inexistente:</p> <ol style="list-style-type: none"> 1. El usuario introduce usuario y contraseña. 2. Se valida el usuario y contraseña. 3. El usuario no existe. Se informa del error y se vuelve al comienzo.
<p>Flujo alternativo 3: usuario ya logueado:</p> <ol style="list-style-type: none"> 1. El usuario introduce usuario y contraseña. 2. Se valida el usuario y contraseña. 3. Ya existe una sesión para ese usuario. Se cierra dicha sesión y se crea una nueva. 4. Se muestran los proyectos en los que el usuario está dado de alta. 5. El usuario elige uno de los proyectos. 6. El usuario accede al sistema para trabajar sobre ese proyecto.
<p>Flujo alternativo 4: no existen proyectos:</p> <ol style="list-style-type: none"> 1. El usuario introduce usuario y contraseña. 2. Se valida el usuario y contraseña. 3. Se crea una sesión para el usuario. 4. Se consultan los proyectos en los que el usuario está dado de alta. 5. El usuario no tiene ningún proyecto. Se informa del error y se vuelve al comienzo.

Tabla 5.8: Especificación del caso de uso *Login*

Nombre: Cerrar sesión (Logout)
Descripción: Funcionalidad para que un usuario pueda desconectarse del sistema
Precondiciones: El usuario debe estar conectado en el sistema
Post-condiciones: El usuario se desconecta del sistema
Flujo principal: <ol style="list-style-type: none"> 1. El usuario cierra sesión. 2. Se elimina la sesión del usuario. 3. Se elimina al usuario del sistema. 4. Se vuelve a la pantalla de inicio.

Tabla 5.9: Especificación del caso de uso *Logout*

En la Figura 1.30 se muestra el diagrama de clases de análisis para el subsistema cliente.

En la Figura 1.31 se muestra el diagrama de clases de análisis para el subsistema servidor.

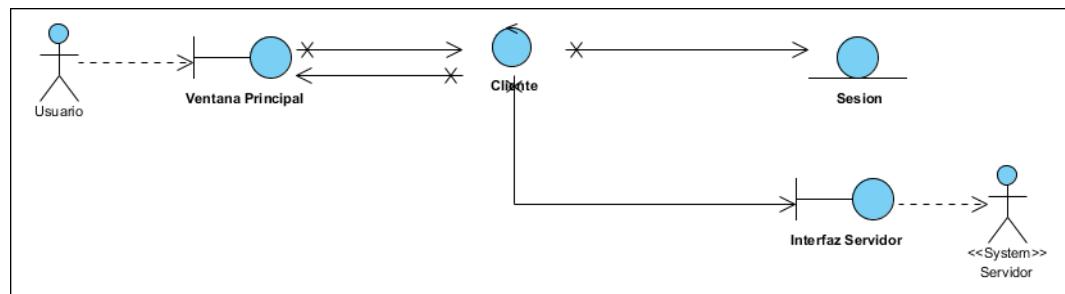


Figura 5.30: Diagrama de clases de análisis - Cliente - Logout

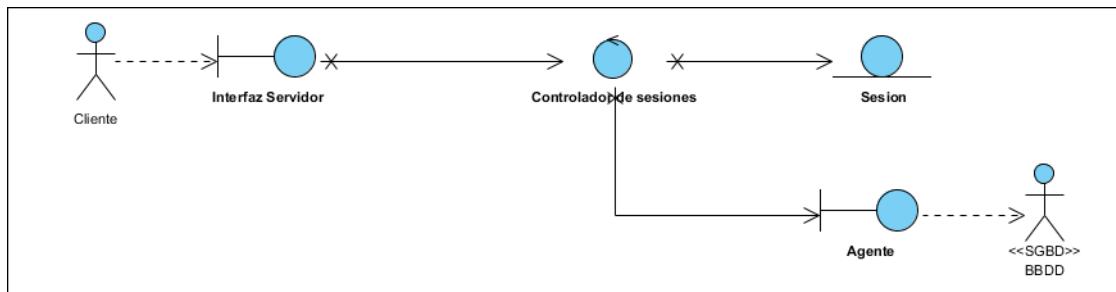


Figura 5.31: Diagrama de clases de análisis - Servidor - Logout

5.2.1.4.2 Visualizar decisiones

Consultar decisiones

En la Tabla 1.10 se describe el caso de uso *Consultar decisiones*.

Nombre: Consultar decisiones
Descripción: Funcionalidad para consultar las decisiones (y toda su información relacionada) de un proyecto.
Precondiciones: Que el usuario haya accedido al sistema y tenga permisos para realizar la operación.
Post-condiciones: Se consultan las decisiones de un proyecto y se muestran.
<p>Flujo principal:</p> <ol style="list-style-type: none"> 1. El usuario inicia la acción para consultar las decisiones de un proyecto. 2. El sistema consulta las decisiones del proyecto. 3. El sistema consulta la información asociada a las decisiones. 4. Se muestran las decisiones encontradas. 5. Se muestra la información asociada a las decisiones. <p>Flujo alternativo 1: no existen decisiones:</p> <ol style="list-style-type: none"> 1. El usuario inicia la acción para consultar las decisiones de un proyecto. 2. El sistema consulta las decisiones del proyecto. 3. Se muestra mensaje de que no hay ninguna decisión.

Tabla 5.10: Especificación del caso de uso *Consultar decisiones*

La Figura 1.32 representa el diagrama de clases de análisis para el subsistema cliente. En la Figura 1.33 se muestra el diagrama de clases de análisis para el subsistema servidor.

Consultar compañía

La Figura 1.34 representa el diagrama de clases de análisis para el subsistema cliente. En la Figura 1.35 se muestra el diagrama de clases de análisis para el subsistema servidor.

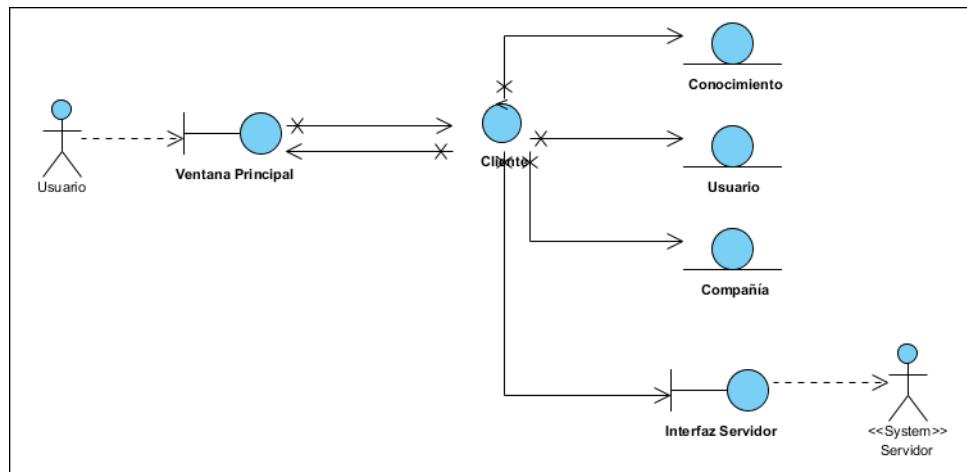


Figura 5.32: Diagrama de clases de análisis - Cliente - Consultar Decisiones

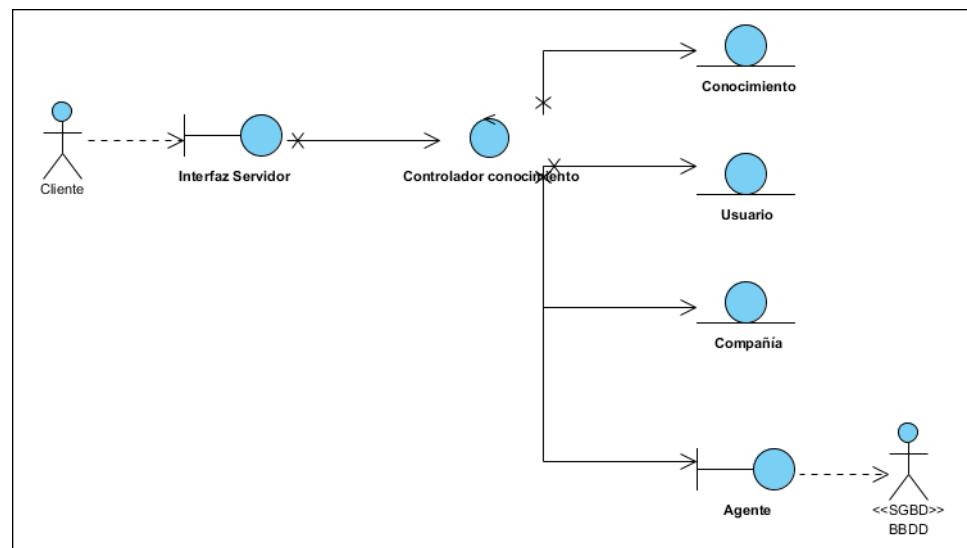


Figura 5.33: Diagrama de clases de análisis - Servidor - Consultar Decisiones

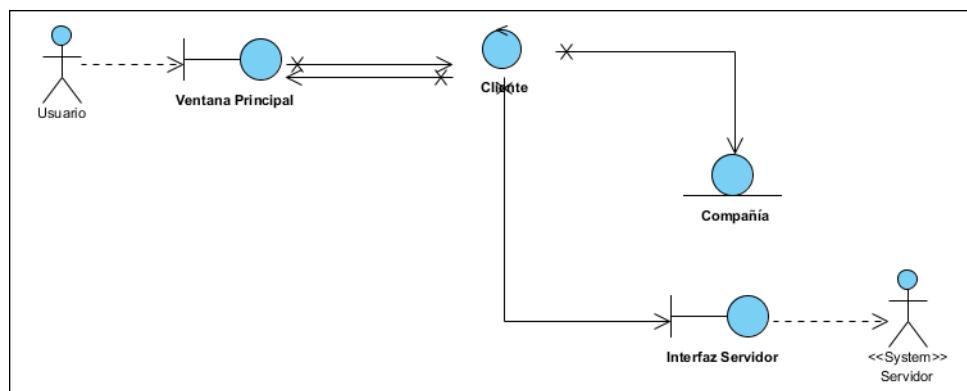


Figura 5.34: Diagrama de clases de análisis - Cliente - Consultar compañía

Nombre: Consultar compañía
Descripción: Funcionalidad para consultar los detalles de una compañía.
Precondiciones: Que el usuario haya accedido al sistema y tenga permisos para realizar la operación.
Post-condiciones: Se consultan los detalles de una compañía y se muestran.
<p>Flujo principal:</p> <ol style="list-style-type: none"> 1. El usuario inicia la acción para consultar los detalles de una compañía. 2. El sistema consulta los datos de la compañía. 3. Se muestran los datos de la compañía.

Tabla 5.11: Especificación del caso de uso *Consultar compañía*

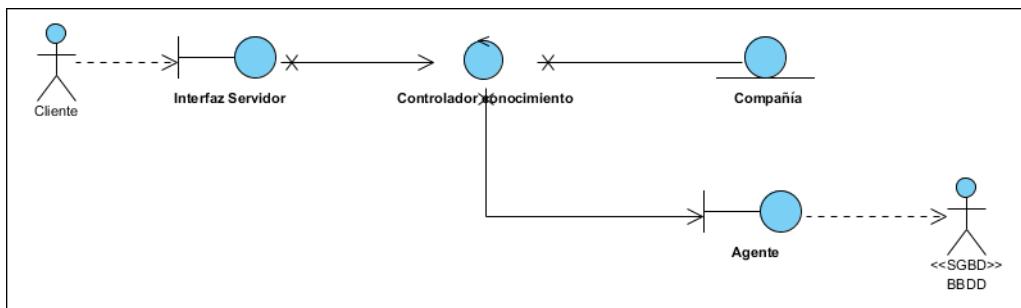


Figura 5.35: Diagrama de clases de análisis - Servidor - Consultar compañía

5.2.1.4.3 Gestión decisiones

Crear decisión

Este es un caso de uso del que heredan otros tres casos, según el tipo de decisión que se desee crear (ver Figura 1.24). A continuación se describen cada uno de los casos de uso.

En la Tabla 1.12 se describe el caso de uso *Crear Decisión - Crear Tema*.

Nombre: Crear Decisión - Crear Tema (<i>Topic</i>)
Descripción: Funcionalidad para crear un nuevo Tema en un proyecto.
Precondiciones: Que el usuario haya accedido al sistema y tenga permisos para realizar la operación.
Post-condiciones: Se crea un nuevo Tema para un proyecto.
<p>Flujo principal:</p> <ol style="list-style-type: none"> 1. El jefe de proyecto introduce un título y descripción para el Tema. 2. El sistema valida los datos introducidos. 3. Se crea el nuevo Tema para el proyecto. 4. El sistema crea una notificación para los usuarios dados de alta en el proyecto. 5. Se refrescan y actualizan las decisiones.
<p>Flujo alternativo 1: datos incompletos:</p> <ol style="list-style-type: none"> 1. El jefe de proyecto introduce un título y descripción para el Tema. 2. El sistema valida los datos introducidos. 3. Los datos son incompletos. Se muestra un mensaje y se vuelven a solicitar los datos del Tema.
<p>Flujo alternativo 2: Tema ya existente:</p> <ol style="list-style-type: none"> 1. El jefe de proyecto introduce un título y descripción para el Tema. 2. El sistema valida los datos introducidos. 3. Ya existe ese Tema. Se muestra un mensaje y se vuelven a solicitar los datos.

Tabla 5.12: Especificación del caso de uso *Crear decisión - Crear Tema*

En la Figura 1.36 se muestra el diagrama de clases de análisis para el subsistema cliente.

En la Figura 1.37 se muestra el diagrama de clases de análisis para el subsistema servidor.

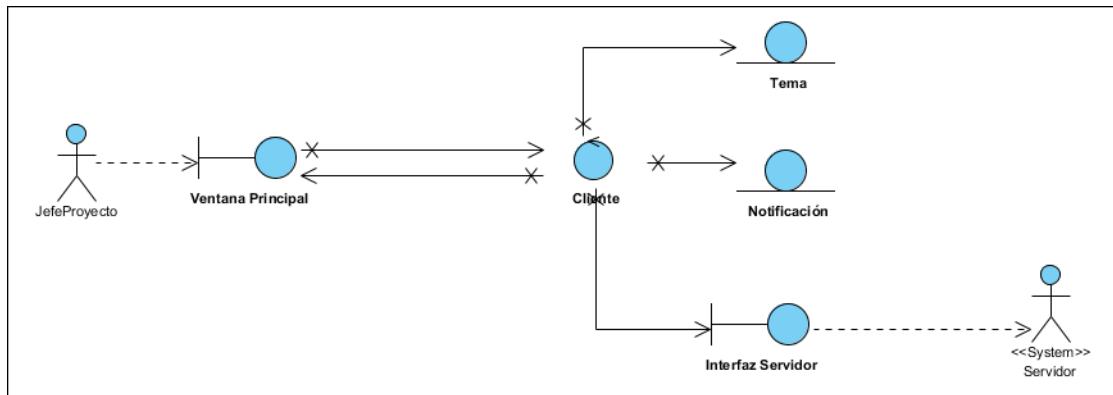


Figura 5.36: Diagrama de clases de análisis - Cliente - Crear Tema

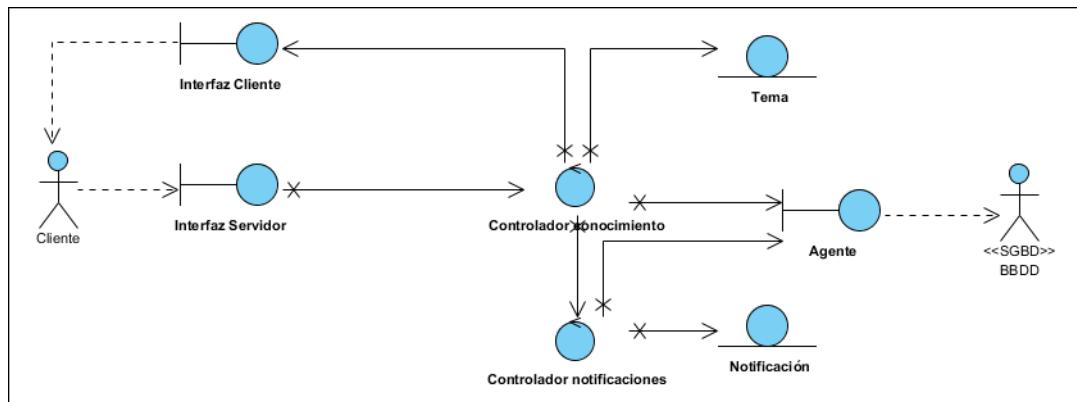


Figura 5.37: Diagrama de clases de análisis - Servidor - Crear Tema

En la Tabla 1.13 se describe el caso de uso *Crear Decisión - Crear Propuesta*.

Nombre: Crear Decisión - Crear Propuesta (<i>Proposal</i>)
Descripción: Funcionalidad para crear una nueva Propuesta en un proyecto.
Precondiciones: Que el usuario haya accedido al sistema y tenga permisos para realizar la operación.
Post-condiciones: Se crea una nueva Propuesta para un proyecto.
<p>Flujo principal:</p> <ol style="list-style-type: none"> 1. El usuario selecciona un Tema. 2. El usuario introduce los datos de la nueva Propuesta. 3. El sistema valida los datos introducidos. 4. Se crea la nueva Propuesta para el Tema seleccionado. 5. El sistema crea una notificación para los usuarios dados de alta en el proyecto. 6. Se refrescan y actualizan las decisiones.
<p>Flujo alternativo 1: datos incompletos:</p> <ol style="list-style-type: none"> 1. El usuario selecciona un Tema. 2. El usuario introduce los datos de la nueva Propuesta. 3. El sistema valida los datos introducidos. 4. Los datos son incompletos. Se muestra un mensaje y se vuelven a solicitar los datos de la Propuesta.
<p>Flujo alternativo 2: Propuesta ya existente:</p> <ol style="list-style-type: none"> 1. El usuario selecciona un Tema. 2. El usuario introduce los datos de la nueva Propuesta. 3. El sistema valida los datos introducidos. 4. Ya existe esa Propuesta. Se muestra un mensaje y se vuelven a solicitar los datos.

Tabla 5.13: Especificación del caso de uso *Crear decisión - Crear Propuesta*

En la Figura 1.38 se muestra el diagrama de clases de análisis para el subsistema cliente.

En la Figura 1.39 se muestra el diagrama de clases de análisis para el subsistema servidor.

En la Tabla 1.14 se describe el caso de uso *Crear Decisión - Crear Respuesta*.

En la Figura 1.40 se muestra el diagrama de clases de análisis para el subsistema cliente.

En la Figura 1.41 se muestra el diagrama de clases de análisis para el subsistema servidor.

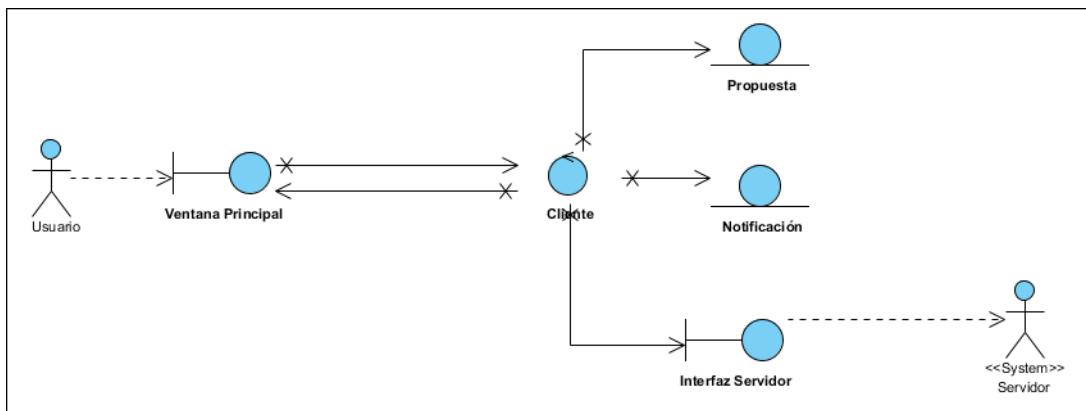


Figura 5.38: Diagrama de clases de análisis - Cliente - Crear Propuesta

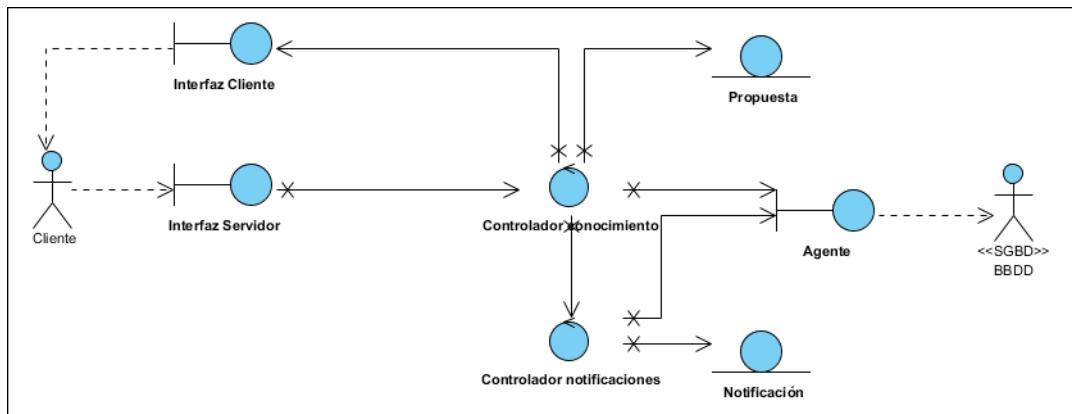


Figura 5.39: Diagrama de clases de análisis - Servidor - Crear Propuesta

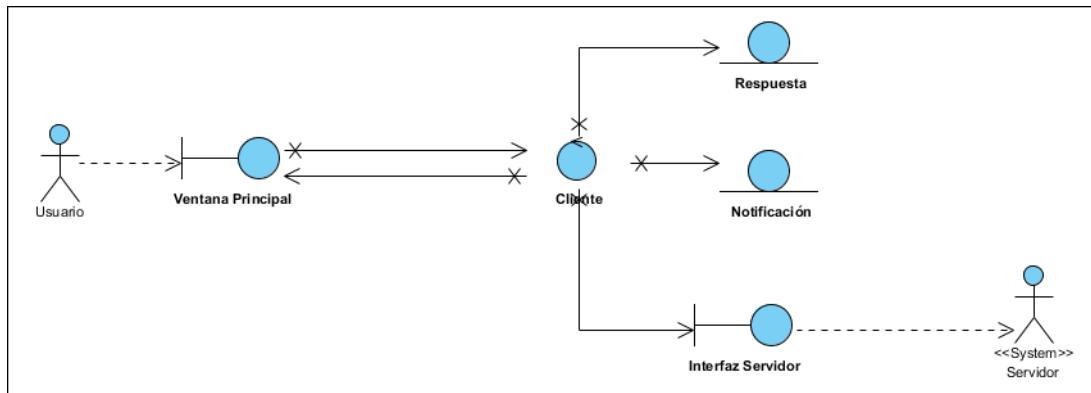


Figura 5.40: Diagrama de clases de análisis - Cliente - Crear Respuesta

Nombre: Crear Decisión - Crear Respuesta (<i>Answer</i>)
Descripción: Funcionalidad para crear una nueva Respuesta en un proyecto.
Precondiciones: Que el usuario haya accedido al sistema y tenga permisos para realizar la operación.
Post-condiciones: Se crea una nueva Respuesta para un proyecto.
Flujo principal: <ol style="list-style-type: none">1. El usuario selecciona una Respuesta.2. El usuario introduce los datos de la nueva Respuesta.3. El sistema valida los datos introducidos.4. Se crea la nueva Respuesta para la Propuesta seleccionada.5. El sistema crea una notificación para los usuarios dados de alta en el proyecto.6. Se refrescan y actualizan las decisiones.
Flujo alternativo 1: datos incompletos: <ol style="list-style-type: none">1. El usuario selecciona una Propuesta.2. El usuario introduce los datos de la nueva Respuesta.3. El sistema valida los datos introducidos.4. Los datos son incompletos. Se muestra un mensaje y se vuelven a solicitar los datos de la Respuesta.
Flujo alternativo 2: Respuesta ya existente: <ol style="list-style-type: none">1. El usuario selecciona una Propuesta.2. El usuario introduce los datos de la nueva Respuesta.3. El sistema valida los datos introducidos.4. Ya existe esa Respuesta. Se muestra un mensaje y se vuelven a solicitar los datos.

Tabla 5.14: Especificación del caso de uso *Crear decisión - Crear Respuesta*

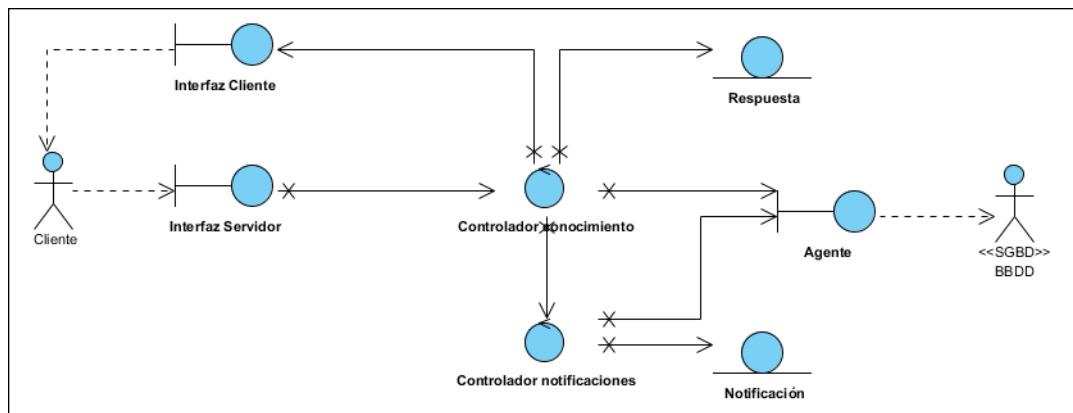


Figura 5.41: Diagrama de clases de análisis - Servidor - Crear Respuesta

Del mismo modo que los anteriores, se detallan los escenarios y se crean sus diagramas de clases de análisis para los siguientes casos de uso:

- **Gestión de decisiones**

- Modificar Tema
- Modificar Propuesta
- Modificar Respuesta
- Eliminar Tema
- Eliminar Propuesta
- Eliminar Respuesta
- Aceptar decisión
- Adjuntar fichero

- **Gestión de notificaciones**

- Consultar notificación
- Modificar notificación
- Eliminar notificación

- **Gestión de proyectos**

- Consultar proyectos
- Crear proyecto
- Modificar proyecto
- Seleccionar proyecto activo
- Aconsejar decisiones

- **Generación de estadísticas**

- Generación de estadísticas

- **Generación de informes**

- Generación de informes

- **Gestión de idiomas**

- Cambiar idioma

- **Exportar conocimiento**

- Exportar conocimiento

Se ha decidido no incorporar todos estos diagramas por la gran cantidad de diagramas que existen, para no aumentar excesivamente la longitud del documento.

5.2.1.5 Arquitectura del sistema

Al final de esta primera iteración de la fase de elaboración, se comienza con el diseño del sistema. En este caso se define y diseña su arquitectura. En posteriores iteraciones y fases, se continuará con la labor de diseño del sistema, creando los diagramas de secuencia para cada caso de uso y el diagrama de clases de diseño, siendo la base para la posterior implementación.

5.2.1.5.1 Arquitectura cliente-servidor

Como se comentó en el apartado 1.1.1.2, el sistema a desarrollar debe ser distribuido, para poder ser utilizado desde localizaciones diferentes, por lo que se ha decidido utilizar una arquitectura cliente-servidor. De esta forma, el sistema completo se divide en dos subsistemas:

- **Subsistema Cliente:** este subsistema es el utilizado por los miembros de los equipos de desarrollo deslocalizados. Este subsistema es el que se encarga de mostrar la interfaz gráfica de usuario, recoger las acciones que el usuario desea hacer y enviar dicha acción al servidor, esperando su respuesta para actualizar la interfaz gráfica en consecuencia.
- **Subsistema Servidor:** este subsistema se encarga de recibir las peticiones del subsistema cliente, procesarlas, almacenar y recuperar información de su base de datos y enviar la respuesta al cliente. También se encarga de que todas las acciones queden registradas.

Así, cada uno de los subsistemas se puede encontrar en máquinas diferentes, pues, siguiendo la arquitectura cliente-servidor, se comunican a través de **RMI** (ver sección ??). Además, la base de datos utilizada por el servidor también se pueden encontrar en una máquina diferente a la máquina donde se encuentre el servidor.

Por tanto, algunas ventajas de utilizar este enfoque distribuido siguiendo la arquitectura cliente-servidor son:

- **Centralización del control:** toda la lógica de dominio y control está centralizada en el servidor, por lo que el sistema cliente es totalmente independiente de la implementación del servidor. Del mismo modo, el sistema cliente es independiente de como se realiza la gestión del conocimiento en el servidor. Además, los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un cliente defectuoso o no autorizado no pueda dañar el sistema.
- **Escalabilidad:** se pueden añadir nuevos tipos de sistemas clientes para que se comuniquen con el servidor.
- **Fácil mantenimiento:** al estar los sistemas distribuidos en diferentes máquinas, es posible reemplazar, reparar o actualizar el servidor, mientras que sus clientes no se verán afectados por ese cambio.

En la Figura 1.42 se muestra una vista de los subsistemas que integran el sistema global y como se comunican dichos sistemas a través de interfaces, facilitando la distribución de cada uno de los sistemas en diferentes máquinas.

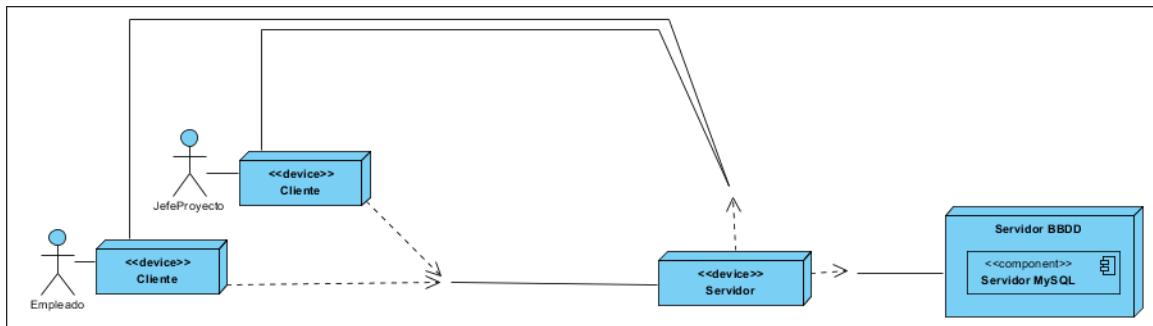


Figura 5.42: Arquitectura cliente-servidor

5.2.1.5.2 Arquitectura multicapa

En cuanto a la arquitectura de implementación, ambos sistemas, cliente y servidor, serán desarrollados siguiendo una arquitectura multicapa, de modo que se pueda aislar la capa de presentación de la de la lógica de dominio y ésta de la capa de persistencia (ver Figura 1.43).

De este modo, las capas con las que cuenta cada subsistema son las siguientes:

- Cliente: comunicaciones, dominio y presentación.

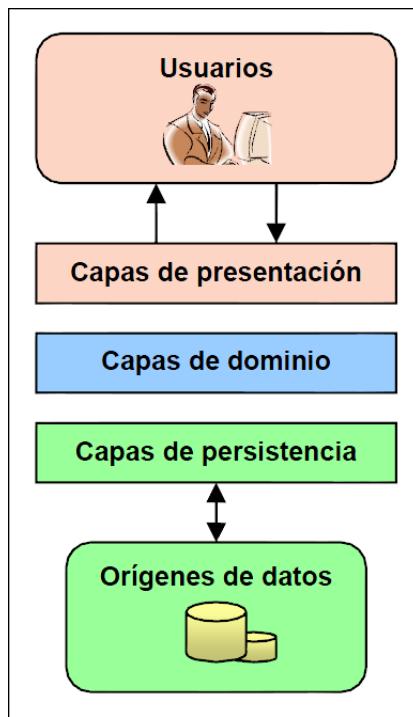


Figura 5.43: Arquitectura multicapa

- Servidor: comunicaciones, dominio, persistencia y presentación.

Más concretamente, estas capas se traducirán a paquetes de implementación, donde cada uno agrupara los siguientes elementos:

- **Comunicaciones:** contiene las clases e interfaces necesarias para la comunicación de los subsistemas a través de RMI.
- **Dominio:** contiene todos los objetos del dominio de la aplicación.
- **Persistencia:** contiene las clases encargadas de gestionar la persistencia de los objetos de dominio.
- **Presentación:** contiene todas las vistas de la interfaz gráfica de usuario, organizadas en subpaquetes.

En la Figura 1.44 puede observarse esta arquitectura multicapa y las relaciones entre dichas capas.

Utilizando este enfoque multicapa, se sigue el principio de mínimo acoplamiento y máxima cohesión, desacoplando los elementos de una capa de los de otra. De este modo, se

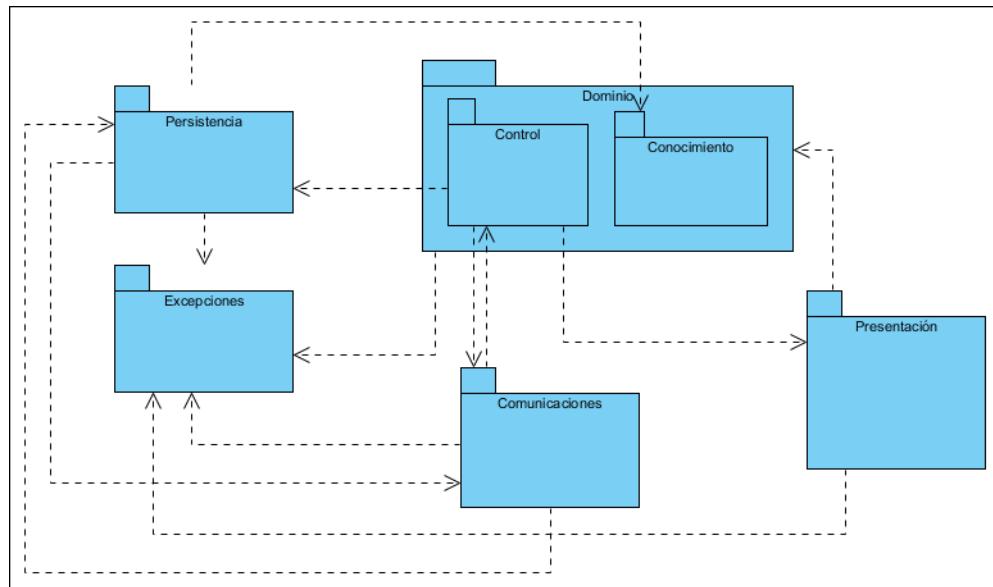


Figura 5.44: Arquitectura multicapa

facilita el mantenimiento y extensibilidad del sistema, pues cuando se realice el cambio en algún elemento de una capa, el resto de capas no se verán afectadas.

5.2.1.5.3 Patrones utilizados

En cuanto a los patrones de diseño, se han utilizado los siguientes para el desarrollo del sistema:

- **Singleton:** patrón que asegura que para una determinada clase solo exista una única instancia y provee un punto de acceso a ella. Se utiliza para implementar el patrón Agente y para otras clases que deben ser únicas en el sistema.
- **Agente:** patrón utilizado para gestionar acceso y manipulación a ficheros.
- **DAO (Data Access Object):** patrón que permite separar los objetos de dominio de su persistencia, haciendo independiente cómo se almacenan. Se ha utilizado este patrón para gestionar la persistencia de los objetos de dominio.
- **Observador:** patrón que permite que los cambios sobre un objeto se notifiquen directamente al conjunto de objetos dependientes (observados). Se utiliza para gestionar varias conexiones de bases de datos y para notificar a los clientes cambios que se produzcan en el sistema.

- **Fachada:** patrón que proporciona una interfaz de alto nivel que delega posteriormente en una o varias clases. Se ha utilizado para implementar la interfaz de operaciones del servidor, que el subsistema cliente puede utilizar.
- **Proxy:** patrón que proporciona el acceso a otro objeto, normalmente remoto. Se ha utilizado para comunicar ambos subsistemas por RMI.
- **Iterator:** patrón que proporciona un mecanismo para acceder y recorrer objetos secuencialmente, sin exponer su representación. Se ha utilizado para recorrer enumeraciones y colecciones.

5.2.2 Segunda iteración

Al finalizar la iteración anterior, en la que se identificaron nuevos requisitos, éstos se añadieron como nuevos casos de uso, se crearon las clases de análisis y se especificó la arquitectura del sistema, se realizó otra reunión de seguimiento antes de comenzar la nueva iteración, para revisar y validar todos estos artefactos obtenidos.

En dicha reunión, ya no se identificaron más requisitos y se validó el plan de iteraciones y el resto de artefactos obtenidos, por lo que en sucesivas iteraciones y fases se comenzó con el diseño, implementación y pruebas de los casos de uso identificados.

Cabe destacar que, como se explicó en el apartado 1.2.1.5, al utilizar una arquitectura cliente-servidor, el subsistema del servidor es el encargado de toda la lógica y control de dominio y persistencia, por lo que es donde más hincapié se hará en los diagramas de diseño, mientras que el cliente se encarga de proporcionar la interfaz gráfica del sistema, validar los datos que introduce el usuario y enviar y recibir las peticiones del servidor.

Señalar que la implementación de ambos subsistemas se irá haciendo en paralelo, es decir, cada caso de uso se desarrollará para el servidor y para el cliente, cerrando así la implementación completa de ese caso de uso.

Las tareas a realizar en esta iteración son:

- Diseño de clases del dominio.
- Diseño de la base de datos.

- Definición y diseño de la arquitectura del sistema.
- Implementación del control de acceso al sistema, así como la implementación de la arquitectura cliente-servidor.
- Diseño e implementación de pruebas unitarias y funcionales relativas a la comunicación entre ambos subsistemas y del acceso al sistema.

5.2.2.1 Diagrama de clases de dominio

En esta segunda iteración de la fase de elaboración, se realiza un diagrama de clases de diseño de alto nivel, reflejando las clases de los objetos de dominio y sus relaciones. Este diagrama se irá detallando y refinando durante las iteraciones de la fase de construcción.

Así, según la especificación de requisitos de la sección 1.1.1.1, se han modelado las siguientes clases de conocimiento, junto con sus relaciones:

- **User**: una clase abstracta que representa los usuarios que pueden acceder y hacer uso del sistema. De ella heredan dos clases, que son *Employee* y *ChiefProject*, representando los diferentes roles de usuarios que existen en el sistema. Estas clases concretas implementan el método abstracto *getRole()*, devolviendo el rol correspondiente a cada clase. Es una clase persistente.
- **Company**: clase que representa una sede de una compañía donde trabajan los diferentes usuarios del sistema. Es una clase persistente.
- **Address**: representa la dirección física de una sede de una compañía, representando su dirección, ciudad, país, etc.. Es una clase persistente.
- **Project**: es la clase que representa un proyecto software donde trabajan los usuarios del sistema. Es una clase persistente.
- **Session**: es la clase encargada de almacenar toda la información sobre las sesiones que se inicien en el sistema.
- **Knowledge**: es una clase abstracta que representa el conocimiento del sistema. En este caso, son las decisiones tomadas en cada proyecto por los diferentes usuarios que en

participan en los proyectos. De esta clase heredan las clases concretas *Topic*, *Proposal* y *Answer*, formando la jerarquía de decisiones que se detallará en el apartado 1.3.1.1.2. Es una clase persistente.

- **TopicWrapper:** es una clase que representa el conjunto de temas (*topics*) presentes en un proyecto.
- **File:** clase que representa un fichero que puede adjuntarse a una decisión. Es una clase persistente.
- **Notification:** clase que representa una notificación o alerta que se crea en el sistema cuando se provoca un cambio en las decisiones de un proyecto. Es una clase persistente.
- **PDFElement:** esta clase representa los elementos que van a componer un documento PDF. De ella heredan los diferentes elementos usados en un documento PDF, como son *PDFTable*, *PDFText* y *PDFTitle*. Gracias a esta herencia, se podrá usar el polimorfismo a la hora de componer las secciones del documento PDF.
- **PDFSection:** representa una sección de un documento PDF, compuesta de diferentes elementos.
- **PDFConfiguration:** clase que representa la información de un documento PDF, compuesto de un conjunto de secciones.
- **Coordinates:** esta clase representa las coordenadas geográficas de una dirección de una sede de una compañía.
- **Operation:** clase que representa las acciones que se pueden realizar en el sistema.
- **LogEntry:** por la especificación de requisitos, se debe mantener un histórico de todas las acciones realizadas en el sistema. Por tanto, esta es la clase que almacena información sobre quién, cuándo y qué acción se ha realizado. Es una clase persistente.

En cuanto a las relaciones entre las clases anteriores, se han modelado las siguientes:

- Un usuario se relaciona con una compañía en la que trabaja, y se relaciona con uno o más proyectos en los que participa. A su vez, en cada proyecto trabajan varios usuarios.

- Una compañía se asocia con una dirección.
- Una sesión se relaciona con un usuario, que es el que se identifica e inicia sesión en el sistema.
- Cada clase que representa una decisión (*Knowledge*) se asocia con un usuario, que es el autor que la crea.
- Cada tema (*Topic*) se asocia con 0 o más propuestas (*Proposal*) y con un proyecto, que es donde se crea.
- Cada propuesta se asocia con 0 o más respuestas (*Answer*).
- El conjunto de temas (*TopicWrapper*) se asocia con 0 o más temas.
- Un fichero se adjunta para una decisión, y cada decisión puede tener 0 o más ficheros adjuntos.
- Una notificación se asocia con una decisión, un proyecto y uno o más usuarios (todos los que trabajan en ese proyecto).
- Una tabla del documento PDF (*PDFTable*) se asocia con un proyecto, para conocer y generar toda la información de ese proyecto.
- Una sección del documento PDF (*PDFSection*) se compone de uno o más elementos del documento PDF (*PDFElement*), para componer la sección.
- Un documento PDF (*PDFConfiguration*) se compone de una o más secciones del documento (*PDFSection*).
- La clase *LogEntry* utiliza la clase usuario para consultar el nombre de usuario que ha realizado una acción en el sistema.

En la Figura 1.45 se puede observar este diagrama de clases de dominio.

5.2.2.2 Diseño de la base de datos

A partir de las clases de dominio persistentes y sus relaciones, se modela y diseña la base de datos para que los objetos de dominio puedan ser persistentes. Para realizar esto, se han tenido en cuenta las siguientes consideraciones a la hora de diseñar la base de datos:

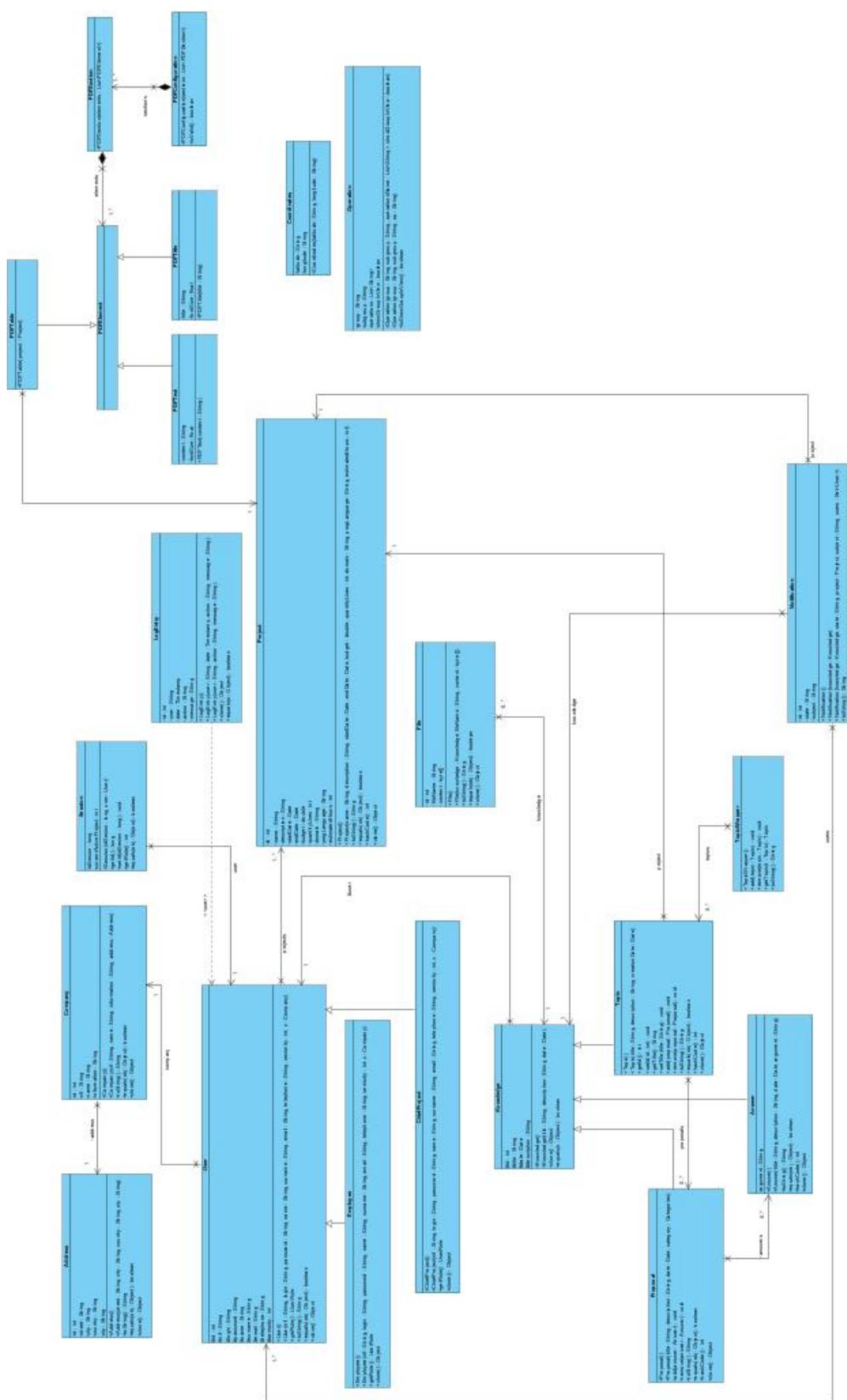


Figura 5.45: Diagrama de clases de dominio

- Para representar la jerarquía de herencia de las clases *User*, *Employee* y *ChiefProject*, se ha utilizado el patrón de persistencia **1 árbol de herencia, 1 tabla**, por lo que sólo se creará la tabla *Users*, que agrupará los atributos de todas las entidades anteriores. Sin embargo, es necesario añadir un nuevo atributo a la tabla *Users* para indicar el rol del usuario, correspondiente a cada una de las subclases. La razón de utilizar dicho patrón es agrupar en única tabla toda la jerarquía de herencia que existe entre esas clases en el modelo de dominio, pues ninguna de las clases *Employee* ni *ChiefProject* añaden nuevos atributos a la clase *User*, por lo que no se van a obtener atributos (columnas) nulas en la tabla resultante.
- En la tabla *LogEntries*, obtenida al transformar la clase *LogEntry* en una tabla, la columna "usuario" puede ser vacía, pues hay acciones que no están asociadas a ningún usuario del sistema, como, por ejemplo, iniciar o detener el servidor.
- Debido a las asociaciones n:m (o *muchos a muchos*) entre las clases de *User* y *Project*, y entre las de *Notification* y *User*, es necesario modelar tablas adicionales que permitan modelar dichas asociaciones, utilizando claves ajenas a las tablas que representan las clases que participan en esas asociaciones.
- En la tabla *NotificationsUsers*, que modela la relación *muchos a muchos* entre usuarios y notificaciones, se ha creado un *trigger* para borrar automáticamente una notificación cuando todos los usuarios a los que iba dirigida ya la han borrado. Esto se comentará más en detalle en el apartado 1.3.2.2.2.

Teniendo en cuenta dichas consideraciones, se obtiene el modelo EER (Entidad-Interrelación Extendido) de la base de datos, mostrado en la Figura 1.46.

Para terminar, cabe destacar que este modelo se ha creado utilizando la herramienta **MySQL Workbench** (ver sección ??), la cual permite generar el código SQL necesario para crear las tablas y relaciones de la base de datos a partir de ese modelo, aplicando ingeniería directa.

5.2.2.3 Funcionalidad de Acceso al sistema

Una vez se ha definido la arquitectura del sistema, se han modelado los objetos de dominio y se ha diseñado la base de datos a utilizar en el sistema, se pasa a desarrollar la funcionalidad

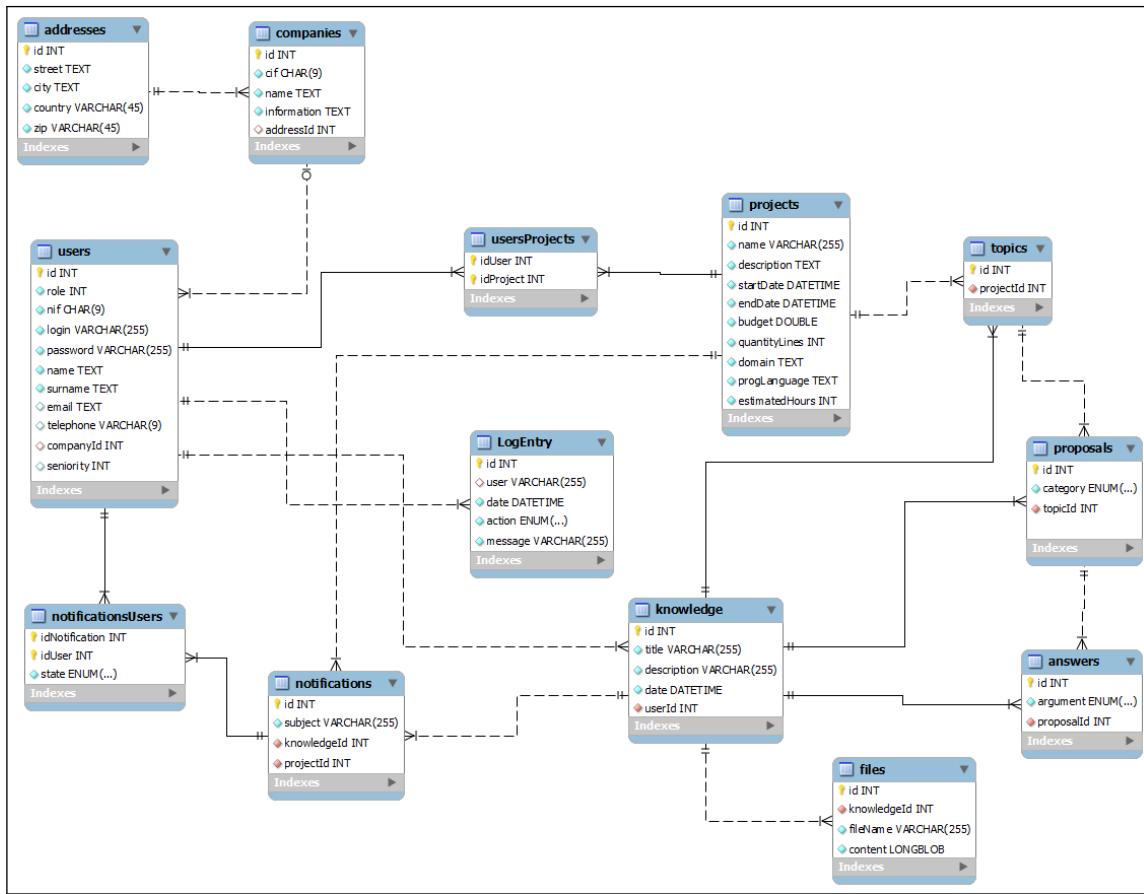


Figura 5.46: Diagrama EER de la base de datos

planificada para esta iteración, que es el acceso al sistema. Por tanto, durante esta iteración, se diseña, implementa y prueba los casos de uso que componen la funcionalidad de *Acceso al sistema*, mostrados en la Figura 1.5.

5.2.2.3.1 Diagramas de secuencia

En primer lugar, se modela el funcionamiento de los casos de uso que la componen (ver Tabla 1.5) a través de diagramas de secuencia, tanto para el subsistema cliente como para el servidor.

Login

En la Figura 1.47 se muestra el diagrama de secuencia para el caso de uso *Login* en el cliente, mientras que en la Figura 1.48 se muestra el diagrama de secuencia para el servidor.

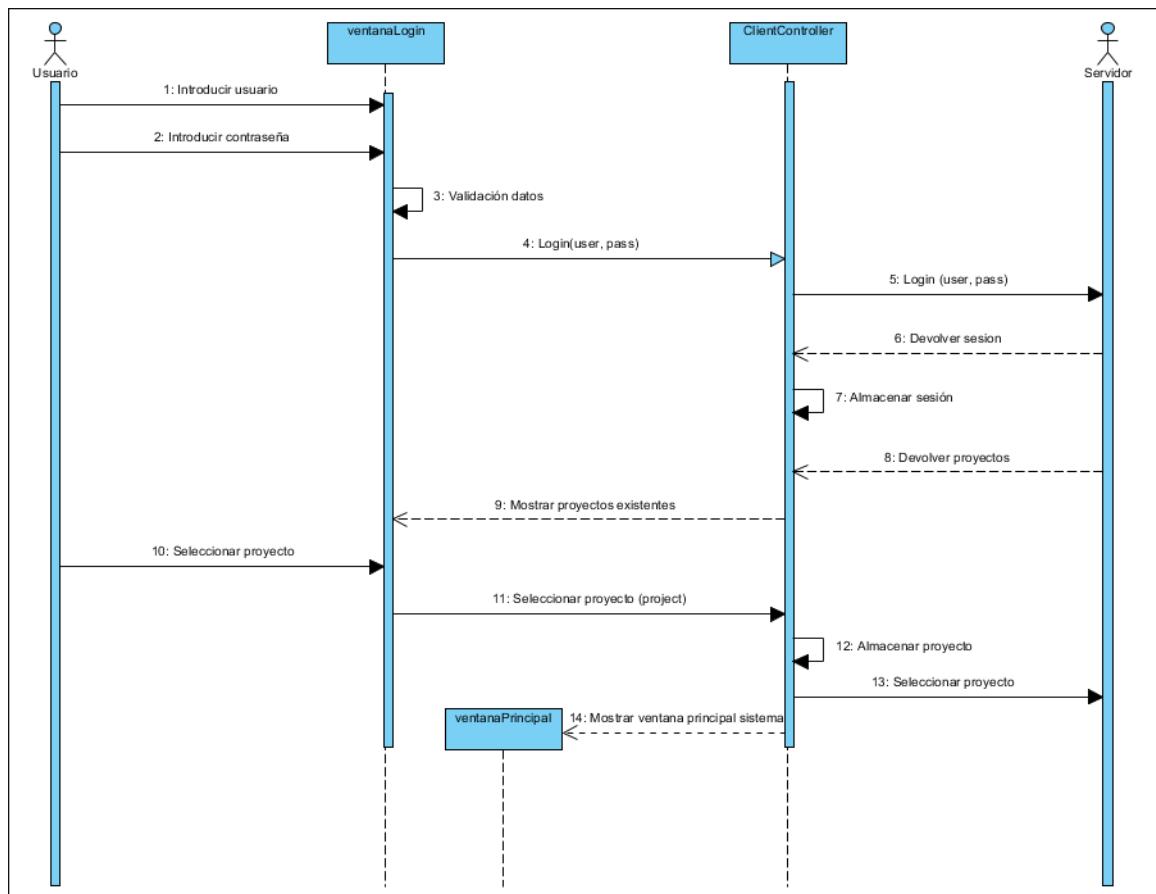


Figura 5.47: Diagrama de secuencia - Cliente - Login

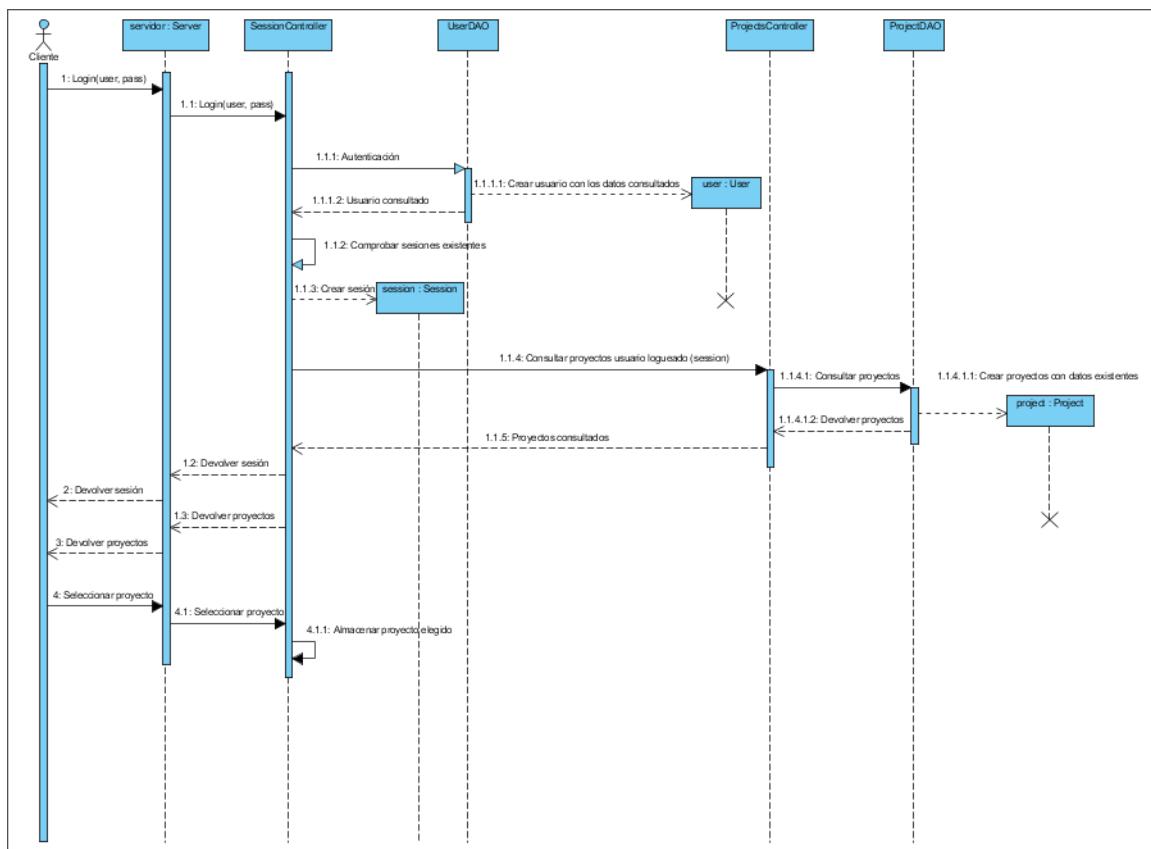


Figura 5.48: Diagrama de secuencia - Servidor - Login

Logout

En la Figura 1.49 se muestra el diagrama de secuencia para el caso de uso *Logout* en el cliente, mientras que en la Figura 1.50 se muestra el diagrama de secuencia para el servidor.

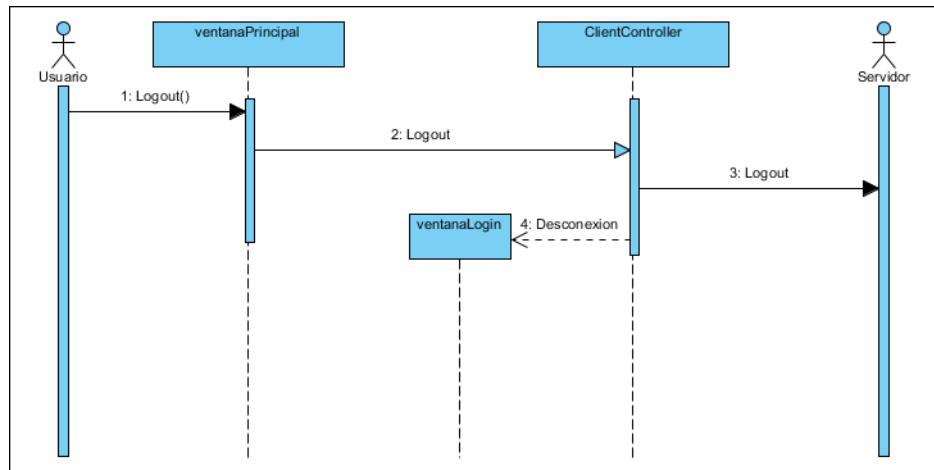


Figura 5.49: Diagrama de secuencia - Cliente - Logout

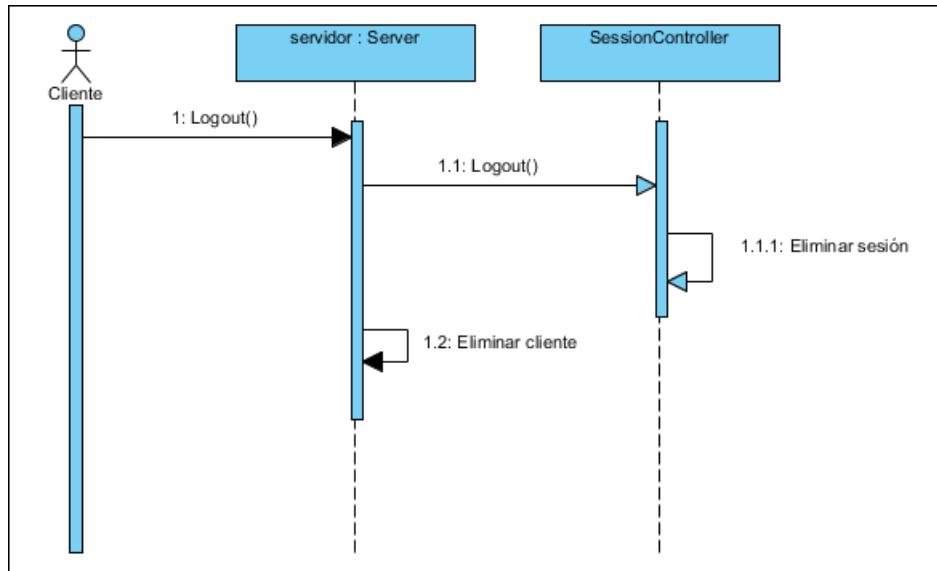


Figura 5.50: Diagrama de secuencia - Servidor - Logout

5.2.2.3.2 Diseño e Implementación de la arquitectura cliente-servidor

Antes de comenzar con la implementación de la funcionalidad del acceso al sistema, es necesario diseñar e implementar la comunicación entre los subsistemas cliente y servidor,

siguiendo la arquitectura cliente-servidor definida anteriormente, así como la comunicación entre el servidor y su base de datos.

Por tanto, a continuación se comentan los aspectos reseñables del diseño e implementación de la comunicación entre sistemas.

Comunicación entre cliente y servidor

El primer paso es hacer que las clases de dominio mostradas en la Figura 1.45 sean serializables, para poder ser exportadas y enviarse de un sistema a otro. Para ello, dichas clases implementan la interfaz *Serializable* de Java.

A continuación, se diseña e implementa la capa de comunicaciones de ambos sistemas, haciendo uso del patrón **Proxy**, responsable de establecer conexión con las clases remotas exportadas por cada uno de los subsistemas. Además, para conocer los métodos que pueden invocarse utilizando RMI, se crean interfaces para cada uno de los subsistemas, implementando la interfaz *Remote* provista por RMI.

De este modo, el proxy del servidor implementa la interfaz remota del servidor, la cual contiene todas las operaciones que el cliente puede solicitar al servidor, por lo que es un patrón **Fachada**. Así, el cliente puede invocar al proxy del servidor como si éste fuera local y estuviera en la misma máquina. De un modo análogo, se implementa el proxy y la interfaz del cliente, para que el servidor pueda enviar y notificar los resultados al cliente.

En la Figura 1.51 se muestra el diagrama de clases para esta capa de comunicaciones.

Conviene destacar que, con el fin de que el sistema funcione correctamente si alguno de los subsistemas pertenece a varias redes, al exportar los objetos remotos se recorren todas las interfaces de red para buscar una IP según el siguiente orden:

1. Si el ordenador pertenece a una red pública, se usa una IP pública.
2. Si el ordenador no pertenece a una red pública pero sí a una privada, se utiliza una IP privada.
3. Si el ordenador no está conectado a ninguna red, se emplea la IP *localhost* (127.0.0.1).

Además, para que la comunicación con los objetos remotos se establezca correctamente, no sólo es necesario indicar la IP al exportar los objetos, sino que también hace falta modificar

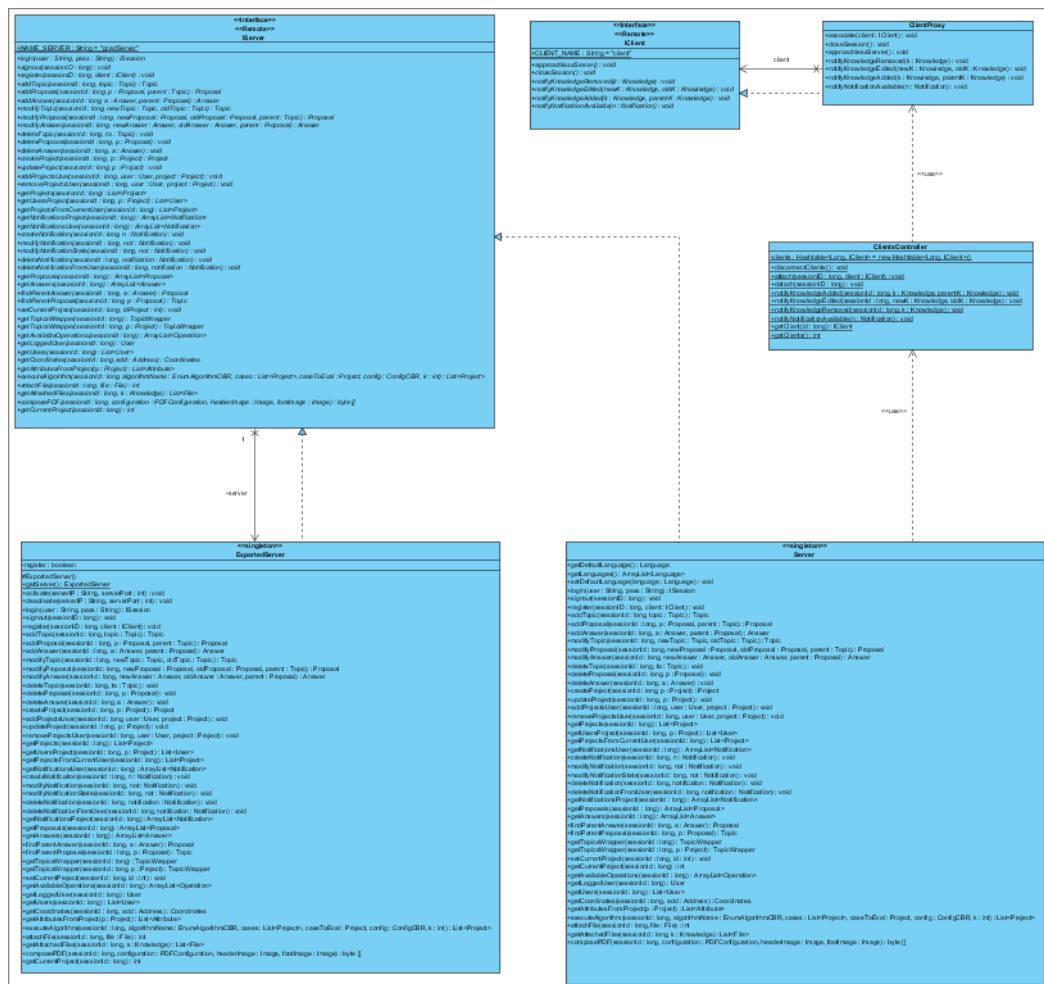


Figura 5.51: Diagrama de clases - Capa de comunicación cliente-servidor

la propiedad `java.rmi.server.hostname` de la máquina virtual de Java, que representa la IP del servidor RMI que contiene los objetos.

En el fragmento de código 1.1 se muestra como se exporta el objeto que representa al servidor y como se modifica la propiedad `java.rmi.server.hostname` de la máquina virtual de Java, para que el servidor sea accesible y sus métodos puedan ser invocados por el cliente.

```
/**  
 * Class that exports the server instance to be used by clients to execute operations from  
 * the server facade (interface)  
 */  
  
public class ExportedServer extends UnicastRemoteObject implements IServer {  
  
    ....  
  
    public void activate(String serverIP, int serverPort) throws MalformedURLException,  
        RemoteException {  
        // If the server is already exports, don't throw exception  
        try {  
            if(!register) {  
                LocateRegistry.createRegistry(serverPort);  
                register = true;  
            }  
            exportObject(this, serverPort);  
        } catch(ExportException ex) {  
            if(!ex.getMessage().toLowerCase().equals("object already exported")) {  
                throw ex;  
            }  
        }  
        try {  
            Naming.bind("rmi://" + serverIP + ":" + String.valueOf(serverPort) + "/" +  
                NAME_SERVER , this);  
        } catch(AlreadyBoundException ex) {  
            Naming.rebind("rmi://" + serverIP + ":" + String.valueOf(serverPort) + "/" +  
                NAME_SERVER, this);  
        }  
    }  
  
    ....  
  
    public class ServerController {  
        public void startServer(ServerConfiguration configuration) throws RemoteException,  
            MalformedURLException, SQLException {  
            serverIP = CommunicationsUtilities.getHostIP();  
  
            // Indicate to RMI that it have to use the given IP as IP of this host in remote
```

```
communications.  
// This instruction is necessary because if the computer belongs to more than one  
// network, RMI may take a private IP as the host IP  
// and incoming communications won't work  
System.setProperty("java.rmi.server.hostname", serverIP);  
  
....
```

Listado 5.1: Proceso para exportar un objeto utilizando RMI

Como se puede observar en el fragmento anterior, para exportar el objeto del servidor se utiliza el método *bind* de la clase *Naming* de RMI, indicando la IP, el puerto y el nombre del objeto exportado. De este modo, en el cliente se utiliza el método *lookup* de la clase *Naming* para localizar ese objeto exportado, utilizando la IP, el puerto y el nombre con el que se exportó. En el fragmento de código 1.2 se muestra un ejemplo de cómo realizar esta acción.

```
public class ProxyServer implements IServer {  
  
    ....  
  
    public void connectServer(String ip, int port) throws MalformedURLException,  
        RemoteException, NotBoundException {  
        String url;  
  
        url = "rmi://" + ip + ":" + String.valueOf(port) + "/" + NAME_SERVER;  
        server = (IServer)Naming.lookup(url);  
    }  
  
    ....
```

Listado 5.2: Proceso para localizar un objeto remoto utilizando RMI

Comunicación entre servidor y base de datos

En el paquete de comunicaciones también se ha utilizado el patrón **Observador** para crear un gestor de conexiones de bases de datos. De este modo, se consigue extensibilidad en las comunicaciones con bases de datos, pues haciendo uso de este observador, se podrían añadir más de una base de datos y este observador sería el encargado de enviar las peticiones a todas ellas.

Para ello, se ha creado una interfaz que agrupa las operaciones típicas de una base de

datos (*CRUD - Create, Read, Update, Delete*), implementada por las diferentes conexiones a bases de datos que pudieran existir. También se ha creado un gestor de conexiones de bases de datos (el observador), que es el encargado de enviar las peticiones a cada una de esas conexiones, utilizando su interfaz. En la Figura 1.52 se puede observar el diagrama de clases que representa este observador.

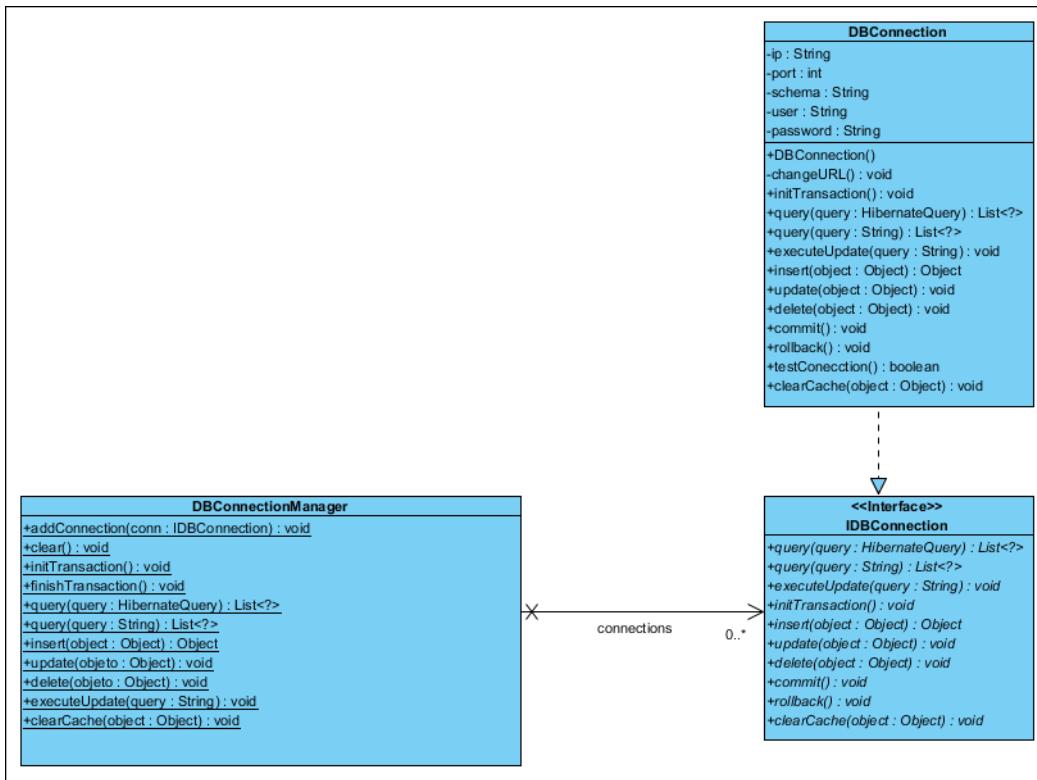


Figura 5.52: Diagrama de clases - Capa de comunicación para bases de datos

En el caso del sistema a desarrollar, sólo existe una base de datos, cuyas operaciones *CRUD* y conexión con la base de datos MySQL es gestionada por **Hibernate**. Cabe señalar algunos problemas que se detectaron al trabajar con RMI e Hibernate:

- Hay que clonar las referencias que devuelve Hibernate, para que sean serializables por RMI.
- Al actualizar un objeto, hay que buscarlo primero en la base de datos, pues la referencia que llega por RMI es diferente a la que utiliza Hibernate.
- Hay que limpiar las cachés que mantiene Hibernate tras hacer una consulta a la base de datos para evitar problemas de referencias.

De un modo similar al anterior, se ha utilizado también el patrón **Observador** para gestionar el *log*, ya que el servidor debe registrar todas las operaciones realizadas por los usuarios. Para ello, se ha creado un gestor de log (el observador) que utiliza interfaces para poder enviar las entradas del log que hay que registrar tanto a la base de datos como a la interfaz gráfica del servidor. De este modo, utilizando este patrón y las interfaces, las entradas de log se crean automáticamente tanto en la base de datos como en la interfaz gráfica, siendo independiente de su implementación. En la Figura 1.53 se puede observar el diagrama de clases que representa este observador.

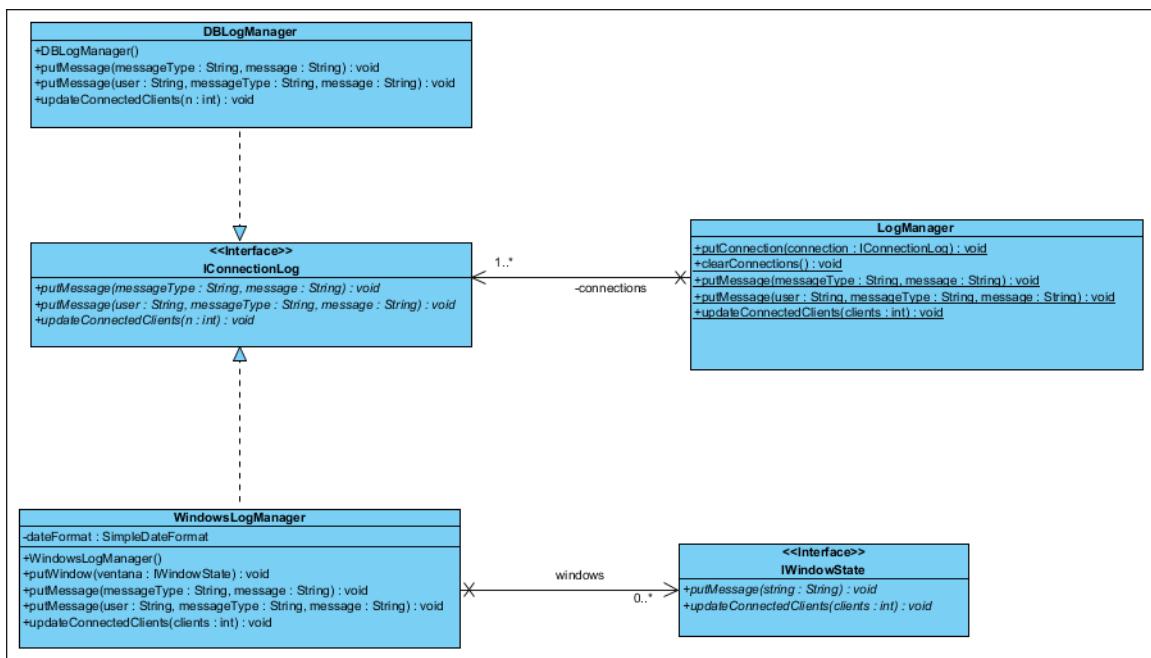


Figura 5.53: Diagrama de clases - Capa de comunicación para gestionar el log

5.2.2.3.3 Diseño e implementación de acceso al sistema

Una vez implementada la arquitectura cliente-servidor, y teniendo ya la comunicación entre sistemas y con la base de datos, se siguen los diagramas de secuencia mostrados en el apartado 1.2.2.3.1 para diseñar e implementar esos casos de uso de la funcionalidad de acceso al sistema en ambos sistemas. A continuación se muestran algunos aspectos a destacar al implementar esta funcionalidad.

Servidor

La clase *Usuario* es abstracta ya que no se van a instanciar objetos directamente de esa clase, sino de una de sus subclases, que representan los diferentes usuarios que pueden existir en el sistema, según la especificación de requisitos (ver sección 1.1.1.1), como se observa en el diagrama de clases de la Figura 1.45.

Además, esta solución propuesta facilita la extensibilidad futura del sistema, ya que el rol de cada usuario se define mediante una enumeración, de tal modo que cada subclase de la clase *Usuario* redefinirá el método abstracto *getRol()*, devolviendo el valor correspondiente a su rol en la enumeración. Por tanto, para añadir un nuevo rol de usuario, basta con añadir una nueva clase que herede de la superclase y un nuevo rol a la enumeración.

Por otra parte, para aumentar la seguridad del sistema, se decidió encriptar la contraseña de los usuarios, utilizando para ello el algoritmo *SHA1*. Algunas de las razones para utilizar una encriptación por código y no delegar esta responsabilidad en el sistema gestor de base de datos, son las siguientes:

1. Si se quiere cambiar la encriptación a una más segura, no haría falta más que cambiar el método que encripta la contraseña.
2. Puede que otros SGBD que no sean MySQL no tengan encriptación incorporada.
3. El número de encriptaciones que incorpora un SGBD es limitado.

Tras acceder al sistema, la interfaz gráfica de usuario del subsistema cliente se adaptará a las operaciones del usuario logueado, según su rol. Para ello, el gestor de sesiones utiliza archivos XML donde están definidos los perfiles existentes en el sistema y las operaciones que puede realizar este perfil. Así, las operaciones se han dividido en categorías, de tal modo que los elementos de los menús de la aplicación, submenús, *toolbar*, etc, se generarán de manera automática tras acceder al sistema y conocer las operaciones que puede realizar un cierto rol.

De este modo, además de permitir una interfaz de usuario totalmente flexible y adaptable, se facilita la extensibilidad, ya que, además de los cambios mencionados anteriormente, solamente habría que añadir el nuevo perfil y sus operaciones a los ficheros XML alojados en el servidor. Para gestionar estos archivos XML, se ha utilizado el patrón **Agente** para encapsular en un clase el acceso a los ficheros XML y todas las operaciones a realizar con

ellos, como consultas al archivo XML, gestión de XPath, etc, utilizando JDOM y Jaxen (ver sección ??).

En la Figura 1.54 se muestra el diagrama de clases para el gestor de sesiones.

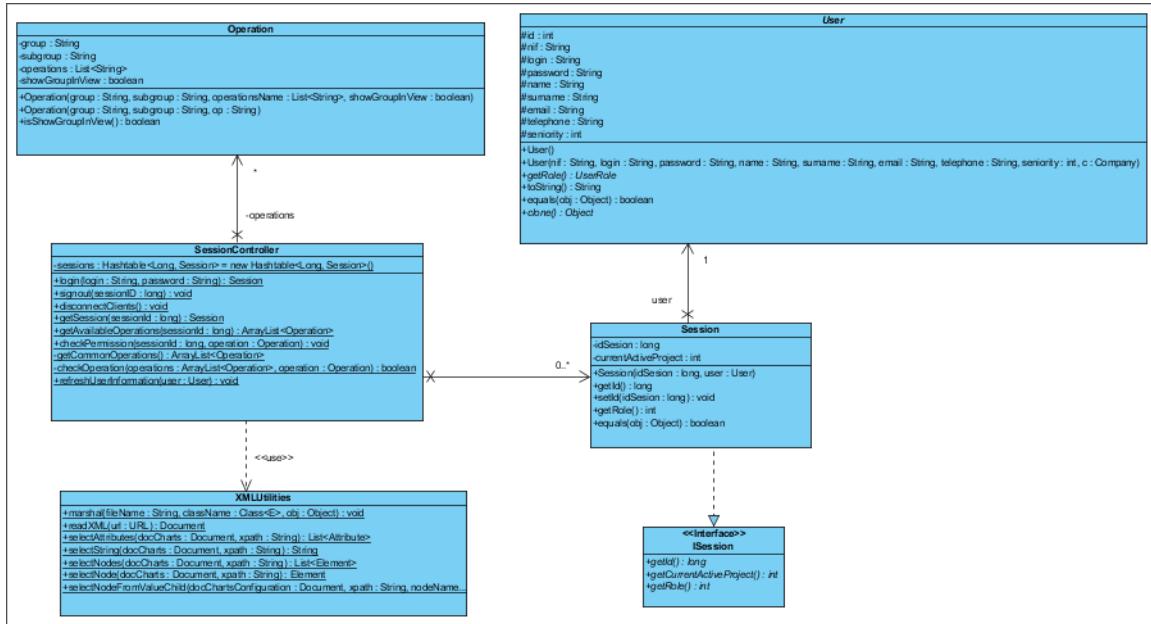


Figura 5.54: Diagrama de clases - Gestor de sesiones

Cliente

Cabe destacar que la interfaz gráfica de usuario del subsistema cliente se ha diseñado e implementado siguiendo una estructura de composición de clases, de tal modo que la ventana (*o frame*) principal se compone de paneles, y éstos, a su vez, de otros paneles y elementos gráficos. De este modo se consigue una interfaz gráfica ampliamente extensible y adaptable.

Gracias a esta estructura, según el rol del usuario que accede al sistema, los menús y diferentes elementos de la interfaz gráfica son fácilmente adaptables a las operaciones que ese usuario pueda realizar en el sistema. Además, para tener en cuenta el multi-idioma, la interfaz se adaptará al idioma elegido y todos sus menús, etiquetas, texto, etc, se mostrarán en el idioma preferido. Este aspecto de internacionalización se detallará en la fase de Construcción.

Para terminar, en lo que respecta a esta funcionalidad de acceso al sistema en el cliente, se ha diseñado e implementado una ventana para que el usuario pueda introducir sus datos, validarlos y enviarlos al servidor, accediendo al sistema si todo es correcto. En este aspecto

cabe destacar el uso del método *invokeLater* de la librería Swing (ver sección ??) que permite manipular la interfaz gráfica mientras se realiza una tarea de larga duración en un hilo separado. Por tanto, la acción de validar los datos para acceder al sistema y enviarlos al servidor, se realiza en un hilo separado, mientras que en la interfaz gráfica se muestra un panel con un *spinner* de carga (ver Figura 1.55) animado, para proporcionar al usuario un *feedback* visual y saber que la operación se está realizando y que debe esperar.

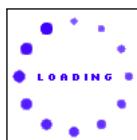


Figura 5.55: Ejemplo de *spinner* de carga

En el fragmento de código 1.3 se muestra como se ha utilizado el método *invokeLater* para crear un nuevo hilo para realizar el acceso al sistema, mostrando el panel con el *spinner* animado.

```
....  
  
this.glassPane = new InfiniteProgressPanel(ApplicationInternationalization.getString("glassLogin"));  
getMainFrame().setGlassPane(glassPane);  
  
....  
  
@Action  
public void loginAction() {  
    ....  
  
    // Invoke a new thread in order to show the panel with the loading  
    // spinner  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() {  
            glassPane.setColorB(241);  
            glassPane.start();  
            Thread performer = new Thread(new Runnable() {  
                public void run() {  
                    perform(txtUserName.getText(), new String(txtPass.getPassword()), ip,  
                           port);  
                }  
            }, "Performer");  
            performer.start();  
        }  
    });  
}  
....
```

```
        }
    });

private void perform(String user, String pass, String ip, String port) {
    // Login
    ClientController.getInstance().initClient(ip, port, user, pass);
    glassPane.stop();
    getMainFrame().setEnabled(true);
    getMainFrame().requestFocus();

    ....
}
```

Listado 5.3: Fragmento de código para acceder al sistema en el cliente

5.2.2.3.4 Pruebas

5.3 Fase de construcción

En esta fase intervienen las disciplinas de Diseño, Implementación y Pruebas con mayor influencia, como se muestra en la Figura 1.56. Esta fase, formada por 5 iteraciones, es la fase de más duración del desarrollo del producto software.

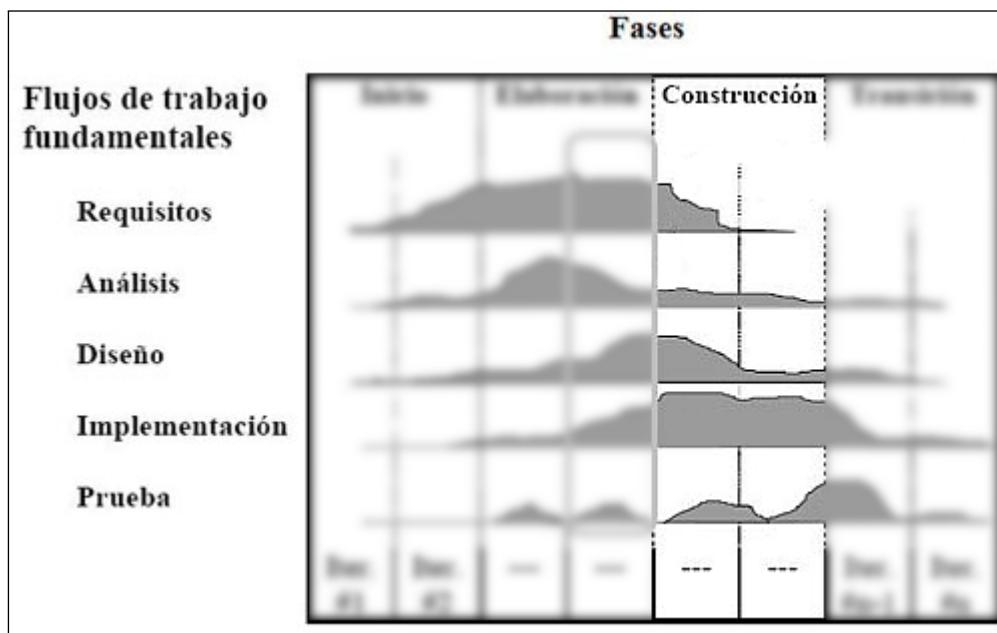


Figura 5.56: Fase de construcción en el PUD

Durante las diferentes iteraciones, se irán diseñando e implementando las funcionalidades planificadas para cada iteración (ver Tabla 1.7), obteniendo incrementos en la funcionalidad del sistema hasta llegar a tener el producto totalmente implementado y probado.

5.3.1 Primera iteración

Siguiendo los casos de uso de la funcionalidad de *Visualizar decisiones* (ver Figura 1.22), así como su análisis realizado, se abordan las siguientes tareas en esta primera iteración de la fase de Construcción:

- Diseño de la funcionalidad relativa a la visualización de información en el sistema, como son las decisiones y toda su información asociada.
- Implementación de dicha funcionalidad.
- Diseño e implementación de pruebas unitarias y funcionales relativas a la visualización de información.

5.3.1.1 Funcionalidad de *Visualizar información*

5.3.1.1.1 Diagramas de secuencia

En los siguientes apartados se muestran los diagramas de secuencia para el cliente y servidor de los casos de uso que componen esta funcionalidad. Dichos diagramas se modelan siguiendo la descripción de los casos de uso realizada en el apartado 1.2.1.4.

Consultar decisiones

En la Figura 1.57 se muestra el diagrama de secuencia para el caso de uso *Consultar decisiones* en el cliente, mientras que en la Figura 1.58 se muestra el diagrama de secuencia para el servidor.

Visualizar compañía

En la Figura 1.59 se muestra el diagrama de secuencia para el caso de uso *Consultar compañía* en el cliente, mientras que en la Figura 1.60 se muestra el diagrama de secuencia para el servidor.

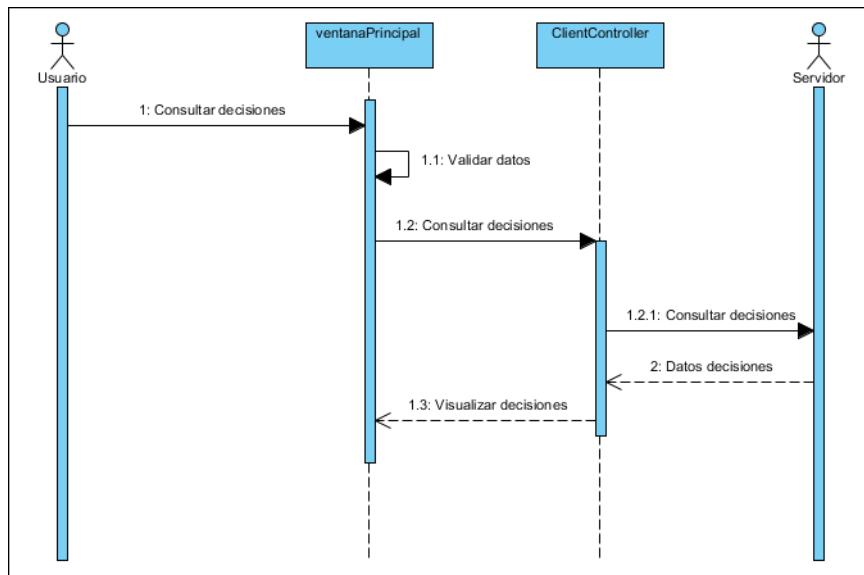


Figura 5.57: Diagrama de secuencia - Cliente - Consultar decisiones

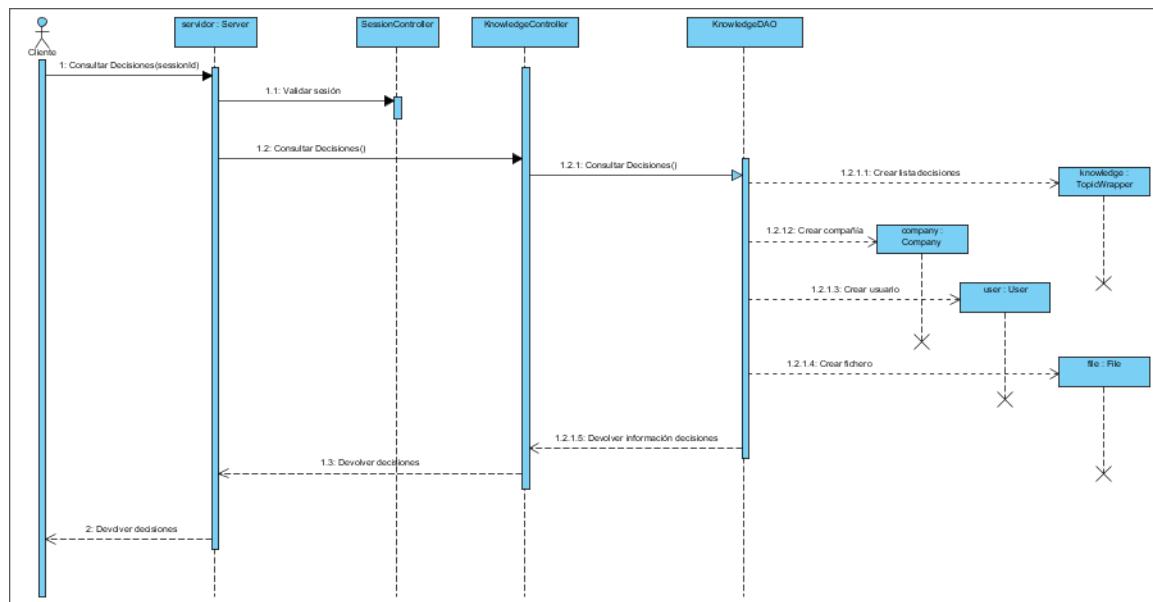


Figura 5.58: Diagrama de secuencia - Servidor - Consultar decisiones

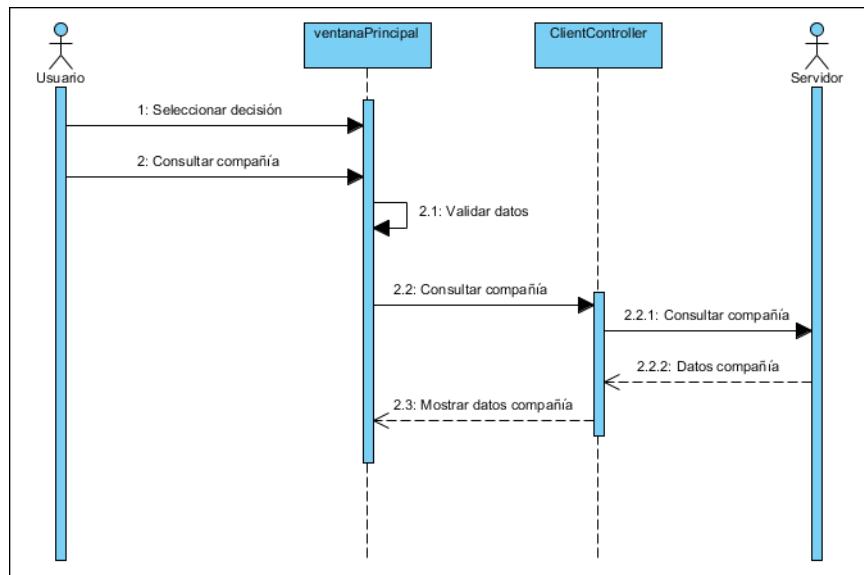


Figura 5.59: Diagrama de secuencia - Cliente - Consultar compañía

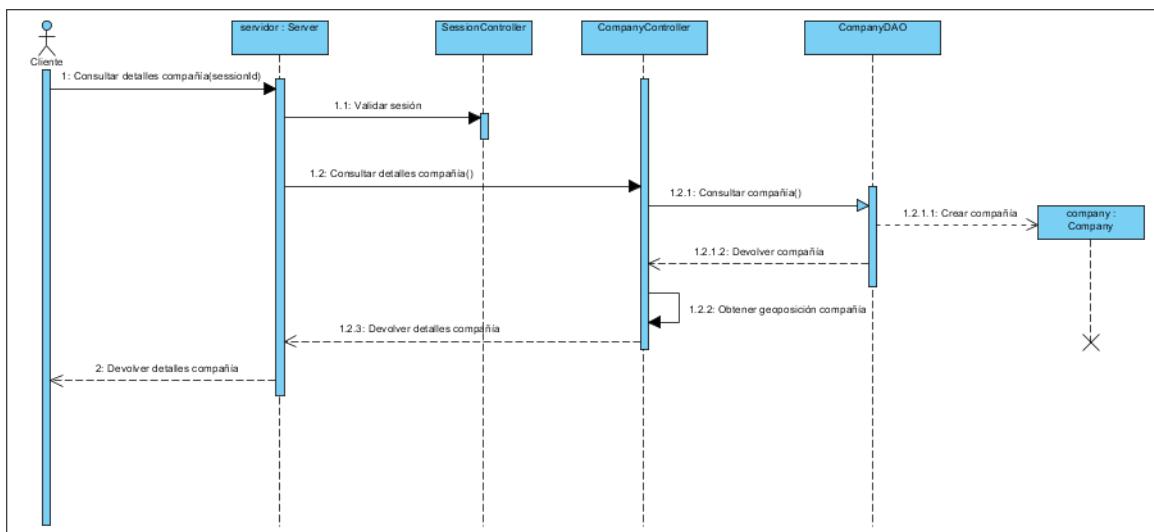


Figura 5.60: Diagrama de secuencia - Servidor - Consultar compañía

Descargar ficheros adjuntos

En la Figura 1.61 se muestra el diagrama de secuencia para el caso de uso *Descargar ficheros adjuntos* en el cliente, mientras que en la Figura 1.62 se muestra el diagrama de secuencia para el servidor.

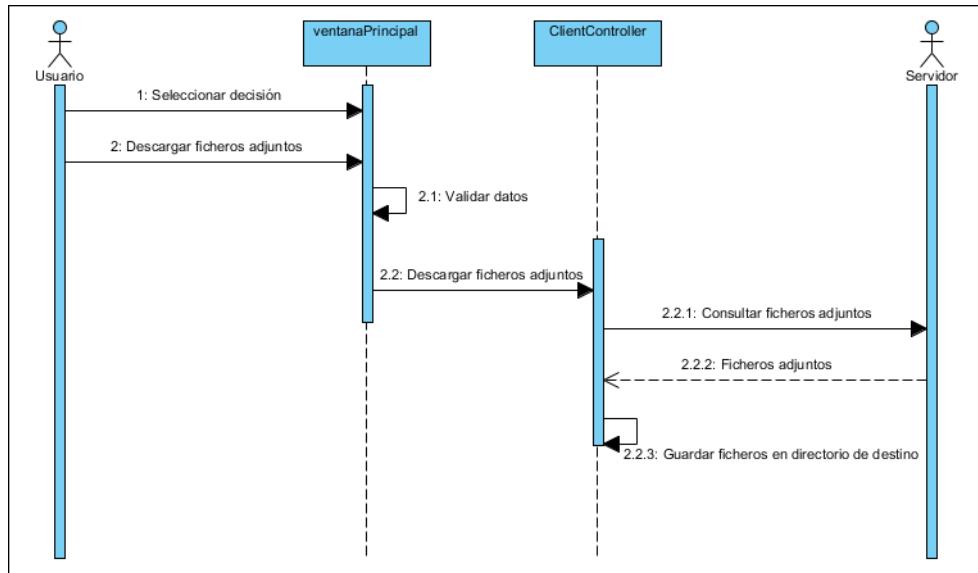


Figura 5.61: Diagrama de secuencia - Cliente - Descargar ficheros adjuntos

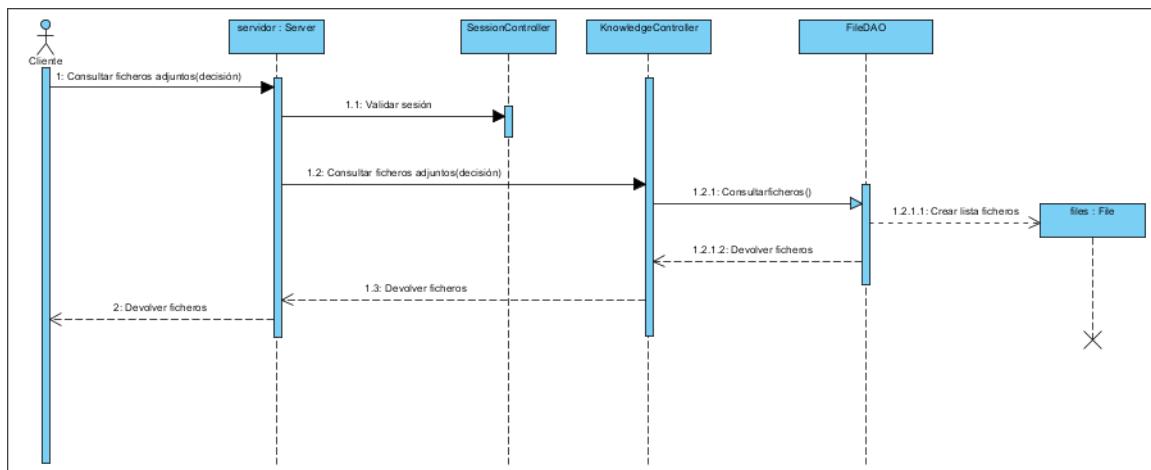


Figura 5.62: Diagrama de secuencia - Servidor - Descargar ficheros adjuntos

5.3.1.1.2 Diseño e implementación

Siguiendo los diagramas de secuencia, se realiza el diseño de los casos de uso utilizando un diagrama de clases de diseño y se procede a la implementación de dichos casos de uso, para

obtener las clases Java que dan soporte a estas funcionalidades. A continuación se destacan algunos aspectos tenidos en cuenta al diseñar e implementar estos casos de uso.

Cabe destacar que para mantener la seguridad en el sistema, siempre se comprobará en el servidor la sesión del usuario antes de realizar cualquier operación, tanto para comprobar que esa sesión existe como para validar si se tiene permiso para ejecutar dicha acción en el sistema. Por ello, los gestores (o controladores) de ésta y del resto de funcionalidades del sistema tendrán una relación con el gestor de sesiones, descrito en la segunda iteración de la fase anterior (ver Figura 1.54).

Servidor

Como se puede apreciar en el diagrama de clases de la Figura 1.63, las decisiones (o conocimiento) de un proyecto siguen una jerarquía de herencia, existiendo la clase abstracta *Knowledge* de las que heredan otras tres clases: *Topic*, *Proposal* y *Answer*. Se ha decidido diseñar esta herencia para facilitar la futura extensibilidad del sistema, ya que se podría incluir nuevo tipo de conocimiento creando una nueva subclase. Además, para mejorar la organización de las decisiones, estas tres subclases están asociadas de una manera jerárquica, del siguiente modo:

- **Topic** (o Tema): esta clase, que representa un Tema, está compuesta por un conjunto de decisiones del tipo *Proposal* dentro de un proyecto.
- **Proposal** (o Propuesta): esta clase, que representa una Propuesta, esta compuesta por un conjunto de decisiones del tipo *Answer*, y están agrupadas bajo un *Topic* común.
- **Answer** (o Respuesta): esta clase, que representa una Respuesta están agrupadas bajo una *Proposal* común y ya no pueden tener más hijos.

De este modo, como se muestra en la Figura 1.63, para cada proyecto, existirán una serie de *Topics*, donde cada *Topic* estará compuesto por una serie de *Proposals* y éstas, a su vez, estarán compuestas por un conjunto de *Answers*. Por tanto, se sigue una estructura de composición y jerarquía de conocimiento. Un ejemplo de esta estructura jerárquica puede encontrarse en los foros de debate presentes en la Web (ver Figura 1.64).

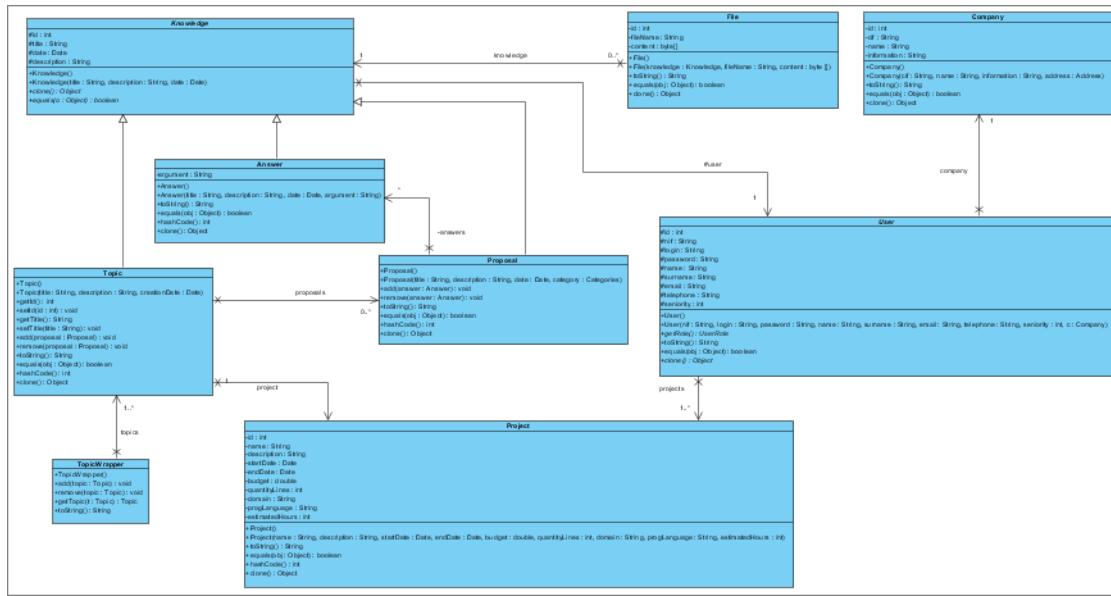


Figura 5.63: Diagrama de clases - Jerarquía de decisiones

All Documents

New Topic

Main Topic
Dec 2, 2009 7:53 PM | Responses: 4

Response To main
Dec 2, 2009 7:53 PM

another response
Dec 2, 2009 7:54 PM

Response to Response
Dec 2, 2009 7:54 PM

yet another response
Dec 2, 2009 7:54 PM

Steve Castledine (repetido para cada respuesta)

Previous 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 ... Next

Figura 5.64: Ejemplo de jerarquía de decisiones en foros de debates

Además, existe la clase *TopicsWrapper*, que engloba todos los *Topic* de un proyecto, facilitando su recuperación y necesario además para exportar todo el conocimiento a un fichero XML, como se detallará en el posterior apartado 1.3.5.1.

Siguiendo esta estructura descrita, en lo que respecta al caso de uso *Consultar Decisiones*, se recupera y visualiza esta composición jerárquica de decisiones y toda su información asociada, como es el usuario, la compañía y los posibles ficheros adjuntos (ver diagrama de la Figura 1.63). Sin embargo, al utilizar el framework de persistencia *Hibernate*, no hace falta hacer varias consultas para recuperar toda esta información ya que, al existir asociaciones entre los objetos, Hibernate se encargará de recuperar todos ellos al consultar las decisiones existentes en un proyecto.

En lo que concierne al caso de uso *Consultar Compañía*, que sirve para consultar y mostrar los detalles de una compañía, cabe destacar como aspecto de implementación que se ha utilizado un servicio Web para obtener la posición geográfica de la dirección de la compañía, para poder mostrarla posteriormente en un mapa, en el subsistema cliente. Para ello, como se vió en el marco tecnológico de la sección ??, se ha utilizado el servicio web **Yahoo! PlaceFinder** para obtener las coordenadas geográficas a partir de una dirección.

En un primer momento se pensó en utilizar el API de geolocalización de Google para realizar esta tarea. Sin embargo, atendiendo a las condiciones de uso de Google [?], esta API debe usarse en conjunto con el API de *Google Maps*, que a su vez sólo se puede utilizar en aplicaciones Web. Por tanto, se descartó esta opción para no violar las condiciones de uso de Google y se buscó una alternativa, encontrando el servicio web de Yahoo!, cuyos términos de uso permitían utilizarlo en el sistema [?].

Para invocar el servicio *Yahoo! PlaceFinder*, se necesita una URL del tipo:

```
http://where.yahooapis.com/geocode?[parameters] &appid=APPID
```

donde *APPID* es un código proporcionado para desarrolladores para poder utilizar su API, y *parameters* es la dirección a buscar.

Un ejemplo de URL sería:

```
http://where.yahooapis.com/geocode?location=701+First+Sunnyvale&APPID
```

Al invocar el servicio web, éste devuelve una respuesta en formato XML indicando una serie de parámetros, como se muestra en el Listado 1.4. Entre ellos, interesa el código de error

y las coordenadas geográficas de la dirección, formadas por longitud y latitud.

```
<?xml version="1.0" encoding="UTF-8"?>
<ResultSet version="1.0">
    <Error>0</Error>
    <ErrorMessage>No error</ErrorMessage>
    <Locale>us_US</Locale>
    <Quality>87</Quality>
    <Found>1</Found>
    <Result>
        <quality>87</quality>
        <latitude>37.416275</latitude>
        <longitude>-122.025092</longitude>
        <offsetlat>37.416397</offsetlat>
        <offsetlon>-122.025055</offsetlon>
        <radius>500</radius>
        <name>
        </name>
        <line1>701 1st Ave</line1>
        <line2>Sunnyvale, CA 94089-1019</line2>
        <line3>
        </line3>
        <line4>United States</line4>
        <house>701</house>
        <street>1st Ave</street>
        <xstreet>
        </xstreet>
        <unittype>
        </unittype>
        <unit>
        </unit>
        <postal>94089-1019</postal>
        <neighborhood>
        </neighborhood>
        <city>Sunnyvale</city>
        <county>Santa Clara County</county>
        <state>California</state>
        <country>United States</country>
        <countrycode>US</countrycode>
        <statecode>CA</statecode>
        <countycode>
        </countycode>
        <uzip>94089</uzip>
        <hash>DDAD1896CC0CDC41</hash>
        <woeid>12797150</woeid>
        <woetype>11</woetype>
    </Result>
</ResultSet>
```

Listado 5.4: Respuesta del servicio Web Yahoo! PlaceFinder

En el Listado 1.5 se muestra un fragmento de código utilizado para invocar el servicio web desde Java y obtener su respuesta XML, que será tratada con JDOM para extraer los datos necesarios (las coordenadas) para poder mostrar en un mapa la posición exacta de la compañía.

```
/**  
 * Class used to obtain geographic coordinates from an address. To do so, uses the Web  
 * Service "Yahoo! PlaceFinder"  
 */  
public class GeoCoder {  
  
    // Yahoo! application ID  
    private static final String APPID = "NzaqCw38";  
    // Yahoo! Web Service base url  
    private static final String BASE_URL = "http://where.yahooapis.com/geocode";  
  
    public static Coordinates getGeoCoordinates(Address address) throws  
        NonExistentAddressException, WSResponseException, IOException, JDOMException {  
        URL url;  
        Coordinates coor = null;  
        StringBuffer ad = new StringBuffer();  
        ad.append(address.getStreet() != null ? address.getStreet().replace(" ", "+") +  
            "+" : "");  
        ad.append(address.getCity() != null ? address.getCity().replace(" ", "+") +  
            "+" : "");  
        ad.append(address.getZip() != null ? address.getZip() + "+" : "");  
        ad.append(address.getCountry() != null ? address.getCountry().replace(" ", "+") + "+"  
            : "");  
        if (ad.toString().endsWith("+"))  
            ad = ad.replace(ad.length() - 1, ad.length(), "");  
  
        String request = BASE_URL + "?q=" + ad.toString() + "&appid=" + APPID;  
        url = new URL(request);  
        // Connect and get response stream from Web Service  
        InputStream in = url.openStream();  
        // The response is an XML  
        SAXBuilder builder = new SAXBuilder();  
        // Uses JDOM and XPath in order to parser Xml  
        Document doc;  
        doc = builder.build(in);  
        // Get values from XML using XPath  
        String status = ((Element) XPath.selectSingleNode(doc, "/ResultSet/Error")).
```

```

getContent(0).getValue();
if (!status.equals("0"))
    throw new WSResponseException();

String found = ((Element) XPath.selectSingleNode(doc, "/ResultSet/Found")).getContent
(0).getValue();
if (found.equals("0"))
    throw new NonExistentAddressException(AppInternationalization.
getString("AddressNotFound_Exception"));

String latitude = ((Element) XPath.selectSingleNode(doc, "/ResultSet/Result/latitude"
)).getContent(0).getValue();
String longitude = ((Element) XPath.selectSingleNode(doc, "/ResultSet/Result/
longitude")).getContent(0).getValue();
in.close();
coor = new Coordinates(latitude, longitude);
return coor;
}
}

```

Listado 5.5: Invocación del servicio Web Yahoo! PlaceFinder

Cliente

En el subsistema cliente, cuando se consultan las decisiones, éstas se muestran de manera jerárquica (en árbol) y en forma de grafo, utilizando **JUNG** (ver sección ??). Se ha decidido utilizar ambos tipos de visualización para ofrecer una mayor flexibilidad e información al usuario, ya que gracias a la representación jerárquica, se puede apreciar toda la estructura de decisiones de un simple vistazo, mientras que con el grafo, se pueden observar cómo están asociadas dichas decisiones, mostrándose de manera mas detallada. Además de mostrar las decisiones, también se muestra toda su información asociada, utilizando paneles expandibles, para flexibilizar y personalizar la información que en cada momento el usuario quiere mostrar.

Un tipo de información que se muestra para cada decisión son los archivos adjuntos que tiene, que pueden ser descargados por el usuario, guardando una copia local que envía el sistema servidor al hacer la petición.

Como se comentó en la última iteración de la fase de elaboración, la interfaz del cliente sigue una jerarquía de composición de clases para mostrar la información y construir su interfaz gráfica. En el diagrama de la Figura 1.65 se puede apreciar esta organización de clases de la interfaz gráfica.

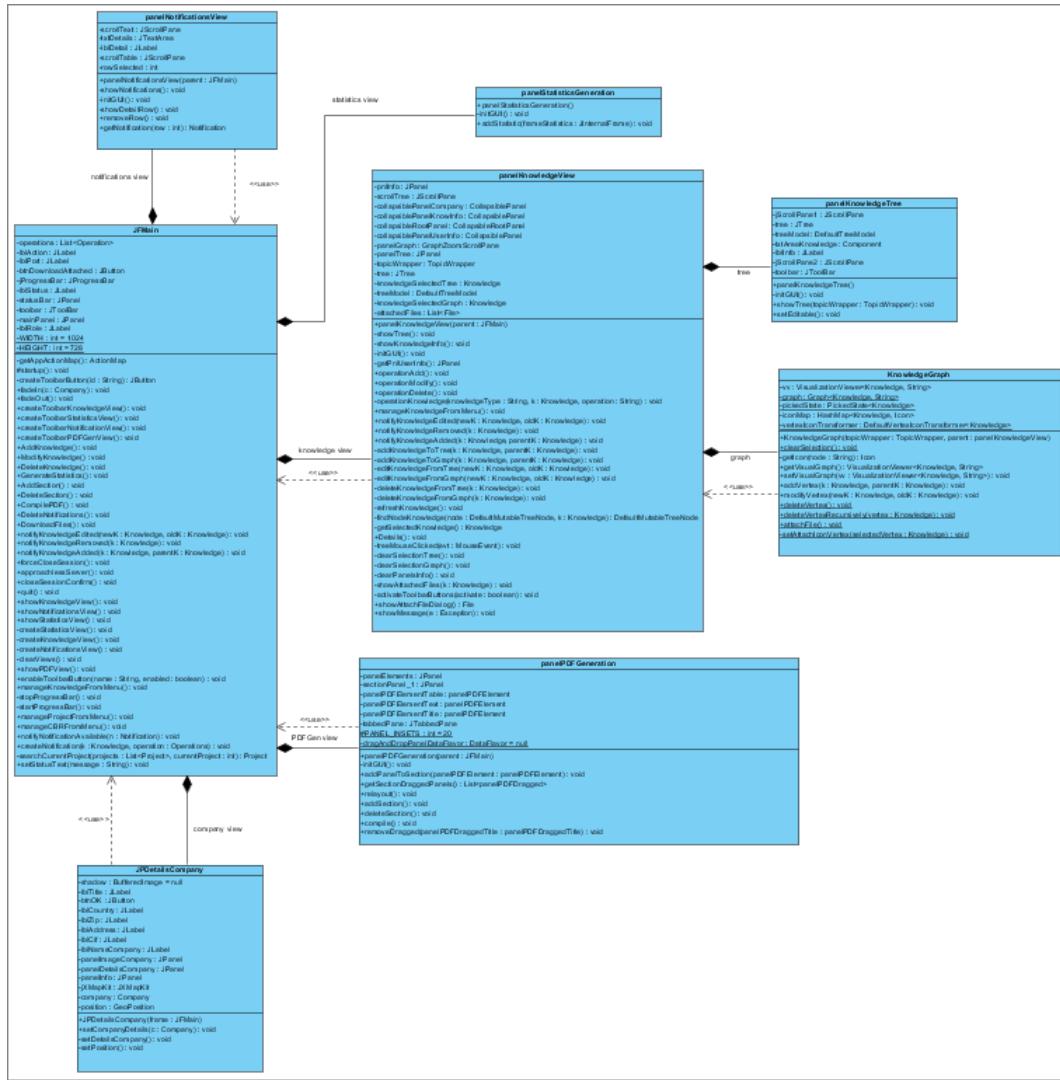


Figura 5.65: Diagrama de clases - Composición de clases de interfaz gráfica

Para el resto de funcionalidades del sistema, el diseño de la interfaz sigue una estructura similar a la mostrada, por lo que no se mostrarán sus diagramas de clases si no hay algo que destacar, para no extender excesivamente la longitud del presente capítulo.

Para terminar, en lo que se refiere al caso de uso de *Consultar compañía*, se ha decidido mostrar en un mapa la posición geográfica (además de otros datos) de la compañía consultada, para poder conocer su información exacta y real. Para ello se han utilizado los mapas proporcionados por **OpenStreetMaps** (ver sección ??), ya que, como se comentó anteriormente, debido a los términos de uso de Google, los mapas proporcionados por *Google Maps* sólo pueden utilizarse en aplicaciones Web, por lo que se decidió optar por esta alternativa.

Para mostrar los mapas proporcionados por *OpenStreetMaps*, se ha utilizado un contenedor gráfico llamado **JXMapKit**, proporcionado por la librería *Swingx*, el cuál puede ser configurado para proporcionarle un proveedor de mapas para visualizar. Además, una vez calculadas las coordenadas geográficas en el sistema servidor, este elemento permite añadir un marcador, llamado *WayPointer* para marcar exactamente esas coordenadas, correspondientes a la posición real de la compañía. En el fragmento de código 1.6 se muestra esta implementación y uso de *JXMapKit*.

```
JXMapKit jXMapKit = new JXMapKit();
// Configure maps provider
jXMapKit.setDefaultProvider(org.jdesktop.swingx.JXMapKit.DefaultProviders.OpenStreetMaps);
jXMapKit.setAutoscrolls(true);
jXMapKit.setZoomButtonsVisible(false);

.....
// Get the coordinates from server and set the wayPointer in the map
Coordinates coor;
coor = GeoCoder.getGeoCoordinates(company.getAddress());
double latitude = Double.parseDouble(coor.getLatitude());
double longitude = Double.parseDouble(coor.getLongitude());
position = new GeoPosition(latitude, longitude);

jXMapKit.setAddressLocation(position);
Set<Waypoint> waypoints = new HashSet<Waypoint>();
waypoints.add(new Waypoint(latitude, longitude));

WaypointPainter<?> painter = new WaypointPainter();
painter.setWaypoints(waypoints);
```

```
jXMapKit.getMainMap().setOverlayPainter(painter);  
jXMapKit.getMainMap().setZoom(2);  
jXMapKit.setAddressLocationShown(true);
```

Listado 5.6: Fragmento de código para mostrar mapas geoposicionados

5.3.1.1.3 Pruebas

5.3.2 Segunda iteración

Siguiendo los casos de uso de la funcionalidad de *Gestión de decisiones* (ver Figura 1.24) y *Gestión de notificaciones* (ver Figura ??), así como el análisis realizado, se abordan las siguientes tareas en esta iteración:

- Diseño de la funcionalidad relativa a la gestión de decisiones y alertas del sistema.
- Implementación de dichas funcionalidades.
- Diseño e implementación de pruebas unitarias y funcionales relativas a la gestión de decisiones y alertas.

5.3.2.1 Funcionalidad de *Gestión de decisiones*

5.3.2.1.1 Diagramas de secuencia

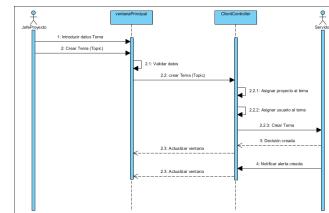
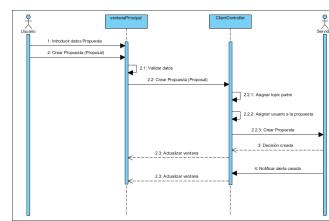
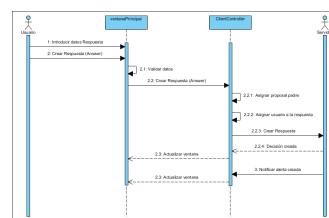
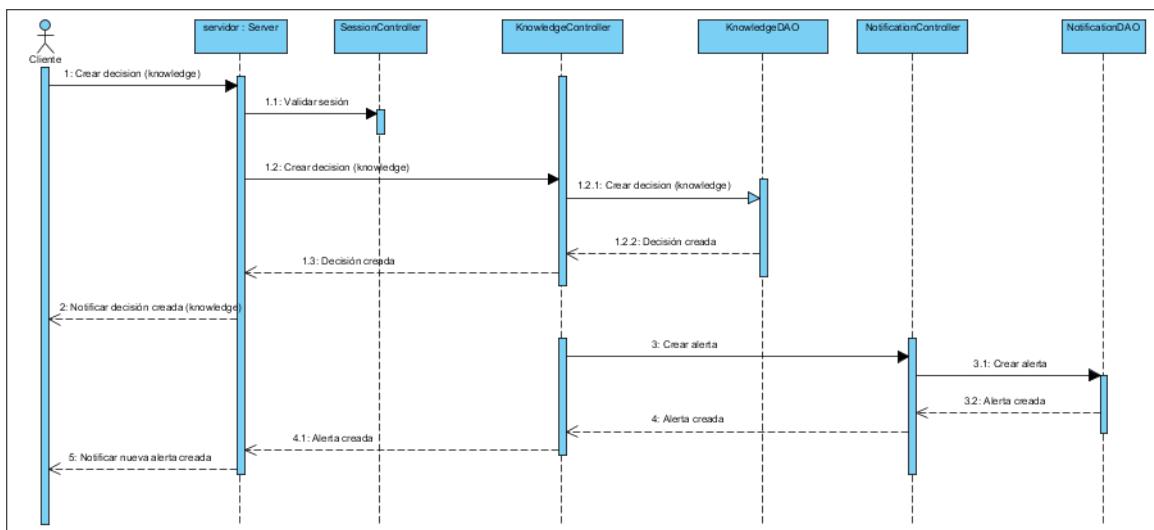
En los siguientes apartados se muestran los diagramas de secuencia para el cliente y servidor de los casos de uso que componen esta funcionalidad. Dichos diagramas se modelan siguiendo la descripción de los casos de uso realizada en el apartado 1.2.1.4.

Crear decisión

En las Figuras 1.66, 1.67 y 1.68 se muestran los diagramas de secuencia para el caso de uso *Crear decisión* en el subsistema cliente, mientras que en la Figura 1.69 se muestra el diagrama de secuencia para el servidor. Señalar que en el caso del subsistema cliente, se han creado tres diagramas de secuencia, para representar los diferentes actores (o roles en el sistema) que participan en el caso de uso *crear decisión*, según el tipo de decisión. En el servidor, sin embargo, no es necesario separarlo, ya que la secuencia de acciones a realizar es la misma, independientemente del tipo de decisión creada.

Modificar y eliminar decisión

Los diagramas de secuencia de estos dos casos de uso, para el subsistema cliente y servidor, son iguales que en caso de uso anterior, sólo que la acción realizada es modificar o eliminar una decisión, en vez de crearla.

**Figura 5.66:** Diagrama de secuencia - Cliente - Crear decisión (*Topic*)**Figura 5.67:** Diagrama de secuencia - Cliente - Crear decisión (*Topic*)**Figura 5.68:** Diagrama de secuencia - Cliente - Crear decisión (*Topic*)**Figura 5.69:** Diagrama de secuencia - Servidor - Crear decisión

Cambiar estado decisión

En la Figura ?? se muestra el diagrama de secuencia para el caso de uso *Cambiar estado decisión* en el cliente, mientras que en la Figura ?? se observa el diagrama de secuencia para el servidor.

Adjuntar ficheros

En la Figura 1.70 se muestra el diagrama de secuencia para el caso de uso *Adjuntar ficheros* en el cliente, mientras que en la Figura 1.71 se muestra el diagrama de secuencia para el servidor.

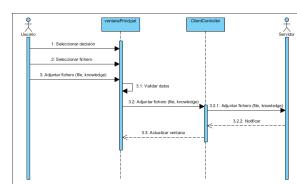


Figura 5.70: Diagrama de secuencia - Cliente - Adjuntar ficheros

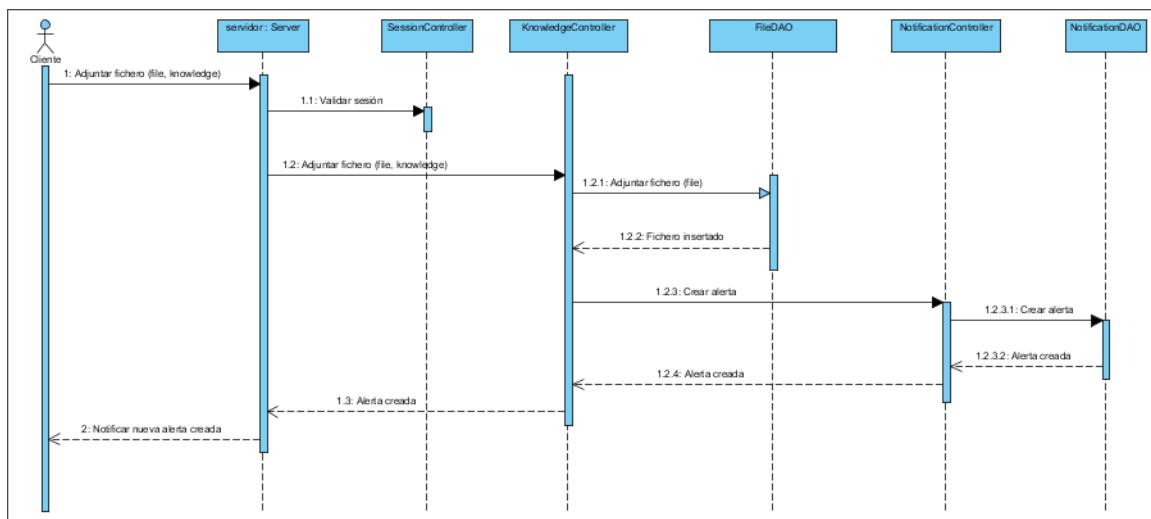


Figura 5.71: Diagrama de secuencia - Servidor - Adjuntar ficheros

5.3.2.1.2 Diseño e implementación

Servidor

En el diseño e implementación de estos casos de uso, cabe destacar la creación de una alerta o notificación de manera automática por parte del servidor al realizar cualquier acción sobre las decisiones. Dicha alerta se crea para el proyecto en el cuál se ha creado, modificado o eliminado una decisión, y para todos los usuarios que en dicho proyecto participan. De este modo, se notifica a los empleados que trabajan en ese proyecto que nuevo conocimiento está disponible, indicando en la alerta el tipo de decisión afectada, su autor, fecha y otros detalles. Esto facilita la comunicación asíncrona, ya que cuando un usuario vuelva a iniciar sesión, podrá comprobar sus nuevas alertas y adquirir conciencia de los cambios y nuevo conocimiento disponible o modificado.

En la Figura 1.72 se muestra el diagrama de clases para la funcionalidad de gestión de decisiones, mostrando las asociaciones entre clases y entre los controladores de decisiones y de notificaciones. Como en el resto de casos, las operaciones de bases de datos se delegan en el gestor de bases de datos y éste, a su vez, delega en el framework **Hibernate**.

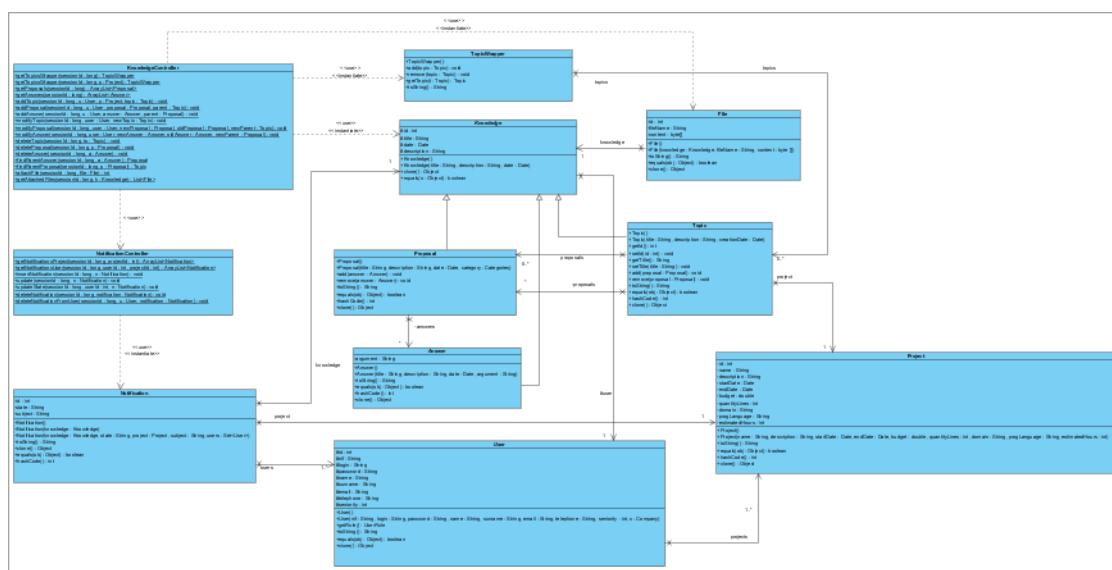


Figura 5.72: Diagrama de clases - Gestión de decisiones

También cabe destacar otra decisión de diseño que se ha tenido en cuenta para implementar otro de los requisitos del sistema, que es la notificación de información de manera síncrona. Para ello, cuando un cliente crea, modifica o elimina una decisión y envía la petición al

servidor, éste, además de crear la alerta, notifica a los clientes conectados que se ha producido un cambio, para que éstos puedan actualizar su vista de la interfaz gráfica y puedan reflejar los cambios sobre esa decisión en tiempo real. Para ello, el servidor lanza un hilo por cada uno de los clientes registrados en el sistema y les envía la información necesaria. Se utilizan hilos para no bloquear el servidor mientras manda actualizaciones a los clientes y pueda seguir atendiendo otras peticiones.

La clase controlador encargada de realizar esta tarea constituye además un patrón **observador**, el cuál se utiliza para registrar los clientes autenticados en el sistema y notificar y actualizar su estado. En el diagrama de la Figura 1.73 se muestra este patrón y las clases que lo forman.

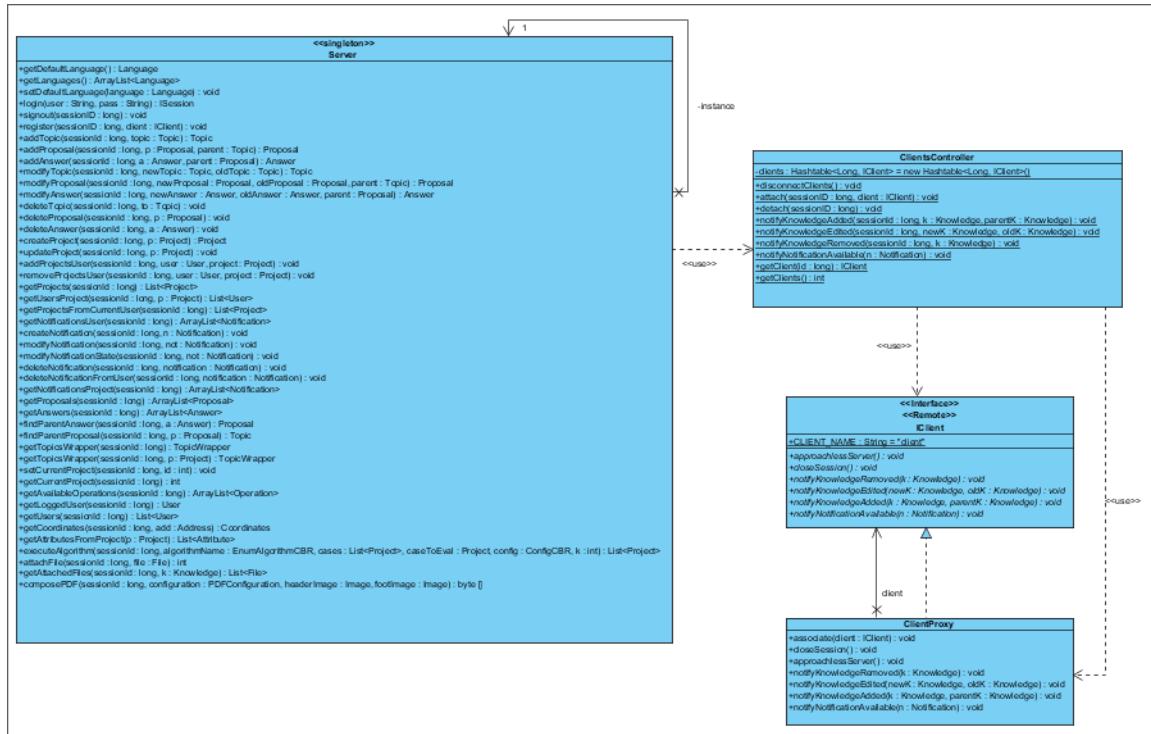


Figura 5.73: Diagrama de clases - Observador para actualizar clientes conectados

En el fragmento de código 1.7 se muestra parte de la implementación de la clase controlador de los clientes, y en el fragmento 1.8 como se ha implementado la gestión de hilos para notificar a los clientes. La clase encargada de esto último además representa un patrón **proxy**, ya que los clientes son remotos.

```
public class ClientsController {
```

```
private static Hashtable<Long, IClient> clients = new Hashtable<Long, IClient>();  
  
public static void attach(long sessionId, IClient client) {  
    clients.put(sessionId, client);  
}  
  
public static void detach(long sessionId) {  
    clients.remove(sessionId);  
}  
  
public static void notifyKnowledgeAdded(long sessionId, Knowledge k, Knowledge parentK)  
    throws RemoteException {  
    // Notify the clients (except the client that launched the operation) about the  
    // operation, in order to refresh their view  
    for(Long id : clients.keySet())  
        if (id != sessionId)  
            clients.get(id).notifyKnowledgeAdded(k, parentK);  
}  
  
public static void notifyKnowledgeEdited(long sessionId, Knowledge newK, Knowledge oldK)  
    throws RemoteException {  
    // Notify the clients (except the client that launched the operation) about the  
    // operation, in order to refresh their view  
    for(Long id : clients.keySet())  
        if (id != sessionId)  
            clients.get(id).notifyKnowledgeEdited(newK, oldK);  
}  
  
public static void notifyKnowledgeRemoved(long sessionId, Knowledge k) throws  
    RemoteException {  
    // Notify the clients (except the client that launched the operation) about the  
    // operation, in order to refresh their view  
    for(Long id : clients.keySet())  
        if (id != sessionId)  
            clients.get(id).notifyKnowledgeRemoved(k);  
}  
  
....  
}
```

Listado 5.7: Fragmento de código del controlador de clientes

```
public class ClientProxy implements IClient {  
  
    ....
```

```
@Override
public void notifyKnowledgeAdded(Knowledge k, Knowledge parentK) throws RemoteException
{
    Thread thread;

    // Launch the operation on another thread for not stopping the server
    thread = new Thread(new notifyKnowledgeAddedThread(client, k, parentK));
    thread.start();
}

@Override
public void notifyKnowledgeEdited(Knowledge newK, Knowledge oldK) throws RemoteException
{
    Thread thread;

    // Launch the operation on another thread for not stopping the server
    thread = new Thread(new notifyKnowledgeEditedThread(client, newK, oldK));
    thread.start();
}

@Override
public void notifyKnowledgeRemoved(Knowledge k) throws RemoteException {
    Thread thread;

    // Launch the operation on another thread for not stopping the server
    thread = new Thread(new notifyKnowledgeRemovedThread(client, k));
    thread.start();
}

private class notifyKnowledgeAddedThread implements Runnable {

    private IClient client;
    private Knowledge k;
    private Knowledge parentK;

    public notifyKnowledgeAddedThread(IClient client, Knowledge k, Knowledge parentK) {
        this.client = client;
        this.k = k;
        this.parentK = parentK;
    }

    public void run() {
        try {
            client.notifyKnowledgeAdded(k, parentK);
        } catch(Exception e) {
        }
    }
}
```

```
private class notifyKnowledgeEditedThread implements Runnable {

    private IClient client;
    private Knowledge newK;
    private Knowledge oldK;

    public notifyKnowledgeEditedThread(IClient client, Knowledge newK, Knowledge oldK) {
        this.client = client;
        this.newK = newK;
        this.oldK = oldK;
    }

    public void run() {
        try {
            client.notifyKnowledgeEdited(newK, oldK);
        } catch(Exception e) {
        }
    }
}

private class notifyKnowledgeRemovedThread implements Runnable {

    private IClient client;
    private Knowledge k;

    public notifyKnowledgeRemovedThread(IClient client, Knowledge k) {
        this.client = client;
        this.k = k;
    }

    public void run() {
        try {
            client.notifyKnowledgeRemoved(k);
        } catch(Exception e) {
        }
    }
}

....
```

Listado 5.8: Soporte multi-hilo para actualizar el estado de clientes

Cliente

En el subsistema cliente cabe destacar la utilización del API de Java **Reflection** para poder configurar el diálogo para gestionar las decisiones (creación y modificación) en tiempo de ejecución. Así, cuando el usuario selecciona una decisión a crear o modificar (*tema, Propuesta o Respuesta*), la interfaz se adaptará a ese tipo de decisión, mostrando los elementos oportunos. Por tanto, se utiliza la reflexión de Java para instanciar el panel gráfico correspondiente y visualizarlo cuando el usuario haya elegido una acción, en tiempo de ejecución.

En el fragmento de código 1.9 se presenta un ejemplo de uso de la reflexión para crear un componente visual conocido en tiempo de ejecución, según el valor de la variable *subgroup*.

```
Constructor c = Class.forName("presentation.panelsManageKnowledge.JPManage"+subgroup) .  
getConstructor(  
    new Class [] {JFMain.class, JDialog.class, Object.class, String.class});  
component = (Component) c.newInstance(new Object [] {parentFrame, dialog, data,  
operationToDo});
```

Listado 5.9: Fragmento de código utilizando *reflection*

Por otra parte, en el cliente, cuando el servidor le notifica que se ha producido un cambio en las decisiones, producido por otro cliente, se refrescan las decisiones en la vista de visualización de decisiones (comentada en la iteración anterior), para poder reflejar este cambio de manera síncrona, refrescando el grafo y árbol de decisiones, así como la información asociada a cada una de ellas.

Para terminar, en lo que concierne a la gestión de notificaciones o alertas, se ha diseñado e implementado una vista en el cliente para poder mostrar esas alertas, de manera similar a una vista de correo electrónico, mostrando las alertas leídas y no leídas, la información de dichas alertas, su autor, etc.

5.3.2.2 Funcionalidad de *Gestión de notificaciones*

5.3.2.2.1 Diagramas de secuencia

Consultar notificaciones

En la Figura 1.74 se muestra el diagrama de secuencia para el caso de uso *Consultar notificaciones* en el cliente, mientras que en la Figura 1.75 se muestra el diagrama de secuencia para el servidor.

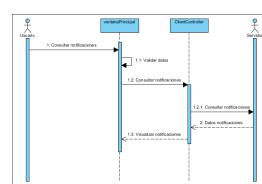


Figura 5.74: Diagrama de secuencia - Cliente - Consultar notificaciones

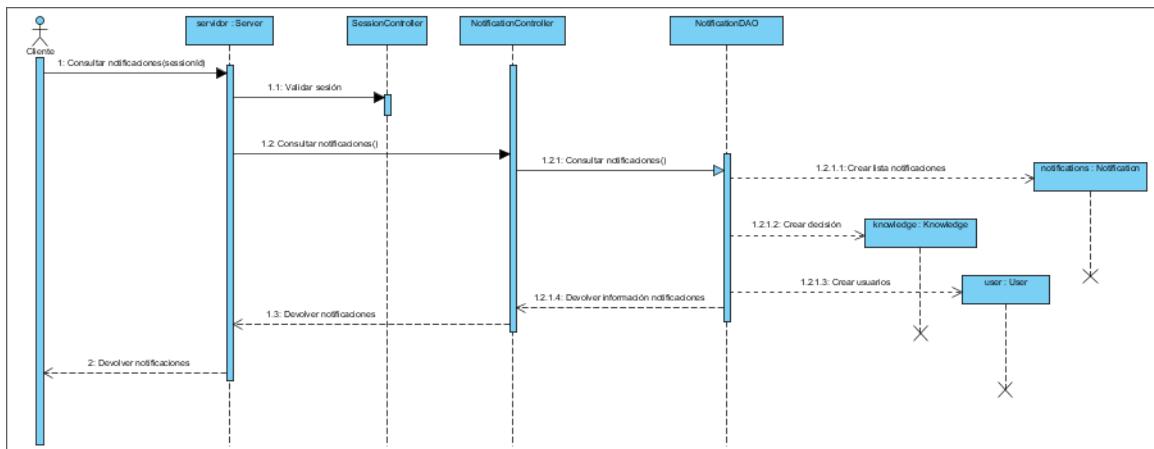


Figura 5.75: Diagrama de secuencia - Servidor - Consultar notificaciones

Modificar notificaciones

En la Figura 1.76 se muestra el diagrama de secuencia para el caso de uso *Modificar notificaciones* en el cliente, mientras que en la Figura 1.77 se muestra el diagrama de secuencia para el servidor.

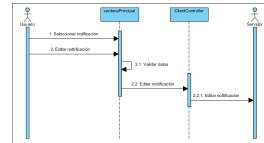


Figura 5.76: Diagrama de secuencia - Cliente - Modificar notificaciones

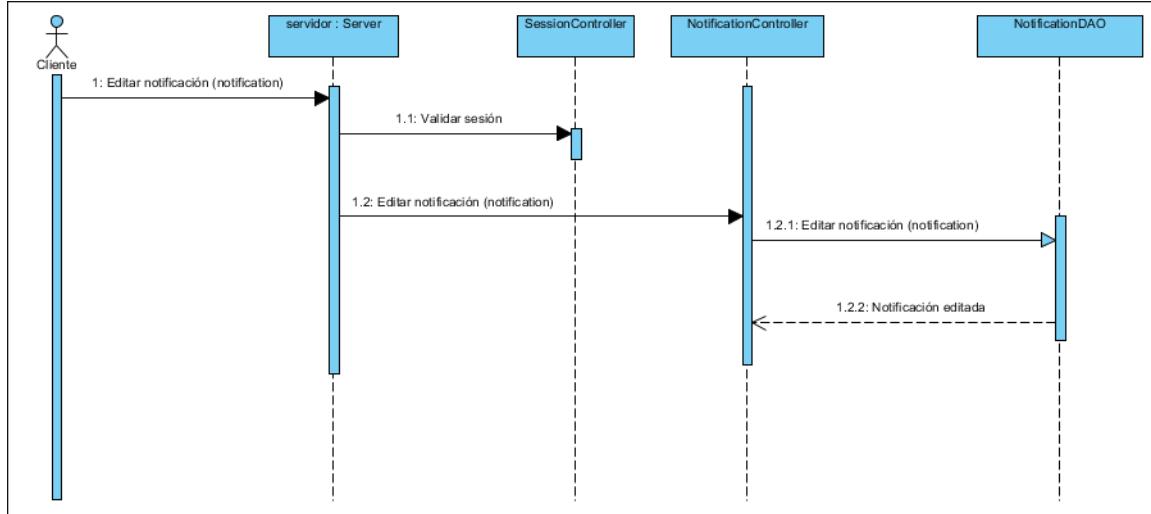


Figura 5.77: Diagrama de secuencia - Servidor - Modificar notificaciones

Eliminar notificaciones

En la Figura 1.78 se muestra el diagrama de secuencia para el caso de uso *Eliminar notificaciones* en el cliente, mientras que en la Figura 1.79 se muestra el diagrama de secuencia para el servidor.

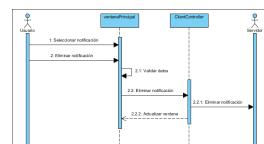


Figura 5.78: Diagrama de secuencia - Cliente - Eliminar notificaciones

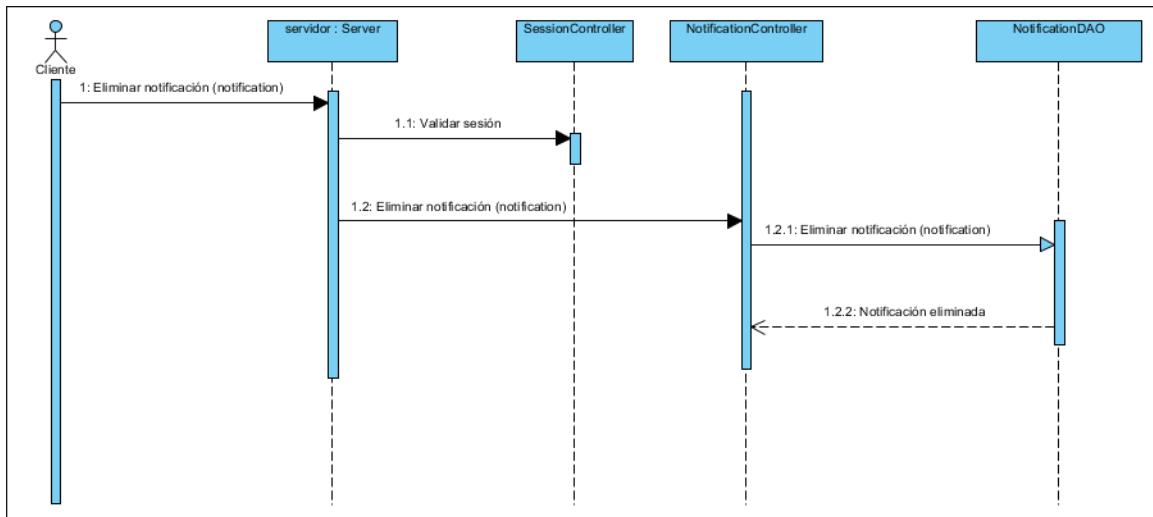


Figura 5.79: Diagrama de secuencia - Servidor - Eliminar notificaciones

5.3.2.2.2 Diseño e implementación

Servidor

Como se ha detallado en el apartado 1.3.2.1.2, se crean alertas (o notificaciones) de manera automática para todos los usuarios del proyecto que ha sufrido cambios en sus decisiones, como se muestra en el diagrama de la Figura 1.72. Por tanto, la misma alerta debe crearse para todos los usuarios de ese proyecto, pero, para evitar que la información de esa alerta esté repetida, la base de datos se ha diseñado de tal modo que la alerta sólo se crea una vez y se hace referencia a ella para todos los usuarios, utilizando la tabla *notificationsUsers* con claves ajena, como se muestra en el diseño de base de datos de la Figura 1.46.

De este modo, cada usuario podrá editar y eliminar su propia alerta, sólo eliminando la alerta original cuando ningún usuario tenga ya referencias a ella, es decir, cuando todos los usuarios del proyecto hayan borrado esa alerta. Esta tarea se ha delegado al SGBD de MySQL, mediante la creación de un *trigger*, mostrado en el fragmento de código 1.10. Dicho trigger será el encargado de borrar la alerta original cuando ya no existan referencias a ella por parte de ningún usuario.

```

CREATE TRIGGER DeleteEntity
AFTER DELETE ON notificationsUsers
FOR EACH ROW
BEGIN

```

```
DECLARE
    cont INT;
SELECT COUNT(*) INTO cont FROM notificationsUsers WHERE idNotification = OLD.
    idNotification;
IF (cont = 0) THEN
    DELETE FROM notifications WHERE id = OLD.idNotification;
END IF;
END$$
```

Listado 5.10: Trigger de base de datos para gestionar la eliminación de alertas

Cliente

Siguiendo los diagramas de secuencia para este subsistema, se diseña e implementa la interfaz gráfica de usuario para dar soporte a cada caso de uso, enviando peticiones al servidor y obteniendo los datos que éste devuelve, mostrándolos en la interfaz.

5.3.2.2.3 Pruebas

5.3.3 Tercera iteración

Siguiendo los casos de uso de la funcionalidad de *Gestión de proyectos* (ver Figura ??), así como su análisis realizado, se abordan las siguientes tareas en esta iteración:

- Diseño de la funcionalidad relativa a la gestión de proyectos.
- Implementación de dicha funcionalidad.
- Diseño e implementación de pruebas unitarias y funcionales relativas a la gestión de proyectos.

5.3.3.1 Funcionalidad de *Gestión de proyectos*

5.3.3.1.1 Diagramas de secuencia

En los siguientes apartados se muestran los diagramas de secuencia para el cliente y servidor de los casos de uso que componen esta funcionalidad. Dichos diagramas se modelan siguiendo la descripción de los casos de uso realizada en el apartado 1.2.1.4.

Consultar proyectos

En la Figura 1.80 se muestra el diagrama de secuencia para el caso de uso *Consultar proyectos* en el cliente, mientras que en la Figura 1.81 se muestra el diagrama de secuencia para el servidor.

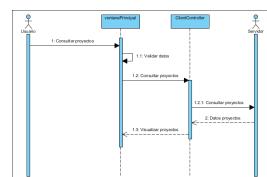


Figura 5.80: Diagrama de secuencia - Cliente - Consultar proyectos

Consultar usuarios

En la Figura 1.82 se muestra el diagrama de secuencia para el caso de uso *Consultar usuarios* en el cliente, mientras que en la Figura 1.83 se muestra el diagrama de secuencia para el

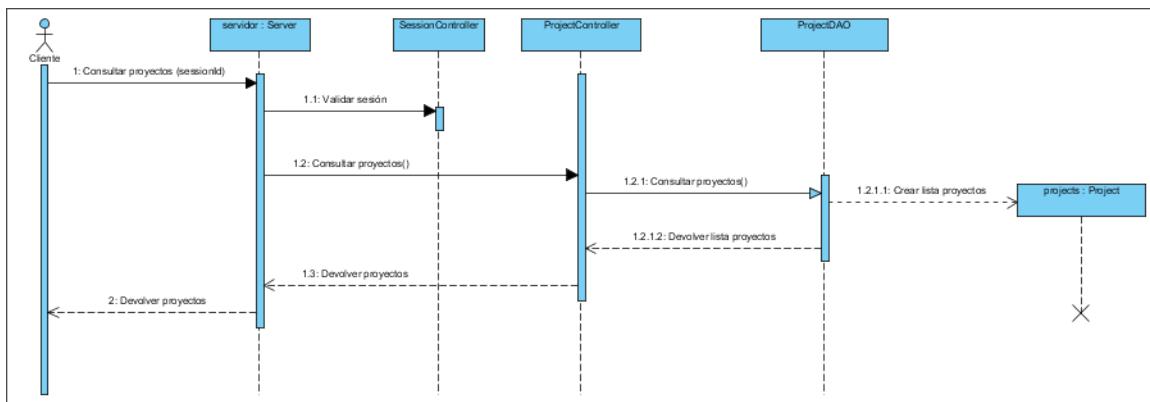


Figura 5.81: Diagrama de secuencia - Servidor - Consultar proyectos

servidor.

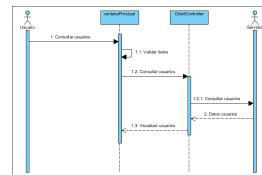


Figura 5.82: Diagrama de secuencia - Cliente - Consultar usuarios

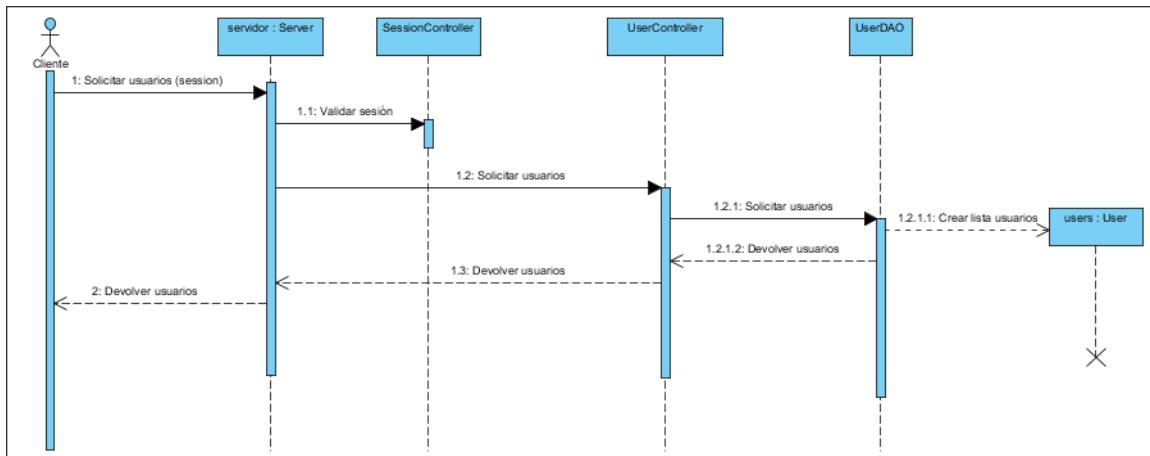


Figura 5.83: Diagrama de secuencia - Servidor - Consultar usuarios

Crear proyecto

En la Figura 1.84 se muestra el diagrama de secuencia para el caso de uso *Crear proyecto* en el cliente, mientras que en la Figura 1.85 se muestra el diagrama de secuencia para el servidor.

En este caso, se incluye el comportamiento del caso de uso *consultar usuarios*, por lo que ese comportamiento se muestra de manera simplificada en este diagrama de secuencia.

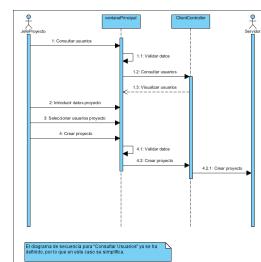


Figura 5.84: Diagrama de secuencia - Cliente - Crear proyecto

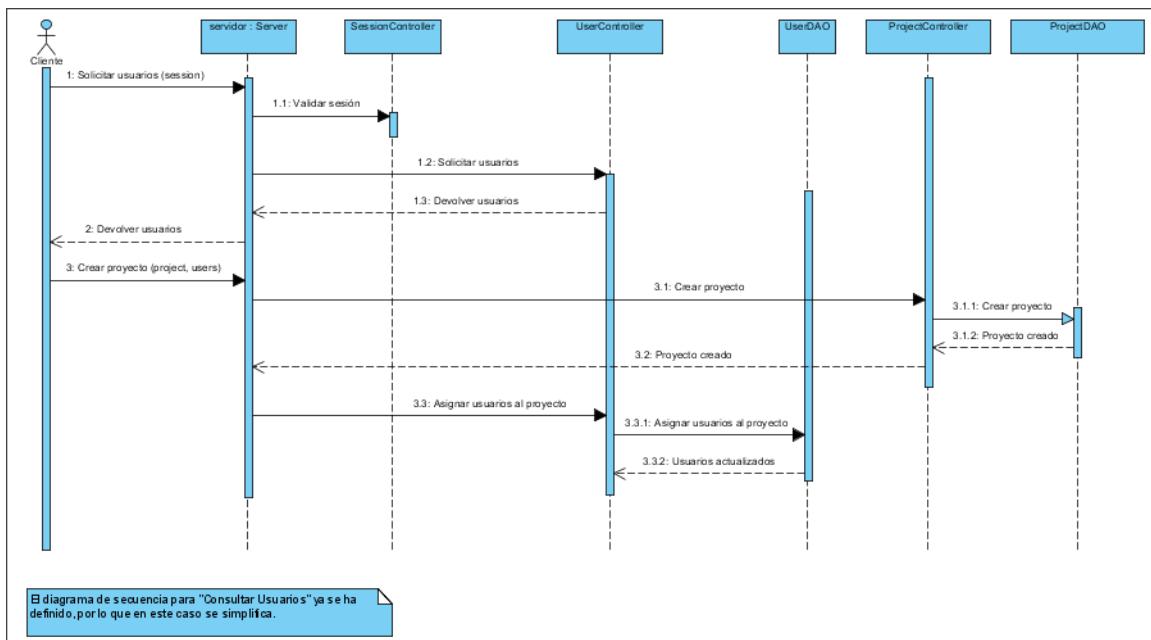


Figura 5.85: Diagrama de secuencia - Servidor - Crear proyecto

Modificar proyecto

En la Figura 1.86 se muestra el diagrama de secuencia para el caso de uso *Consultar proyectos* en el cliente, mientras que en la Figura 1.87 se muestra el diagrama de secuencia para el servidor.

En este caso, se incluye el comportamiento de los casos de uso *consultar proyectos* y *consultar usuarios*, por lo que dichos comportamientos se muestran de manera simplificada en este diagrama de secuencia.

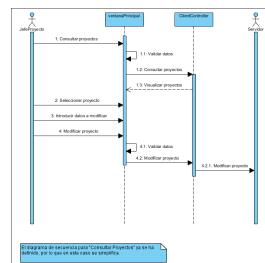


Figura 5.86: Diagrama de secuencia - Cliente - Modificar proyecto

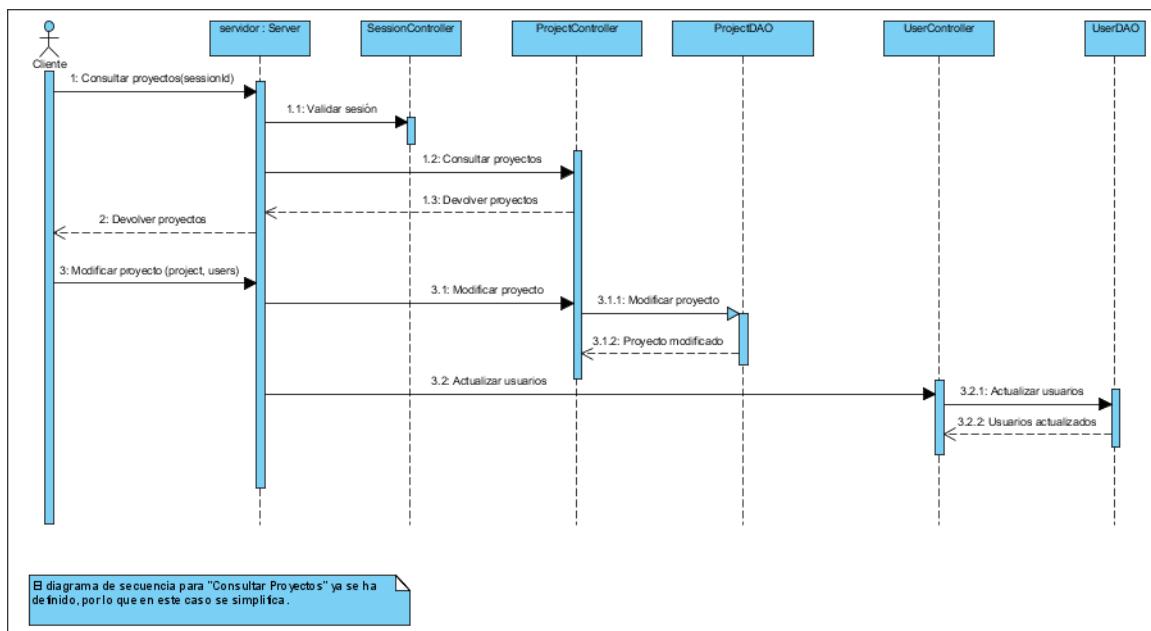


Figura 5.87: Diagrama de secuencia - Servidor - Modificar proyecto

Seleccionar proyecto activo

En la Figura 1.88 se muestra el diagrama de secuencia para el caso de uso *Consultar proyectos* en el cliente, mientras que en la Figura 1.81 se muestra el diagrama de secuencia para el servidor.

En este caso, se incluye el comportamiento del caso de uso *consultar proyectos*, por lo que ese comportamiento se muestra de manera simplificada en este diagrama de secuencia.

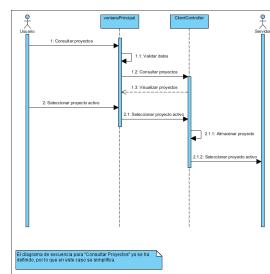


Figura 5.88: Diagrama de secuencia - Cliente - Seleccionar proyecto activo

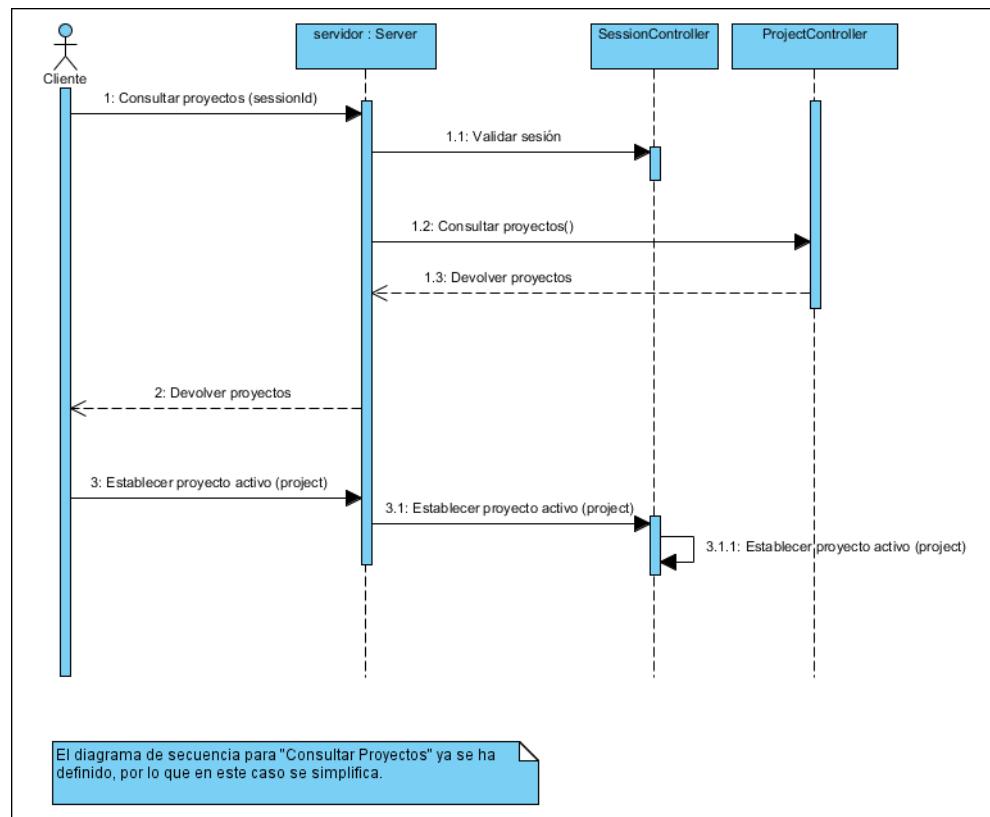


Figura 5.89: Diagrama de secuencia - Servidor - Seleccionar proyecto activo

Aconsejar decisiones

En la Figura 1.80 se muestra el diagrama de secuencia para el caso de uso *Consultar proyectos* en el cliente, mientras que en la Figura 1.81 se observa el diagrama de secuencia para el servidor.

5.3.3.1.2 Diseño e implementación

Servidor

Como se comentó en el modelo de negocio en el apartado ??, cada usuario trabaja en 1 o más proyectos, y en cada proyecto participan varios usuarios, por lo que, como se refleja en los diagramas de secuencia anteriores, para crear y modificar un proyecto hay que asignar los usuarios que en dicho proyecto trabajan. En el diagrama de la Figura ?? se observan las clases que participan en la gestión de proyectos y las asociaciones que existen entre los usuarios y los proyectos, y sus clases controladoras correspondientes.

Sin embargo, en esta funcionalidad, lo que cabe destacar es el diseño e implementación del caso de uso de *Aconsejar decisiones*. Este caso de uso representa el requisito de recuperar proyectos similares a uno dado, para poder mostrar las decisiones que en él se tomaron, cuáles fueron aceptadas o rechazadas, etc, para poder tenerlas en cuenta en un nuevo proyecto. De este modo, a partir de un nuevo proyecto, y basándose en proyectos anteriores ya terminados, se recuperan aquellos que sean más similares al proyecto dado y se muestran sus decisiones. Para ello, se utiliza el Razonamiento Basado en Casos, o CBR, detallado en la sección ??.

Cada caso del CBR representará un proyecto, definido por un conjunto de atributos, que serán los utilizados para poder comparar proyectos y calcular la semejanza entre casos.

En lo que respecta al CBR en el subsistema servidor, éste recibe la configuración definida en el cliente, el tipo de algoritmo a utilizar, la lista de proyectos terminados (los casos o experiencias pasadas), el proyecto a evaluar y el número de proyectos (o casos) que se desean recuperar y visualizar. A continuación, se explican cada uno de estos elementos que recibe el método del servidor encargado de esta funcionalidad de CBR.

- Número de proyectos a recuperar, definidos por una variable k . Es el número de proyectos similares que se recuperan en la fase de *Recuperación* del CBR. Pueden

recuperarse todos los proyectos similares, o sólo algunos, según el valor de k .

- El proyecto a evaluar es el nuevo proyecto, o caso, del que se desean obtener sus decisiones, basándose en proyectos ya pasados y similares a éste.
- La lista de proyectos ya terminados, que componen lo que se llama la *base de casos* en CBR. Son todos los proyectos pasados con los que se calculará la semejanza o similitud con el caso nuevo a evaluar.
- El algoritmo a utilizar para calcular la semejanza entre el nuevo caso y cada uno de los casos pasados.
- La configuración necesaria para utilizar en los algoritmos de CBR.

Cabe destacar este último elemento, que es la configuración utilizada en los algoritmos del CBR. Dicha configuración contiene el conjunto de pesos que dichos atributos van a tomar al calcular el valor final de la semejanza entre casos. Además contiene, para cada atributo, una función que indica como valorar y comparar dicho atributo entre casos, llamada *función de semejanza local*. Dicha función establece como comparar los atributos entre dos casos, devolviendo el valor de semejanza entre esos atributos. Existen tres tipos de función:

- **Diferencia:** esta función devuelve la diferencia entre dos valores numéricos o fechas. Si los valores son de tipo cadena, devuelve 1 si las cadenas son iguales, o 0 en caso contrario.
- **Igualdad:** función que devuelve 1 si los valores son iguales, y en caso contrario.
- **Intervalo:** función utilizada para evaluar si la diferencia de dos valores numéricos se encuentran en un determinado margen, devolviendo la desviación con respecto a ese margen.

En lo que respecta a los algoritmos utilizados para calcular el valor de la semejanza final, o global, entre casos, se han diseñado e implementado diferentes algoritmos, basándose en la literatura estudiada y comentada en la sección ???. Dichos algoritmos son:

- **Nearest neighbor:** este algoritmo, conocido también como *NN Method*, calcula el valor de semejanza global entre dos casos realizando la media aritmética de los valores

de los atributos. Así, se calcula el sumatorio del producto del valor de cada atributo (calculado por la función de semejanza local) por su peso, y se divide por la suma del peso total de todos los atributos. En la ecuación 1.1 se muestra esta función de semejanza global utilizada en este algoritmo, siendo $c1$ y $c2$ los casos a evaluar; w_i el peso de cada atributo, y $sem(att1_i, att2_i)$ el valor de semejanza calculado por la función de semejanza local entre los atributos de ambos casos.

- **Euclidean Distance;** este algoritmo calcula el valor de semejanza global entre dos casos realizando la distancia euclídea de los valores de los atributos. Así, se calcula el sumatorio del producto del valor de cada atributo (calculado por la función de semejanza local) elevado al cuadrado, por su peso. En la ecuación 1.2 se muestra esta función de semejanza global utilizada en este algoritmo, siendo $c1$ y $c2$ los casos a evaluar; w_i el peso de cada atributo, y $sem(att1_i, att2_i)$ el valor de semejanza calculado por la función de semejanza local entre los atributos de ambos casos.

$$sem(c1, c2) = \sum_{i=1}^n (w_i * sem(att1_i, att2_i)) / \sum_{i=1}^n w_i \quad (5.1)$$

$$sem(c1, c2) = \sum_{i=1}^n (w_i * sem(att1_i - att2_i)^2) \quad (5.2)$$

Una vez calculada la semejanza global entre el caso a evaluar (el nuevo proyecto) y cada uno de los proyectos pasados, obteniendo una lista de proyectos con su valor de semejanza, dicha lista se ordena de mayor a menor y se toman los k primeros, en caso de haber establecido el valor de k . Con ello, termina esta fase del CBR, que es la fase de **Recuperación**, de la que se encarga el sistema servidor. Acto seguido, esta lista de proyectos, ordenados de mayor a menor semejanza con el nuevo caso a evaluar, se devuelven al cliente.

En la Figura 1.90 se muestra el diagrama de clases de diseño de la funcionalidad del CBR. Como se puede observar, se han diseñado interfaces para implementar las funciones de semejanza local y global. De este modo, se podrían extender estas funciones, añadiendo nuevos métodos de comparación, simplemente implementando estas interfaces, consiguiendo que el controlador de CBR sea extensible a nuevas funciones de semejanza.

Para terminar, en el fragmento de código 1.11 se muestra la implementación del algoritmo *NN*, haciendo uso del diseño y elementos comentados anteriormente. Un aspecto a señalar es

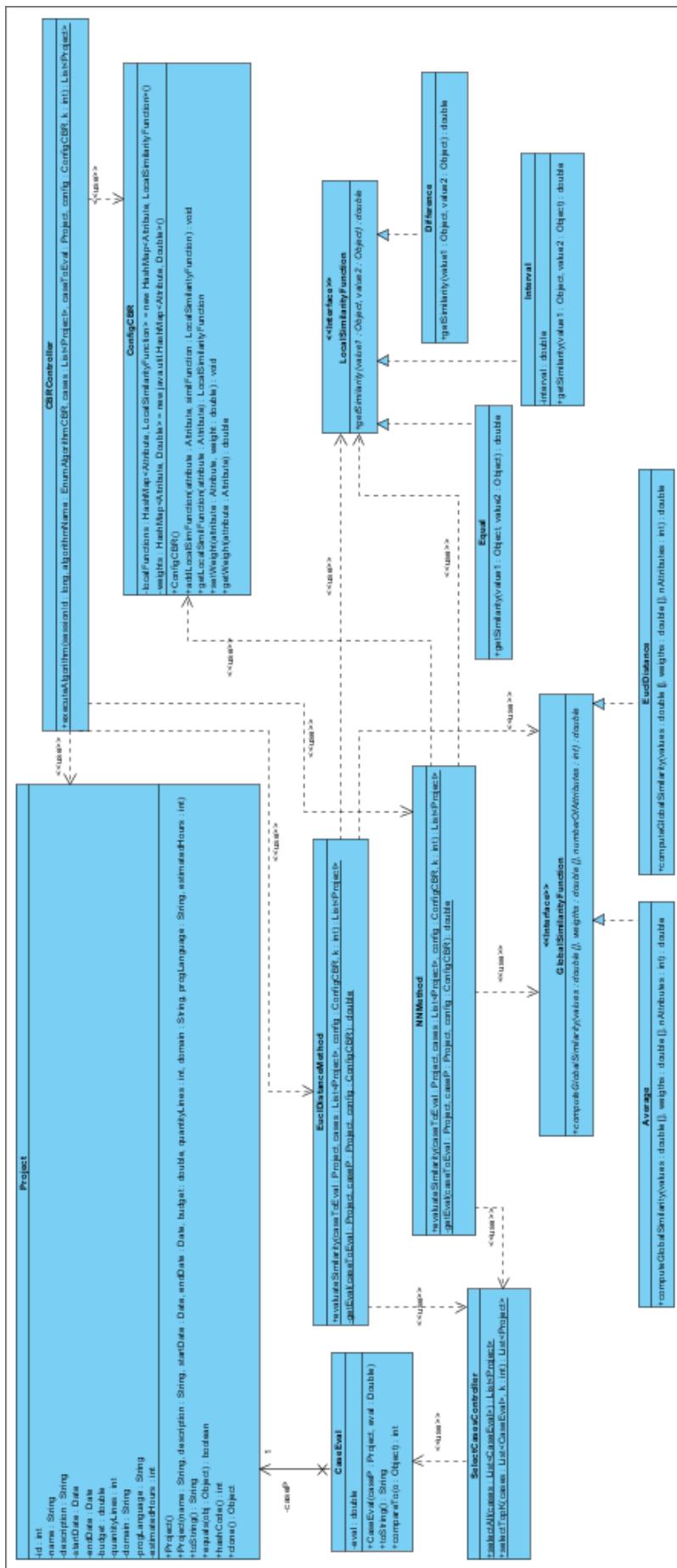


Figura 5.90: Diagrama de clases - Servidor - Razonamiento Basado en Casos

el uso del API de reflexión de Java, utilizado para acceder a los atributos de un proyecto en tiempo de ejecución.

```
/**  
 * Class used to apply the Nearest Neighbor algorithm  
 */  
  
public class NNMethod {  
  
    /* Apply the algorithm over the cases.  
     * Return the similarity cases */  
    public static List<Project> evaluateSimilarity(Project caseToEval, List<Project> cases,  
        ConfigCBR config, int k) throws Exception  
    {  
        List<Project> similCases = new ArrayList<Project>();  
        List<CaseEval> result = new ArrayList<CaseEval>();  
        for(Project caseP: cases)  
        {  
            result.add(new CaseEval(caseP, getEval(caseToEval, caseP, config)));  
        }  
        // Sort the result  
        Collections.sort(result);  
  
        if (k == 0)  
            similCases = SelectCasesController.selectAll(result);  
        else  
            similCases = SelectCasesController.selectTopK(result, k);  
  
        return similCases;  
    }  
  
    /*  
     * Get the evaluation for each attribute of the cases to compare  
     */  
    private static double getEval(Project caseToEval, Project caseP, ConfigCBR config)  
        throws Exception {  
        LocalSimilarityFunction lsf = null;  
        GlobalSimilarityFunction gsf = null;  
  
        // Take attributes from each case  
        List<Attribute> attributesCaseToEval = ProjectController.getAttributesFromProject(  
            caseToEval);  
        List<Attribute> attributesCase = ProjectController.getAttributesFromProject(caseP);  
        // Global similarity function used in NN Method  
        gsf = new Average();  
  
        // Evaluation for each attribute (ignore id and serialVersionUID)  
        double[] values = new double[attributesCaseToEval.size() - 2];
```

```
// Weights for each attribute (ignore id and serialVersionUID)
double[] weights = new double[attributesCaseToEval.size() - 2];

int nAttributes = 0;
for(int i=2; i<attributesCaseToEval.size(); i++)
{
    Attribute attCase1 = attributesCaseToEval.get(i);
    Attribute attCase2 = attributesCase.get(i);

    // Evaluation of the attributes using local similarity function
    if ((lsf = config.getLocalSimilFunction(attCase1)) != null) {
        // Using reflection in order to get the attribute value
        Field attField1 = Project.class.getDeclaredField(attCase1.getName());
        Field attField2 = Project.class.getDeclaredField(attCase2.getName());
        attField1.setAccessible(true);
        attField2.setAccessible(true);
        // Calculate similarity using the local similarity function
        values[i - 2] = lsf.getSimilarity(attField1.get(caseToEval), attField2.get(
            caseP));
        weights[i - 2] = config.getWeight(attCase1);
        nAttributes++;
    }
}

// Return the similarity between both cases, applying the global function (average,
// in this algorithm)
return gsf.computeGlobalSimilarity(values, weights, nAttributes);
}
```

Listado 5.11: Fragmento de código para el algoritmo *NN* del CBR

Cliente

5.3.3.1.3 Pruebas

5.3.4 Cuarta iteración

Siguiendo los casos de uso de la funcionalidad de *Generación de informes* (ver Figura 1.27) y *Generación de estadísticas* (ver Figura 1.25), así como el análisis realizado, se abordan las siguientes tareas en esta iteración:

- Diseño de la funcionalidad relativa a la generación de informes y gráficos estadísticos.
- Implementación de dichas funcionalidades.
- Diseño e implementación de pruebas unitarias y funcionales relativas a la generación de informes y gráficos estadísticos.

5.3.4.1 Funcionalidad de *Generación de informes*

5.3.4.1.1 Diagramas de secuencia

En los siguientes apartados se muestran los diagramas de secuencia para el cliente y servidor de los casos de uso que componen esta funcionalidad. Dichos diagramas se modelan siguiendo la descripción de los casos de uso realizada en el apartado 1.2.1.4.

Generar informe

Como se puede observar en el modelo de casos de uso de la Figura 1.27, este caso de uso cuenta con un punto de extensión, donde el caso de uso *Consultar proyectos* extiende su funcionalidad. Por tanto, en la Figura 1.91 se muestra el diagrama de secuencia para el comportamiento normal de este caso de uso en el cliente, y en la Figura 1.92 se muestra su comportamiento normal en el servidor.

Por otra parte, en la Figura 1.93 se muestra el diagrama de secuencia para el comportamiento extendido de este caso de uso en el cliente, y en la Figura 1.94 se muestra su comportamiento extendido en el servidor.

Hay que señalar que en esta funcionalidad se incluye el comportamiento del caso de uso *consultar decisiones*, por lo que dicho comportamiento se muestra de manera simplificada en estos diagramas de secuencia.

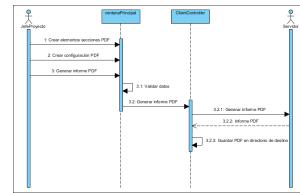


Figura 5.91: Diagrama de secuencia - Cliente - Generar informe (comportamiento normal)

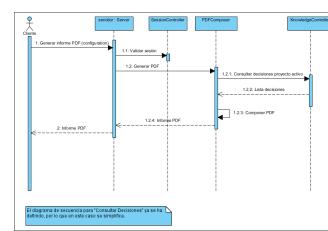


Figura 5.92: Diagrama de secuencia - Cliente - Generar informe (comportamiento normal)

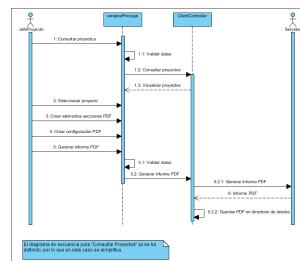


Figura 5.93: Diagrama de secuencia - Cliente - Generar informe (comportamiento extendido)

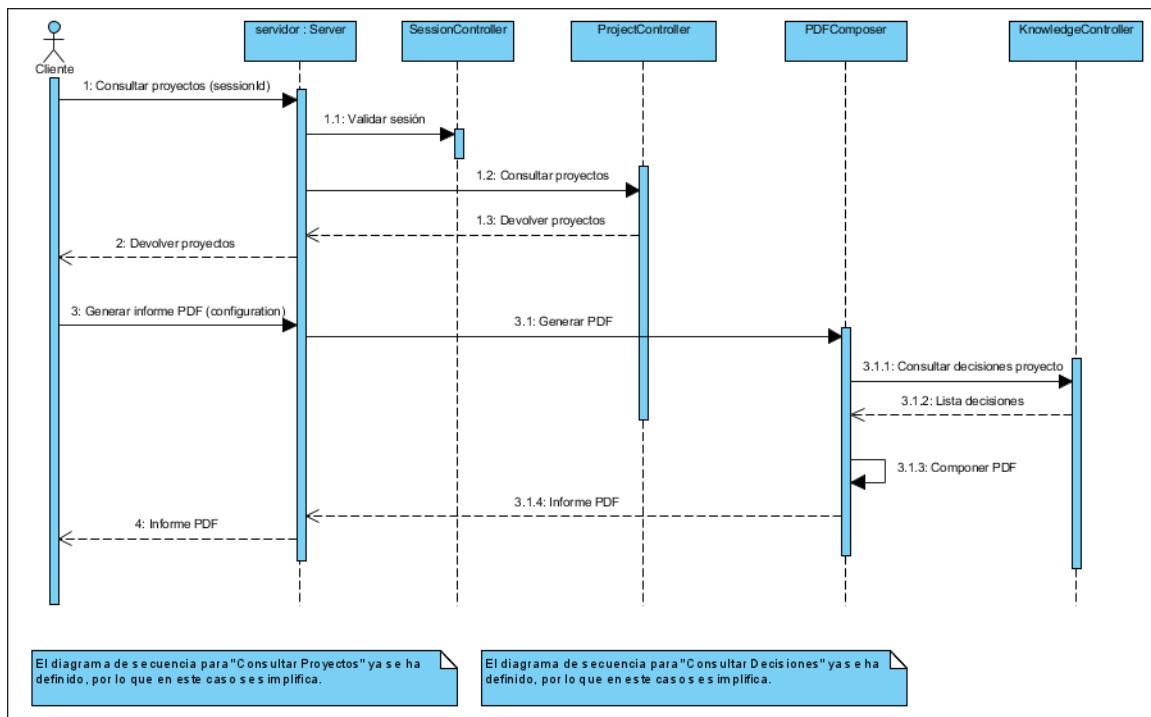


Figura 5.94: Diagrama de secuencia - Servidor -Generar informe (comportamiento extendido)

5.3.4.1.2 Diseño e implementación

Servidor

Siguiendo los diagramas de secuencia, se realiza el diseño de los casos de uso utilizando un diagrama de clases de diseño y se procede a la implementación de dichos casos de uso, para obtener las clases Java que dan soporte a estas funcionalidades. En la Figura 1.95 se muestra el diagrama de clases de diseño para la funcionalidad de generación de informes en PDF.

Se ha utilizado una jerarquía de herencia para diseñar los elementos que forman parte de las secciones que componen un documento PDF, para poder utilizar la capacidad de polimorfismo de las clases de Java y poder componer una sección del PDF de diferentes elementos, con una superclase de la que heredan. De este modo, una sección se puede componer de:

- Un título de sección, que tiene un tipo de fuente determinada.
- Texto, que compone el texto de la sección, también con un tipo de fuente determinada.
- Una tabla, donde se van a mostrar el conjunto de decisiones y toda su información asociada del proyecto seleccionado.

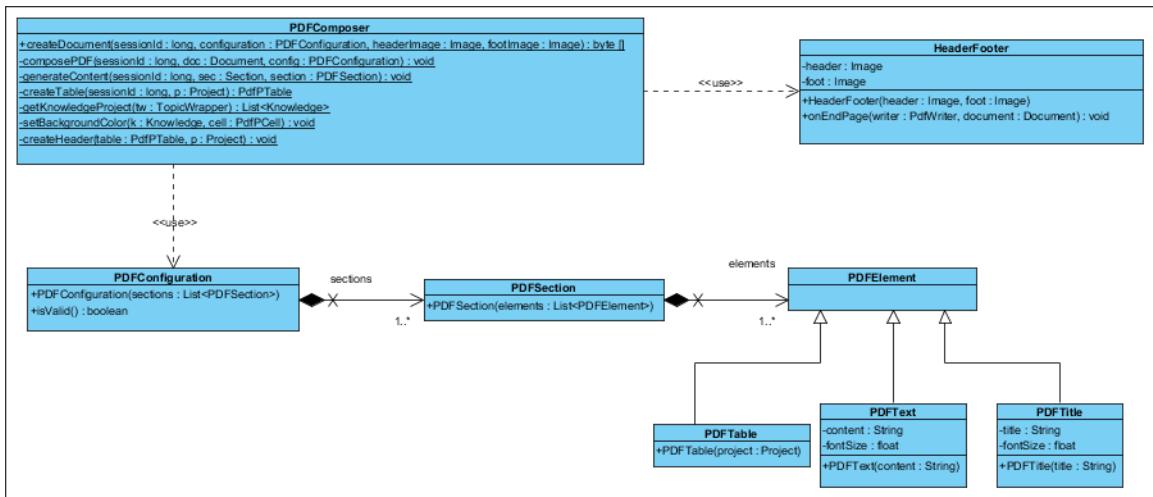


Figura 5.95: Diagrama de clases - Generación de documentos PDF - Servidor

En cuanto al comportamiento normal o extendido de esta funcionalidad que se comentó en el apartado anterior, ésto depende si se selecciona o no proyecto para generar su tabla de decisiones e información asociada. Por tanto, el comportamiento por defecto es trabajar con el proyecto activo del usuarios en ese momento, pudiendo seleccionar otro proyecto para generar la tabla de decisiones en el informe PDF.

Por otra parte, como se observa en la Figura 1.95, cabe señalar el uso de la clase *HeaderFooter* para colocar una imagen en el encabezado y pie de página de cada página del documento PDF. Esto es útil, por ejemplo, para colocar el logo de una compañía en cada página del informe.

En cuanto a aspectos de implementación, se utiliza **iText** (ver sección ??) para crear el documento PDF y para ir componiendo sus secciones con los diferentes elementos que las componen, según hayan sido configuradas en el cliente. En el listado de código ?? se muestra un fragmento de código de como crear el documento y como se insertan en él los elementos.

```

/**
 * Class used to generate a PDF document from the user-entered configuration
 */
public class PDFComposer {

    public static byte[] createDocument(long sessionId, PDFConfiguration configuration,
        Image headerImage, Image footImage) throws NumberFormatException, RemoteException,
        SQLException, NonPermissionRoleException, NotLoggedException, Exception {
        ....
    }
}
  
```

```
Document doc = new Document(PageSize.A4, 20, 20, marginTop, marginBottom);

ByteArrayOutputStream buffer = new ByteArrayOutputStream();
PdfWriter pdfWriter = PdfWriter.getInstance(doc, buffer);

// Event used to add header image and foot image
HeaderFooter event = new HeaderFooter(headerImage, footImage);
pdfWriter.setPageEvent(event);

doc.open();
composePDF(sessionId, doc, configuration);
doc.close();

return buffer.toByteArray();
}

private static void composePDF (long sessionId, Document doc, PDFConfiguration config)
throws NumberFormatException, RemoteException, SQLException,
NonPermissionRoleException, NotLoggedException, Exception {
int count = 1;
// Create the chapter
Chapter ch = new Chapter(count);
ch.setNumberDepth(0);
// Create the different sections
for (PDFSection section : config.getSections()) {
    Section s = ch.addSection(4f, "");
    generateContent(sessionId, s, section);
    doc.add(ch);
}
}

private static void generateContent(long sessionId, Section sec, PDFSection section)
throws NumberFormatException, RemoteException, SQLException,
NonPermissionRoleException, NotLoggedException, Exception {
for(PDFELEMENT element : section.getElements()){
    if (element instanceof PDFTITLE) {
        Font f = FontFactory.getFont(FontFactory.HELVETICA, ((PDFTITLE)element).
            getFontSize(), Font.BOLD, new BaseColor(Color.BLACK));
        Paragraph p = new Paragraph(((PDFTITLE)element).getTitle().toUpperCase(), f);
        sec.setTitle(p);
    }
    else if (element instanceof PDFTEXT) {
        Font f = FontFactory.getFont(FontFactory.HELVETICA, ((PDFTEXT)element).
            getFontSize(), Font.BOLD, new BaseColor(Color.BLACK));
        Paragraph p = new Paragraph(((PDFTEXT)element).getContent(), f);
        sec.add(p);
    }
    else if (element instanceof PDFTABLE) {
```

```
        PdfPTable table = createTable(sessionId, ((PDFTable)element).getProject());
        sec.add(table);
    }
}

private static PdfPTable createTable(long sessionId, Project p) throws RemoteException,
        SQLException, NonPermissionRoleException, NotLoggedException, Exception {
    PdfPTable table = new PdfPTable(6);
    createHeader(table, p);
    TopicWrapper tw;
    // Take knowledge from project
    tw = Server.getInstance().getTopicsWrapper(sessionId, p);
    List<Knowledge> knowledge = getKnowledgeProject(tw);

    for(Knowledge k: knowledge) {
        Image image = null;
        if (k instanceof Topic)
            image = Image.getInstance(ImagesUtilities.getPathImage("Topic.png"));
        if (k instanceof Proposal)
            image = Image.getInstance(ImagesUtilities.getPathImage("Proposal.png"));
        if (k instanceof Answer)
            image = Image.getInstance(ImagesUtilities.getPathImage("Answer.png"));

        PdfPCell cell = new PdfPCell(image, false);
        setBackgroundColor(k, cell);
        table.addCell(cell);

        cell = new PdfPCell(new Paragraph(k.getTitle()));
        setBackgroundColor(k, cell);
        table.addCell(cell);

        cell = new PdfPCell(new Paragraph(k.getDescription()));
        setBackgroundColor(k, cell);
        table.addCell(cell);

        cell = new PdfPCell(new Paragraph(k.getDate().toString()));
        setBackgroundColor(k, cell);
        table.addCell(cell);

        cell = new PdfPCell(new Paragraph(k.getUser().getName() + ", " + k.getUser().
                getSurname()));
        setBackgroundColor(k, cell);
        table.addCell(cell);

        cell = new PdfPCell(new Paragraph(""));
        table.addCell(cell);
    }
}
```

```
    return table;
}

.....
}
```

Listado 5.12: Fragmento de código para la generación de documentos PDF

Cliente

Para configurar los elementos que van a formar las secciones del PDF, se han utilizado paneles para representar cada uno de los tres elementos que componen el PDF (tabla, título y texto) y éstos se irán insertando en los paneles que representan las secciones, de tal modo que se pueden ir configurando dichas secciones de una manera visual e intuitiva.

En este aspecto de configuración de las secciones, cabe destacar el uso de la técnica de *Drag & Drop* (*arrastrar y soltar*), que permite la reordenación de los paneles que ya se han insertado en una sección. De este modo, por ejemplo, si en una sección ya se han incorporado los paneles que representan un título, un texto y una tabla, se puede arrastrar y soltar el panel que representa el texto para colocarlo antes que la tabla. Esto es útil para configurar de manera sencilla el orden en que se quiere que aparezcan los diferentes elementos en las secciones que componen el informe PDF.

En la Figura 1.96 se muestra el diagrama de clases para esta funcionalidad en el sistema cliente.

A continuación, se explican las clases involucradas en el proceso de *Drag & Drop*:

- **panelPDFGeneration:** es el panel que representa la vista de la interfaz gráfica para configurar las secciones del PDF con los elementos que en ellas pueden colocarse. Por tanto, se compone de paneles que representan las secciones, y hace uso de la clase *panelPDFFElement*, para mostrar los elementos que en dichas secciones pueden insertarse.
- **panelPDFFElement:** esta clase representa los elementos que pueden insertarse en las secciones, pero que aún no han sido colocados en el panel que representa una sección.

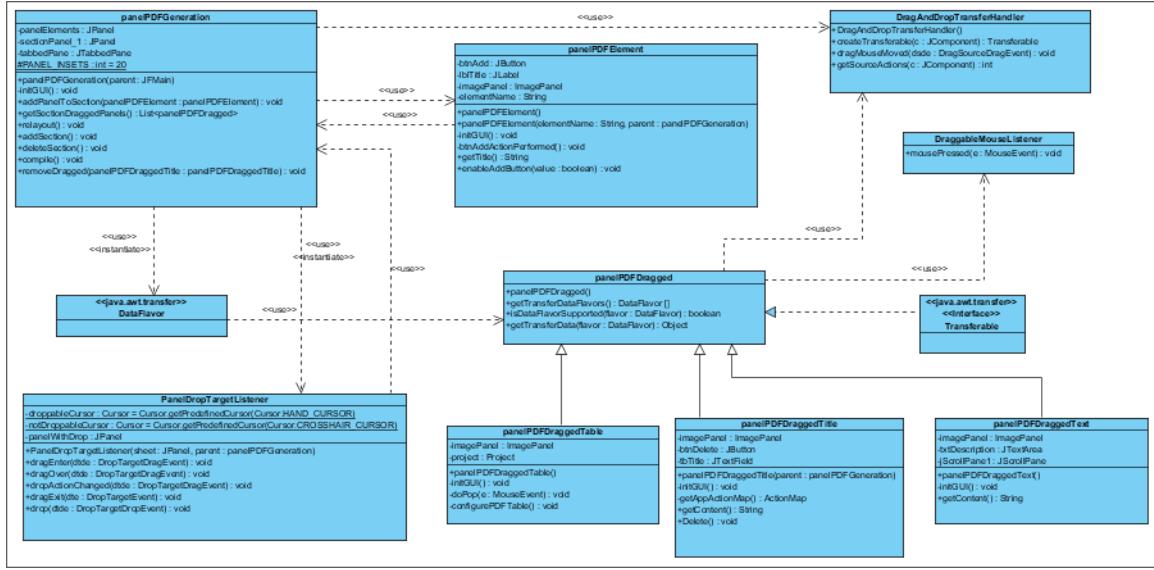


Figura 5.96: Diagrama de clases - Generación de documentos PDF - Cliente

- **panelPDFDragged:** esta clase representa los elementos del PDF que ya han sido insertados en las secciones, y por lo tanto ya tienen una configuración (texto, fuente, proyecto asignado, etc). Esta clase es la superclase de las clases *panelPDFDraggedTitle*, *panelPDFDraggedText* y *panelPDFDraggedTable*, que representan los tres elementos que se utilizan para componer las secciones del PDF. Esta superclase implementa la interfaz *Transferable*.
- **Transferable:** es una interfaz que deben implementar los objetos sujetos a realizar un *Drag*, y que permite conocer cuál es su *DataFlavor* asociado y devolver el objeto correspondiente cuando se realice el evento de *Drop*.
- **DataFlavor:** es una clase de Java, del paquete *java.awt.transfer*, que indica que datos se transmiten cuando el evento del *Drop* se produce.
- **DraggableMouseListener:** es una clase que cuando detecta el evento del ratón *pressed* inicia el proceso de *Drag & Drop*, creando el objeto que implementa la interfaz *Transferable*, que será el origen del *Drag*.
- **DragandDropTransferHandler:** clase controladora del proceso *Drag & Drop* que devuelve el objeto *Transferable* apropiado y controla el modo del proceso (copia o movimiento).

- **DropTargetListener:** es la clase que contiene la lógica para manejar el evento del *Drop*.

Una vez detalladas las clases involucradas en el proceso *Drag & Drop*, éste se detalla:

1. Se muestra la vista de la interfaz gráfica que permite la configuración del informe PDF. En ella, se muestran los paneles que representan las secciones y los paneles que representan los elementos que en ellas se pueden insertar (*panelPDFElement*).
2. Se inserta un elemento a una de las secciones, creándose un objeto de la clase *panelPDFDragged*. Este objeto es el que implementa la interfaz *Transferable* y el que iniciará el evento de *Drag*.
3. Se hace clic con el ratón en uno de los paneles (*panelPDFDragged*) que ya están insertados en una sección. En ese momento, se inicia el proceso de *Drag & Drop*, siendo ese panel el origen del *Drag*. Por tanto, se crea un *DataFlavor*, contenido dicho panel y se asocia con la clase *DragandDropTransferHandler*, en modo copia.
4. El panel se va arrastrando hasta colocarlo en la posición deseada dentro de la sección. Durante este arrastre, la clase *DragandDropTransferHandler* va controlando el objeto *Transferable* que se está moviendo.
5. Cuando se suelta el ratón, se produce el evento *Drop*. En ese momento, se extrae el objeto del *DataFlavor*, que era el panel que se estaba arrastrando y se coloca en la nueva posición dentro del panel de la sección, recolocando el resto de elementos que pudieran estar insertados.

En la Figura 1.97 se muestra visualmente el proceso de *Drag & Drop*, de manera simplificada.

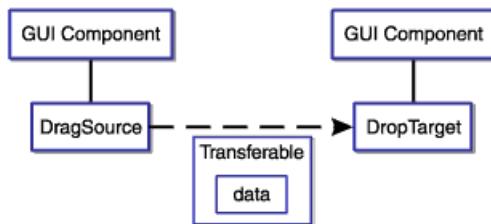


Figura 5.97: Proceso simplificado de *Drag & Drop*

5.3.4.1.3 Pruebas

5.3.4.2 Funcionalidad de *Generación de estadísticas*

5.3.4.2.1 Diagrama de secuencia

En la Figura 1.98 se muestra el diagrama de secuencia para el caso de uso *generar estadísticas* en el cliente.

En este caso, se incluyen los comportamientos de los casos de uso *consultar usuarios*, *consultar proyectos* y *consultar decisiones*, ya definidos, por lo que dichos comportamientos se muestran de manera simplificada en este diagrama de secuencia.

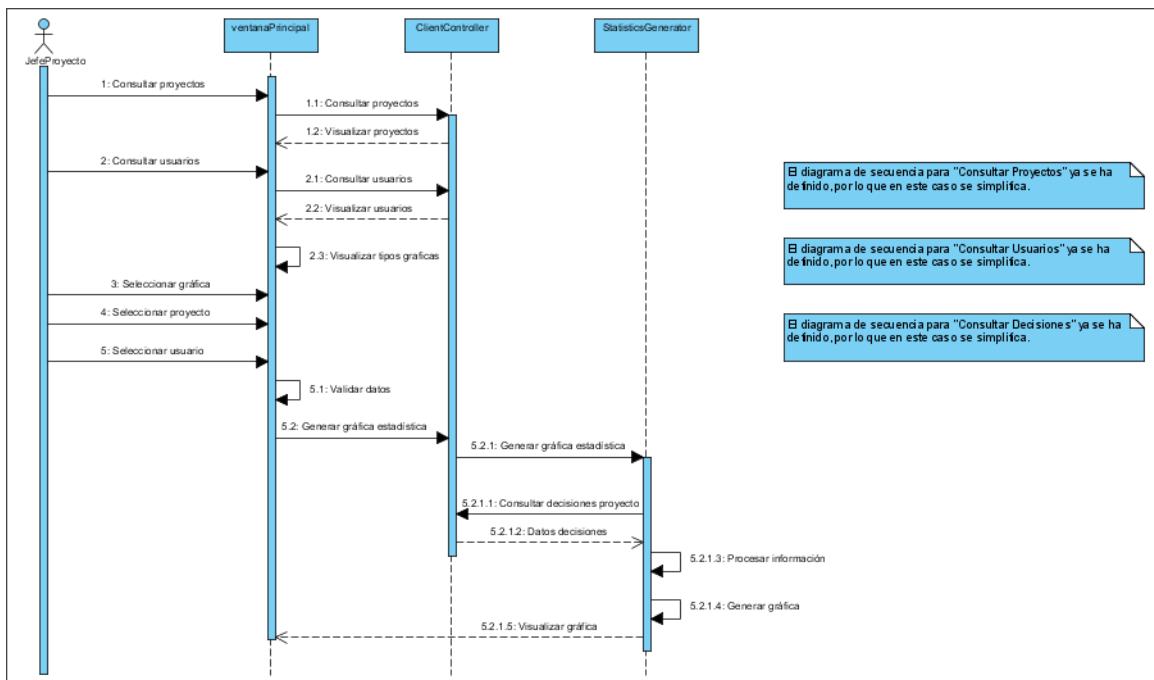


Figura 5.98: Diagrama de secuencia - Cliente - Generar estadísticas

5.3.4.2.2 Diseño e implementación

Como se ha comentado anteriormente, la generación de los gráficos estadísticos es una funcionalidad exclusiva del cliente, ya que los gráficos dependen de la tecnología empleada para representarlos, que en este caso es la librería de gráficos **JFreeChart** (ver sección ??). Por tanto, el cliente, consultando los datos necesarios para crear un gráfico al servidor, es el encargado de generar dichos gráficos y visualizarlos.

En primer lugar, cabe destacar que el tipo de gráficos que pueden generarse y visualizarse están definidos en un fichero XML, por lo que fácilmente se podrán añadir nuevos tipos de gráficos, añadiéndolos en este XML, facilitando su extensibilidad. En este XML se define el tipo de gráfico (de barras, pastel, líneas, etc), un ícono, su nombre y descripción.

Para la creación de los gráficos estadísticos, se hace uso del concepto de *dataset*, que representa el conjunto de datos que se van a representar en un gráfico. En este caso, se han utilizado tres tipos de *dataset*, proporcionados por la librería JFreeChart:

- *DefaultPieDataset*: es el conjunto de datos utilizado para crear y visualizar un gráfico de tipo pastel (o *pie*). Contiene los datos que representan cada una de las porciones del gráfico. Hereda de la clase *AbstractDataSet*.
- *DefaultCategoryDataset*: es el conjunto de datos utilizado para crear y visualizar un gráfico de barras. Contiene los datos que corresponden a un valor del eje X en el eje Y. Hereda de la clase *AbstractDataSet*.
- *CategoryDataset*: es el conjunto de datos utilizado para crear y visualizar un gráfico de líneas. Contiene los datos que corresponden a un valor del eje X en el eje Y. Hereda de la clase *AbstractDataSet*.

En la Figura ?? se muestra el diagrama de clases de diseño para esta funcionalidad, donde se puede observar como la clase controladora de esta funcionalidad (*StatisticsGenerator*) hace uso de estos *datasets*. Además, hace uso de una clase que permite leer y extraer información de archivos XML, utilizada para leer y extraer la información de los gráficos definidos en el fichero XML comentado con anterioridad.

Por tanto, la clase controladora se encarga de realizar las peticiones al servidor para consultar los datos que se necesitan para poder componer los *datasets* necesarios para generar

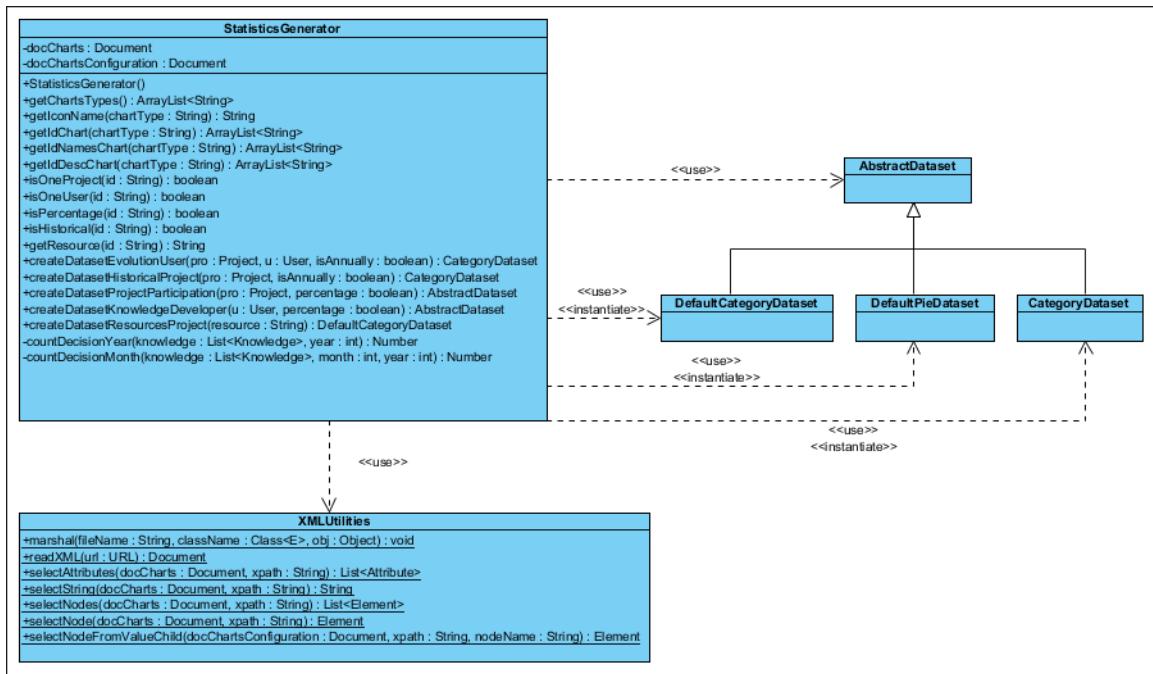


Figura 5.99: Diagrama de clases - Cliente - Generar estadísticas

los gráficos. Los datos consultados dependerán del tipo de gráfico a generar, pudiendo representar la cantidad de decisiones realizadas por un usuario, la cantidad de decisiones realizadas en un proyecto, un histórico de un proyecto, etc. En el fragmento de código 1.13 se muestra un ejemplo de como generar un *dataset* para un diagrama de barras y de pastel que sirve para representar el número de decisiones que un determinado usuario ha realizado en todos los proyectos en los que participa.

```

// Create dataset for the chart of number of knowledge made on each project for that user
public AbstractDataset createDatasetKnowledgeDeveloper(User u, boolean percentage) throws
    RemoteException, NonPermissionRoleException, NotLoggedException, SQLException,
    Exception {
    AbstractDataset dataset = null;
    if (percentage)
        dataset = new DefaultPieDataset();
    else
        dataset= new DefaultCategoryDataset();

    int totalCount = 0;
    int knowledgeUserCount;
    Hashtable<String, Integer> parcial_counts = new Hashtable<String, Integer>();

    // Get all projects
    List<Project> projects = ClientController.getInstance().getProjects();

```

```

List<User> usersInProject;
for (Project p: projects) {
    // Get all the users that work in that project
    usersInProject = ClientController.getInstance().getUsersProject(p);
    if (usersInProject.contains(u)){
        // Get knowledge from user
        TopicWrapper tw = ClientController.getInstance().getTopicsWrapper(p);
        List<Knowledge> knowledgeUser = ClientController.getInstance().getKnowledgeUser(tw
            , u);
        knowledgeUserCount = knowledgeUser.size();
        parcial_counts.put(p.getName(), knowledgeUserCount);
        // Use totalCount in order to calculate the percentage in the case of PieDataSet
        totalCount += knowledgeUserCount;
        // Bar chart case
        if (!percentage)
            ((DefaultCategoryDataset)dataset).addValue(knowledgeUserCount, p.getName(), p.
                getName());
    }
}
// Pie Chart case
if (percentage) {
    for (String projectName: parcial_counts.keySet()) {
        double value = ((parcial_counts.get(projectName) * 100.0) / totalCount);
        ((DefaultPieDataset)dataset).setValue(projectName, value);
    }
}
return dataset;
}

```

Listado 5.13: Fragmento de código para la generación de *datasets*

Para terminar, en el fragmento de código 1.14 se muestra como crear y representar el gráfico estadístico cuando ya se ha generado su *dataset*. Para ello, se hace uso de la clase *ChartFactory* de la librería JFreeChart.

```

private JFreeChart generatePieChart(String title, DefaultPieDataset dataset, boolean
    showLegend) {
    final JFreeChart chart = ChartFactory.createPieChart(
        title,
        dataset,
        showLegend, // legend
        true, // tooltips
        false // URLs
    );
    return chart;
}

```

Listado 5.14: Fragmento de código para la generación de gráficos

5.3.4.2.3 Pruebas

5.3.5 Quinta iteración

Siguiendo los casos de uso de la funcionalidad de *Exportar información* (ver Figura ??) y *Gestión de idiomas* (ver Figura ??), así como el análisis realizado, se abordan las siguientes tareas en esta iteración:

- Diseño de la funcionalidad relativa a la gestión de idiomas y a exportar la información de las decisiones.
- Implementación de dichas funcionalidades.
- Diseño e implementación de pruebas unitarias y funcionales relativas a la gestión de idiomas y a exportar la información de las decisiones.

5.3.5.1 Funcionalidad de *Exportar conocimiento*

5.3.5.1.1 Diagramas de secuencia

En la Figura ?? se muestra el diagrama de secuencia para el caso de uso *Adjuntar ficheros* en el cliente, mientras que en la Figura ?? se muestra el diagrama de secuencia para el servidor.

5.3.5.1.2 Diseño e implementación

Servidor

En el diseño e implementación de esta funcionalidad, cabe destacar la utilización de **JAXB** (ver sección ??) para serializar las clases que componen la jerarquía de decisiones y toda su información relacionada (ver Figura 1.63) a un archivo XML. Para ello, se utilizan anotaciones sobre las clases, indicando que esa clase y sus atributos deben convertirse a un nodo XML. Además, es necesario que exista una clase que represente al nodo raíz del fichero XML y que, por tanto, contenga toda la lista de decisiones. Dicha clase, como ya se ha comentado anteriormente, es la clase llamada *TopicWrapper*.

En el fragmento de código 1.15 se muestran las anotaciones de JAXB realizadas sobre la clase *TopicWrapper*, que engloba a todos los *topics* de un proyecto. En el fragmento de código 1.16 se muestran las anotaciones de JAXB para la clase *Knowledge*, para poder serializar sus

atributos. Del mismo modo se realizan las anotaciones en el resto de clases que componen el diagrama de clases mostrado en la Figura 1.63.

```
/**  
 * This class represents a set of Topics for one Project  
 */  
@XmlRootElement (name = "Topics")  
@XmlAccessorType( XmlAccessType.FIELD )  
public class TopicWrapper implements Serializable {  
  
    /**  
     *  
     */  
    private static final long serialVersionUID = -2825778853241760000L;  
  
    @XmlElement( name = "Topic" )  
    private ArrayList<Topic> topics = new ArrayList<Topic>();
```

Listado 5.15: Anotaciones de JAXB sobre la clase *TopicWrapper*

```
/**  
 * This class represents the knowledge to manage, which can be a Topic, a Proposal or an  
 * Answer.  
 */  
@XmlAccessorType( XmlAccessType.FIELD )  
public abstract class Knowledge implements Serializable {  
  
    private static final long serialVersionUID = -7039151251262020404L;  
  
    protected int id;  
    protected String title;  
    protected Date date;  
    protected String description;  
    @XmlElement protected User user;
```

Listado 5.16: Anotaciones de JAXB sobre la clase *Knowledge*

Cliente

5.3.5.1.3 Pruebas

5.3.5.2 Funcionalidad de *Gestión de idiomas*

5.3.5.2.1 Diagramas de secuencia

En la Figura 1.100 se muestra el diagrama de secuencia para el caso de uso *Adjuntar ficheros* en el cliente, mientras que en la Figura 1.101 se muestra el diagrama de secuencia para el servidor.

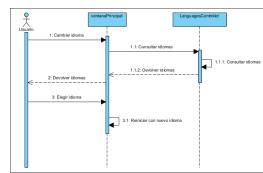


Figura 5.100: Diagrama de secuencia - Cliente - Gestión de idiomas

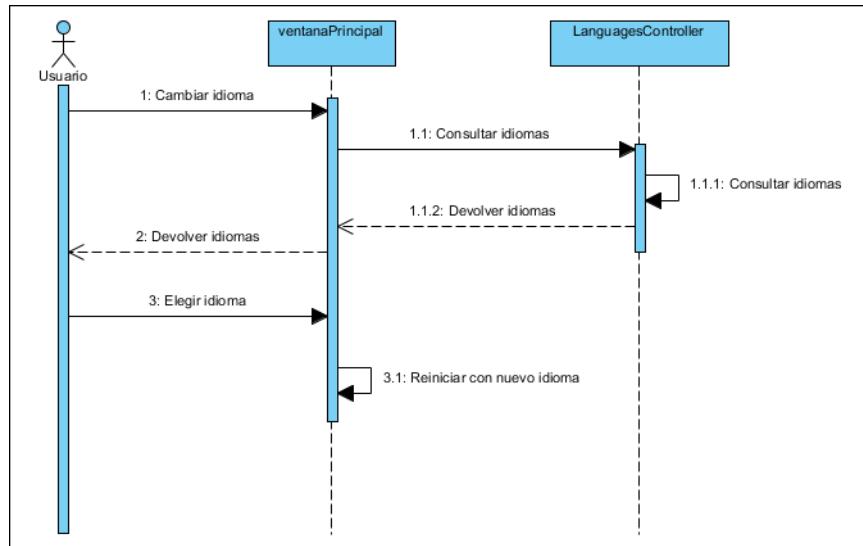


Figura 5.101: Diagrama de secuencia - Servidor - Gestión de idiomas

5.3.5.2.2 Diseño e implementación

Ambos subsistemas, tanto el cliente como el servidor, deben soportar multi-idioma, por lo que se ha diseñado e implementado un gestor de lenguajes. Dicho gestor será el encargado de,

según el idioma elegido en la aplicación, mostrar la diferente información de la interfaz gráfica de usuario en ese idioma. Para ello, se han creado ficheros de propiedades (o *properties*) que contienen las cadenas de texto a internacionalizar, utilizando uno u otro según el idioma escogido. El nombre de dichos ficheros terminan con el código del idioma de cada país, para poder cargar uno u otro según el idioma.

En el fragmento de código 1.17 se muestra cómo utilizar los archivos *properties* para dar soporte al multi-idioma.

```
public class ApplicationInternationalization {

    public static String getString(String key) {
        try {
            // Retrieve the current language
            String languageCode = LanguagesUtilities.getDefaultLanguage().getCode();
            // It indicates the package which contains the "properties" file where it defines
            // the names to be internationalized
            ResourceBundle RESOURCE_BUNDLE = ResourceBundle.getBundle("internationalization.
                app_" + languageCode);
            return RESOURCE_BUNDLE.getString(key);
        } catch (MissingResourceException e) {
            return '!' + key + '!';
        } catch (JDOMException e) {
            return '!' + key + '!';
        } catch (IOException e) {
            return '!' + key + '!';
        }
    }
}
```

Listado 5.17: Soporte multi-idioma

Para terminar, señalar que los idiomas disponibles para configurar la aplicación se encuentran un fichero XML, para que no sea necesario una conexión a base de datos y se pueda modificar el idioma de la aplicación sin tener que acceder previamente a dicha base de datos. Además, se facilita la extensibilidad y adición de nuevos idiomas, ya que sólo abría que crear su fichero de propiedades y agregar ese idioma al fichero XML.

5.3.5.2.3 Pruebas

Capítulo 6

Conclusiones y Propuestas

Bibliografía
