



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

**INGENIERÍA
EN INFORMÁTICA**

PROYECTO FIN DE CARRERA

**DESGLOSA: Un sistema de visualización 3D para dar
soporte al Desarrollo Global de Software**

Jose Domingo López López

Febrero, 2012



UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

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

PROYECTO FIN DE CARRERA

**DESGLOSA: Un sistema de visualización 3D para dar
soporte al Desarrollo Global de Software**

Autor: Jose Domingo López López

Directora: M^a Ángeles Moraga de la Rubia

Febrero, 2012

TRIBUNAL:

Presidente: _____

Vocal 1: _____

Vocal 2: _____

Secretario: _____

FECHA DE DEFENSA: _____

CALIFICACIÓN: _____

PRESIDENTE

VOCAL 1

VOCAL 2

SECRETARIO

Fdo.: _____

Fdo.: _____

Fdo.: _____

Fdo.: _____

Resumen

La globalización es un *fenómeno que está afectando a aspectos económicos, tecnológicos, sociales, políticos y culturales que se dan entre las personas, empresas y gobiernos de todo el mundo*. Esto provoca que la industria y el comercio, entre otros, rompan con las fronteras nacionales y se expandan a todo el mundo con el objetivo de llegar a un mayor público y reducir costes.

La industria del software no es una excepción y también se está amoldando a las nuevas modas que se imponen con el paso del tiempo. Por ello, el software está pasando de ser desarrollado en una única ubicación a ser desarrollado en múltiples ubicaciones dispersas por todo el mundo. Este concepto recibe el nombre de *Desarrollo Global de Software* (DSG).

El DSG aporta una gran cantidad de ventajas a las empresas que desarrollan software bajo este contexto. No obstante, también introduce una serie de factores que pueden afectar negativamente a aspectos de calidad y productividad, entre otros, si no se controlan adecuadamente. Para ello, son necesarios entornos y herramientas que ayuden a asegurar la calidad del software y a mitigar el efecto de la distribución de recursos en el proceso de desarrollo y en la gestión del conocimiento.

En este Proyecto Fin de Carrera se propone desarrollar una herramienta que facilite la gestión organizacional, en el contexto del desarrollo global, y el seguimiento de los proyectos globalizados. Así pues, se dará soporte a las distintas actividades que se llevan a cabo en este tipo de escenarios y se ofrecerán medios para la representación gráfica de información relevante, entre los que destaca la visualización de medidas e indicadores de calidad y productividad utilizados en este paradigma. Gracias a ello será posible predecir y detectar distintos riesgos y anomalías, así como ofrecer ayuda en la toma de decisiones de las actividades de la organización, con el objetivo de incrementar la competitividad de las organizaciones a nivel internacional.

Abstract

Globalization is defined as *a phenomenon that affects economic, technological, social, and political factors among people, corporations, and governments around the world*. This makes industry and commerce, among others, break national frontiers and get expanded worldwide in order to reach a wider public and to reduce expenses.

Software industry is not an exception and it is getting adapted to the new fashions coming with time. Therefore, software is being developed not in just a single location but in multiple places around the world. This concept is called **Global Software Development** (GSD).

GSD provides many advantages to those companies developing software in this context. However, it also introduces a set of factors that could negatively affect aspects such as quality and productivity, if they are not properly managed. Hence, GSD requires tools and environments that make it possible to guarantee software quality and to mitigate the impact of resource distribution in the development process and knowledge management.

The system developed throughout this project is a tool that facilitates organizational management, in the context of global development, and the monitoring of globalized projects. Thus, the various activities carried out in this type of scenarios will be supported and means for the graphical representation of outstanding information will be offered, such as displaying quality and productivity measures used in this paradigm. Consequently, it will be possible to predict and detect different risks and anomalies, as well as to support in decision making in organizational activities, with the aim of increasing the worldwide competitiveness of companies.

*A papá y a mamá.
Porque todo es posible gracias a vosotros.*

*A mi hermano.
Porque estoy orgulloso de ti y eres un ejemplo a seguir.*

Agradecimientos

Son muchas las personas que he conocido a lo largo de mi vida a las que tendría que agradecerles multitud de cosas que han hecho posible alcanzar este hito. Desde aquellas responsables de los buenos momentos, como las responsables de los no tan buenos, ya que me han servido de experiencia y de ello he aprendido. Sin embargo, en este libro me dirijo a aquellos que, de algún modo, más calor me habéis regalado desde que entrasteis en mi vida.

A Félix Óscar García y Manuel Ángel Serrano, que tanto tiempo habéis invertido en sacar ideas para que todo esto salga adelante. A M^a Ángeles Moraga, por el trabajo realizado con Félix y Manuel, por brindarme la oportunidad de realizar este trabajo, por confiar en mí y por el esfuerzo invertido en guiarme durante la elaboración de este libro y en su revisión. A Ismael Caballero, porque no sólo es un buen profesor, sino también un gran amigo que ha sabido en qué momento decir las palabras oportunas para levantarme el ánimo. A Félix Úbeda, por ofrecerme unos magníficos años de convivencia y demostrarme que es un excelente amigo. A Juan Andrada y Juan José Antequera, por estar ahí siempre que se necesita un amigo y por ser unas de las personas más trabajadoras que he conocido. Habéis convertido mis últimos años de carrera en un constante proceso de superación. A Francisco José Oteo, alias iBoy, y Jose Luis Hernández, que tantas horas de pesca hemos compartido para liberarnos del estrés que nos ha generado la carrera. Al grupo de investigación Alarcos y todos sus miembros, por mantener un ambiente de trabajo estupendo. A Ana García, por todo el apoyo me has dado en este último sprint y aguantarme en una de las épocas más estresantes de mi vida. A Maribel, por hacernos felices a mí y a los míos y entender mis sentimientos con sólo mirarme. A mi hermano, porque juntos hemos salido de lo peor y no habrá nada que nos tumbe si nos mantenemos unidos. A Papá, por preocuparse por mí como nadie lo hará y haberme enseñado a ser tan perseverante.

A Mamá, porque nunca me olvidaré de ti.

Índice general

Índice de figuras	xviii
Índice de tablas	xxii
Índice de listados	xxiii
1 Introducción	1
1.1 Estructura del documento	5
2 Motivación y Objetivos del Proyecto	7
2.1 Motivación	8
2.2 Objetivos	9
2.2.1 Objetivo principal	9
2.2.2 Objetivos parciales	12
2.2.3 Limitaciones	13
3 Estado del Arte	15
3.1 Desarrollo Global de Software (DGS)	15
3.1.1 Ventajas del Desarrollo Global de Software	17
3.1.2 Desafíos del Desarrollo Global de Software	19
3.2 Calidad y medidas software	21
3.3 Visualización de medidas software	26
3.4 Gráficos por computador y Librerías de gráficos 3D	31
3.4.1 Direct3D	33
3.4.2 OpenGL	34
4 Método de Trabajo	39
4.1 Marco de investigación preliminar	39
4.2 Proceso Unificado de Desarrollo (PUD)	39
4.2.1 Fases del Proceso Unificado de Desarrollo	41
4.2.2 Disciplinas del flujo de trabajo fundamental	43
4.3 Patrones de diseño	44
4.4 Marco tecnológico	47
4.4.1 Herramientas de gestión de proyectos	47
4.4.2 Herramientas de modelado de software y elaboración de documentación	49
4.4.3 Herramientas, tecnologías y frameworks de desarrollo software . . .	51

5 Resultados	61
5.1 Fase de inicio	62
5.1.1 Captura de requisitos	62
5.1.2 Identificación de requisitos	67
5.1.3 Modelo de Casos de Uso	71
5.1.4 Glosario de términos	74
5.1.5 Gestión del riesgo	75
5.1.6 Plan de iteraciones	76
5.2 Fase de elaboración	88
5.2.1 Iteración 1	89
5.2.2 Iteración 2	95
5.2.3 Iteración 3	104
5.3 Fase de construcción	110
5.3.1 Iteración 4	110
5.3.2 Iteración 5	115
5.3.3 Iteración 5 (repetición)	121
5.3.4 Iteración 6	124
5.3.5 Iteración 7	131
5.3.6 Iteración 8	146
5.3.7 Iteración 9	154
5.4 Fase de transición	154
5.4.1 Iteración 10	155
6 Conclusiones y Propuestas	157
6.1 Aspectos destacables de la solución propuesta	157
6.2 Trabajo actual y futuras mejoras	159
6.3 Conocimientos adquiridos	160
Bibliografía	162
A Informes de pruebas y cobertura de código	169
B Manual de Instalación y Despliegue	173
B.1 Instalación de las librerías de JOGL en el repositorio local de Maven	173
B.2 Construcción de los proyectos	174
C Manual de Usuario	177
C.1 Acceso al sistema	177
C.2 Visualización	180
C.2.1 Visualización mediante el Sistema de Información Geográfica	180
C.2.2 Visualización de medidas mediante gráficos en 3D	182
C.3 Perfiles de visualización	191
C.3.1 Configuración de perfiles de visualización	192
C.4 Gestión de la estructura organizacional del DGS	199
C.4.1 Gestión de factorías software	199
C.4.2 Gestión de medidas	202

D Código Fuente	205
D.1 Configuración de Spring Security	205
D.2 Automatización de pruebas funcionales de una aplicación web mediante Maven	208

Índice de figuras

2.1	Esquema de la estructura organizacional de las entidades que intervienen en el desarrollo global de software	11
3.1	Mapa del mundo que refleja la dispersión geográfica de factorías involucradas en desarrollo global [4]	16
3.2	Tira de cómic que refleja las necesidades de un producto desde el punto de vista de distintos <i>stakeholders</i> . De izquierda a derecha y de arriba a abajo representa: cómo lo explicó el cliente, cómo lo entendió el jefe de proyecto, cómo lo diseñó el analista, cómo lo desarrolló el programador, cómo lo describió el consultor, cómo fue documentado el proyecto, lo que fue instalado, por lo que pagó el cliente, cómo fue el soporte, y lo que el cliente realmente necesitaba	23
3.3	Características y subcaracterísticas del modelo de calidad del software propuesto por el estándar ISO/IEC 9126 [67]	24
3.4	Relación entre las normas ISO/IEC 9126 (modelo de calidad del producto software) e ISO/IEC 1498 (evaluación del producto software) [67]	25
3.5	(a) Nodo de una vista polimétrica en la que se pueden representar hasta siete medidas. (b) Vista polimétrica aplicada una jerarquía de herencia entre clases. Cada clase es representada por un nodo en el que el ancho muestra el número de atributos de la clase, el alto indica el número de métodos y el color el número de líneas de código. [13]	28
3.6	Ejemplo del nivel de ruido introducido por las asociaciones entre entidades [63]	28
3.7	Ejemplo que activa un proceso preatento. En la imagen de la izquierda el objetivo se encuentra fácilmente; en la de la derecha, el objetivo se camufla entre el resto de elementos	29
3.8	Metáfora de una ciudad para representar sistemas software [66]	30
3.9	Representación visual de medidas software de un sistema. Cada cubo representa una parte del sistema. El color verde podría significar un <i>valor aceptable</i> de una medida definida; el color azul indicaría un <i>valor marginal</i> de dicha medida; y el color rojo representaría un <i>valor inaceptable</i> [63]	31
3.10	Pila de recursos que intervienen en la generación de gráficos	32
3.11	Colección de API de DirectX	33
3.12	Ejemplo renderizado con OpenGL que muestra texturas, sombras y reflejos [31]	35
4.1	Mapa conceptual del PUD	41
4.2	Grado de participación de las disciplinas del flujo de trabajo fundamental en las iteraciones del PUD [45]	45
4.3	Captura de pantalla de la herramienta MySQL Workbench 5.2 CE	50

4.4	Captura de pantalla de la herramienta Visual Paradigm 8.0	50
5.1	Prototipo a mano alzada para diseñar el modelo gráfico de una factoría software	64
5.2	Prototipo a mano alzada para diseñar la el modelo gráfico de un producto software	65
5.3	Prototipo a mano alzada para diseñar un modelo gráfico genérico mediante una torre	66
5.4	Esquema de las asociaciones o <i>mapeos</i> entre los atributos de una entidad y las dimensiones de un modelo gráfico	67
5.5	Prototipo a mano alzada para diseñar la navegación entre las distintas metáforas de visualización	68
5.6	Modelo de casos de uso de la aplicación web	72
5.7	Modelo de casos de uso del motor gráfico	72
5.8	Gráfica de líneas que representa la duración de cada fase en iteraciones . . .	88
5.9	Gráfica de barras que representa el grado de participación de las iteraciones en cada disciplina	88
5.10	Gráfica de áreas que representa el grado de participación de las iteraciones en cada disciplina	89
5.11	Diagrama de casos de uso para un mini-proyecto de funcionalidad reducida, basado en JOGL	90
5.12	Diagrama de clases de análisis para un mini-proyecto de funcionalidad reducida, basado en JOGL	91
5.13	Diagrama de comunicación para un mini-proyecto de funcionalidad reducida, basado en JOGL	91
5.14	Diagrama de clases de diseño para un mini-proyecto de funcionalidad reducida, basado en JOGL	92
5.15	Diagrama de secuencia de la inicialización del hilo de JOGL	93
5.16	Diagrama de secuencia de ejecución del hilo de JOGL	93
5.17	Resultado de la disciplina de implementación de la iteración 1	95
5.18	Diagrama de casos de uso para CdU1.1 y CdU1.2	96
5.19	Esquema de los vectores directores que mantienen la orientación de la cámara	99
5.20	Cálculo de los vectores directores de orientación de la cámara después de una operación de rotación. Los vectores <i>arriba</i> ₀ y <i>arriba</i> ₁ no se visualizan ya que se consideran perpendiculares y en sentido saliente al papel	99
5.21	Diseño de las clases GLCamera y Vector3f	100
5.22	Diagrama de clases de diseño para CdU1.1 y CdU1.2	101
5.23	Diagrama de clases de diseño para CdU1.5	101
5.24	Diagrama de clases de la anotación GLDimension y su correspondiente analizador	102
5.25	Modelo Entidad-Relación Extendido	106
5.26	Diagrama de clases de conocimiento para el contexto del desarrollo global .	107
5.27	Diagrama de clases de la anotación Property y su correspondiente analizador	108
5.28	Estructura de un proyecto Maven con arquetipo <i>struts2-archetype-starter</i> .	109
5.29	Esquema conceptual para el posicionamiento de los modelos gráficos en CdU1.5	111
5.30	Diagrama de clases para CdU1.5 (1/2)	112
5.31	Diagrama de clases para CdU1.5 (2/2)	113
5.32	Gráficos generados después de la disciplina de implementación de la cuarta iteración	114

5.33 Diagrama de comunicación para CdU1.3 y 1.7	116
5.34 Diagrama de flujo para CdU1.4	117
5.35 Diagrama de clases de diseño para CdU1.3 y CdU1.7	118
5.36 Diagrama de secuencia para CdU1.3 y 1.7	119
5.37 Diagrama de componentes del sistema	120
5.38 Gráficos generados después de la disciplina de implementación de la sexta iteración: a la izquierda sin sombras ni <i>skybox</i> ; a la derecha con ambas características	123
5.39 Gráficos generados después de la disciplina de implementación de la sexta iteración. De izquierda a derecha se muestran las metáforas de ciudad y polígono industrial y una representación de proyectos software	123
5.40 Diagrama de casos de uso para CdU2.1 (Iniciar sesión)	125
5.41 Diagrama clases de análisis para CdU2.1 (Iniciar sesión)	126
5.42 Diagrama de comunicación para CdU2.1 (Iniciar sesión)	127
5.43 Diagrama de secuencia para CdU2.1 (Iniciar sesión)	127
5.44 Diagrama de clases de diseño para CdU2.1 (GBAC mediante Spring Security)	129
5.45 Modelo Entidad-Relación Extendido para CdU2.1 (GBAC mediante Spring Security)	130
5.46 Captura de pantalla de la integración de Google Maps en la aplicación web	131
5.47 Diagrama de casos de uso para CdU2.2	132
5.48 Diagrama de comunicación para CdU2.2 (Crear perfil de visualización)	133
5.49 Prototipo de la interfaz gráfica para CdU2.2 (Crear perfil de visualización)	133
5.50 Diagrama de paquetes de la aplicación web	135
5.51 Diagrama de clases de dominio para CdU2.2	137
5.52 Diagrama de clases de la arquitectura de pruebas unitarias	141
5.53 Estructura de los casos de prueba para CdU2.1	145
5.54 Informe de pruebas para CdU2.1 (Vista global)	147
5.55 Informe de pruebas para CdU2.1 (Caso de prueba 1 -Tabla 5.24-)	148
5.56 Informe de pruebas para CdU2.1 (Caso de prueba 4 -Tabla 5.27-)	149
5.57 Diagrama de casos de uso para CdU2.3 y CdU2.4	150
5.58 Diagrama de comunicación para CdU2.3 (Crear compañía)	150
5.59 Diagrama de comunicación para CdU2.3 (Ver compañía)	151
5.60 Diagrama de secuencia para CdU2.3 (Crear compañía)	151
5.61 Diagrama de secuencia para CdU2.3 (Ver compañía)	152
5.62 Resultado de procesar la cadena de texto en formato JSON (Listado 5.12)	153
5.63 Captura de pantalla de un fragmento de la página que permite consultar información de una factoría de software	155
A.1 Informes generados mediante Maven	169
A.2 Fragmento del informe de resultados de la ejecución de pruebas	170
A.3 Informe de cobertura de código del paquete <i>action</i>	171
A.4 Informe de cobertura de código del paquete <i>control</i>	171
A.5 Informe de cobertura de código del paquete <i>model.util</i>	171
C.1 Página de inicio de la aplicación web	178
C.2 Formulario de autenticación	179
C.3 Errores obtenidos en el proceso de autenticación: (a) El usuario o la contraseña introducidos son incorrectos (b) La cuenta de usuario ha sido deshabilitada (c) La cuenta de usuario ha sido bloqueada (d) La cuenta de usuario ha caducado	179

C.4	Página de inicio para un usuario autenticado. En la parte inferior izquierda se muestra su nombre de usuario y en la parte superior derecha los menús configurados para su rol	180
C.5	Página de visualización basada en Google Maps	181
C.6	Resultado de seleccionar la <i>compañía Compañía 1</i> en el filtro organizacional	182
C.7	Panel que muestra información detallada del proyecto seleccionado	183
C.8	Pestaña de <i>información global</i> que ofrece acceso a la visualización de datos globalizados	183
C.9	Pestaña <i>compañía</i> del panel <i>más información</i> que ofrece acceso a la visualización de datos globalizados	184
C.10	Mensaje de confirmación para la instalación de librerías de JOGL y librerías nativas	184
C.11	Ventana de selección de perfil y método de agrupación para la visualización de medidas e indicadores a nivel de compañía	185
C.12	Visualización de compañías mediante la metáfora de un polígono industrial. Cada industria representa una compañía	186
C.13	Componente que muestra los controles que pueden emplearse para interaccionar con la escena 3D	187
C.14	Visualización, mediante la metáfora de un polígono industrial, de los proyectos en los que trabajan las factorías de la compañía <i>Compañía 1</i>	188
C.15	Visualización, mediante un modelo diseñado para representar proyectos software, de los proyectos en los que participa una factoría determinada	189
C.16	Visualización, mediante la metáfora de ciudad, de los subproyectos que conforma un proyecto. La agrupación se muestra en base a la factoría que desarrolla cada subproyecto	190
C.17	Visualización, mediante la metáfora de ciudad, de todos los subproyectos dados de alta en el sistema. La agrupación se muestra en base a al proyecto al que pertenecen	191
C.18	Panel de control para la gestión de perfiles de visualización	192
C.19	Primer paso del asistente de configuración de perfiles de visualización . . .	193
C.20	Ejemplo para la configuración de reglas mediante las cuales se asignan colores a valores booleanos	194
C.21	Ejemplo para la configuración de reglas mediante las cuales se asignan valores en punto flotante a intervalos de números enteros	197
C.22	Segundo paso del asistente de configuración de perfiles de visualización . .	198
C.23	Panel de control para la gestión de factorías software	200
C.24	Fragmento de la página de consulta de una factoría software	200
C.25	Mensaje de confirmación para la eliminación de una factoría software	201
C.26	Primer paso del asistente de creación de una factoría software	201
C.27	Segundo paso del asistente de creación de una factoría software	202
C.28	Tercer paso del asistente de creación de una factoría software	203
C.29	Panel de control para la gestión de proyectos software	203
C.30	Página para la consulta de medidas de un proyecto software	204
C.31	Formulario para la configuración de valores de las medidas de un proyecto software	204

Índice de tablas

3.1 Clasificación de desarrollo de software según dispersión geográfica [41]	16
3.2 Modelos y variantes de Desarrollo Global de Software	17
5.1 Mapeo inicial de una factoría software con el modelo gráfico de factoría software	63
5.2 Mapeo inicial de un proyecto software con el modelo gráfico de un proyecto software	65
5.3 Mapeo inicial de un subproyecto software con el modelo gráfico de una torre	66
5.4 Requisitos funcionales del sistema	69
5.5 Roles identificados en la aplicación web	70
5.6 Roles identificados en el motor gráfico	70
5.7 Denominación y priorización de los casos de uso	74
5.8 Ficha de la iteración preliminar	77
5.9 Ficha de la iteración 1	78
5.10 Ficha de la iteración 2	79
5.11 Ficha de la iteración 3	80
5.12 Ficha de la iteración 4	81
5.13 Ficha de la iteración 5	82
5.14 Ficha de la iteración 6	83
5.15 Ficha de la iteración 7	84
5.16 Ficha de la iteración 8	85
5.17 Ficha de la iteración 9	86
5.18 Ficha de la iteración 10	87
5.19 Descripción del caso de uso CdU1.1	97
5.20 Descripción del caso de uso CdU1.2	98
5.21 Informe de pruebas para CdU1.1	115
5.22 Descripción del caso de uso CdU2.1 (Iniciar sesión)	126
5.23 Descripción del caso de uso CdU2.2 (Crear perfil de visualización)	132
5.24 Descripción del caso de prueba 1 para CdU2.1	145
5.25 Descripción del caso de prueba 2 para CdU2.1	145
5.26 Descripción del caso de prueba 3 para CdU2.1	146
5.27 Descripción del caso de prueba 4 para CdU2.1	146
C.1 Compatibilidades entre tipos de atributo de las entidades y tipos de dimensión de los modelos gráficos	196
C.2 Tipos de atributos y dimensiones incompatibles	198

Índice de listados

4.1	Ejemplo de configuración de un fichero <i>pom.xml</i> [52]	48
4.2	Ejemplo de estructuras condicionales con JSTL	56
5.1	Configuración de pom.xml para añadir JOGL como dependencia del proyecto	94
5.2	Ejemplo de texto en formato JSON (http://json.org/example.html)	102
5.3	Fragmento de código Java de la clase GLCamera	103
5.4	Código fuente de la anotación personalizada Property	110
5.5	Ejemplo de clase Java anotada con la anotación GLDimension	113
5.6	Fragmento de código Java para la implementación de CdU1.4	120
5.7	Ejemplo de un perfil de visualización en formato XML	136
5.8	Código fuente del fichero login.jsp para autenticación mediante Spring Security	138
5.9	Fichero de configuración de struts-menu (menu-config.xml)	139
5.10	Configuración de pom.xml para añadir JUnit como dependencia del proyecto	141
5.11	Configuración de pom.xml para añadir Cobertura como plugin de generación de informes	143
5.12	Fragmento de la representación en formato JSON de una instancia de la clase City	152
B.1	Comandos para la instalación de librerías en el repositorio local de Maven .	174
D.1	Fichero de configuración de Spring Security: applicationContext-security.xml	205
D.2	Snippet para la automatización de pruebas funcionales de la aplicación web mediante Maven	208

Capítulo 1

Introducción

Durante las últimas décadas, se ha dado un fenómeno denominado *globalización*. La **globalización**, según la Real Academia Española, es la “*tendencia de los mercados y de las empresas a extenderse, alcanzando una dimensión mundial que sobrepasa las fronteras nacionales*” (DRAE 2006, 23^a edición). Existen otras definiciones más amplias que no se limitan únicamente a hablar de *mercados* y *empresas*, abordando este concepto desde un punto de vista más genérico y definiéndolo como “*un proceso económico, tecnológico, social, político y cultural de interacción e integración entre personas, empresas y gobiernos de todo el mundo*” [10].

Si se analiza la evolución del mundo con el paso de los años, se puede ver que la globalización está presente en -prácticamente- todas las áreas de la vida humana: la competencia por un mismo mercado a escala global entre pequeñas empresas y grandes multinacionales, la forma en que las empresas colaboran entre sí para llegar a un mayor grupo de clientes, los medios utilizados para la transacción de bienes y servicios entre distintos países, cómo fluye la información en tiempo real cuando sucede un acontecimiento importante, etcétera. Estos ejemplos corroboran, tal y como se pautualiza en la segunda definición, que, efectivamente, la globalización afecta a ámbitos sociales, tecnológicos, políticos y culturales, entre otros.

Según Thomas Friedman [20], esta situación es posible gracias a un conjunto de acontecimientos que han logrado la convergencia y la globalización del mundo, contribuyendo así con la aparición de nuevos modelos políticos, sociales y comerciales. Entre estos acontecimientos destacan los lanzamientos de Microsoft Windows 95 y Netscape, el primer navegador web, ya que desde ese momento los usuarios disponen de un sistema operativo y un medio de acceso a Internet. Esto despierta la necesidad de investigar protocolos y estándares que permitan la comunicación entre ordenadores a través de la red, como el protocolo SMTP (*Simple Mail Transfer Protocol*) utilizado para mensajería o el lenguaje de marcas HTML (*HyperText*

Markup Language) utilizado en elaboración de páginas web.

Poco después surge el **open sourcing**. Este movimiento se alza con la aparición de comunidades de usuarios y profesionales que crean proyectos de código abierto, en ocasiones gratuitos, que permiten que cientos de personas puedan acceder a esos recursos con el fin de aprender, colaborar y aportar su grano de arena. Apache, GNU y Wikipedia, entre otros, son algunos de los ejemplos más destacados. Friedman denomina este instante como el *origen de la tierra plana* porque es el momento a partir del cual es posible trabajar simultáneamente en numerosos proyectos y con personas de todo el mundo.

Acto seguido aparece otro movimiento denominado **outsourcing**. Este modelo permite que las compañías realicen subcontrataciones y deleguen parte de sus servicios en otras empresas para ahorrar costes y obtener una mano de obra especializada. De esta manera es posible llegar a un mayor grupo de clientes adaptando los productos a sus necesidades y ofreciendo precios más competitivos.

A partir del *outsourcing* nace el concepto del **offshoring**. En este caso las compañías no sólo trasladan parte de sus servicios a otras empresas, sino que trasladan fábricas enteras a otros países con el fin de abaratar costes de mano de obra o adquisición de materias primas.

Las sucesivas investigaciones que se realizan para lograr ser más competitivos dan lugar al **insourcing**. El *insourcing*, del mismo modo que el *outsourcing*, consiste en delegar un conjunto de tareas o servicios en empresas subcontratadas pero en este caso el trabajo será realizado en las propias instalaciones de la empresa contratante. Así, la empresa contratante mantiene un control sobre el proceso, que es asumido y gestionado por completo por la compañía subcontratada.

Otro acontecimiento al que Friedman hace especial mención es la **producción y distribución en cadena**. Este proceso, cuya base es la cadena de montaje de la producción industrial, delega el desempeño de tareas específicas a empresas o personas especializadas y cualificadas. Así pues, se da un intercambio de productos, información y servicios entre distintas empresas y proveedores para satisfacer las necesidades del cliente.

Por último, destacan los acontecimientos que Friedman denomina, que son los avances tecnológicos que han permitido transformar contenidos analógicos en formatos digitales, y el **in-forming**, que es el término utilizado para designar la búsqueda de conocimiento en Internet y utilizar la red como herramienta para la investigación **esteroides** [20].

Todos estos acontecimientos, que se encuentran íntimamente ligados al crecimiento y

evolución que sufren las tecnologías y telecomunicaciones, tienen por objetivo la *ruptura* de las *barreras o fronteras nacionales* para abrirse al mundo. Esto afecta no sólo a aspectos políticos, sociales y culturales, sino que tiene una gran importancia en cómo conocemos la industria y el comercio de hoy en día.

El modo en que la globalización ha afectado a la industria y el comercio en general, y a la **industria y el comercio del software** en particular, no sólo ha tenido impacto en la venta y distribución de los productos, que han tenido que ser adaptados a las necesidades lingüísticas y culturales del usuario final, sino también en la manera en que se conciben, diseñan, construyen y prueban [25]. Esto se debe a que el software no se desarrolla por un grupo de personas ubicado en la misma oficina, sino que es llevado a cabo por varios equipos de trabajo que se encuentran dispersos geográficamente. En otras palabras, el software se desarrolla de forma distribuida en distintos países.

Este tipo de desarrollo distribuido, que recibe el nombre de **Desarrollo Global de Software** (DGS o GSD, en su acrónimo inglés Global Software Development) [25] proporciona un conjunto de beneficios, entre los que destacan los siguientes:

- Acceso a profesionales y mano de obra especializada a lo largo de todo el mundo, ahorrando en tiempo y coste del traslado de esas personas [11].
- Mejoras en la productividad dada la posibilidad de extender las jornadas de trabajo, cuando los desarrolladores se encuentran distribuidos en sitios con amplia diferencia horaria [25].
- Compartición de experiencias, conocimiento técnico y destrezas de los stakeholders distribuidos [42].
- Mayor proximidad al cliente, lo que facilita la adaptación del producto a sus necesidades y a las leyes y políticas locales [9].

Por estas razones, entre otras, la tendencia es a llevar a cabo la fabricación y el desarrollo de software de manera globalizada. Sin embargo, la globalización presenta en la práctica una serie de problemas relacionados con la dificultad de comunicación y gestión del conocimiento, ya que durante el desarrollo del software se genera gran cantidad de información que proviene de diferentes fuentes y etapas y que a menudo no se almacena y actualiza convenientemente [7].

Además, la distribución de recursos que se lleva a cabo en este tipo de desarrollos, cuando se realizan de forma globalizada, trae consigo numerosos problemas operacionales derivados, entre otros, de las diferencias horarias, culturales e incluso de conocimiento tecnológico y organizacional entre distintos centros u oficinas, además de otros factores como las dificultades físicas en las comunicaciones y el trabajo colaborativo [33].

También es común que ciertos centros de desarrollo funcionen muchas veces como entes prácticamente independientes, con escasa comunicación con el resto de la organización, tratando con clientes locales. Esto evidencia la necesidad de homogeneizar el funcionamiento de todas las factorías de software de la organización y la comunicación entre ellas. Este problema cobra más relevancia si en el desarrollo distribuido entran en juego factorías pertenecientes a distintas organizaciones, cada una con sus metodologías y procesos de negocio particulares [30].

Para evitar que surjan determinados problemas relacionados con el DGS o mitigar sus efectos en caso de aparición, es recomendable utilizar entornos y herramientas adaptadas al contexto del desarrollo global, que permitan **aumentar la productividad** de las actividades de desarrollo software en escenarios globales y **asegurar la calidad** del software desarrollado bajo este paradigma. Para ello, es clave sintetizar la gran cantidad de información con la que cuentan los profesionales e investigadores a cargo de este tipo de actividades y visualizarla gráficamente de la forma más apropiada de acuerdo a sus necesidades. Esta representación visual facilitará las tareas de análisis y gestión, y ayudará en la toma de decisiones. De este modo, se podrá determinar cuáles son las factorías más apropiadas para llevar a cabo una determinada tarea, conocer qué ocurre durante el ciclo de vida del software, predecir posibles situaciones de riesgo para tomar las mejores decisiones, etcétera [17].

Mediante este **Proyecto Fin de Carrera**, en adelante PFC, se pretende desarrollar un sistema de visualización orientado al paradigma del DGS, que facilite la gestión organizacional y el seguimiento de los proyectos *globalizados*. De esta forma, la herramienta permitirá que cada usuario, en función de su cargo o *rol*, obtenga información relevante para llevar a cabo correctamente sus actividades. Cuestiones como la productividad de las factorías de software, las colaboraciones con otras empresas o las incidencias que han surgido a lo largo del ciclo de vida de un proyecto, pueden ser de vital importancia en este contexto. Toda esta información, que deberá ser definida y suministrada por un agente externo, será representada gráficamente mediante el uso de distintas técnicas de visualización, tratando de proporcionar un entorno

intuitivo, eficiente y efectivo.

Finalmente, cabe destacar que este PFC se encuentra enmarcado dentro de un proyecto de I+D+i (investigación, desarrollo e innovación) denominado ORIGIN (ORganizaciones Inteligentes Globales INnovadoras) [54]. ORIGIN pertenece al Fondo Tecnológico del CDTI (Centro para el Desarrollo Tecnológico Industrial) y se desarrolla en el período 2010-2013 con la colaboración de Sicaman, Alhambra Eidos, SIGTEL, Treelogic, la Universidad de Oviedo, la Universidad de Castilla-La Mancha e Indra Software Labs, siendo ésta última con la que principalmente se ha interaccionado para realizar el desarrollo del mismo.

El objetivo principal del proyecto ORIGIN consiste en crear un entorno para la gestión y fabricación de software en escenarios globales. Este entorno permitirá aumentar la productividad de las actividades de desarrollo de software en escenarios globales y mejorar la calidad del software desarrollado, prestando atención tanto a las necesidades de los clientes, como de los proveedores en este tipo de escenarios.

1.1 Estructura del documento

En el capítulo 1 se expone de un modo resumido los orígenes de la globalización, cómo ha afectado a la industria del software y la necesidad de obtener entornos y herramientas que permitan desarrollar aplicaciones con calidad y productividad bajo el paradigma del **Desarrollo Global de Software**. La motivación que da lugar a este proyecto y los objetivos perseguidos en el mismo son descritos con un mayor grado de detalle en el capítulo 2. En el capítulo 3 se describe el estado del arte relacionado con las principales temáticas a abordar: desarrollo global de software, medidas software, técnicas de visualización y librerías de generación de gráficos. A continuación, en el capítulo 4, se detalla la metodología que se seguirá para el desarrollo del PFC, así como los patrones y el marco tecnológico que se emplearán. En el capítulo 5 se mostrará en detalle los diferentes módulos y funcionalidades que se han desarrollado. En el capítulo 6 se ofrece un resumen de las conclusiones obtenidas tras la elaboración de este proyecto y algunas ideas y mejoras que pueden ser elaboradas en futuras líneas de trabajo. Para finalizar, se incluye una serie de anexos que se corresponden con informes de pruebas, el manual de usuario de la herramienta, el manual de instalación y partes de código fuente y ficheros de configuración que se consideran relevantes.

Capítulo 2

Motivación y Objetivos del Proyecto

A medida que la industria del software se ha visto afectada por la globalización, las empresas han tenido que modificar sus metodologías y procesos de gestión de productos software para adaptarse a las nuevas tendencias. Inicialmente, estas metodologías estaban adecuadas para gestionar proyectos que se desarrollaban en una única ubicación, pero actualmente estos trabajos se realizan de forma distribuida, en la que los equipos de trabajo no se encuentran co-ubicados. Esta descentralización complica la gestión del software y puede afectar negativamente en aspectos de calidad y productividad.

Bajo este escenario surge el proyecto ORIGIN (ORganizaciones Inteligentes Globales INnovadoras) [54], dentro del cual se enmarca este PFC, cuyo objetivo se centra en aumentar la productividad de las actividades de desarrollo de software en escenarios globales y mejorar la calidad de los productos desarrollados. De esta forma se trata de incrementar el nivel competitivo a nivel internacional de las empresas compuestas por factorías de software.

Para conseguir este objetivo se desarrollará un conjunto de herramientas conceptuales y metodológicas, así como sistemas software que permitan optimizar la fabricación de software en este tipo de escenarios, paliando problemas de comunicación y gestión de conocimiento. Por lo tanto, las tecnologías desarrolladas facilitarán la utilización del conocimiento organizacional en las actividades de desarrollo de software para dar soporte a la toma de decisiones en las actividades de la organización e incrementar su competitividad. Entre ellas destacan el diseño y construcción de métodos de pruebas distribuidos, **herramientas para visualizar la calidad de los procesos y productos software y tecnologías para mitigar el efecto de la distribución de recursos en el proceso de desarrollo y en la gestión del conocimiento** [48].

2.1 Motivación

Cuando se desea desarrollar un software de forma globalizada, el personal a cargo de la gestión del mismo se enfrenta a ciertos problemas que condicionan su ciclo de vida. Cuestiones como la determinación de las factorías en la que se puede desarrollar o problemas en el desarrollo de un determinado producto (el cual se puede estar ocasionando en una determinada factoría), entre otros, son de vital importancia en este contexto, ya que afectan directamente a la productividad de las actividades de desarrollo y la calidad del software desarrollado. Sin embargo, estas tareas no son triviales ya que requieren habilidades de gestión organizacional, así como conciencia de la propia estructura organizacional y del conocimiento manejado, cuya complejidad aumenta si participan varias empresas estructuradas en base a factorías. Por tanto, es necesario que el personal a cargo de este tipo de actividades disponga de la información apropiada en el momento que lo requiera. Pero disponer de la información precisa no es suficiente si no se proporciona de un modo adecuado, ya que la gran cantidad de datos que se puede llegar a manejar puede ser muy extensa.

Las herramientas de visualización son sistemas que permiten el análisis de datos complejos por medio de la exploración visual. Se utilizan en multitud de áreas para realizar distintas simulaciones y representaciones, como en medicina y aeronáutica. Estas áreas tienen en común la existencia de modelos, objetos tangibles, reglas o propiedades medibles que facilitan su estudio y posterior representación visual. Por el contrario, en la ingeniería del software se presenta el problema de la intangibilidad del código, entre otros, y la ausencia de alguna entidad real que se le asemeje y que permita compararlo para realizar mediciones o establecer un patrón para su representación gráfica.

La motivación de este PFC viene dada por la necesidad de desarrollo de una herramienta, que se encuentra enmarcada dentro del proyecto ORIGIN, para la gestión organizacional en el paradigma de desarrollo global y para el seguimiento de los proyectos globalizados. Para ello, una característica esencial consiste en la posibilidad de visualizar distintas medidas e indicadores indicadores de calidad y productividad utilizados en este contexto.

2.2 Objetivos

A continuación, y bajo el subtítulo de “objetivo principal”, se describirá la meta final que se persigue en la elaboración de este PFC, que consiste en obtener un producto software. Un producto software no se constituye únicamente de código fuente ejecutable, sino que también se compone de la documentación, diagramas y modelos -entre otros- que se han generado en cada una de las fases que conforman su ciclo de vida. Ello acarrea la adquisición de una serie de competencias y la compleción de un conjunto de tareas que serán expuestas bajo el título de “objetivos parciales”.

2.2.1 Objetivo principal

El objetivo principal del PFC consiste en:

Elaborar una herramienta que facilite las tareas de gestión organizacional en el contexto del **Desarrollo Global de Software** y de seguimiento de los proyectos globalizados mediante diversas técnicas de visualización.

Para cumplir este objetivo se dará soporte a los profesionales de este sector permitiéndoles obtener información relevante en función de las actividades que deseen desempeñar y se ofrecerán medios para representar dicha información gráficamente empleando distintas técnicas de visualización. Entre estas técnicas destacan la geolocalización de factorías software, que puede ser de vital importancia cuando se desea alargar la jornada laboral, y la visualización de medidas e indicadores de calidad y productividad empleados en escenarios globales. De este modo, se proveerá un conjunto de medios para asegurar la calidad del software desarrollado, aumentar la productividad de sus actividades, ofrecer soporte en la toma de decisiones de las mismas, y detectar distintos riesgos y anomalías que pueden surgir en un determinado proyecto o factoría, mitigando, así, el efecto de la distribución de recursos en el proceso de desarrollo.

Las medidas e indicadores de calidad y productividad pueden representar diversas características relativas a los distintos elementos que intervienen en este paradigma, tales como la productividad de las factorías de software, las incidencias que han surgido a lo largo del ciclo de vida de un producto o la portabilidad de un determinada funcionalidad de un proyecto, por ejemplo. De este modo, se facilitarán las tareas de gestión y análisis de la calidad del software

dando soporte a la toma de decisiones en las actividades de la organización y haciendo posible detectar o predecir anomalías o incidencias con un golpe de vista.

Para abordar el problema, será necesario realizar un estudio previo que permita modelar una abstracción de las entidades que participan en el desarrollo global de software. En la figura 2.1 se muestra un esquema de la estructura organizacional que puede darse en este contexto. Así pues, será posible establecer distintos niveles de abstracción que permitan obtener información cada vez más detallada siguiendo un planteamiento *top-down*. A modo de ejemplo, se consideran los siguientes niveles:

- **Nivel de compañía.** El DGS puede implicar la colaboración entre varias organizaciones y cada organización puede estar compuesta por una o más factorías de software. Este nivel es muy importante ya que si se representa la estructura organizacional de un escenario global en una pirámide, se corresponde con la cima de la misma. Desde aquí será posible visualizar, por ejemplo, información relativa a las factorías pertenecientes a una compañía o a los proyectos en los que participa.
- **Nivel de factoría.** Una factoría puede estar desarrollando varios proyectos, o parte de ellos, de forma simultánea. Por tanto, en este nivel se podrá acceder a la información relativa a una factoría determinada o a un conjunto de factorías. Características como el mercado de especialización de una factoría, dónde se encuentra ubicado el centro, la productividad del mismo, o el número de empleados del que dispone, pueden ser de vital importancia cuando se trata de seleccionar dónde se desarrollará un software determinado.
- **Nivel de proyecto.** Un proyecto puede estar desarrollándose en múltiples factorías pertenecientes a una misma organización o a distintas organizaciones. Desde este nivel se podrá visualizar la información relativa a los proyectos software para realizar un seguimiento del estado y evolución del mismo. Esta característica permitirá detectar, entre otros, problemas en su desarrollo que pueden estar ocasionados en una determinada factoría, proyectos en los que surge un alto grado de incidencias y baja resolución de las mismas, retrasos en los plazos de entrega, diferentes características de calidad del código fuente, etcétera.

Para permitir que los distintos usuarios del sistema puedan compartir y manejar la información globalizada, la herramienta se diseñará como una **aplicación web**, que deberá ser

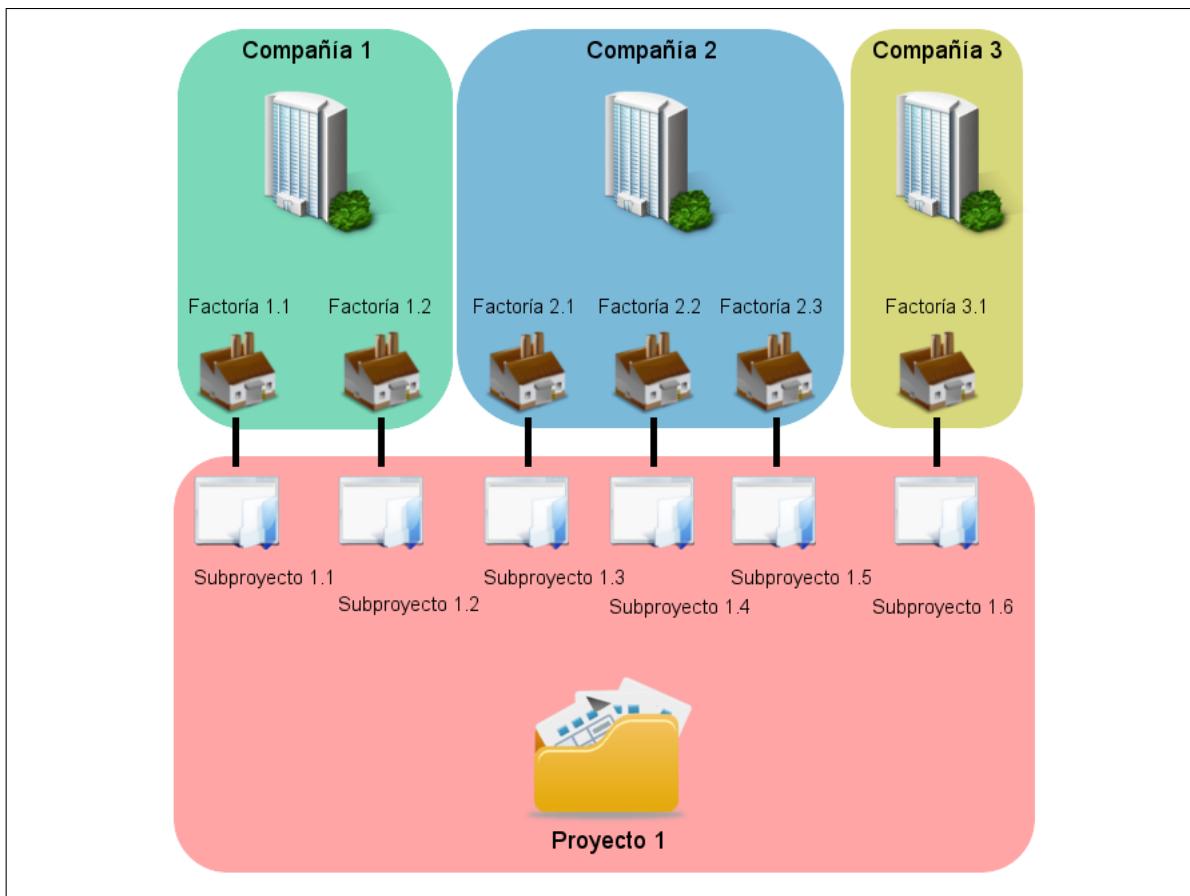


Figura 2.1: Esquema de la estructura organizacional de las entidades que intervienen en el desarrollo global de software

accesible por medio de un navegador web convencional. Además, a fin de desarrollar un sistema extensible y que satisfaga los requisitos de visualización más exigentes, se desarrollará un **motor gráfico** basado en alguna librería de gráficos 3D, que será integrado posteriormente en la aplicación web. El motor gráfico implementará un conjunto de **metáforas de visualización** sobre las cuales se representarán visualmente la información que se considere relevante. Al tratarse de un entorno en tres dimensiones sobre el que se podrá representar una gran cantidad de información, debe permitir que el usuario navegue a través de él sin perder el sentido de la orientación.

Es importante destacar que este PFC se centra en dar soporte a tareas de gestión, visualización y personalización de la visualización. Por lo tanto, no proporcionará medios para calcular los valores de los datos que maneja, tales como las medidas e indicadores de calidad del software, que deberán ser aportados por un agente externo. Para ello, se facilitará una interfaz gráfica a modo de formulario en la que los usuarios podrán aportar dichos datos.

Para conseguir este objetivo se deben llevar a cabo los objetivos parciales que se abordan

en las siguiente subsección.

2.2.2 Objetivos parciales

La consecución del objetivo principal acarrea la adquisición y compleción de las siguientes competencias y tareas:

- Modelar una abstracción del mundo real relativa al contexto del DGS, para lograr un diseño conceptual que abstraiga la información relevante. Acto seguido será posible obtener sus correspondientes diseños lógico y físico.
- Analizar técnicas de visualización para satisfacer los requisitos del sistema.
- Seleccionar la información más relevante utilizada en el contexto empresarial bajo el paradigma del DGS para su posterior representación visual.
- Estudiar la forma de visualizar la información anteriormente seleccionada, entre lo que destaca la definición de metáforas.
- Diseñar un método que permita asociar las dimensiones de los elementos de las metáforas y la información a visualizar.
- Diseñar el modelo de datos y crear una base de datos que permita la gestión de la lógica de negocio utilizada por la herramienta.
- Investigar sobre librerías para la generación de gráficos 3D y técnicas de visualización de datos empleando gráficos 3D.
- Definir una arquitectura estable y extensible de un motor gráfico que permita la representación de modelos para escenificar metáforas de visualización, empleando buenas prácticas de programación así como patrones de diseño.
- Desarrollar un **motor gráfico** para la visualización de datos empleando modelos en 3D.
- Analizar tecnologías y *frameworks* para el desarrollo de aplicaciones Web y tecnologías para la generación de gráficos y modelos de visualización en la Web.
- Definir una arquitectura estable y extensible de una aplicación web, utilizando patrones de diseño.

- Desarrollar una **aplicación web** que permita realizar la gestión de la lógica de negocio, así como la ejecución de las distintas tareas de visualización y generación personalizada de gráficos en tres dimensiones. Para ello será necesario integrar el motor gráfico desarrollado en la aplicación web.
- Desarrollar las **pruebas funcionales y unitarias** del software.
- Elaborar la documentación técnica asociada al proyecto.

2.2.3 Limitaciones

Dada la naturaleza de este PFC y su encuadre dentro del proyecto ORIGIN, las empresas colaboradoras en este proyecto imponen como requisito que el lenguaje de programación sea Java. Por esta razón, para alojar la aplicación web será necesario disponer de un servidor web que implemente la especificación JSP (Java Server Pages) de Oracle. Además, para mantener el estado de la lógica de negocio y hacerlo persistente es necesario un SGDB (Sistema Gestor de Bases de Datos) que cumpla el estándar SQL (Structured Query Language). Ambos servicios podrán estar instalados y ejecutados en la misma máquina o en máquinas independientes. Cualquiera que sea la solución, es necesario que la máquina que los ejecute disponga de conexión a Internet para que los usuarios puedan acceder a la aplicación por medio de un navegador web convencional.

Por último, destacar que el usuario deberá disponer de un entorno de ejecución Java (JRE, en su acrónimo inglés Java Runtime Environment) en su sistema operativo para poder ejecutar el motor generador de gráficos 3D.

Capítulo 3

Estado del Arte

En este capítulo se muestran los conocimientos teóricos obtenidos de manera previa a la elaboración de este PFC. Entre ellos se encuentran los temas de Desarrollo Global de Software (DGS), calidad y medidas software, visualización de medidas software, qué son los gráficos por computador y las librerías más importantes para generar gráficos por computador.

3.1 Desarrollo Global de Software (DGS)

La globalización económica ha hecho que la industria y el comercio se adapte a nuevos modelos de negocio. En el capítulo 1 de este documento se ha hecho mención a algunos de los acontecimientos que, según Thomas Friedman [20], dieron lugar a este fenómeno. Estos acontecimientos también afectaron a la industria del software que a día de hoy ha roto con cualquier frontera física que se le pudiera imponer gracias a su fuerte relación con las innovaciones en tecnología y telecomunicaciones.

Como prueba de ello, hoy en día es posible realizar software en todo momento y en cualquier parte del mundo (Fig. 3.1), lo que introduce el concepto de *Desarrollo Global de Software* (DGS o GSD, en su acrónimo inglés Global Software Development). Para llegar a este modelo ha sido necesario evolucionar a través de dos predecesores: el *modelo tradicional*, en el que los equipos de trabajo encargados de desarrollar un software se encontraban co-localizados en un mismo edificio; y el *Desarrollo Distribuido de Software*, en el que esos equipos de trabajo se encontraban dispersos en una misma ciudad o incluso en distintas ciudades de un mismo país [25]. A modo de síntesis, en la Tabla 3.1 se muestra una clasificación de los distintos tipos de desarrollo de software en base a la dispersión geográfica.

Otro factor a tener en cuenta en el DGS, a parte de la dispersión geográfica de los

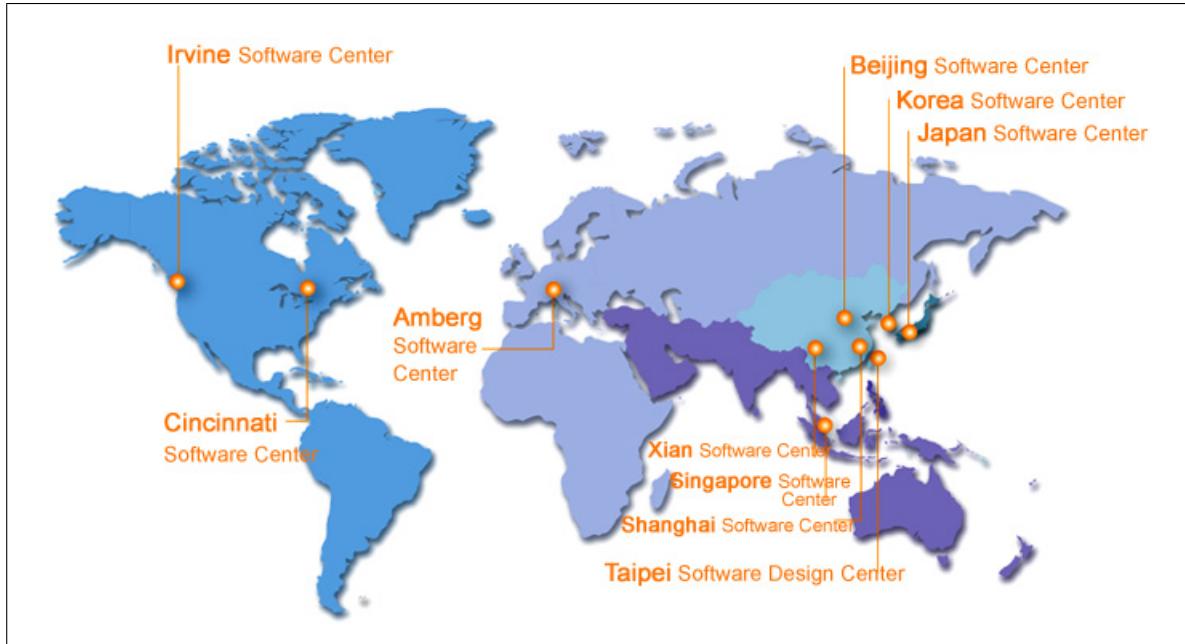


Figura 3.1: Mapa del mundo que refleja la dispersión geográfica de factorías involucradas en desarrollo global [4]

Localización	Tipo de desarrollo
Mismo lugar	Desarrollo tradicional
Mismo país	Desarrollo Distribuido de Software (DSD)
Distintos países	Desarrollo Global de Software (DGS)

Tabla 3.1: Clasificación de desarrollo de software según dispersión geográfica [41]

*stakeholders*¹, es el número de compañías que colaboran en un mismo proyecto. Se puede dar el caso de que el proyecto esté siendo llevado a cabo por una sola compañía o, por el contrario, que exista un conjunto de compañías subcontratadas (compañías subsidiarias) que colaboren entre sí junto con la compañía contratante [44]. De esta característica surgen nuevos modelos y variantes dentro del DGS: outsourcing, inshoring, offshoring y el nearshoring.

El término **outsourcing** se utiliza cuando una compañía realiza una externalización de uno o varios de los servicios que proporciona. Estos servicios son delegados sobre una compañía subcontratada. En el contexto del DGS, se asume que la compañía contratante y compañía subcontratada se encuentran sitas en distintos países. Si ambas compañías se encuentran localizadas en el mismo país, se habla de **inshoring**. Las principales ventajas del

¹Se define el término *stakeholder* como aquellos que pueden afectar o son afectados por las actividades de una empresa [19].

outsourcing se basan en una reducción de costes debido a que la mano de obra en el país de la compañía subcontratada puede ser más barata que en el país de la compañía contratante, y poder llegar a un mayor grupo de clientes adaptando los productos a sus necesidades. Estas dos características permiten ofrecer productos especializados y precios más competitivos [43].

De la idea del *outsourcing* surge el **offshoring**. Este concepto va más allá que su predecesor y no se limita a externalizar servicios, si no que trasladan fábricas enteras a otros países. Si los países implicados en este proceso son relativamente cercanos al país de origen de la compañía, se habla de **nearshoring** [43]. Estos modelos introducen una serie de ventajas sobre el outsourcing tales como la obtención de mano de obra, en ocasiones escasa, necesaria para el proyecto en países donde se está avanzando notablemente en I+D; disminución de las dificultades para construir los métodos necesarios para gestionar el conocimiento y coordinar los distintos grupos de trabajo, abstrayéndose de la distancia geográfica [32]; y reducción de costes y gastos derivados de los viajes a los países con los que se colabora.

Estos conceptos se muestran sintetizados en la Tabla 3.2.

Externalización de servicios	Mismo país	Inshoring
	Distintos países	Outsourcing
Traslación de fábricas	Países vecinos	Nearshoring
	Países distantes	Offshoring

Tabla 3.2: Modelos y variantes de Desarrollo Global de Software

Como se puede apreciar, todas estas modalidades de DGS introducen una serie de ventajas o beneficios, así como un conjunto de inconvenientes o desafíos que afectan al desarrollo y gestión de este tipo proyectos.

En las siguientes subsecciones se detallan tanto las ventajas como los inconvenientes que introduce el DGS.

3.1.1 Ventajas del Desarrollo Global de Software

A pesar de los desafíos que introduce el DGS, que serán abordados en la siguiente subsección de este documento, existen varios motivos por los que este concepto de desarrollo de software resulta muy atractivo. Los más destacables según [9], [11], [25] y [42] son los siguientes:

- **Reducción de costes en salarios.** Dado que el objetivo de las empresas es optimizar la relación beneficios/costes, esta es una de las principales motivaciones del DGS. La reducción del coste se debe a la posibilidad de contratar la mano de obra necesaria en aquellos países donde los salarios son más bajos. Esta situación se da en muchos países asiáticos e India en comparación con países de Europa o los Estados Unidos.
- **Acceso a personal y recursos más cualificado.** Existen países en los que el número de ingenieros que se forman cada año es insuficiente para la demanda que se requiere. Por tanto, es posible contratar personal de aquellos países en los que la oferta de ingenieros sea más elevada.
- **Facilidad para formar equipos de trabajo.** Dado que la compañía dispone de una cantidad de recursos dispersos por todo el mundo, es posible reasignar estos recursos con el objetivo de formar un equipo de trabajo virtual, no siendo necesario que los componentes del equipo se encuentren co-localizados.
- **Mejoras en la productividad.** Existe la posibilidad de establecer un flujo de trabajo mediante el cual se pueden establecer jornadas de trabajo más largas. Un ejemplo sería el método *follow-the-sun* (en español, *seguir el sol*) que permite que se esté desarrollando durante las 24 horas del día gracias a la disposición de distintos equipos de trabajo en diferentes husos horarios.
- **Avances tecnológicos y en infraestructura.** Los avances en telecomunicaciones así como en herramientas y entornos que facilitan la gestión del desarrollo global, son un aspecto muy importante que está permitiendo que este paradigma se lleve a cabo.
- **Proximidad al mercado y al cliente.** Esta característica resulta realmente atractiva ya que se dispone de personal cercano al cliente. Esta situación abre la posibilidad de conocer las necesidades reales del mismo y adaptar el producto a sus necesidades. Además, permite un conocimiento de las leyes y políticas locales, lo cual puede repercutir en cómo se fabrica y distribuye un mismo producto en distintos países.
- **Innovación.** Disponer de equipos de trabajo formados por personas con diferentes experiencias y culturas permite que se compartan las mejores prácticas y hábitos.

3.1.2 Desafíos del Desarrollo Global de Software

A pesar de las ventajas que supone, en este paradigma de desarrollo de software también existen una serie de vulnerabilidades a determinados factores que pueden afectar a un incremento del presupuesto del producto final, así como retrasos en su entrega. Según [33], [35] y [25] estos problemas se pueden clasificar de la siguiente forma:

- **Problemas estratégicos.** Este tipo de problemas surgen a la hora de dividir un proyecto en partes (fases, subproyectos, etcétera) y decidir a qué factoría se asigna cada una de éstas. Para hacer esta asignación, es necesario tener en cuenta los recursos disponibles de cada factoría, así como el grado de experiencia en cada tecnología, la infraestructura, y otros factores que pueden ser determinantes. El objetivo es tratar de lograr una situación en la que las factorías puedan trabajar del modo más independiente posible y que dispongan de un mecanismo de comunicación efectivo, fácil y flexible. Otros problemas pueden surgir debido a la actitud del personal si experimenta que no tiene todo el control que debería sobre su trabajo, que puede perder su puesto de trabajo, que lo trasladen a otro destino o incluso que se requiera que viaje a otras ciudades o países.
- **Diferencias culturales.** Las diferencias culturales se pueden dar en mayor o menor grado en función de la región, la religión, el sexo, la generación y la clase social. Estas diferencias pueden proporcionar beneficios, tales como la compartición de experiencias y buenas prácticas de trabajo. Sin embargo, también plantean un conjunto de problemas -como las malas interpretaciones debido al estilo de comunicación o una actitud inadecuada de un empleado ante su superior- que, según [26], se categorizan en cuatro áreas:
 - **Distancia de la autoridad.** Contempla la actitud de las personas ante sus superiores, y su acepción de que el poder no se distribuye de forma equitativa.
 - **Evasión de la incertidumbre.** Contempla el grado en el que una persona se siente amenazada ante una situación desconocida o incómoda, y trata de evitarla. En este área se incluyen características como el modo de resolución de conflictos, el control de la agresividad y la expresión de sentimientos.
 - **Masculinidad y feminidad.** Contempla el modo en el que se valora la figura del hombre y la figura de la mujer, es decir, si existe algún tipo de discriminación

hacia una figura u otra o si existe igualdad entre ambos géneros.

- **Individualismo y colectivismo.** En este área entran en juego la importancia de los retos y aspiraciones individuales en contraposición con la lealtad y el apoyo de las metas grupales.
- **Comunicación inadecuada.** Las personas que llevan a cabo un proyecto de desarrollo global trabajan en diferentes partes del producto, pero que son dependientes unas de otras. Esta dependencia hace necesario que se establezca una buena comunicación entre las partes involucradas para que sea posible establecer una buena coordinación y evitar que surjan problemas relacionados con el incumplimiento de plazos y el incremento del presupuesto.

En el desarrollo de software suelen darse dos tipos de comunicación. La primera y más formal se rige en base a determinados criterios y métodos impuestos por la organización. Estos métodos suelen realizarse mediante el uso de herramientas de gestión que permitan guardar trazas y registros de las operaciones llevadas a cabo. En esta categoría pueden destacarse tareas como la de actualizar el estado del proyecto o asignar recursos para la resolución de errores. El segundo tipo se da en un contexto más informal. El principal método se basa en la comunicación directa cara a cara, es decir, una conversación espontánea que puede producirse en el área de trabajo o incluso en el tiempo de descanso. Esta situación no se produce cuando las personas se encuentran localizadas en distintas ubicaciones, pero puede darse en forma de videoconferencias y llamadas telefónicas. No obstante, es susceptible a problemas de comunicación debido a las diferencias culturales e idiomáticas, o incluso a diferencias temporales debido a los husos horarios.

Este problema adquiere más agravantes si se introducen factores de propiedad intelectual e información de productos, mercados y clientes.

- **Problemas de gestión del conocimiento.** La gestión del conocimiento y de la información es un factor determinante en cualquier contexto, no sólo en el DGS. Si esta tarea no se aborda de un modo eficiente, pueden darse casos en los que los equipos pierdan oportunidades de reutilizar trabajo que fue completado previamente, dando lugar a pérdidas de tiempo y dinero. Además, pueden surgir situaciones en las que no se haya determinado de un modo correcto qué tareas se encuentran en el camino

crítico de un flujo de trabajo, o que la información de los clientes no se haya distribuido eficientemente a todos los equipos de trabajo afectados.

Por otro lado, una documentación vaga puede ocasionar que el desarrollo colaborativo no se lleve a cabo de un modo eficiente. Por tanto, es muy importante que la documentación se encuentre actualizada para prevenir errores y facilitar la mantenibilidad.

- **Dificultades de gestión del proyecto y de los procesos.** En este aspecto destacan los problemas de sincronización que se da entre los procesos que son llevados a cabo en distintas localizaciones. Para facilitar esta sincronización es crucial establecer un conjunto de hitos así como los entregables de entrada y salida de cada proceso. Pero esta tarea no es fácil si no se dispone de un entorno y herramientas que permitan una comunicación formal y el trabajo colaborativo en tiempo y espacio.
- **Problemas técnicos.** En esta categoría destacan problemas relacionados con intercambio de datos con formatos incompatibles, incompatibilidad entre las versiones de las herramientas o librerías utilizadas, control de versiones del producto y la importancia de que todos los equipos de desarrollo sean conscientes de los cambios, o infraestructuras de red que en ocasiones son lentas y dificultan el trabajo colaborativo.

3.2 Calidad y medidas software

La calidad es uno de los factores más importantes que hacen que los usuarios y clientes se decanten por un producto u otro. Por tanto, las organizaciones deben asegurar de algún modo la calidad de los productos y servicios que ofrecen, ya que de ello depende su supervivencia y competitividad en el mercado [39].

Según el Diccionario de la Rea Academia Española de la Lengua (DRAE 2006, 23^a edición), y en el contexto que interesa tratar en este libro, se define el término **calidad** como:

1. Propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor.
Esta tela es de buena calidad.
2. Buena calidad, superioridad o excelencia. *La calidad del vino de Jerez ha conquistado los mercados.*
3. Condición o requisito que se pone en un contrato.

A lo largo de los años, distintos gurús expertos en el tema han tratado de definir el concepto de *calidad*. A continuación se citarán las afirmaciones que se consideran más representativas para este caso de estudio [28]:

- “*El primer supuesto erróneo es que calidad significa bueno, lujoso, brillo o peso...* Definimos calidad como «conformidad con los requisitos »... La no conformidad detectada es una ausencia de calidad.”. Philip B. Crosby, “*Quality is Free*”, Mc Graw Hill, New York, 1979.
- “*La dificultad de definir calidad es traducir las necesidades futuras del usuario en características medibles. Sólo así un producto puede ser diseñado y fabricado para dar satisfacción al cliente al precio que está dispuesto a pagar.*”. W. Edwards Deming, *Fuera de la crisis*, MIT, 1988.
- “*Debemos enfatizar en la orientación hacia el cliente... Cuando desarrolle un nuevo producto, el fabricante deberá anticipar los requisitos y necesidades del cliente... Cómo uno interprete el término «calidad »es importante... De un modo sencillo, calidad significa calidad del producto. De un modo más amplio, calidad significa calidad de trabajo, calidad del servicio, calidad de información, calidad de proceso, calidad del personal, calidad del sistema, calidad de la empresa, calidad de objetivos, etc.*”. Kaoru Ishikawa, *¿Qué es Control Total de la Calidad? El modelo japonés*, Prentice Hall, 1985.
- “*La palabra calidad tiene múltiples significados. Dos de ellos son los más representativos: 1. La calidad consiste en aquellas características del producto que satisfacen las necesidades del cliente y que por eso brindan satisfacción con el producto. 2. La calidad consiste en la ausencia de deficiencias.*”. Joseph M. Juran, *Manual de Control de Calidad*, 4^aedición, McGraw Hill, 1988.

Tras analizar estas definiciones y la primera acepción de calidad propuesta por la DRAE, y encuadrarlas en el contexto del desarrollo de software, se entiende que es necesario determinar un conjunto de propiedades que permita determinar la calidad de un producto software y que permita conseguir que sea mejor que otro [40]. Pero esto es relativo y depende del punto de vista utilizado, ya que los clientes y las organizaciones no tienen los mismos criterios y no conciben la calidad del mismo modo (Fig. 3.2). Para los primeros, un producto de calidad es aquel que satisface sus necesidades en cuanto a uso o consumo. Para los segundos, un

producto de calidad es aquel sobre el que se han medido determinadas características que satisfacen un conjunto de especificaciones numéricamente definidas [28]. Esta es la razón por la que surge la necesidad de medir la calidad de un producto software y, por tanto, la creación de normas y estándares que definen un modelo de calidad del software y su proceso de evaluación.

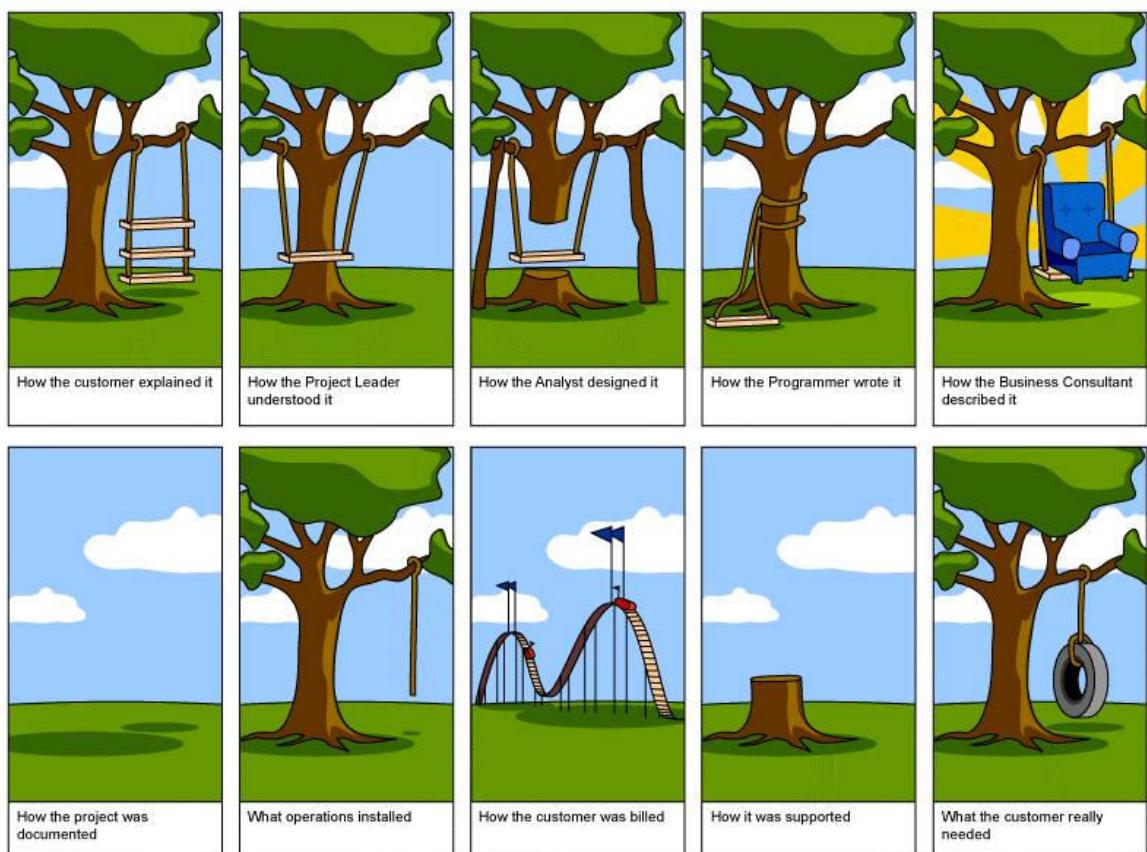


Figura 3.2: Tira de cómic que refleja las necesidades de un producto desde el punto de vista de distintos *stakeholders*. De izquierda a derecha y de arriba a abajo representa: cómo lo explicó el cliente, cómo lo entendió el jefe de proyecto, cómo lo diseñó el analista, cómo lo desarrolló el programador, cómo lo describió el consultor, cómo fue documentado el proyecto, lo que fue instalado, por lo que pagó el cliente, cómo fue el soporte, y lo que el cliente realmente necesitaba

Inicialmente surge una primera generación de estándares de calidad del producto software, entre los que destacan las normas ISO/IEC 9126 e ISO/IEC 14598. Por un lado, la norma ISO/IEC 9126, que se subdivide en cuatro partes, define un modelo de calidad del producto software, métricas externas, métricas internas y métricas de calidad en uso. El modelo de

calidad se descompone en un conjunto de seis características que se subdividen en múltiples subcaracterísticas y éstas a su vez en atributos (Fig. 3.3) [3]. Las características que define son las siguientes:

- **Funcionalidad.** Engloba un conjunto de atributos relacionados con la capacidad del software para satisfacer los requisitos funcionales que debe cumplir el sistema.
- **Fiabilidad.** Se refiere a la capacidad del software de proporcionar servicio durante un tiempo establecido y bajo un conjunto de condiciones definidas.
- **Usabilidad.** Indica el esfuerzo que invierte el usuario para utilizar el producto debidamente.
- **Eficiencia.** Establece una relación entre el rendimiento del sistema y los recursos empleados bajo un conjunto de condiciones definidas.
- **Mantenibilidad.** Indica el esfuerzo necesario para adaptar el producto a nuevos requisitos y especificaciones, así como para corregir errores.
- **Portabilidad.** Se refiere a la posibilidad de transferir el software a un entorno diferente.

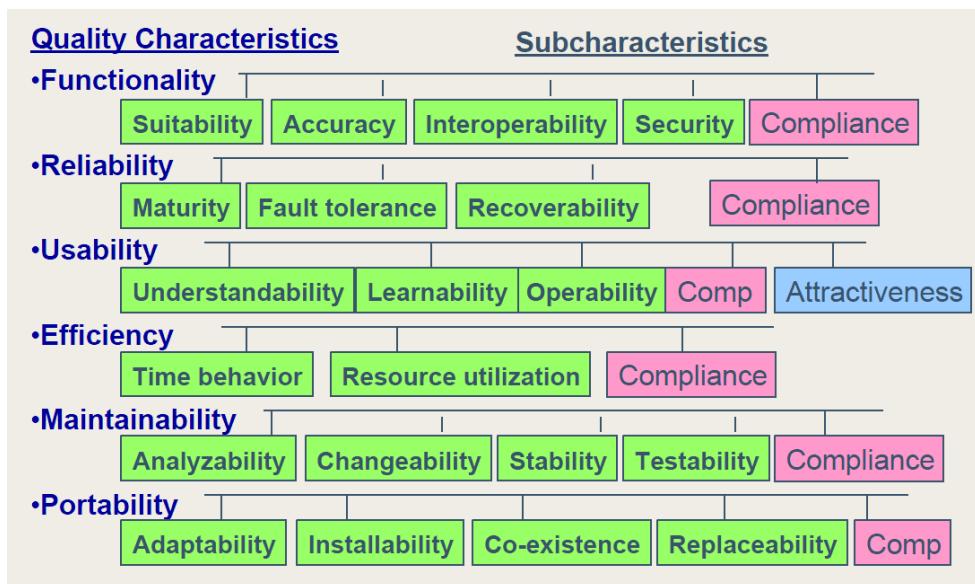


Figura 3.3: Características y subcaracterísticas del modelo de calidad del software propuesto por el estándar ISO/IEC 9126 [67]

Por otro lado, la norma ISO/IEC 14598, que en gran parte no es específica del producto software, define su relación con el modelo de calidad propuesto en la norma ISO/IEC 9126

(Fig. 3.4). Para ello proporciona un marco de evaluación de la calidad de todo tipo de producto software y establece los métodos necesarios para la medición de la calidad del mismo [2].

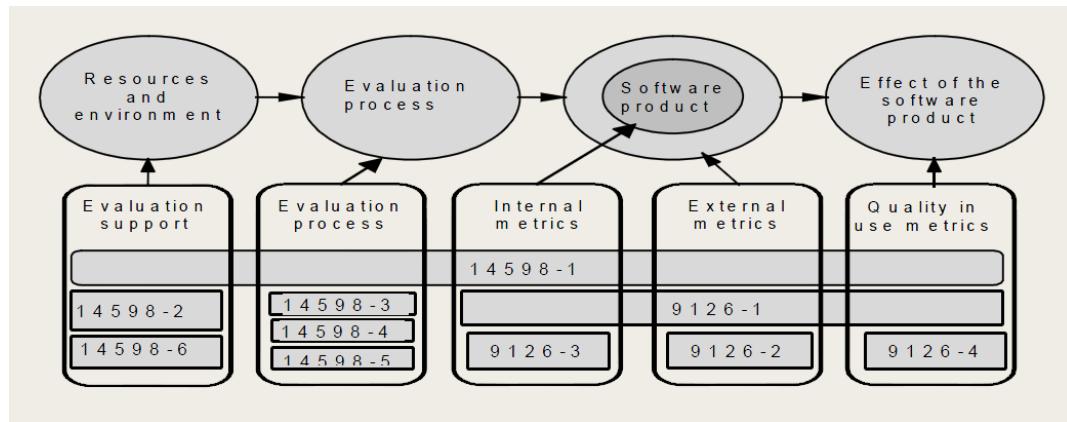


Figura 3.4: Relación entre las normas ISO/IEC 9126 (modelo de calidad del producto software) e ISO/IEC 1498 (evaluación del producto software) [67]

Con el objetivo de crear una única norma que englobe a la ISO/IEC 9126 y a la ISO/IEC 14598, surge la norma ISO/IEC 25000, también conocida como SQuaRE (Software Product Quality Requirements and Evaluation). Esta norma, que consta de cinco divisiones, tiene como objetivo crear un conjunto de estándares específicos de productos y sistemas software, así como establecer un glosario de términos que no contenga conceptos genéricos [1]. Las divisiones de SQuaRE son las siguientes:

- ISO/IEC 2501n: División del modelo de calidad, que reemplaza a la ISO/IEC 9126.
- ISO/IEC 2502n: División de la medición de calidad.
- ISO/IEC 2503n: División de requisitos de calidad.
- ISO/IEC 2504n: División de evaluación de la calidad, que reemplaza a la ISO/IEC 14598.
- ISO/IEC 25050 - 25099: División de extensión de SQuaRE.

Una vez que se dispone de un modelo de calidad del producto software y de métodos de evaluación del mismo, es necesario abordar cuestiones de medición.

La **medición** es una disciplina fundamental en cualquier ingeniería, y la ingeniería del software no es una excepción [38]. Este aspecto está demostrando ser muy eficaz en grandes

proyectos de bases de datos, en el desarrollo y mantenimiento del software y la evaluación y garantía de la calidad de sistemas [8]. Además, la ingeniería del software presenta el problema de que los objetos o entidades que se desean medir no son tangibles, es decir, no se pueden palpar y medir con herramientas convencionales. Esto hace aún más importante la medición del software, ya que es un mecanismo que permite comprobar y asegurar la calidad del mismo.

Cuando se habla de *medidas software*, es frecuente encontrar en la literatura el término *métrica software* en su lugar. Sin embargo, en este PFC se utilizará el concepto de *medida*, ya que de acuerdo a la **Ontología de la Medición del Software** (*Software Measurement Ontology* o SMO) es más adecuado [22].

En la SMO se afirma que el objetivo de una medición es *satisfacer ciertas necesidades de información*. Para ello, es necesario identificar las entidades que se quieren medir y sus atributos, siendo un atributo una propiedad, física o abstracta, que se puede medir. Por ejemplo, en el contexto del software orientado a objetos, una *clase* podría ser una entidad y, las *líneas de código*, un atributo de dicha entidad.

Según [37] y [17], el uso de medidas en el software es una buena práctica para conocer qué ocurre durante el proceso de desarrollo y mantenimiento del software. Gracias a ello, es posible predecir lo que puede ocurrir y, por consiguiente, tomar las mejores decisiones cuando se considere oportuno.

3.3 Visualización de medidas software

Como se ha explicado en la sección 3.2, realizar mediciones sobre el software es un desafío. Cuando se completa un proceso de medición, es conveniente generar un informe que muestre los resultados obtenidos para facilitar su posterior análisis. La importancia o el grado de interés de dichos resultados, varía en función de la experiencia y la necesidad del usuario que los estudie. Con el objetivo de filtrar información no relevante en determinadas ocasiones y destacar aquella que proceda, se propone el uso de herramientas de visualización que, además de proporcionar dichas funcionalidades, explotan la capacidad de percepción del sistema visual humano [8].

Los sistemas de visualización son herramientas que permiten el análisis de datos complejos por medio de la exploración visual. Este tipo de sistemas se han aplicado en multitud de áreas, como medicina y aeronáutica, para realizar representaciones y simulaciones. Estas

áreas tienen en común la existencia de modelos, objetos o reglas del mundo real que facilitan su representación visual en un entorno 3D, dado que son objetos tangibles o propiedades medibles mediante algún tipo de utensilio. Por el contrario, cuando tratamos de representar el análisis de datos relativos a un programa software, se presenta el problema de la intangibilidad del código y la ausencia de alguna entidad real que se le asemeje y que nos permita establecer un patrón para su representación gráfica [8].

En la literatura se encuentran distintas técnicas que permiten representar visualmente determinados tipos de información relacionada con el software. En [16] y [27] utilizan modelos en tres dimensiones y texturas para mostrar información estadística. En [34] se combinan vistas polimétricas con medidas software de las entidades conceptuales que intervienen en programas software orientado a objetos.

Para facilitar la comprensión de este último caso se adjunta la figura 3.5 (a). En ella, la vista polimétrica puede escenificar una clase, un atributo o un método, y representará medidas software relativas a la entidad que escenifica.

- Si la entidad escenificada es una **clase**, puede representar la profundidad del árbol de herencia, el número de métodos heredados, el número de métodos totales, el número de atributos, el número de líneas de código en toda la clase, entre otros. La figura 3.5 (b) muestra un ejemplo concreto de este caso.
- Si la entidad escenificada es un **método**, puede representar el número de líneas de código, el número de parámetros de entrada, el número de acceso a los atributos, etcétera.
- Si la entidad escenificada es un **atributo**, puede representar el número de veces que se ha accedido desde fuera de la clase o desde la propia clase.

En definitiva, la estrategia que se persigue consiste en asociar atributos gráficos (textura, color, ubicación, tamaño, etcétera) con los datos que se quiere representar, pero hay que controlar determinadas características para evitar introducir ruido y exceso de información que dificulte la visualización. Estos dos factores interfieren en la inspección visual haciendo que no sea eficiente. Por ejemplo, en la figura 3.6 se muestra el ruido introducido al tratar de representar las asociaciones entre un conjunto de entidades.

Para facilitar la comprensión de la información, los sistemas de visualización deben proporcionar vistas intuitivas, efectivas y eficaces que hagan del análisis de datos una tarea

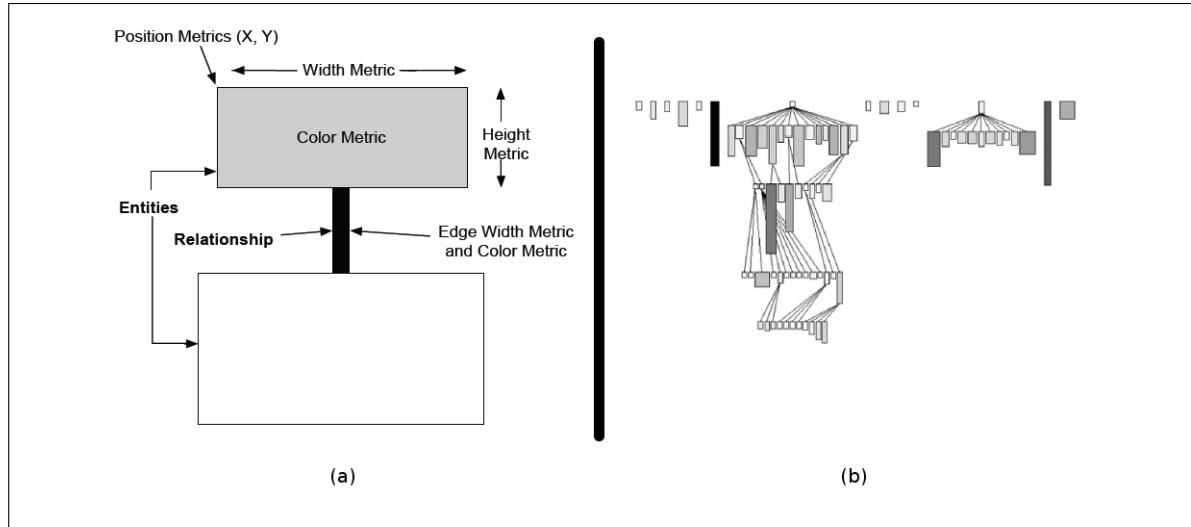


Figura 3.5: (a) Nodo de una vista polimétrica en la que se pueden representar hasta siete medidas.

(b) Vista polimétrica aplicada una jerarquía de herencia entre clases. Cada clase es representada por un nodo en el que el ancho muestra el número de atributos de la clase, el alto indica el número de métodos y el color el número de líneas de código.

[13]

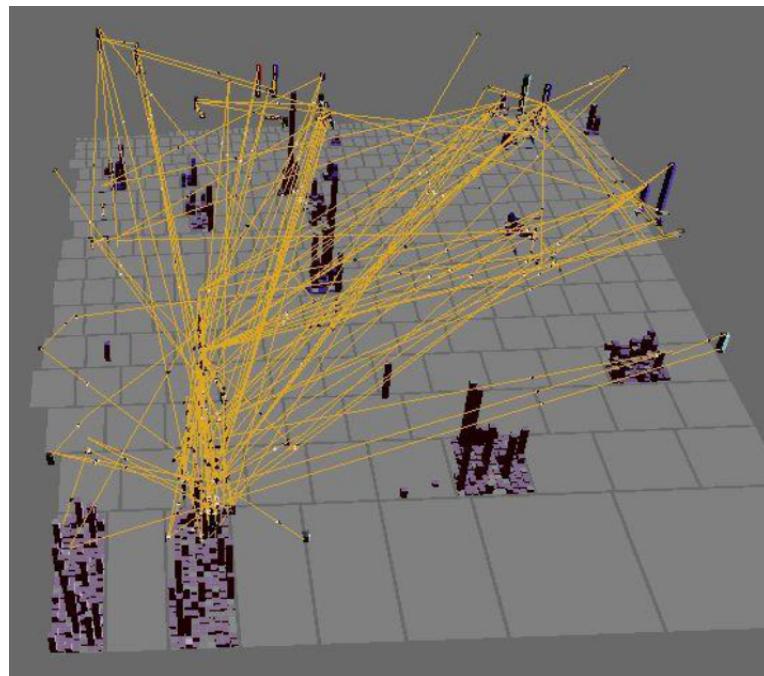


Figura 3.6: Ejemplo del nivel de ruido introducido por las asociaciones entre entidades [63]

rápida y precisa. Un método para lograr esta característica consiste en explotar una cualidad del sistema visual humano denominado **proceso preatento** [24]. Los procesos preatentos son procesos que “saltan a la vista” (Fig. 3.7), siendo detectados inmediatamente y evitando que

el usuario deba centrar su atención en una determinada región de una imagen para captar la presencia o ausencia de una característica determinada [13].

Existen cuatro categorías básicas de características que se procesan de forma preatenta [65]: color, forma, movimiento y localización espacial. De cada una de estas características se puede obtener un gran conjunto de subcaracterísticas como curvatura, tamaño, orientación, longitud, etcétera.

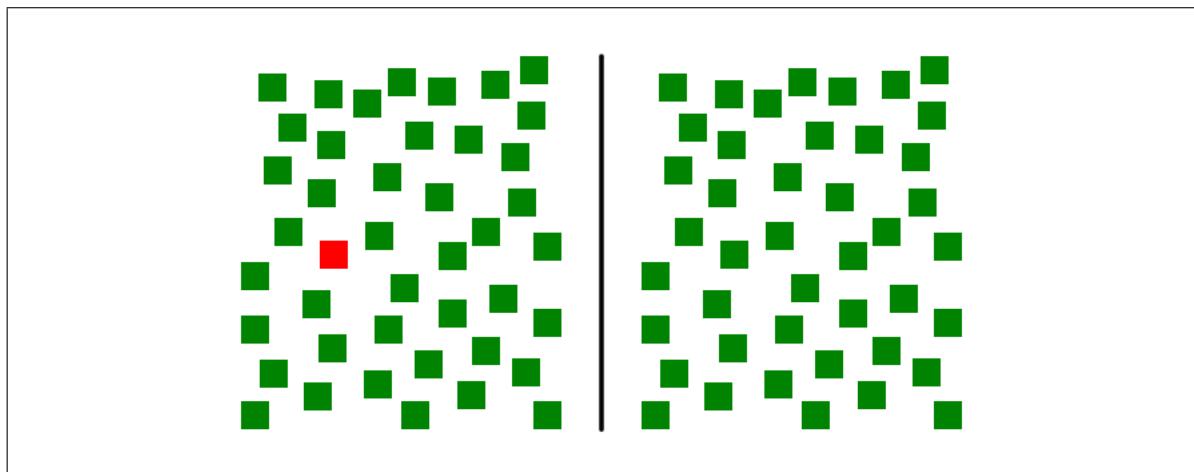


Figura 3.7: Ejemplo que activa un proceso preatento. En la imagen de la izquierda el objetivo se encuentra fácilmente; en la de la derecha, el objetivo se camufla entre el resto de elementos

Las técnicas de visualización expuestas en este capítulo se están empleando actualmente en escenarios en los que se desarrollan proyectos software tradicionales basados en programación orientada a objetos. Sin embargo, estos métodos pueden ampliarse y adaptarse al software que está siendo desarrollado de forma global. Por tanto, es necesario incluir información relativa a la estructura organizacional y la información relevante que intervienen en este paradigma, como las empresas y factorías de software involucradas en cada proyecto global, así como distintas medidas de calidad de software y productividad, y dotarla de una representación visual.

Antes de diseñar una representación visual, se requiere estudiar si un modelo 2D satisface las necesidades de la abstracción del mundo que se quiere escenificar, o si es necesario una escena más compleja en 3D. Por un lado, las escenas 2D son sencillas de implementar pero hacen que el usuario tenga la sensación de ver el sistema desde fuera y no se sienta inmerso en el mismo. Por otro lado, las escenas 3D sí producen esta sensación de inmersión pero pueden provocar que el usuario se desoriente. Para evitar este problema, en [66] se propone realizar

la visualización 3D en base a la metáfora de una ciudad, que es un entorno conocido por el usuario, para que pueda ser explorada de forma interactiva. En [66] los autores emplean la metáfora de una ciudad en tres dimensiones para representar medidas software (Fig. 3.8).

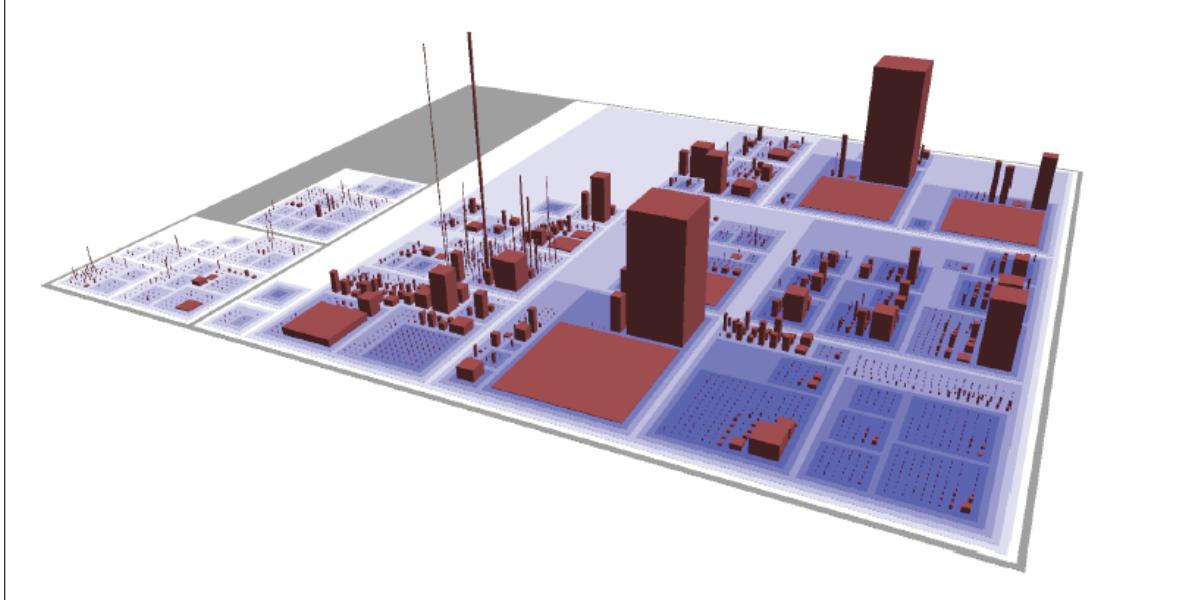


Figura 3.8: Metáfora de una ciudad para representar sistemas software [66]

Para representar una **medida software** visualmente es necesario identificar qué entidad se quiere escenificar por medio de un modelo gráfico y cuáles son los atributos más relevantes de dicha entidad. De este modo, es posible establecer un conjunto de asociaciones entre los atributos de la entidad y las dimensiones del modelo gráfico. Este concepto recibe el nombre de *vista polimétrica* [34] y permite que los modelos gráficos adopten una determinada forma, orientación o color en función de los resultados de las mediciones. Esta característica se puede combinar con los procesos preatentos, permitiendo que los sistemas de visualización adquieran un gran potencial y flexibilidad. Por ejemplo, un jefe de proyecto podría detectar a *golpe de vista* el cumplimiento de plazos de cada una de las partes de un proyecto (Fig. 3.9). Otro ejemplo podría consistir en asociar el número de líneas de código a la altura de un cubo, y el número de comentarios a la base del mismo. De este modo, si el cubo es muy alto y muy fino, significa que tiene pocos comentarios para demasiadas líneas de código; si por el contrario, el cubo es muy bajo y muy ancho, significa que tiene demasiados comentarios.

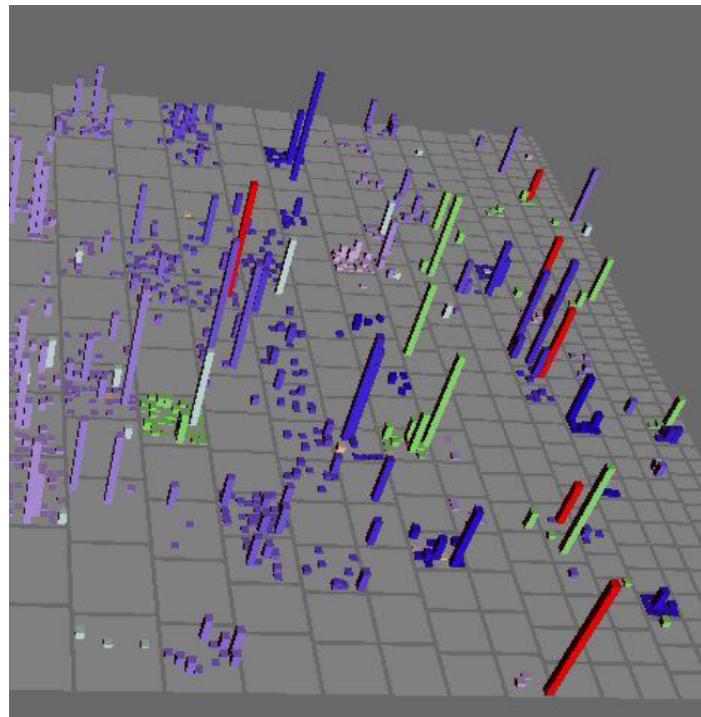


Figura 3.9: Representación visual de medidas software de un sistema. Cada cubo representa una parte del sistema. El color verde podría significar un *valor aceptable* de una medida definida; el color azul indicaría un *valor marginal* de dicha medida; y el color rojo representaría un *valor inaceptable* [63]

3.4 Gráficos por computador y Librerías de gráficos 3D

Los **gráficos por computador** es un término cuya definición ha evolucionado en las últimas décadas. De un modo sencillo, puede definirse como *imágenes generadas o modificadas mediante el uso de un ordenador*. Estas imágenes pueden ser representaciones visuales de entidades del mundo real o abstracciones de las mismas [23]. Pero este concepto ha progresado desde imágenes simples en dos dimensiones hasta entornos complejos tridimensionales de alta calidad. Por tanto, en su definición habría que incorporar el aspecto de la generación de modelos de objetos que varían su geometría y comportamiento a lo largo del tiempo [18].

Cuando se trata de pensar en el uso que se le pueden dar a los gráficos por computador, es fácil pensar en los videojuegos o en películas de animación como Avatar y Shrek, pero su uso va más allá y son una herramienta crucial en numerosas áreas. Por ejemplo, en medicina se utilizan para proporcionar métodos no invasivos que permitan probar e investigar características del cuerpo humano; en educación son una excelente herramienta para representar

conceptos; en entornos empresariales se utilizan para crear diagramas y gráficas que permiten el análisis de datos; en aviación sirven para la generación de simuladores de vuelo, etcétera [23].

Todos los usos listados anteriormente y muchos más, son razones más que suficientes para que las empresas y las comunidades de desarrolladores empleen recursos, tiempo y esfuerzo en elaborar soluciones eficientes para la generación y manipulación de gráficos por computador. Por ello, a lo largo de los últimos años han ido surgiendo estándares y librerías de gráficos que satisfagan las necesidades de hoy en día. Sin estos estándares y librerías, los desarrolladores de software debían implementar distintas versiones de sus aplicaciones para soportar las múltiples plataformas hardware en las que fuese a ser ejecutado dicho software.

Una **librería de gráficos 3D** es una *interfaz* software que proporciona un medio para *renderizar*² gráficos en una pantalla, estableciendo un medio de comunicación entre las aplicaciones y los controladores de la tarjeta gráfica (Fig. 3.10). Su principal cometido es ofrecer un nivel mayor de abstracción y otorgar independencia tanto a nivel de hardware -de entrada y de salida- como a nivel de aplicación ya que la librería es accedida a través de un interfaz único.



Figura 3.10: Pila de recursos que intervienen en la generación de gráficos

Existen numerosas librerías destinadas a generar gráficos tanto 2D como 3D. En este PFC se han analizado las dos más conocidas y potentes hoy en día: Direct3D y OpenGL.

²Generar imágenes a partir de un modelo.

3.4.1 Direct3D

Direct3D es un API (*Application Programming Interface*) que forma parte de la colección de librerías multimedia llamada DirectX (Fig. 3.11). Esta colección de librerías, propiedad de Microsoft, está diseñada específicamente para desarrollar aplicaciones interactivas, especialmente videojuegos [36]. Proporciona al programador un medio de acceso a los dispositivos de entrada y salida, como la pantalla y la tarjeta de sonido, para lograr generar gráficos en tres dimensiones realistas, música y efectos de sonido envolventes. Oficialmente sólo se encuentra disponible para plataformas basadas en Microsoft Windows y Xbox, aunque existe proyectos de código abierto que se han encargado de llevar a cabo proyectos de emulación y compatibilidad, como Wine, y hacer posible la portabilidad a plataformas basadas en Unix.

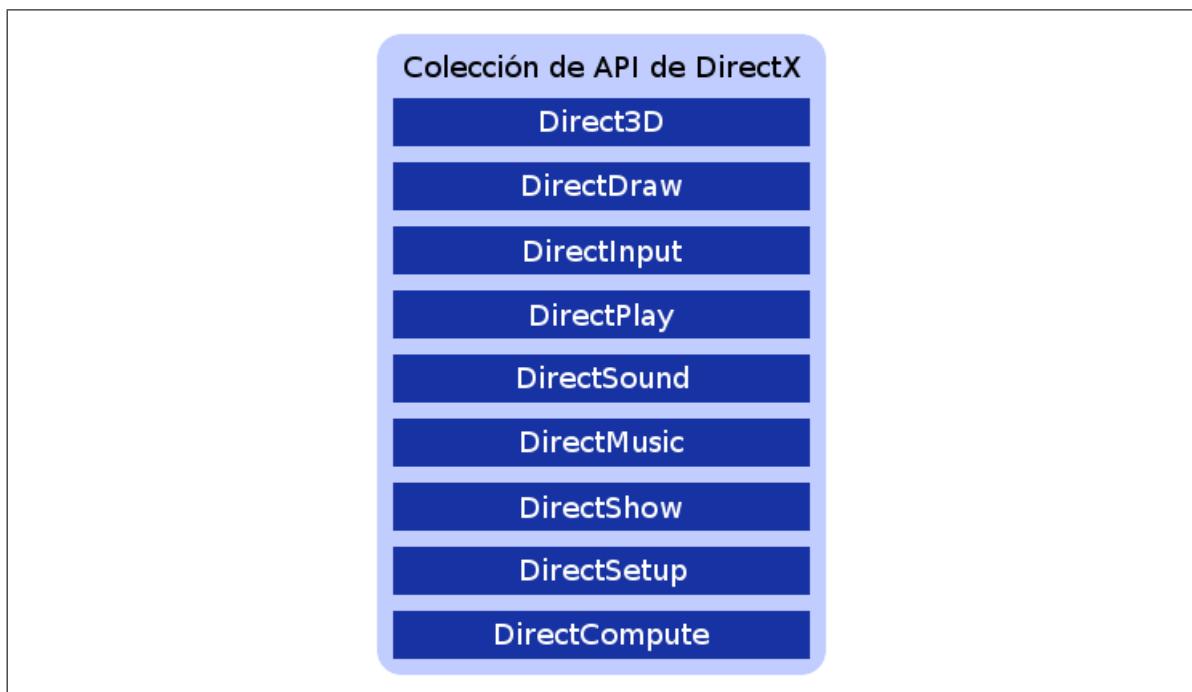


Figura 3.11: Colección de API de DirectX

DirectX contiene, entre otros, librerías para manejar dispositivos de entrada y salida (DirectInput), para comunicaciones en red (DirectPlay), para reproducción y grabación de sonidos (DirectSound), etcétera. **Direct3D**, en concreto, es la librería que proporciona un interfaz de programación para generar gráficos en tres dimensiones [64].

En las versiones 6 y 7 de DirectX, Direct3D implementaba dos API denominadas *modo retenido* y *modo inmediato*. El *modo inmediato* era un API flexible pero difícil de utilizar ya que estaba diseñada para trabajar a bajo nivel. Este modo daba soporte a todas las primitivas

de procesamiento 3D que permiten las tarjetas gráficas: iluminación, materiales, *z-buffering*³, entre otros. El **modo retenido** era una capa de abstracción que envolvía al *modo inmediato* y se basaba en una estructura de datos predefinida con pocas posibilidades de personalización, pero su uso y aprendizaje era más sencillo que el modo inmediato. Por tanto, los programadores necesitaban un API con la facilidad de uso del *modo retenido* pero que ofreciese la potencia y flexibilidad del *modo inmediato*. Esto hizo que Microsoft abandonase el mantenimiento del *modo retenido* y se centrarse en extender y mejorar el *modo inmediato* [14].

En las sucesivas versiones, Microsoft ha incorporado numerosas características a Direct3D así como optimizado las existentes para estar a la altura de su competidor más directo: OpenGL.

3.4.2 OpenGL

OpenGL (*Open Graphics Library*) [53] es una **especificación estándar** que define un API (Application Programming Interface) independiente de la plataforma y del sistema operativo, para generar gráficos en dos y tres dimensiones (Fig. 3.12) acelerados por el hardware de la máquina. También es independiente del sistema gestor de ventanas y del protocolo de red empleados [46].

El API de OpenGL comenzó siendo una iniciativa de SGI (Silicon Graphics, Inc.) para crear un API estándar e independiente de los fabricantes, que permitiese desarrollar aplicaciones con gráficos 2D y 3D. Antes del lanzamiento de OpenGL, las compañías que desarrollaban dispositivos hardware disponían de sus propias librerías de gráficos. Esta situación era frustrante para los desarrolladores de software debido a la necesidad de adaptar sus aplicaciones para soportar múltiples plataformas hardware. Esto requería portar una aplicación de una plataforma a otra, un proceso difícil y costoso. Por tanto, SGI observó la necesidad de un API de gráficos estándar y decidió formar un grupo de desarrollo para crearlo [56].

El resultado de este trabajo sucedió en Enero de 1992 y fue el API de OpenGL, que comenzó como una especificación. Después, SGI publicó un software de ejemplo que podía ser utilizado como guía de referencia por los fabricantes de hardware para crear nuevas implementaciones del API a modo de drivers entre OpenGL y su propio hardware. Además,

³El *z-buffering* es un término utilizado en el contexto de gráficos en tres dimensiones. Este concepto también se conoce como *buffer de profundidad* ya que los gráficos en dos dimensiones se basan en coordenadas (*x*, *y*) y la coordenada *z* es introducida como una tercera dimensión.

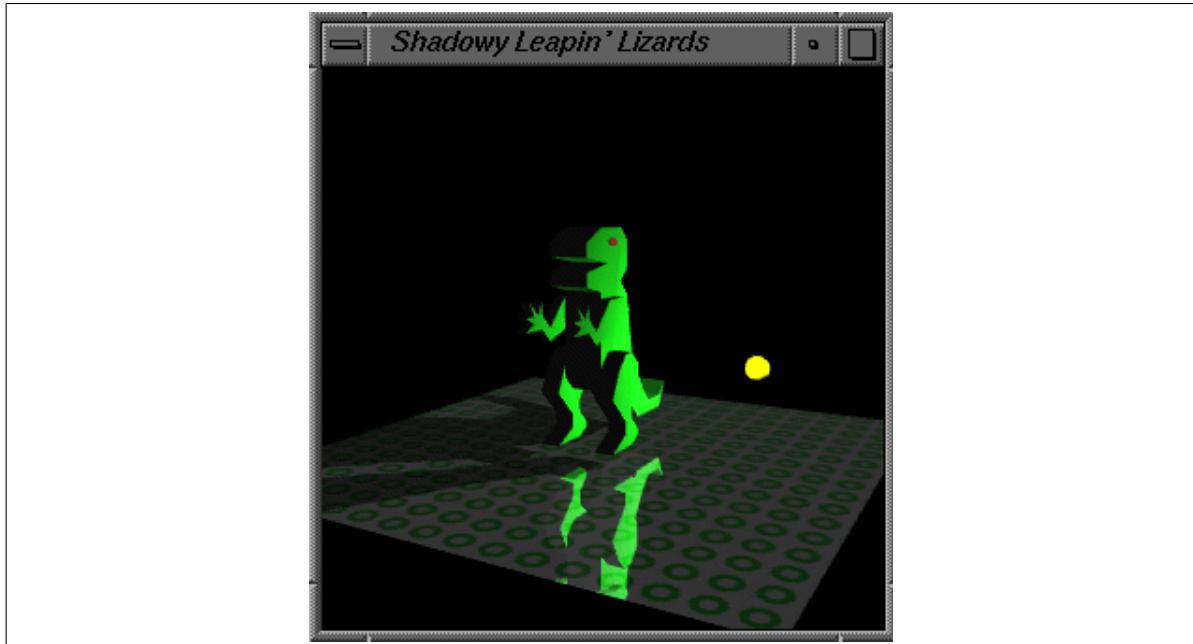


Figura 3.12: Ejemplo renderizado con OpenGL que muestra texturas, sombras y reflejos [31]

la publicación se produjo bajo una licencia de código abierto para asegurar que, por medio de los desarrolladores de este tipo de comunidades, existiesen drivers de gran calidad y alto rendimiento disponibles para Linux [55].

Desde entonces, OpenGL 1.x ha evolucionado constatemente, primero bajo la tutela de ARB (OpenGL Architecture Review Board) y después mediante el Grupo Khronos, un consorcio independiente sin ánimo de lucro que vela por crear estándares abiertos relativos a áreas multimedia para distintos dispositivos y plataformas.

En la fecha de la edición de este documento, la versión actual de OpenGL es la 4.2 y es totalmente compatible con la especificación de OpenGL ES 2.0. Cabe destacar que los desarrolladores de software no necesitan adquirir ningún tipo de licencia para utilizar OpenGL en sus aplicaciones. Sin embargo, los fabricantes de hardware sí necesitan una licencia para crear una implementación de OpenGL para su arquitectura.

Por otra parte y cambiando a un contexto más tecnológico, las aplicaciones basadas en OpenGL pueden hacer uso de una gran variedad de librerías que extienden sus funcionalidades. Las más importantes son *OpenGL Utility Library (GLU)* y *OpenGL Utility Toolkit (GLUT)*. La primera de ellas, **GLU**, proporciona rutinas de más alto nivel que las que proporciona OpenGL inicialmente. Destacan aquellas encargadas de generar texturas, NURBS (en inglés *Non Uniform Rational B-splines*)⁴, dibujar superficies cuádricas como esferas, conos y cilindros,

⁴Modelo matemático utilizado en computación gráfica para generar y representar curvas y superficies.

entre otros [46]. Por otra parte, **GLUT** proporciona métodos sencillos para manejar las ventajas de la interfaz gráfica de usuario y los dispositivos de entrada como el teclado y el ratón [23].

Según el sitio web oficial de OpenGL [53], las ventajas de este API son las siguientes:

1. **Estándar.** La especificación de OpenGL es mantenida por un grupo independiente sin ánimo de lucro (Khronos Group) con el apoyo de numerosas empresas como IBM, Sun Microsystems y Hewlett-Packard. Se trata de un estándar abierto, independiente de la plataforma y del sistema operativo.
2. **Estable.** Las implementaciones de OpenGL han estado disponibles durante más de siete años para una gran variedad de plataformas. Los cambios que han surgido a lo largo del tiempo han sido controlados y se han anunciado con antelación para que los programadores puedan adaptarse a ellos. No obstante, la *retrocompatibilidad* ha sido un compromiso para asegurar que las aplicaciones existentes no se vuelvan obsoletas.
3. **Fiable y Portable.** Todas las aplicaciones de OpenGL producen contenido visual sin importar el hardware, el sistema operativo o el sistema de ventanas sobre el que se ejecuten.
4. **Evolución.** OpenGL incorpora un mecanismo de extensibilidad para permitir a los programadores y fabricantes de hardware la posibilidad de incorporar nuevas funcionalidades.
5. **Escalable.** Las aplicaciones basadas en OpenGL pueden ejecutarse en una gran variedad de dispositivos. Esto incluye dispositivos móviles, ordenadores personales, *workstations* e incluso superordenadores.
6. **Facilidad de uso.** La convención de nombres utilizada así como el diseño estructurado del que goza OpenGL, permite escribir aplicaciones empleando menos líneas de código que con otras librerías. Además, OpenGL abstrae al programador de las características específicas del hardware.
7. **Documentación.** Existe una extensa bibliografía publicada que cubre numerosos aspectos de OpenGL, así como una gran variedad de *snippets*⁵ a modo de ejemplo.

⁵Un *snippet* es un pequeño ejemplo funcional de código ejecutable.

Por último, destacar que OpenGL inicialmente fue concebido para desarrollar aplicaciones escritas en C y C++. No obstante, existe una gran cantidad de *bindings*⁶ que permiten desarrollar este tipo de aplicaciones empleando numerosos lenguajes de programación como Fortran, Java, Perl, Pike, Python, Delphi, Ada y Visual Basic.

⁶En este contexto, un *binding* se refiere a una envoltura del API que permite el acceso a la misma desde un lenguaje de programación distinto a C o C++.

Capítulo 4

Método de Trabajo

En este capítulo se hace una especial mención al método de investigación preliminar que ha tenido lugar antes de abordar el desarrollo del proyecto. Además, se explica la metodología de desarrollo que se va a aplicar y se introduce el tema de los patrones de diseño. Para finalizar, se detalla el marco tecnológico sobre el cual se encuadra este PFC.

4.1 Marco de investigación preliminar

Tal y como se argumenta en el capítulo 1 y 2, este PFC se encuentra enmarcado dentro del proyecto de investigación **ORIGIN** (ORganizaciones Inteligentes Globales INnovadoras) [54]. Dada la naturaleza de dicho proyecto, ha sido necesario llevar a cabo un conjunto de reuniones entre el responsable del proyecto en Indra Software Labs S.L. y diferentes investigadores del Grupo Alarcos [61], entre los que se encuentran Félix Óscar García Rubio, Manuel Ángel Serrano Martín, M^a Ángeles Moraga de la Rubia (directora del PFC) y Jose Domingo López López (autor del PFC).

El objetivo de estas reuniones ha consistido en determinar los requisitos que la herramienta debe satisfacer, así como las metáforas de visualización a desarrollar y las dimensiones de sus elementos, cuyo resultado se muestra en la sección 5.1.1.

4.2 Proceso Unificado de Desarrollo (PUD)

Debido a la naturaleza de este proyecto y por su encuadre dentro de un proyecto de I+D+i, se ha optado por utilizar una metodología de desarrollo de software genérica que permita adaptarse al mismo. Por estas razones, se ha seleccionado el **Proceso Unificado de Desarrollo** (en adelante PUD) como metodología de trabajo.

El PUD es una evolución del Proceso Unificado de Rational, que define un “*conjunto de actividades necesarias para transformar los requisitos de usuario en un sistema software*”. Este método de trabajo consiste en un marco de desarrollo genérico y extensible que puede ser adaptado a organizaciones o proyectos específicos, independientemente del tamaño o área de aplicación de los mismos [29]. Sus principales características son las siguientes [15]:

- **Dirigido por casos de uso.** Para poder desarrollar un sistema es necesario saber qué necesitan sus usuarios. Un usuario puede ser un ser humano u otro sistema que interacciona con el sistema que se está desarrollando. Las necesidades de un usuario se denominan requisitos funcionales y se representan por medio de casos de uso.

Los casos de uso guían el proceso de desarrollo desde la especificación de requisitos hasta las pruebas y se utilizan para crear los modelos que permitan la construcción e implementación de los mismos. Todos los casos de uso juntos constituyen el **modelo de casos de uso**.

- **Centrado en la arquitectura.** La ingeniería informática, como cualquier otra ingeniería, elabora un diseño de la idea antes de llevarla a cabo. Este diseño, que se construye a partir del modelo de casos de uso e incluye los aspectos estáticos y dinámicos más significativos del sistema, es lo que se conoce como **arquitectura del sistema**. Para diseñar una buena arquitectura hay que tener en cuenta conceptos como los *patrones de diseño* y la *programación contra interfaces*. Estas **buenas prácticas de diseño y programación** tienen un gran impacto sobre ella y, por lo tanto, en su rendimiento, robustez, capacidad de evolución y escalabilidad.
- **Iterativo e incremental.** La estrategia *divide et impera* (en castellano, *divide y vencerás*) consiste en dividir un problema en partes más pequeñas y fáciles de resolver. El PUD (Fig. 4.1) aplica esta estrategia y divide todo el ciclo de vida del software en ciclos más pequeños, que concluyen con una versión del producto. Del mismo modo, los ciclos se dividen en iteraciones. Cada iteración aborda un conjunto de casos de uso y produce un incremento en la funcionalidad del sistema software que se está desarrollando. Dada la importancia de seleccionar y ejecutar las iteraciones de forma planificada, éstas serán distribuidas a lo largo de las cuatro fases que componen un cada ciclo y abordarán, en mayor o menor medida, cada una de las disciplinas del flujo de trabajo fundamental: requisitos, análisis, diseño, implementación y pruebas.

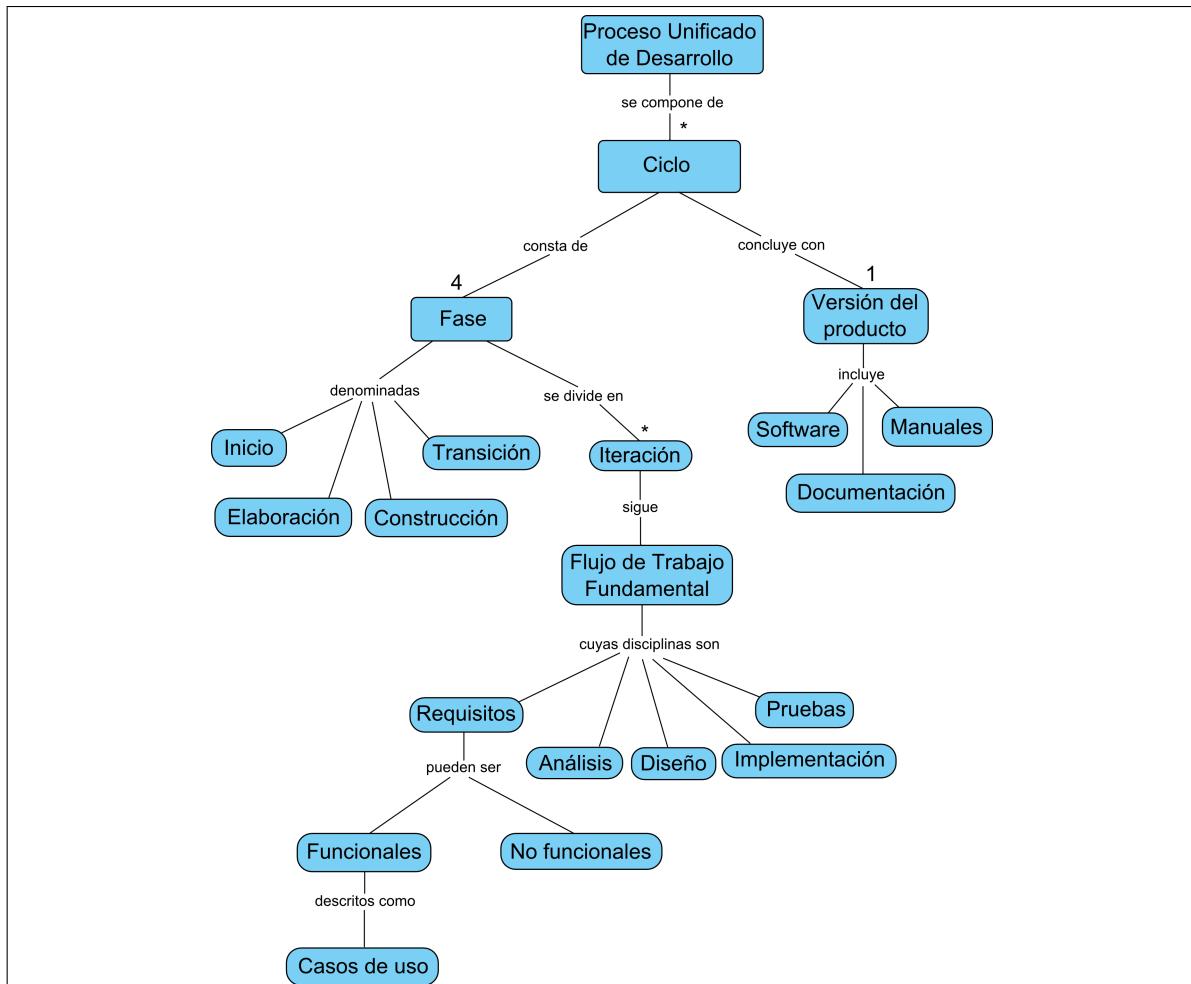


Figura 4.1: Mapa conceptual del PUD

4.2.1 Fases del Proceso Unificado de Desarrollo

La aplicación del PUD para llevar a cabo el desarrollo de un proyecto software implica la realización de una serie de ciclos que -juntos- representan el ciclo de vida completo del proyecto en cuestión. Cada ciclo concluye con una versión entregable del sistema, que incluye el software, los manuales, la documentación y los productos que se han obtenido a lo largo de las cuatro fases de las que consta: inicio, elaboración, construcción y transición.

- **Inicio.** Durante esta fase se define el alcance del proyecto y se identifican los riesgos principales. Esto permitirá llevar a cabo un estudio de viabilidad del sistema.

Los artefactos que se obtienen a lo largo de esta fase son los siguientes [62]:

- **Modelo de casos de uso.** Ya que se ha realizado una captura de requisitos, es posible identificar los requisitos funcionales del sistema y elaborar el diagrama de casos de uso.

- **Gestión del riesgo.** Se identifican y describen los posibles riesgos que puedan afectar al desarrollo del proyecto.
 - **Glosario de términos.** Se elabora un glosario con los conceptos más relevantes del dominio en el que se está trabajando.
 - **Plan de iteraciones.** Las iteraciones dedicadas a la fase de inicio se denominan *iteraciones preliminares*. Será a partir de la fase de elaboración cuando comience una sucesión de iteraciones en las que se describen los objetivos e hitos a alcanzar en las mismas.
- **Elaboración.** Esta fase se compone de un pequeño conjunto de iteraciones en el cual se mejora el modelo de casos de uso y se diseña la arquitectura base del sistema. A continuación, se desarrollan los casos de uso más críticos que se identificaron en la fase de inicio y se obtiene una *línea base* de la arquitectura.

Los artefactos que se obtienen a lo largo de esta fase son los siguientes [62]:

- **Modelo de casos de uso mejorado.** Se trabaja a partir del modelo de casos de uso obtenido en la fase de inicio, al cual se le añaden los requisitos funcionales que se hayan detectado a lo largo de la fase de elaboración.
 - **Modelo de análisis.** Este modelo, sin llegar al nivel de detalle que ofrecen los modelos de diseño, ofrece una vista más formal que el modelo de casos de uso. Se compone de los diagramas de clases de análisis y diagramas de comunicación de los casos de uso más importantes.
 - **Modelo de diseño.** Este modelo se compone por dos categorías de diagramas: estáticos, que muestran una vista estática del sistema -por ejemplo, diagramas de clases-, y dinámicos, que muestran una vista dinámica del sistema -por ejemplo, diagramas de secuencia e interacción-.
 - **Modelo de datos.** Es una abstracción del dominio en el que se trabaja, que describe los elementos que intervienen en la resolución del problema y sus relaciones, tanto a nivel conceptual, como lógico y físico (o persistente).
- **Construcción.** Esta fase, que utiliza como entrada todos los modelos producidos en las fases anteriores, aborda la mayor parte de la implementación de los requisitos

funcionales del sistema. De manera conjunta, diseña y ejecuta las pruebas para las funcionalidades que se implementan.

Los artefactos que se obtienen a lo largo de esta fase son los siguientes [62]:

- **Modelo de diseño mejorado.** Se refina y completa el modelo de diseño obtenido en la fase de elaboración.
- **Modelo de implementación.** Consta de los diagramas de componentes y despliegue, que representan los elementos que participan en la ejecución del sistema (librerías, paquetes, etcétera), sus relaciones y dónde se ejecutan cada uno de ellos.
- **Modelo de pruebas.** Consta de los diferentes diseños de casos de prueba tanto unitarias como funcionales.
- **Transición.** Al comienzo de esta fase se dispone una versión del producto prácticamente completa. Durante el desarrollo de la misma se completa la implementación del sistema y se refinan los modelos obtenidos en la fase de construcción. Cuando esta fase concluye se dispone de una versión del producto lista para ser entregada al cliente, junto con los manuales y documentos obtenidos a lo largo de todo el ciclo.

4.2.2 Disciplinas del flujo de trabajo fundamental

Cada una de las fases explicadas en la sección 4.2.1 se divide en una o varias iteraciones. Como ya se ha explicado anteriormente, una iteración aborda un conjunto de casos de uso y produce un incremento en la funcionalidad del sistema software que se está desarrollando. Esta situación se da porque cada iteración ejecuta por completo el **flujo de trabajo fundamental**, que se compone de cinco disciplinas: requisitos, análisis, diseño, implementación y pruebas (Fig. 4.2).

- **Requisitos.** Esta disciplina, cuyo mayor grado de participación se da en las fases de inicio y elaboración, se centra en la captura e identificación de requisitos. Para ello, el equipo de desarrollo y los clientes establecen comunicaciones por diferentes medios para especificar qué es lo que se espera del sistema que se va a desarrollar.

Los requisitos identificados que describen una funcionalidad del sistema se denominan **requisitos funcionales** y son descritos mediante casos de uso. Por otro lado, cualquier

otra característica requerida del sistema que no forme parte de una funcionalidad, recibe el nombre de **requisito no funcional**.

- **Análisis.** Su mayor grado de participación se da en la fase de elaboración. En ella se elabora una especificación más detallada de los casos de uso tratando de obtener una comprensión más precisa de los requisitos del sistema. Por otro lado, se elabora el modelo de análisis que ofrecerá al equipo de desarrollo una vista más específica del sistema sin llegar al nivel de detalle del modelo de diseño.
- **Diseño.** Durante esta disciplina se elabora el modelo de diseño, que comienza a alejarse de los modelos conceptuales y ofrece un mayor nivel de detalle cada vez más cercano a la implementación.
- **Implementación.** Durante esta disciplina, que toma su mayor grado de participación en la fase de construcción, se implementan las decisiones tomadas en la etapa de diseño. El objetivo que se persigue consiste en construir las funcionalidades que satisfagan los casos de uso identificados en la etapa de análisis.
- **Pruebas.** Según el PUD, los componentes deben ser probados a medida que son implementados. Así pues, esta disciplina se encuentra presente en todas las fases en las que se ha llevado a cabo la implementación de alguna funcionalidad. La disciplina de pruebas tiene por objetivo el diseño e implementación de pruebas unitarias y funcionales del sistema, así como de la automatización de su ejecución siempre y cuando sea posible.

4.3 Patrones de diseño

El desarrollo de sistemas software, en general, y sistemas software orientados a objetos, en particular, es una tarea difícil y costosa en términos económicos y temporales, entre otros. Si además se requiere que el sistema desarrollado o partes del mismo sean reutilizables, se puede convertir en un auténtico desafío.

El diseño de un software orientado a objetos requiere una abstracción y conceptualización del dominio en el que se trabaja para poder identificar los objetos relevantes y diseñar las clases que los representan. Las clases se deben diseñar con la **granularidad** adecuada, ya que un *grano grueso* comprometería la reutilización de las mismas y violaría el principio de **bajo**

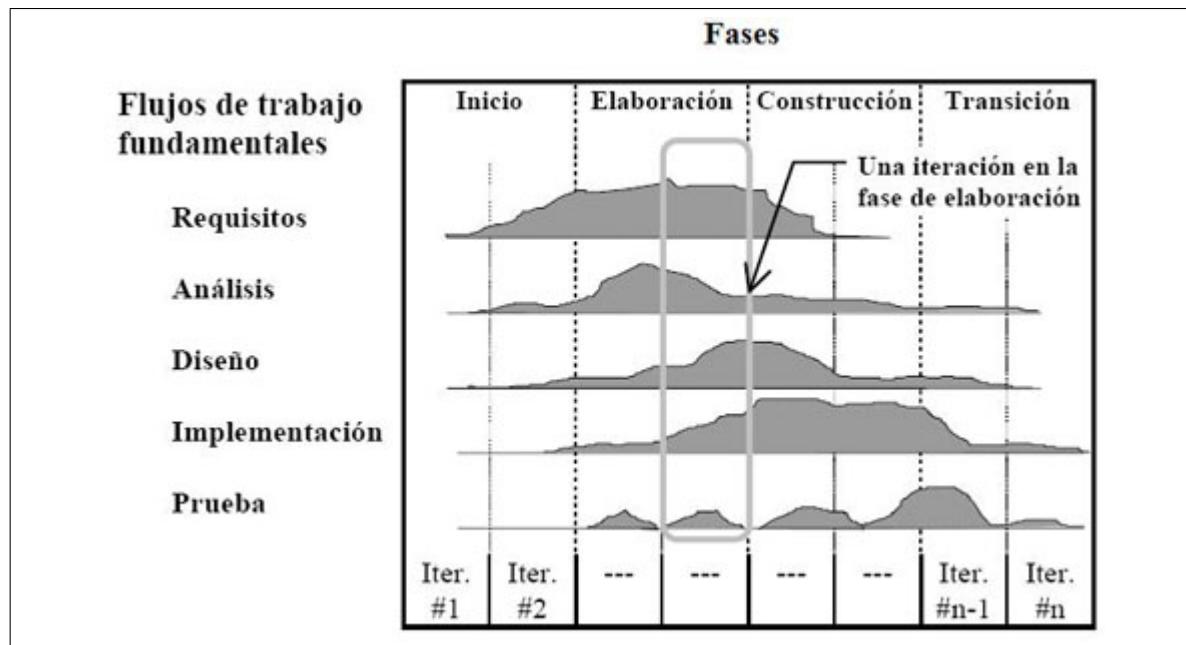


Figura 4.2: Grado de participación de las disciplinas del flujo de trabajo fundamental en las iteraciones del PUD [45]

acoplamiento, y un *grano fino* aumentaría la complejidad del sistema y violaría el principio de **alta cohesión**. Además, es necesario definir interfaces, herencias y las relaciones entre clases. Todos estos factores que intervienen en el diseño de un sistema orientado a objetos han de ser específicos al contexto y, a su vez, lo suficientemente genéricos debido a razones de extensibilidad por futuros problemas o requisitos que puedan surgir [21].

En el diseño de sistemas software, cualquiera que sea su área de aplicación, existen problemas recurrentes que se dan una y otra vez. Por ejemplo, cuando se diseña una nueva aplicación a menudo surge la necesidad de querer notificar a la interfaz gráfica de usuario los cambios que se producen en el estado de determinados objetos, y evitar a toda costa cualquier tipo de dependencia de la capa de dominio con la capa de presentación. Para resolver este problema existen soluciones de las que ya se conoce su eficacia y se ha comprobado que funcionan. Además, este problema y muchos otros, surgen en repetidas ocasiones a la hora de diseñar distintos sistemas, por lo que es fácil encontrar varias situaciones -en el contexto del diseño orientado a objetos- en las que se sigue un *patrón* en el diseño de las clases o la comunicación de los objetos.

En 1977, un arquitecto británico llamado Christopher Alexander definió el término **patrón**, en referencia a problemas relacionados al diseño de edificios y ciudades, de la siguiente manera: “*cada patrón describe un problema que ocurre una y otra vez en nuestro entorno y,*

puede ser estudiado para descubrir la esencia de su solución, por lo que se puede utilizar esta solución un millón de veces sin repetir el estudio una segunda vez.” [6]. Desde entonces surgen definiciones más directas como “un patrón es una solución recurrente para un problema recurrente dentro de un contexto”, “un patrón es una solución recurrente para un problema estándar” o “un patrón resuelve un problema dentro de un contexto” [5].

A mediados de los años 90, un grupo de autores que se hacían llamar **the gang-of-four** (*el grupo de los cuatro*), adaptaron el término a la ingeniería del software y publicaron un conjunto de **patrones software**, basados en buenas prácticas de diseño y programación, que proponían soluciones a problemas comunes. Estos patrones resuelven problemas de diseño específicos y hacen de la programación orientada a objetos una disciplina más flexible, elegante y reutilizable. Así pues, no es coherente tratar de diseñar una solución desde cero y se recomienda crear nuevos diseños a partir de la experiencia previa [21]. La forma de capturar y transmitir experiencias para que puedan ser mejoradas y utilizadas otra vez se registra en forma de **patrones de diseño**, análogamente al modisto que diseña y guarda el patrón de un vestido.

Un patrón de diseño se compone de un nombre, una explicación del problema que pretende resolver, la solución propuesta y una evaluación que indica las ventajas e inconvenientes de su uso. La evaluación es un factor muy importante ya que incluye el impacto que tiene el uso del patrón sobre la flexibilidad, portabilidad y extensibilidad del sistema, y permite estudiar otras alternativas. Por otro lado, los patrones de diseño mejoran la documentación y el mantenimiento ya que proporcionan una especificación explícita de las clases que participan, la comunicación entre objetos y su propósito y, además, son ampliamente conocidos por un gran número de diseñadores y desarrolladores.

A modo de síntesis, a continuación se enumeran las ventajas que proporciona el uso de patrones de diseño [21]:

- **Enriquecen el lenguaje de modelado.** Los patrones de diseño enriquecen los lenguajes de modelado como UML. De hecho, algunas herramientas de modelado permiten crear y aplicar patrones de diseño. Gracias a ellos, no es necesario comenzar los diseños desde cero y a partir de entidades primitivas como clases e interfaces, sino que se pueden combinar en bloques y modelar desde un nivel de abstracción superior.
- **Mejoran la documentación.** Al utilizar patrones de diseño se están empleando -lo que los expertos consideran- buenas prácticas de diseño y programación que son

ampliamente conocidas por otros diseñadores y desarrolladores.

- **Agilizan el diseño y la implementación.** Dado que los patrones de diseño proporcionan un nivel superior de abstracción, no es necesario comenzar el trabajo desde cero ya que fomentan la reutilización de diseños e implementaciones previas.
- **Aumento de la calidad.** Los patrones se basan en la experiencia previa, por lo que reducen fallos, son de eficacia probada y evitan la búsqueda de la solución a un problema que ya fue resuelto. Por lo tanto, son más correctos y robustos que una solución nueva.
- **Capacidad de reutilización.** Permiten reutilizar diseños y código para evitar reinventar soluciones a problemas ya resueltos.

4.4 Marco tecnológico

El conjunto de herramientas, tecnologías y frameworks que se emplean durante todo el ciclo de vida de este PFC es muy extenso. A continuación se enumeran los elementos más destacables, incluyendo una breve descripción de la misma y la versión empleada, si procede.

4.4.1 Herramientas de gestión de proyectos

En esta sección se muestran las herramientas empleadas para la gestión de este PFC durante todo su ciclo de vida.

4.4.1.1 Maven

Versión: 2.2.1

Maven es una herramienta de compartición de conocimiento y gestión de proyectos basados en tecnología Java. Maven establece un modelo (POM, en su acrónimo inglés Project Object Model) para especificar distintos aspectos del proyecto. Este modelo, que se representa en formato XML (Listado 4.1), permite a Maven gestionar distintos aspectos del ciclo de vida del proyecto, como la construcción, generación de informes y documentación, gestión de dependencias, gestión de la configuración, lanzamiento de versiones, distribución, etcétera.

Inicialmente, el objetivo de Maven consistía en conseguir que distintos proyectos de la **Fundación Apache** trabajasen de un modo común. Así, los desarrolladores podrían trabajar en cualquier proyecto sin tener que invertir tiempo en aprender aspectos específicos acerca

de su construcción, ya que todos utilizarían el mismo formato. Después, esta misma idea se aplicó a distintas disciplinas como pruebas, generación de documentación, generación de informes, despliegue, etcétera.

A día de hoy muchas comunidades, empresas y organizaciones gestionan sus proyectos a través de Maven.

Por último, señalar que la funcionalidad de Maven es cada vez más amplia gracias a la posibilidad de creación e incorporación de plugins. En la actualidad ya existe una amplia oferta de ellos.

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4           http://maven.apache.org/xsd/maven-4.0.0.
5           xsd">
6 
7   <modelVersion>4.0.0</modelVersion>
8 
9   <groupId>com.mycompany.app</groupId>
10 
11  <artifactId>my-app</artifactId>
12 
13  <packaging>jar</packaging>
14 
15  <version>1.0-SNAPSHOT</version>
16 
17  <name>Maven Quick Start Archetype</name>
18 
19  <url>http://maven.apache.org</url>
20 
21  <dependencies>
22 
23    <dependency>
24 
25      <groupId>junit</groupId>
26 
27      <artifactId>junit</artifactId>
28 
29      <version>3.8.1</version>
30 
31      <scope>test</scope>
32 
33    </dependency>
34 
35  </dependencies>
36 
37 </project>
```

Listado 4.1: Ejemplo de configuración de un fichero *pom.xml* [52]

4.4.1.2 Apache Subversion

Apache Subversion es el sistema de control de versiones que se emplea a lo largo de todo el ciclo de vida de este PFC. Su uso permite realizar un seguimiento del estado de las versiones y sus cambios, además de la integración de las partes del software (código ejecutable,

documentación, etcétera) en un solo producto.

4.4.1.3 Google Code Project Hosting

Google Code Project Hosting es un servicio -rápido, fiable y sencillo- de alojamiento de proyectos software. Ofrece varias herramientas tales como *Subversion*, *Mercurial* y *Git* para el control de versiones, una *wiki* para la publicación de documentación, una sección de descargas, gestión de problemas (*issue management*), entre otros. Además, permite consultar código fuente en un visor integrado y utilizar herramientas de revisión de código y revisión de contribuciones.

4.4.2 Herramientas de modelado de software y elaboración de documentación

En esta sección se muestran las herramientas empleadas para la elaboración de la documentación del proyecto y de los distintos diagramas y modelos que sean necesarios en cada una de las fases del ciclo de vida de este PFC.

4.4.2.1 MySQL Workbench

Versión: 5.2 CE

MySQL Workbench (Fig. 4.3) es una herramienta que permite diseñar, configurar y administrar bases de datos y servidores de bases de datos de un modo sencillo e intuitivo. Para ello proporciona al usuario funcionalidades para modelado de datos, administración de usuarios, etcétera.

Una de las funcionalidades más interesantes consiste en la generación automática del *script* de instalación de la base de datos a partir del modelo EER (Entidad-Relación Extendido), el cual permite diseñar mediante una herramienta de dibujo.

4.4.2.2 Visual Paradigm for UML

Versión: 8.0

Visual Paradigm (Fig. 4.4) es una herramienta CASE (Computer Aided Software Engineering) profesional para modelado UML (Unified Modeling Language). Ofrece un conjunto de funcionalidades muy amplio, tales como modelado de clases de análisis y diseño o modelado

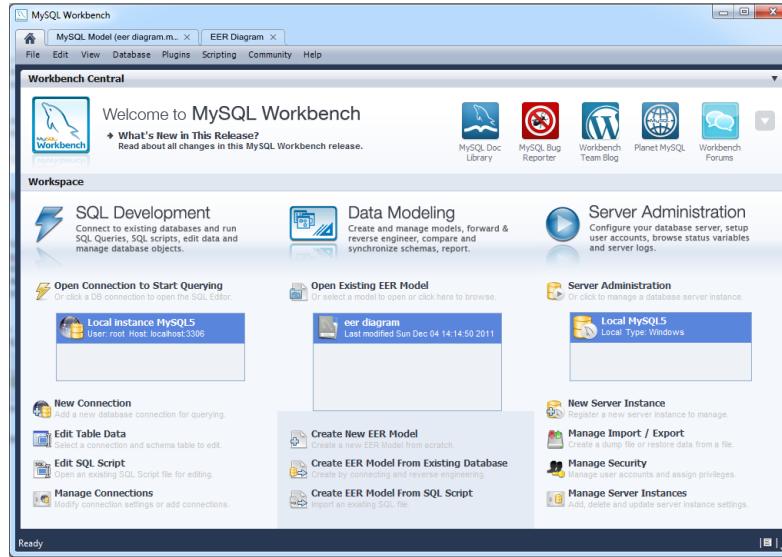


Figura 4.3: Captura de pantalla de la herramienta MySQL Workbench 5.2 CE

de datos, que son necesarios para etapas como la captura de requisitos, la planificación del proyecto y la planificación de las pruebas.

En este PFC se utilizará durante las disciplinas de análisis y diseño con el fin de elaborar los diagramas UML necesarios: casos de uso, clases de análisis, comunicación, clases de diseño, secuencia, interacción, despliegue, componentes, etcétera.

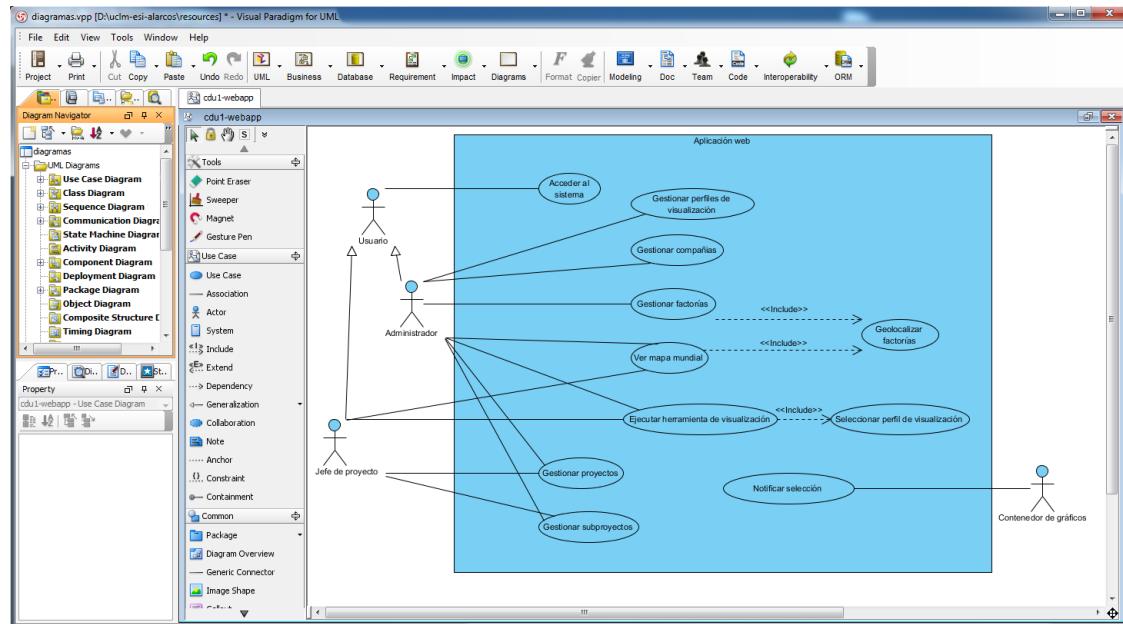


Figura 4.4: Captura de pantalla de la herramienta Visual Paradigm 8.0

4.4.2.3 L^AT_EX

L^AT_EX es un sistema de composición de textos de alta calidad diseñado para la creación de documentos científicos y técnicos. Además, es el estándar *de-facto* para la publicación de documentos científicos.

En este PFC se ha empleado para la elaboración de este documento.

4.4.2.4 BIBL^AT_EX

BIBL^AT_EX es, paralelamente, una herramienta y un formato de fichero que se emplea para gestionar la bibliografía y referencias de documentos generados con L^AT_EX.

4.4.2.5 Inkscape y GIMP

Versión: 0.48 y 2.6.11, respectivamente.

Por un lado, Inkscape es un editor de gráficos vectoriales, con capacidades similares a Adobe Illustrator y Macromedia Freehand. Por otro, GIMP (GNU Image Manipulation Program) es un programa para manipulación y edición de imágenes, con capacidades similares a Adobe Photoshop.

Se trata de dos herramientas de dibujo muy potentes, gratuitas, de código abierto y multiplataforma, que serán empleadas para la generación y manipulación de las imágenes utilizadas tanto en el software como en la elaboración de la documentación asociada.

4.4.3 Herramientas, tecnologías y frameworks de desarrollo software

En esta sección se muestran las herramientas, tecnologías y frameworks empleados para el desarrollo de la herramienta software que se incluye en este PFC.

4.4.3.1 Eclipse IDE

Versión: EE Helios

Eclipse es un IDE (Integrated Development Environment) de código abierto, multiplataforma y ampliable en cuanto a funcionalidades mediante la incorporación de plugins.

En este PFC será utilizado para la construcción e implementación la herramienta y sus componentes asociados. Además, su funcionalidad será ampliada mediante la incorporación del plugin **m2eclipse** (m2e) [51], que proporcionará soporte integrado para Apache Maven.

De este modo será más sencillo editar el fichero *pom.xml*, gestionar las dependencias y construir el proyecto desde el propio IDE, entre otros.

4.4.3.2 MySQL Community Server

Versión: 5.1.11

MySQL Server es un SGBD (Sistema Gestor de Bases de Datos) relacional multi-hilo, multi-usuario y que soporta el lenguaje SQL (Structured Query Language).

Posee doble licencia: una GNU GPL para aquellos usuarios que decidan utilizar el software bajo los términos de la misma, y otra comercial para aquellos que quieran incorporarlo en productos privativos.

4.4.3.3 Apache Tomcat

Versión: 7.0.2

Se trata de un servidor web y contenedor de servlets de código abierto que implementa las especificaciones de los servlets y JSP (JavaServer Pages) de Sun Microsystems.

Este servidor será empleado tanto en el desarrollo como en el despliegue la herramienta.

4.4.3.4 Spring Framework

Versión: 2.5.6

Spring Framework [57] es un popular *framework* de desarrollo de aplicaciones basadas en Java, cuyo objetivo es facilitar el desarrollo de las mismas promoviendo buenas prácticas de diseño y programación. Proporciona un conjunto de mecanismos muy bien documentados y fáciles de usar para acceder a una amplia colección de tecnologías que mejoran la productividad y el desarrollo de sistemas software. De este modo, los desarrolladores pueden centrarse en cuestiones relativas con la lógica de negocio de la aplicación y abstraerse de la conexión entre componentes y sistemas.

Spring es una plataforma diseñada en capas para distintos niveles de abstracción. Una de las características más populares de Spring Framework es la implementación del acceso a datos. Destaca la gestión de transacciones, la abstracción de JDBC que simplifica el manejo de errores y reduce la cantidad de código a implementar, integración con Hibernate, JDO, iBATIS y SQL Maps, y una implementación del patrón MVC para aplicaciones web.

Además existen numerosas extensiones para multitud de propósitos. A continuación se listan algunas de las más importantes:

- **Spring Security.** Es un framework configurable para el control de acceso y autenticación.
- **Spring Mobile.** Es una extensión de la implementación del patrón MVC de Spring, creado específicamente para el desarrollo de aplicaciones web para dispositivos móviles como Android, iOS, Blackberry, Windows Phone 7 o J2ME.
- **Spring Android.** Se trata una extensión de Spring destinada a simplificar el desarrollo de aplicaciones nativas de Android.
- **Spring Social.** Es una extensión de Spring que permite conectar las aplicaciones con proveedores de SaaS (Software-as-a-Service) como Facebook y Twitter. SaaS es un modelo de distribución de software basado en el paradigma cliente-servidor, en el que el servidor aloja el software y proporciona a los clientes el acceso al mismo a través de un navegador web convencional o un cliente específico.
- **Spring Web Flow.** Se trata de una extensión de la implementación del patrón MVC de Spring, que permite implementar el *flujo* de una aplicación web. El flujo encapsula una secuencia de pasos que guían al usuario durante la ejecución de una tarea.

4.4.3.5 Spring Security

Versión: 2.0.4

Spring Security es un *framework* potente y configurable que proporciona mecanismos de autenticación y permite implementar políticas de control de acceso a recursos. Es el estándar *de-facto* para la seguridad de aplicaciones basadas en Spring.

4.4.3.6 Struts

Versión: 2.2.3

Struts es un *framework* de código abierto que permite desarrollar aplicaciones web¹ basadas en tecnología Java, como JSP o servlets, mediante el patrón MVC (Modelo-Vista-

¹Una aplicación web difiere de una página web convencional por su capacidad de crear respuestas dinámicas y no generar únicamente páginas estáticas.

Controlador), permitiendo así reducir el acoplamiento entre la interfaz de usuario de la aplicación web y la lógica de negocio.

La funcionalidad de Struts es similar a Spring MVC, Stripes o Apache Tapestry.

4.4.3.7 Struts-Menu

Versión: 2.4.3

Struts-menu es un framework que permite generar dinámicamente los menús de una página web basada en Struts y JSP.

Proporciona un conjunto de etiquetas que permite crear menús y controlar su renderizado en base a los roles del usuario. Además, está totalmente integrado con el sistema de autenticación del contenedor web, que en este caso es *Spring Security* y sus definiciones se realizan mediante ficheros XML.

4.4.3.8 JSON-lib

Versión: 2.4

JSON-lib es una librería basada en Java que permite transformar distintos tipos de estructuras de datos a formato JSON y viceversa. Entre los tipos de datos que maneja destacan distintos tipos que implementan las interfaces *List*, *Collection* y *Map*, entre otros.

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero. Su funcionalidad es similar a XML, pero es más sencillo de analizar y generar, tanto para humanos como para procesadores de textos, al no contener etiquetas y basarse en separadores de caracteres. Además, es completamente independiente del lenguaje de programación.

4.4.3.9 Struts2-JQuery

Versión: 3.1.0

Se trata de un *plugin* para el *framework* Struts2 que proporciona mecanismos de acceso para las funcionalidades de **Ajax** mediante jQuery 1.5.2, y un conjunto de widgets mediante jQuery UI 1.8.14.

Este plugin será necesario para lograr que la aplicación web goze de dinamismo y tenga una apariencia más atractiva.

4.4.3.10 JUnit

Versión: 4.8.2

Se trata de un framework dedicado a la **automatización de pruebas unitarias** para aplicaciones basadas en Java. Además, se integra perfectamente con Maven para la generación de informes de pruebas y de cobertura de código.

4.4.3.11 Spring Test

Versión: 2.5.6.SEC03

Se trata de una extensión de Spring Framework que permite integrar el desarrollo de pruebas como parte del desarrollo de la aplicación. Para ello permite incorporar el núcleo de Spring (el contendor IoC -Inversion of Control-, responsable de la creación y configuración de objetos) en las pruebas unitarias con el objetivo de que éstas puedan beneficiarse de él.

4.4.3.12 Canoo WebTest

Versión: R_1812

Canoo WebTest es una herramienta de código abierto que facilita la automatización de las pruebas funcionales de aplicaciones web. Permite especificar los casos de prueba mediante ficheros XML y la ejecución automatizada de los mismos mediante tareas **Ant**.

Además, genera los informes con los resultados de ejecución de las pruebas en formato HTML. Por otro lado, goza de una integración perfecta con Maven, gracias al plugin **webtest-maven-plugin**, para realizar las pruebas de integración, es decir, aquellas que se realizan sin la necesidad de desplegar la aplicación web en un servidor web.

4.4.3.13 Google Maps API

Versión: v2

Google Maps es un **Sistema de Información Geográfica** (SIG o GIS, en su acrónimo inglés Geographic Information System) que ofrece un API para desarrolladores basada en JavaScript. Gracias a ella es posible insertar un mapa de Google en una página web, así como manipular el mapa, superponer datos propios y añadir contenido a través de diferentes servicios que proporciona.

En este PFC se empleará para geolocalizar las factorías de software.

4.4.3.14 Displaytag

Versión: 1.2

Se trata de una librería de código abierto, que proporciona un conjunto de etiquetas mediante las cuales es posible mostrar y manipular tablas en una página web, sin violar el principio del patrón MVC.

Soporta operaciones tales como paginación, agrupación de columnas, ordenación de filas, exportación de los resultados a distintos formatos como XLS, PDF, XML o CSV, etcétera.

4.4.3.15 SiteMesh

Versión: 2.4.2

Se trata de un *framework* para desarrollo de aplicaciones web basadas en Java, que gestiona aspectos de decoración y diseño para proporcionar un *look&feel* consistente entre las distintas páginas de un mismo sitio web.

Para configurarlo es necesario definir una página modelo, denominada *decorador*, en la que se especifica el diseño de la página y en qué lugares se insertarán los distintos contenidos. Acto seguido, SiteMesh interceptará todas las peticiones que los distintos clientes hagan al servidor web, tanto de páginas HTML generadas de manera estática como de manera dinámica, para procesar su contenido adaptándolo a la definición del *decorador* y construir el resultado final.

4.4.3.16 JSTL: JavaServer Pages Standard Tag Library

Versión: 1.2

Se trata de una librería que encapsula funcionalidades comunes que son utilizadas en el desarrollo de aplicaciones web. Entre ellas destaca el soporte a la programación en cuanto a estructuras condicionales e iteraciones, manipulación de documentos XML, internacionalización y sentencias SQL.

En el listado 4.2 se muestra un ejemplo de estructuras condicionales con JSTL.

```
1 <c:choose>
2   <c:when test="\${test1}">
3     // Si test1 es verdadero, entonces...
4   </c:when>
5   <c:otherwise>
```

```
6 <c:if test="\${test2}">
7     // Si test1 es falso y test2 es verdadero, entonces....
8 </c:if>
9     // Si test1 es falso e independientemente de test2, entonces
10    ...
11 </c:otherwise>
12 </c:choose>
```

Listado 4.2: Ejemplo de estructuras condicionales con JSTL

4.4.3.17 JavaScript

JavaScript es un lenguaje de programación interpretado que, ejecutado en el lado del cliente, permite desarrollar contenidos web dinámicos evitando recargas innecesarias de la página. De este modo se puede lograr que la interfaz gráfica de usuario de una aplicación web sea tan interactiva como la de una aplicación de escritorio.

El código JavaScript se ejecuta sin la necesidad de ser compilado previamente y su intérprete está implementado en la mayoría de los navegadores web que se utilizan hoy en día.

Para que los programadores no lidien directamente con JavaScript, existen frameworks como JQuery y Dojo que agilizan el desarrollo con este lenguaje y amplían su funcionalidad al automatizar la integración con AJAX (Asynchronous JavaScript And XML).

4.4.3.18 Java Annotations

Una anotación, en el lenguaje de programación Java, es un método para añadir metadatos al código fuente. Estos metadatos, que pueden aplicarse a clases, métodos, variables, parámetros y paquetes, podrán ser analizados en tiempo de compilación o tiempo de ejecución para proporcionar la información que se considere oportuna.

Las anotaciones Java proporcionan información sobre el programa, pero no afectan a su funcionamiento. Además, son completamente accesibles a nivel de aplicación utilizando el método de **introspección**. Por otro lado, la información que proporcionan las anotaciones puede ser utilizada por distintas herramientas para generar código usando el método de **reflexión**.

En este PFC se utiliza esta tecnología para analizar diferentes clases cuyos atributos pueden variar a lo largo del tiempo y condicionan las operaciones de visualización y generación de

escenas en tres dimensiones (Secciones 5.2.2.3 y 5.2.3.3).

4.4.3.19 Java Reflection API

Gracias a este API es posible aplicar las técnicas de **reflexión** e **introspección**. Se trata de dos técnicas muy potentes y complejas, ya que requieren un alto conocimiento del lenguaje de programación, que permiten a las aplicaciones llevar a cabo operaciones que de otro modo sería imposible.

La *reflexión* es una técnica mediante la cual un programa informático puede observar y modificar su estructura y comportamiento en tiempo de ejecución. Esta técnica suele utilizarse junto con la **introspección**, que es la capacidad que poseen algunos lenguajes de programación orientados a objetos para determinar el tipo de un objeto en tiempo de ejecución.

4.4.3.20 JAXB: Java Architecture for XML Binding

Versión: 2.2.4

JAXB es un framework integrado dentro de Java, que permite obtener un documento XML a partir de una jerarquía de objetos Java, y viceversa. El proceso de obtención del documento XML se denomina *serialización*, y recibe los nombres de *marshalling* o *serializing*. El proceso inverso, *deserialización*, se conoce como *unmarshalling* o *deserialization*.

La característica más destacable de esta tecnología es su facilidad de uso por medio de anotaciones. Para ello es necesario anotar las clases correspondientes mediante anotaciones de JAXB e implementar un pequeño adaptador que proporcione acceso a las funciones de *serialización* y *deserialización* de JAXB.

En este PFC, los ficheros XML son utilizados para hacer persistentes los perfiles de visualización.

4.4.3.21 Hibernate

Versión: 3.5.4

Hibernate es un *framework*, de código abierto, de tipo ORM (Object-Relational Mapping). Permite asociar clases Java con tablas de una base de datos relacional para gestionar la persistencia de una aplicación y ejecutar consultas sobre la base de datos. Las consultas se realizan mediante los lenguajes HQL (Hibernate Query Language) y SQL (Structured Query Language).

El objetivo de Hibernate consiste en mecanizar prácticamente todas las tareas de programación relativas a la persistencia. Así pues, el programador no debe implementarlas a mano mediante SQL y JDBC.

Además, Hibernate contiene un paquete de anotaciones de que incluye las anotaciones de JPA (Java Persistence API), que son estándar de Java, y EJB.

4.4.3.22 JOGL: Java binding for the OpenGL API

Versión: 2.0

JOGL [49] es una biblioteca que proporciona acceso a la librería de gráficos OpenGL (Sección 3.4.2) a través del lenguaje de programación Java. De este modo es posible renderizar gráficos 2D y 3D acelerados por hardware desde una aplicación basada en Java.

JOGL proporciona acceso completo a las versiones del API relativas a las especificaciones 1.3 - 3.0, 3.1 - 3.3, ≥ 4.0 , ES 1.x y ES 2.x (Embedded Systems).

Además, está completamente integrado con los componentes proporcionados por Swing y AWT para facilitar la creación de una interfaz gráfica. Aún así, en las últimas versiones también incluye un **gestor de ventanas nativo** de alto rendimiento llamado **NEWT (Native Windowing Toolkit)**.

Capítulo 5

Resultados

En este capítulo se detallan los resultados obtenidos como consecuencia de la aplicación del método de trabajo expuesto en las secciones 4.1 y 4.2, que consiste en una herramienta compuesta de dos componentes bien diferenciados: un motor encargado de generar gráficos 3D y una aplicación web que lo integra permitiendo realizar las tareas de gestión, configuración y visualización. Para su consecución, se han empleado todas y cada una de las herramientas, frameworks y tecnologías expuestas en la sección 4.4.

Según la documentación aportada en la sección 4.2 y la figura 4.1, el desarrollo del software se lleva a cabo en ciclos. En este PFC cada componente puede ser abordado en un ciclo distinto dado que constituyen productos software independientes. No obstante, ya que todo el desarrollo se realiza por el alumno del presente PFC, ambos componentes se desarrollan de forma conjunta en un único ciclo. El producto resultante de este ciclo es la herramienta software desarrollada, así como su documentación, diagramas y manuales.

Un ciclo consta de 4 fases: inicio, elaboración, construcción y transición. Estas fases conforman la estructura básica de este capítulo. Dentro de cada fase se desarrollan una o más iteraciones. En cada iteración se ejecuta el flujo de trabajo fundamental, compuesto por las disciplinas de requisitos, análisis, diseño, implementación y pruebas. La envergadura de cada disciplina, que en determinados casos será nula, está condicionada por la fase a la que pertenece la iteración que se está llevando a cabo.

Para evitar que este capítulo y, como consecuencia, todo el documento se prolongue en exceso, no se profundiza del mismo modo en todas las iteraciones y sus disciplinas del flujo de trabajo ni se incluyen todos los diagramas e informes obtenidos. Por otro lado, sí se incorporan algunos ejemplos de ficheros de configuración y fragmentos de código fuente que pueden resultar de especial interés en la elaboración de futuros proyectos o el mantenimiento de este mismo proyecto.

5.1 Fase de inicio

La fase de inicio goza de diferencias notables con respecto al resto de fases, ya que no se estructura en base a iteraciones y no produce resultados funcionales. Así pues, se centra en la captura e identificación de requisitos, el estudio de viabilidad del sistema y la realización del plan de iteraciones.

5.1.1 Captura de requisitos

Dada la naturaleza de este PFC y su encuadre dentro de un proyecto de I+D+i, ha sido necesario realizar una serie de reuniones entre los investigadores del Grupo Alarcos y el responsable del proyecto en Indra Software Labs S.L.. El objetivo de estas reuniones se ha centrado tanto en identificar las necesidades que el sistema debe satisfacer como en cuestionar y tratar algunas decisiones de diseño. Así pues, esta sección se corresponde con una parte de la captura de requisitos.

Tal y como se ha tratado en la sección 2.2, se desea construir una herramienta de gestión organizacional del desarrollo global y de visualización de información relevante, tales como la geolocalización de las factorías donde se desarrollan las distintas partes de un proyecto o la representación gráfica de diversas medidas e indicadores de calidad y productividad del software. Dichas medidas e indicadores se corresponderán con la información que se considere importante como -por ejemplo- la *productividad*, número de empleados o mercado de especialización de las fábricas de software; o el número de incidencias que han surgido a lo largo del ciclo de vida de un proyecto, cuántas de esas incidencias han sido resueltas, en qué factorías se está llevando a cabo o medidas de calidad tales como *portabilidad*, *usabilidad*, *eficiencia*, etcétera. La cantidad de datos relevantes y susceptibles de representación visual de cada entidad¹ puede variar a lo largo del tiempo, es decir, puede producirse la adición de características para una o varias entidades. Además, puede suceder que en un futuro se deseen añadir más entidades para modelar un contexto más complejo. Por esta razón, es requisito indispensable que la interfaz de usuario se adapte automáticamente, en la medida de lo posible, a los cambios realizados en la lógica de dominio de la herramienta.

Para mostrar la información relativa a medidas e indicadores de calidad y productividad,

¹Se entiende como entidad a aquellas clases que modelan el contexto del DGS: compañía, factoría, proyecto y subproyecto.

se emplearán **metáforas de visualización**. Las metáforas de visualización están compuestas por un conjunto de elementos o modelos gráficos, y cada modelo tiene múltiples propiedades denominadas **dimensiones**. De este modo, el usuario podrá seleccionar qué modelo gráfico desea emplear en la metáfora para representar la información de una entidad determinada. Acto seguido, se establecerán asociaciones entre las distintas características de la entidad (nombre, distintas medidas o indicadores, etcétera) y las dimensiones del modelo gráfico, haciendo que la escenificación del mismo varíe en función de los valores que almacenan los atributos de la entidad. Así pues, se deberá proporcionar un conjunto de modelos gráficos capaces de escenificar los datos, medidas e indicadores definidos previamente.

Del mismo modo que en el caso de las entidades, los modelos gráficos también pueden experimentar variaciones si la empresa lo requiere, así como puede ser necesario que se añadan dimensiones. Por ello, y una vez más, es necesario que la interfaz gráfica de usuario vuelva a adaptarse a dichos cambios.

En este PFC se han elaborado tres modelos gráficos:

- **Modelo gráfico para factorías.** Consiste en un modelo que representa una estructura semejante a una fábrica industrial (Fig. 5.1). Por esta razón, un conjunto de instancias de este modelo escenifican la **metáfora de un polígono industrial**. El modelo está formado por una base que sostiene un edificio y una chimenea. Las dimensiones *mapeables* de este modelo serán la escala de la factoría (tamaño global), el color de la chimenea y la altura de la chimenea. Esta última dimensión también será reflejada en modo texto sobre el edificio. Inicialmente, este modelo ha sido diseñado para conocer el número de proyectos que se están desarrollando en una factoría, el número de empleados que trabajan en ella y el mercado de especialización de la misma (Tabla 5.1).

Característica de la factoría	Dimensión del modelo
Número de trabajadores (número)	Escala
Mercado de especialización (texto)	Color de la chimenea
Número de proyectos en desarrollo (número)	Altura de la chimenea y texto sobre el edificio

Tabla 5.1: Mapeo inicial de una factoría software con el modelo gráfico de factoría software

- **Modelo gráfico para proyectos.** Dada la inexistencia de un objeto real que se asemeje

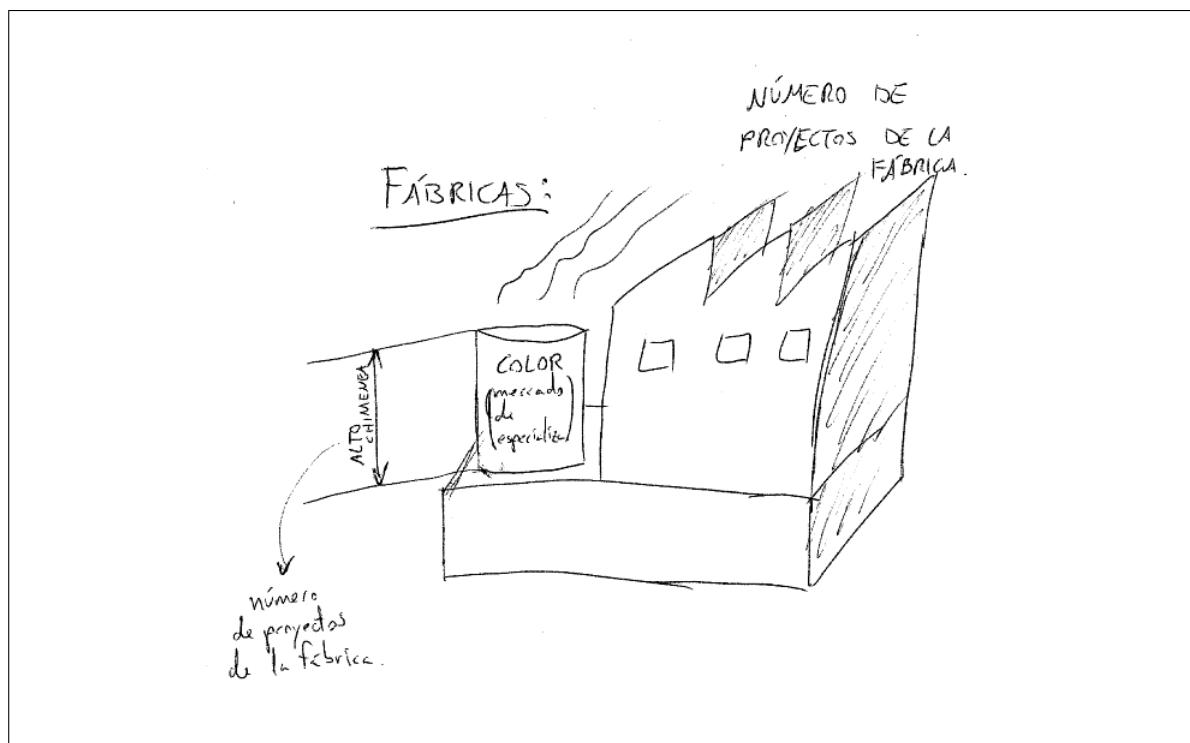


Figura 5.1: Prototipo a mano alzada para diseñar el modelo gráfico de una factoría software

al concepto de *proyecto*, este modelo se ha diseñado para satisfacer las necesidades de visualización que los expertos del sector han especificado. La estructura (Fig. 5.2) está formada por una esfera inferior cuyo tamaño y color son variables. Además, presenta un espacio reservado para mostrar una marca. La marca consiste en el icono de un *check* verde (✓) o una cruz roja (✗). Por otro lado, dispone de dos antenas en la parte superior con una pequeña esfera en cada extremo. Estas pequeñas esferas serán de color verde y rojo respectivamente, y presentarán un número entero en la parte superior. Por último, bajo la esfera inferior, se mostrará una cadena de texto. Inicialmente, este modelo ha sido diseñado para conocer el número de incidencias resueltas y sin resolver de un determinado proyecto, su tamaño en líneas de código, el tipo de mercado al que pertenece y si ha superado el proceso de auditoría (Tabla 5.2).

- **Modelo gráfico de torres.** Consiste un modelo genérico que representa una estructura semejante a la de un rascacielos o una torre (Fig. 5.3). Por esta razón, un conjunto de instancias de este modelo escenifican la **metáfora de una ciudad**. Este modelo se ha diseñado para ser configurado a medida y representar cualquier entidad de un modo genérico. Sus dimensiones son la anchura, profundidad, altura total, altura del relleno y color. Inicialmente, este modelo ha sido diseñado para conocer los resultados de

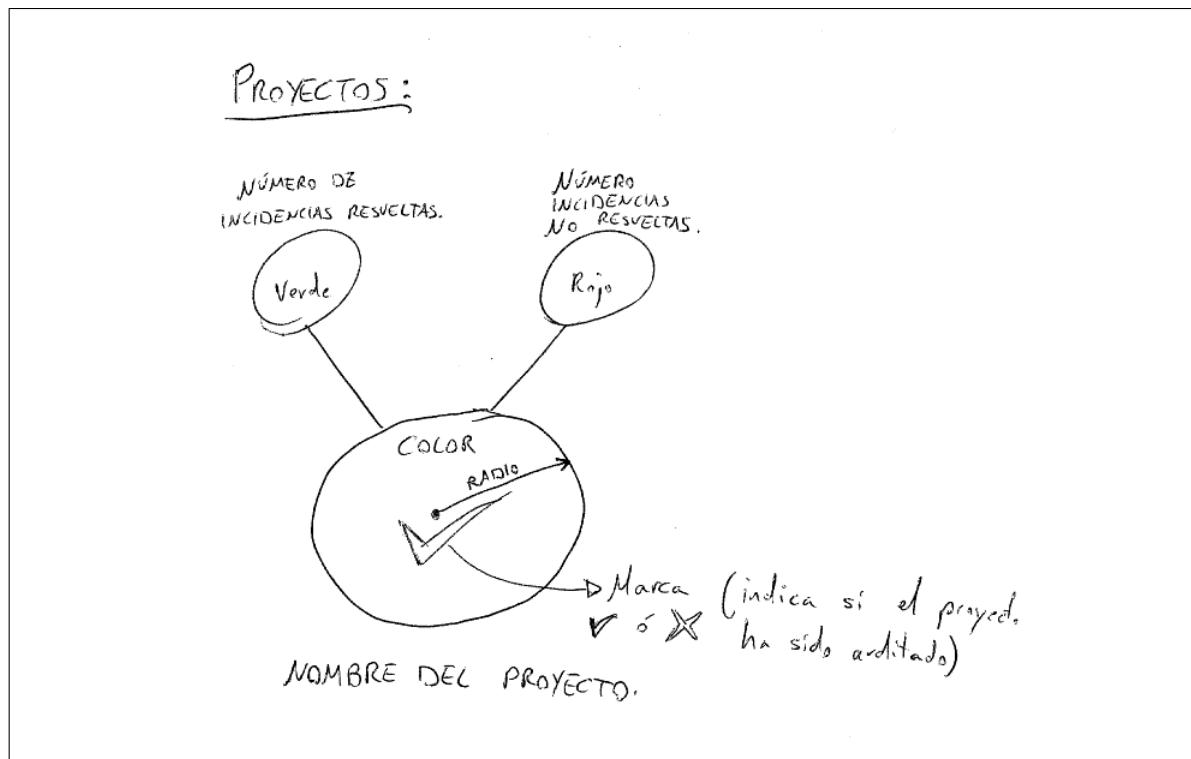


Figura 5.2: Prototipo a mano alzada para diseñar la el modelo gráfico de un producto software

Característica del proyecto	Dimensión del modelo
Líneas de código (número)	Tamaño de la esfera inferior
Proyecto auditado (lógico)	Marca de la esfera inferior
Tipo de mercado (texto)	Color de la esfera inferior
Incidencias resueltas (número)	Texto sobre la esfera superior verde
Incidencias no resueltas (número)	Texto sobre la esfera superior roja
Nombre del proyecto (texto)	Texto debajo del modelo gráfico

Tabla 5.2: Mapeo inicial de un proyecto software con el modelo gráfico de un proyecto software

diversas mediciones de un proyecto software, tales como el fichaje de horas realizado, el fichaje de horas estimado, si se encuentra dentro de plazo y su número de líneas de código y comentarios (Tabla 5.3).

Para maximizar la utilidad de la visualización y dado el alto grado de configuración del sistema, es necesario que la herramienta permita la creación de *profiles de visualización*. Un **perfil de visualización** definirá, como ya ha sido comentado, qué modelo gráfico será utilizado para representar una entidad determinada. Además, deberá contener información relativa a

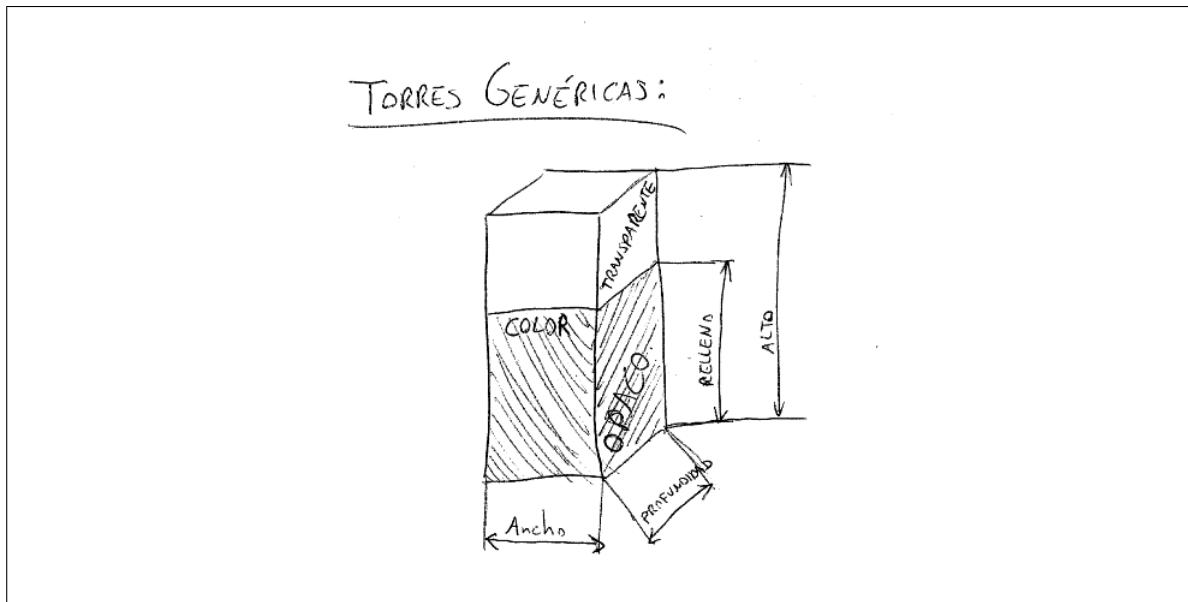


Figura 5.3: Prototipo a mano alzada para diseñar un modelo gráfico genérico mediante una torre

Característica del subproyecto	Dimensión del modelo
Líneas de código (número)	Anchura
Número de comentarios (lógico)	Profundidad
Fichaje de horas realizadas (texto)	Altura del relleno
Fichaje de horas estimadas (número)	Altura total
Retrasado (lógico)	Color

Tabla 5.3: Mapeo inicial de un subproyecto software con el modelo gráfico de una torre

las **asociaciones** que se desean establecer entre los atributos de la entidad y las dimensiones del modelo (Fig. 5.4). Gracias a esta característica, la herramienta de visualización no estará cerrada en cuanto a personalización se refiere, y el usuario será capaz de representar cualquier tipo de información del modo que crea conveniente.

Cabe destacar la posibilidad de crear diversos perfiles de visualización para una misma entidad. De este modo se logra disponer de múltiples vistas que muestren distintos tipos de información. Por ejemplo, puede ser interesante una vista en la que cada torre representa los atributos que conforman la funcionalidad de los proyectos, otra vista para representar los atributos de la portabilidad, y así sucesivamente con cada una de las características que se proponen en los estándares de calidad del producto software.

Para finalizar, se especifica la necesidad de navegar entre los distintos niveles organizacio-

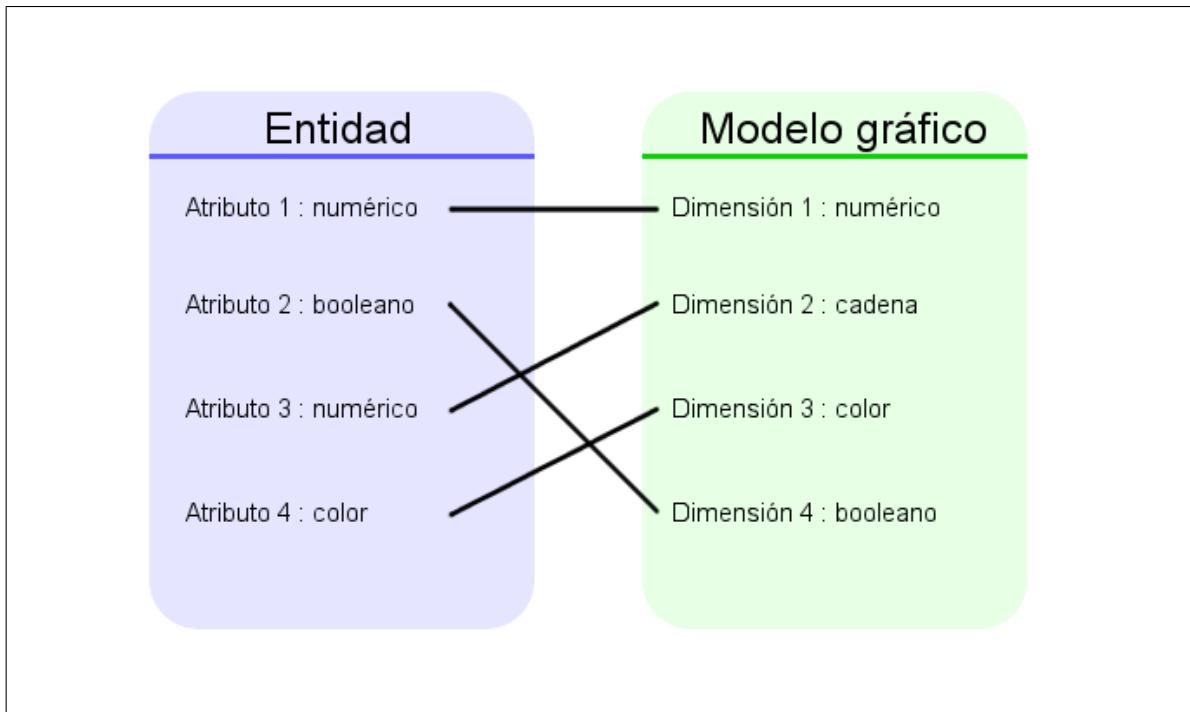


Figura 5.4: Esquema de las asociaciones o *mapeos* entre los atributos de una entidad y las dimensiones de un modelo gráfico

nales de un escenario de desarrollo global. En este caso, la empresa necesita obtener una vista global de sus factorías, a través de la cual podrá acceder a los proyectos que se desarrollan en cada una de ellas, y así poder realizar un seguimiento acerca de los mismos. Para satisfacer este requisito se propone una solución en la que el usuario podrá navegar entre las distintas metáforas de visualización seleccionando con el ratón los modelos gráficos que deseé (Fig. 5.5).

5.1.2 Identificación de requisitos

Para lograr obtener un sistema que satisfaga los requisitos descritos anteriormente, se propone la creación de dos componentes: *Desglosa Graphics Engine*, que se corresponde con un motor de gráficos 3D, y *Desglosa Web*, que se corresponde con una aplicación web. Esta decisión toma fuerza y es aceptada dado el gran potencial de generación de gráficos que aporta un componente específico de visualización y su capacidad de reutilización en otros proyectos de visualización.

Después de una serie importante de reuniones, se han identificado un conjunto de requisitos que serán expuestos en las dos siguientes subsecciones y que dan paso al modelado del diagrama de casos de uso 5.1.3.

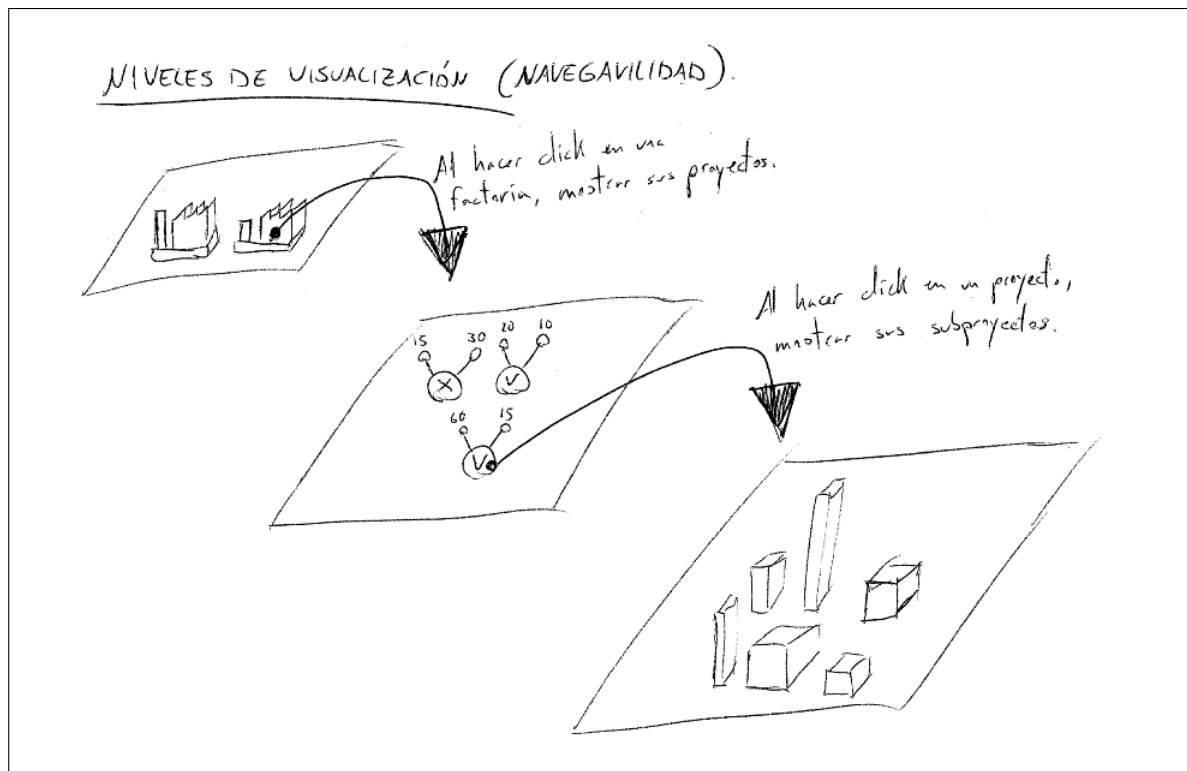


Figura 5.5: Prototipo a mano alzada para diseñar la navegación entre las distintas metáforas de visualización

5.1.2.1 Requisitos funcionales

Las funcionalidades que proporcione la herramienta al usuario dependerán de su rol. Así pues, en función del rol del usuario se tendrá acceso a un mayor o menor conjunto de funcionalidades destinadas a tareas de administración, gestión o análisis.

En la tabla 5.4 se especifican los requisitos funcionales detectados en la fase de inicio, una descripción en lenguaje natural de los mismos y qué usuarios pueden beneficiarse de cada una de las funcionalidades.

Funcionalidad	Descripción	Roles
Acceso al sistema	Control de acceso al sistema basado en grupos. Engloba las funcionalidades de autenticación y fin de sesión.	Cualquier usuario
Gestión de compañías	Engloba las funcionalidades de creación, edición, consulta y supresión de compañías.	Administrador
Gestión de factorías	Engloba las funcionalidades de creación, edición, consulta y supresión de factorías.	Administrador

Funcionalidad	Descripción	Roles
Gestión de proyectos	Engloba las funcionalidades de creación, edición, consulta y supresión de proyectos.	Administrador y jefe de proyecto
Gestión de subproyectos	Engloba las funcionalidades de creación, edición, consulta y supresión de subproyectos.	Administrador y jefe de proyecto
Gestión de perfiles de visualización	Engloba las funcionalidades de creación, consulta y supresión de perfiles de visualización, que configuran la información que se desea mostrar mediante gráficos 3D.	Administrador
Selección de perfiles	Permitirá que el usuario seleccione un perfil de visualización para configurar la información que desea analizar mediante la herramienta de visualización.	Administrador, jefe de proyecto y usuario autenticado
Geolocalización de factorías	Muestra un mapa del mundo con un conjunto de factorías software geolocalizadas.	Administrador, jefe de proyecto y usuario autenticado
Visualización de medidas e indicadores	Generación de gráficos 3D a partir de la información de las medidas e indicadores de las entidades del DGS.	Administrador, jefe de proyecto y usuario autenticado
Selección de objetos 3D	El usuario debe ser capaz de seleccionar los objetos 3D de una escena mediante el puntero del ratón.	Usuario de la herramienta de visualización
Navegabilidad entre niveles de abstracción	Capacidad de navegar, siguiendo un enfoque <i>top-down</i> , entre los distintos niveles de abstracción que representan el paradigma del DGS con la ayuda del puntero del ratón.	Usuario de la herramienta de visualización

Tabla 5.4: Requisitos funcionales del sistema

A modo de síntesis, en las tablas 5.5 y 5.6 se muestran los roles identificados que harán

uso de cada uno de los componentes que conforman el sistema. Nótese que el rol *Usuario* de ambos componentes no se corresponde con el mismo actor, ya que el usuario del motor gráfico engloba a los administradores, jefes de proyecto y usuarios autenticados, en general, de la aplicación web.

Rol del usuario	Descripción
Administrador	Usuarios experto del sistema encargado de gestionar las tareas de configuración críticas y no críticas del sistema y con capacidad de beneficiarse de las funcionalidades de visualización.
Jefe de proyecto	Usuario del sistema encargado de las tareas de configuración no críticas y con capacidad de beneficiarse de las funcionalidades de visualización.
Usuario	Usuario estándar con capacidad de beneficiarse de las funcionalidades de visualización.
Motor gráfico	Componente software que notifica una acción de selección.

Tabla 5.5: Roles identificados en la aplicación web

Rol del usuario	Descripción
Usuario	Usuario capaz de navegar por el entorno 3D e interactuar con los modelos gráficos escenificados en el mismo.
Aplicación contenadora	Componente que alberga al motor gráfico con capacidad de configurar y mostrar la escena que se requiera. Este componente podrá obtener una realimentación o <i>feedback</i> de las distintas acciones de selección de objetos.

Tabla 5.6: Roles identificados en el motor gráfico

5.1.2.2 Requisitos no funcionales

Las empresas colaboradoras en este proyecto imponen como requisito que el lenguaje de programación sea Java. Por lo tanto, ambos componentes estarán basados en tecnología **Java** y su ciclo de vida será gestionado de forma separada mediante **Maven**.

Por otro lado, para que la herramienta sea accesible desde cualquier parte del mundo, siempre y cuando se disponga de una conexión a Internet, será diseñada como una **aplicación web**. Así pues, y con el objetivo de romper con las diferencias idiomáticas, la aplicación web gozará de una característica de *internacionalización*. En este PFC se incluirán los idiomas de Inglés (Americano) y Español (España), siendo únicamente necesario traducir un conjunto de ficheros en texto plano para añadir cuantos idiomas se deseen. Esta característica es fundamental al ser una herramienta orientada al desarrollo global, ya que se podrá utilizar en distintos países.

En cuanto a la generación de gráficos 3D, será necesario encontrar soluciones que abstraigan la generación de los mismos de la arquitectura de la máquina y del sistema operativo.

Otras de las características de las que debe gozar el sistema son las siguientes:

- **Intuitivo.** Para permitir que el análisis de los datos sea una tarea rápida y precisa, el sistema de visualización debe permitir vistas intuitivas y de fácil comprensión.
- **Flexible.** Cabe la posibilidad de que las empresas que utilicen el sistema deseen mostrar información adicional a la seleccionada inicialmente. Por ello, es necesario que el sistema disponga de algún mecanismo que permita esta adición de conocimiento de un modo sencillo.
- **Personalizable.** Algunos sistemas de visualización son efectivos y eficientes pero tienen un diseño muy específico y resulta costoso adaptarlos para manejar nuevos datos. Por ello, el sistema desarrollado debe diseñarse de un modo que permita la adición de nuevos requisitos de un modo sencillo. Acto seguido, los jefes de proyecto podrán crear distintas vistas personalizadas en función de sus preferencias.
- **Extensible.** El sistema debe incorporar una arquitectura bien diseñada que permita, al equipo encargado del mantenimiento del software, la adición de nuevas metáforas de visualización de un modo rápido y fácil.

5.1.3 Modelo de Casos de Uso

A continuación se muestran los diagramas de casos de uso que se han modelado para los dos componentes que conformarán el sistema, a partir de los requisitos funcionales identificados en la sección 5.1.2.1.

La figura 5.6 muestra el modelo de casos de uso de la aplicación web. En este componente se observan cuatro actores, de los cuales tres (usuario, administrador y jefe de proyecto) representan personas físicas y uno (contenedor de gráficos) representa un subsistema software. En la tabla 5.5 se incluye una descripción acerca de cada actor.

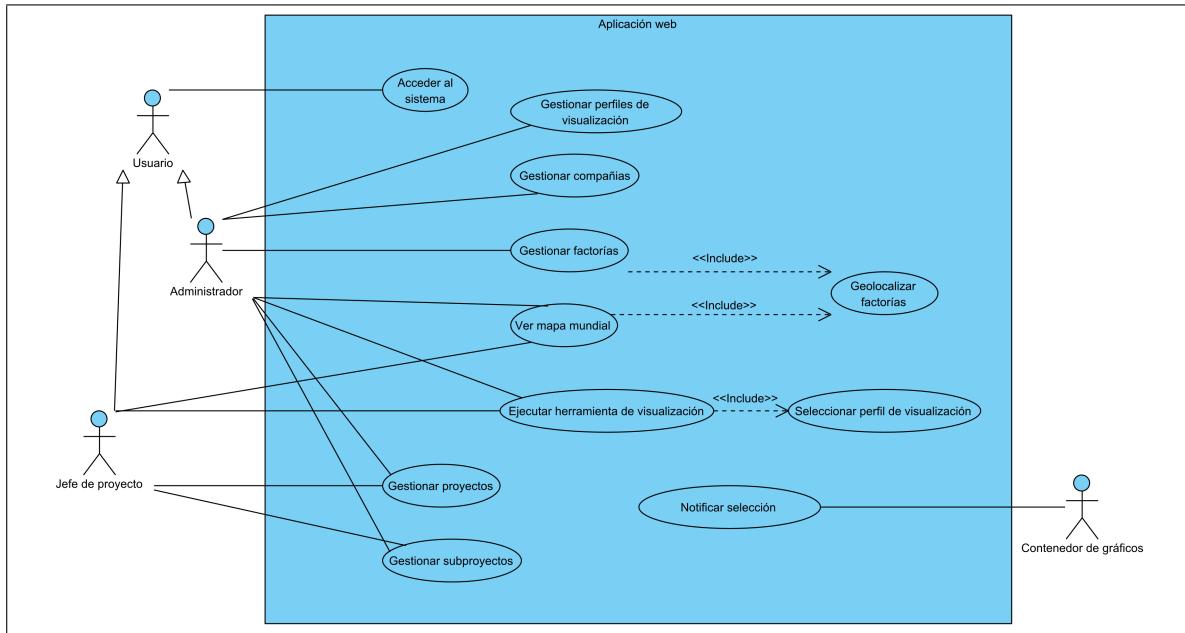


Figura 5.6: Modelo de casos de uso de la aplicación web

Por otro lado, la figura 5.7 presenta dos actores (Tabla 5.6): el usuario que interactúa con la escena, que es una persona física y se corresponde con el usuario al que la aplicación contenedora le haya concedido el privilegio de ejecutar el componente de visualización; y la aplicación contenedora que será la encargada de aportar los parámetros de configuración necesarios para configurar y visualizar la escena deseada.

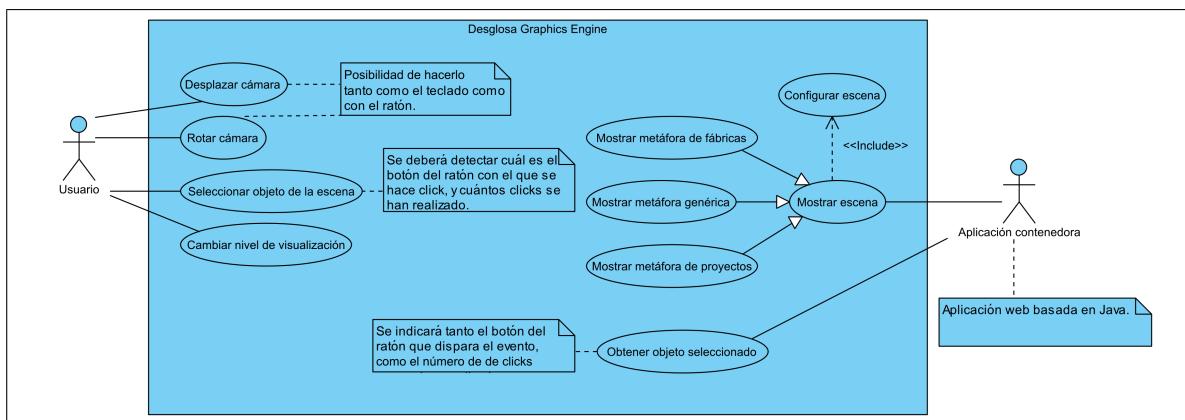


Figura 5.7: Modelo de casos de uso del motor gráfico

Para finalizar, se facilita la tabla 5.7 que muestra los casos de uso identificados, sus tareas

asociadas -en caso de existir-, la prioridad de realización y un código identificativo que será utilizado para denominar las funcionalidades de aquí en adelante. Cabe destacar que los códigos identificativos de formato *CdUI.x* se refieren a casos de uso del motor gráfico y los de *CdU2.x* se refieren a los de la aplicación web.

Requisito	Prior.	Descomposición de tareas	Id.
Desplazar cámara	1	-	CdU1.1
Rotar cámara	2	-	CdU1.2
Seleccionar objetos de la escena	6	-	CdU1.3
Cambiar nivel de visualización	5	-	CdU1.4
Mostrar escena	3	Mostrar metáfora de polígono industrial Mostrar escena con modelos de proyectos Mostrar metáfora de ciudad	CdU1.5
Configurar escena	4	-	CdU1.6
Obtener objeto seleccionado	7	-	CdU1.7
Acceder al sistema	11	Iniciar sesión Finalizar sesión Control de acceso Configuración de menús	CdU2.1
Gestionar perfiles	13	Crear perfiles Ver perfiles Borrar perfiles	CdU2.2
Gestionar compañías	15	Crear compañías Ver compañías Editar compañías Borrar compañías	CdU2.3
Gestionar factorías	16	Crear factorías Ver factorías Editar factorías Borrar factorías	CdU2.4

Requisito	Prior.	Descomposición de tareas	Id.
Gestionar proyectos	17	Crear proyectos Ver proyectos Editar proyectos Borrar proyectos	CdU2.5
Gestionar subproyectos	18	Crear subproyectos Ver subproyectos Editar subproyectos Borrar subproyectos	CdU2.6
Ver mapa mundial	10	-	CdU2.7
Geolocalizar factorías	12	-	CdU2.8
Ejecutar herramienta de visualización	8	-	CdU2.9
Seleccionar perfil de visualización	14	-	CdU2.10
Notificar selección	9	-	CdU2.11

Tabla 5.7: Denominación y priorización de los casos de uso

5.1.4 Glosario de términos

Se ha elaborado un glosario que define y aclara el concepto de algunos de los términos empleados a lo largo de la elaboración del PFC. Así pues, se eliminan posibles ambigüedades o malentendidos.

- **Compañía.** En este PFC se utiliza el término *compañía* para referirse a una empresa, organización o institución involucrada en el desarrollo o mantenimiento de productos software.
- **Entidad.** Se hace referencia a este término en numerosas ocasiones a lo largo del documento, especialmente cuando se aborda la parte de perfiles de visualización. El término *entidad* se refiere a las distintas unidades que se consideran en el contexto del

desarrollo global, tales como las compañías, las factorías, los proyectos y los subproyectos. Cuando se trata de hacer referencia a las propiedades, medidas o indicadores de una entidad, se suele utilizar el término *propiedad* o *atributo*.

- **Factoría.** En este PFC se utiliza el término *factoría software* o *fábrica software* para referirse a un lugar físico, tal como una oficina o edificio, en el que se ubican distintos equipos de trabajo dedicados a mantener o desarrollar software. Una factoría o fábrica software pertenece a una compañía. Por otro lado, una compañía puede disponer de varias fábricas o factorías software para llevar a cabo un mismo proyecto.
- **Metáfora de visualización.** Una metáfora de visualización es una analogía que subyace en la representación gráfica de una entidad o un concepto abstracto. El objetivo de las metáforas de visualización consiste en transferir propiedades del dominio de la representación gráfica a esa entidad o concepto abstracto [12].
- **Motor gráfico.** Se trata de un componente software desarrollado en este PFC cuya finalidad consiste en generar gráficos 3D. El motor gráfico dispone de distintos modelos gráficos y metáforas de visualización que serán utilizados para representar distintos tipos de información.
- **Subproyecto.** Este término se refiere a una de las partes de un proyecto, que puede ser una *fase*, una *funcionalidad* o un *módulo* del mismo.

5.1.5 Gestión del riesgo

Se ha realizado un conjunto de estudios para decidir de qué forma se desarrollará el motor de gráficos 3D. Dado que es requisito que este componente sea independiente de la arquitectura de la máquina y del sistema operativo, consistirá en un **motor gráfico basado en OpenGL** que deberá ser integrado en la aplicación web. Las alternativas de integración que se han contemplado se basan en el uso de un *canvas* de HTML5 (*HyperText Markup Language*), que haría posible el uso de la tecnología WebGL, o un alguna tecnología tipo *applet* o *Java Web Start*, que permitiría desarrollar el motor gráfico con alguna librería basada en Java que permita acceder a OpenGL.

La primera alternativa ha sido desechada ya que en la fecha en la que se aborda el análisis y diseño del motor gráfico, no existe una especificación estándar del API de WebGL (la primera

versión ha sido publicada el 10 de Febrero de 2011). Como consecuencia de ello, a la fecha, las versiones estables de los navegadores web tradicionales (Mozilla Firefox, Google Chrome, Internet Explorer, etcétera) aún no implementan dicha tecnología y la documentación para desarrolladores acerca de la misma no es abundante. Por esta razón, se opta por la posibilidad de utilizar una librería basada en Java e integrar el motor gráfico en la aplicación web mediante un applet o Java Web Start.

Del conjunto de librerías basadas en Java que se han contemplado para generar gráficos 3D mediante OpenGL, se ha seleccionado *JOGL (Java OpenGL)*. **JOGL** es un proyecto gratuito y de código libre, con licencia BSD (Berkeley Software Distribution) y goza de la potencia necesaria para un proyecto de estas características.

5.1.6 Plan de iteraciones

A partir de los estudios elaborados a lo largo de esta fase, es posible establecer un plan de iteraciones. El plan de iteraciones se ha elaborado como una serie de tablas (metáforas de fichas de papel), en la que cada tabla contiene la información relativa a una iteración: número de iteración, fase a la que pertenece, comentarios adicionales, objetivos, hitos a alcanzar y una pequeña gráfica que muestra el grado de participación en las disciplinas del flujo de trabajo fundamental en la iteración (Tablas 5.8 - 5.18).

A modo de síntesis, y para facilitar el análisis del plan de iteraciones se facilitan las figuras 5.8, 5.9 y 5.10. La figura 5.8 es una gráfica de líneas que muestra cuántas iteraciones conlleva cada una de las fases del ciclo de vida. Como se puede observar, las fases de inicio y transición son las más livianas (una sola iteración) y, por el contrario, la fase de construcción es la más laboriosa (seis iteraciones).

En las figuras 5.9 y 5.10 se muestra un diagrama de barras y de áreas, respectivamente. Ambos diagramas muestran el grado de participación de las iteraciones en cada disciplina del flujo de trabajo fundamental. Nótese cómo las disciplinas de requisitos, análisis y diseño toman fuerza al inicio del ciclo de vida y van cediendo el papel protagonista a las disciplinas de implementación y pruebas.

Número de iteración	Preliminar										
Fase a la que pertenece	Inicio										
Comentarios	Esta iteración se completa al finalizar el presente plan de iteraciones.										
Objetivos	Realizar un estudio y la captura de requisitos del sistema que se pretende desarrollar.										
Hitos	<ul style="list-style-type: none"> • Marco de investigación preliminar y captura de requisitos. • Identificación de requisitos funcionales y no funcionales. • Identificación de roles. • Modelo preliminar de casos de uso. • Glosario de términos. • Gestión del riesgo. • Plan de iteraciones 										
Grado de participación	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">Requisitos</td> <td style="width: 70%;"></td> </tr> <tr> <td>Análisis</td> <td></td> </tr> <tr> <td>Diseño</td> <td></td> </tr> <tr> <td>Implementación</td> <td>-</td> </tr> <tr> <td>Pruebas</td> <td>-</td> </tr> </table>	Requisitos		Análisis		Diseño		Implementación	-	Pruebas	-
Requisitos											
Análisis											
Diseño											
Implementación	-										
Pruebas	-										

Tabla 5.8: Ficha de la iteración preliminar

Número de iteración	1										
Fase a la que pertenece	Elaboración										
Comentarios	Iteración centrada en el funcionamiento básico y la arquitectura del motor gráfico. Dado el carácter investigador de esta iteración, no se desarrollará ningún caso de uso.										
Objetivos	Estudiar y entender cómo funciona el hilo de ejecución de un proyecto de Java OpenGL para desarrollar un pequeño proyecto consistente en una ventana que pinte un cubo en tres dimensiones y permita rotarlo.										
Hitos	<ul style="list-style-type: none"> • Diagramas de clases de análisis, comunicación, secuencia de análisis, clases de diseño y secuencia de diseño para el desarrollo de un pequeño proyecto que pinte un cubo en tres dimensiones con tecnología JOGL y la posibilidad rotarlo. • Preparación del entorno de desarrollo y creación del proyecto mediante Maven. • Implementación del mini-proyecto planteado en esta iteración. 										
Grado de participación	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">Requisitos</td> <td style="width: 70%;"></td> </tr> <tr> <td>Análisis</td> <td></td> </tr> <tr> <td>Diseño</td> <td></td> </tr> <tr> <td>Implementación</td> <td></td> </tr> <tr> <td>Pruebas</td> <td></td> </tr> </table>	Requisitos		Análisis		Diseño		Implementación		Pruebas	
Requisitos											
Análisis											
Diseño											
Implementación											
Pruebas											

Tabla 5.9: Ficha de la iteración 1

Número de iteración	2										
Fase a la que pertenece	Elaboración										
Comentarios	Esta iteración tomará como base el mini-proyecto elaborado en la iteración 1 y comenzará a añadirle las funcionalidades que debe satisfacer el motor gráfico.										
Objetivos	<ul style="list-style-type: none"> • Análisis y diseño de CdU1.1, CdU1.2 y CdU1.5. • Implementación de CdU1.1 y CdU1.2. • Implementación de un foco de luz para iluminar la escena. 										
Hitos	<ul style="list-style-type: none"> • Diagramas de clases de análisis, comunicación, secuencia de análisis, clases de diseño y secuencia de diseño de CdU1.1, CdU1.2 y CdU1.5. • Modelo arquitectónico del componente (motor gráfico). 										
Grado de participación	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 40%;">Requisitos</td> <td style="width: 60%;"></td> </tr> <tr> <td>Análisis</td> <td></td> </tr> <tr> <td>Diseño</td> <td></td> </tr> <tr> <td>Implementación</td> <td></td> </tr> <tr> <td>Pruebas</td> <td>-</td> </tr> </table>	Requisitos		Análisis		Diseño		Implementación		Pruebas	-
Requisitos											
Análisis											
Diseño											
Implementación											
Pruebas	-										

Tabla 5.10: Ficha de la iteración 2

Número de iteración	3
Fase a la que pertenece	Elaboración
Comentarios	Esta iteración se centra en la aplicación web, mientras que las iteraciones 1 y 2 se centran en el motor gráfico. Por tanto, la iteración 3 se puede realizar de modo paralelo con sus dos predecesoras si se dispone de recursos.
Objetivos	Diseñar el modelo datos e investigar acerca de las tecnologías que se van a emplear en el diseño y desarrollo web.
Hitos	<ul style="list-style-type: none"> • Modelo de datos. • Preparación del entorno de desarrollo y creación del proyecto mediante Maven.
Grado de participación	<p>Requisitos </p> <p>Análisis </p> <p>Diseño </p> <p>Implementación </p> <p>Pruebas -</p>

Tabla 5.11: Ficha de la iteración 3

Número de iteración	4										
Fase a la que pertenece	Construcción										
Comentarios	Dada la gran dificultad para implementar los casos de prueba que confirmen que la cámara se desplaza correctamente en el entorno virtual, éstas serán llevadas a cabo y validadas por un usuario experto.										
Objetivos	<ul style="list-style-type: none"> • Análisis y diseño de CdU1.6. • Implementación de CdU1.5 y CdU1.6. • Obtener una arquitectura extensible que permita que la tarea de mantenimiento para añadir nuevas metáforas de visualización sea una tarea fácil. • Probar los CdU1.1 y 1.2. 										
Hitos	<ul style="list-style-type: none"> • Diagramas de clases de análisis, comunicación, secuencia de análisis y clases de diseño de CdU1.6. • Modelo arquitectónico del componente (motor gráfico) mejorado. • Informe de verificación y validación de pruebas de los CdU1.1 y CdU1.2. 										
Grado de participación	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">Requisitos</td> <td style="width: 70%;"></td> </tr> <tr> <td>Análisis</td> <td></td> </tr> <tr> <td>Diseño</td> <td></td> </tr> <tr> <td>Implementación</td> <td></td> </tr> <tr> <td>Pruebas</td> <td></td> </tr> </table>	Requisitos		Análisis		Diseño		Implementación		Pruebas	
Requisitos											
Análisis											
Diseño											
Implementación											
Pruebas											

Tabla 5.12: Ficha de la iteración 4

Número de iteración	5										
Fase a la que pertenece	Construcción										
Comentarios	Dada la dificultad para implementar los casos de prueba que confirmen que los gráficos generados mediante los CdU1.5 y 1.6 se visualizan correctamente, éstas serán llevadas a cabo y validadas por un usuario experto.										
Objetivos	<ul style="list-style-type: none"> • Análisis, diseño e implementación de los CdU1.3, CdU1.4 y CdU1.7. • Análisis y diseño de los CdU2.9, CdU2.11 y CdU2.7. • Probar la correcta generación de gráficos de los CdU1.5 y 1.6. 										
Hitos	<ul style="list-style-type: none"> • Diagramas de clases de análisis, comunicación, secuencia de análisis, clases de diseño y secuencia de diseño de los CdU1.3, CdU1.4, CdU1.7, CdU2.9, CdU2.11 y CdU2.7. • Prototipo de la interfaz gráfica de usuario de la herramienta de visualización en la aplicación web. • Informe de verificación y validación de la correcta generación de gráficos de los CdU1.5 y CdU1.6. 										
Grado de participación	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">Requisitos</td> <td style="width: 70%;"></td> </tr> <tr> <td>Análisis</td> <td></td> </tr> <tr> <td>Diseño</td> <td></td> </tr> <tr> <td>Implementación</td> <td></td> </tr> <tr> <td>Pruebas</td> <td></td> </tr> </table>	Requisitos		Análisis		Diseño		Implementación		Pruebas	
Requisitos											
Análisis											
Diseño											
Implementación											
Pruebas											

Tabla 5.13: Ficha de la iteración 5

Número de iteración	6										
Fase a la que pertenece	Construcción										
Comentarios	Dada la dificultad para implementar los casos de prueba que confirmen la correcta integración del motor gráfico en una página web, así como el mecanismos de selección de objetos 3D, las pruebas de CdU1.3, CdU1.4 y CdU1.7 serán llevadas a cabo y validadas por un usuario experto.										
Objetivos	<ul style="list-style-type: none"> • Análisis y diseño de CdU2.1 • Implementación de los CdU2.9, CdU2.11 y CdU2.7, lo cual supone la integración del motor gráfico en la aplicación web. • Probar la integración del motor gráfico en la aplicación web y los mecanismos de selección de objetos 3D, correspondientes a los CdU1.3, CdU1.4 y CdU1.7. 										
Hitos	<ul style="list-style-type: none"> • Diagramas de clases de análisis, comunicación, secuencia de análisis y clases de diseño de CdU2.1. • Informe de verificación y validación de la correcta integración del motor gráfico en la aplicación web y del mecanismo de selección y notificación de objetos 3D. 										
Grado de participación	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 40%;">Requisitos</td> <td style="width: 10%;"></td> </tr> <tr> <td>Análisis</td> <td></td> </tr> <tr> <td>Diseño</td> <td></td> </tr> <tr> <td>Implementación</td> <td></td> </tr> <tr> <td>Pruebas</td> <td></td> </tr> </table>	Requisitos		Análisis		Diseño		Implementación		Pruebas	
Requisitos											
Análisis											
Diseño											
Implementación											
Pruebas											

Tabla 5.14: Ficha de la iteración 6

Número de iteración	7										
Fase a la que pertenece	Construcción										
Comentarios											
Objetivos	<ul style="list-style-type: none"> • Análisis y diseño de CdU2.2 y CdU2.10. • Implementación de CdU2.1. • Configuración del entorno de pruebas con Maven. • Diseño e implementación de pruebas de CdU2.1, CdU2.9, CdU2.11 y CdU2.7. 										
Hitos	<ul style="list-style-type: none"> • Diagramas de clases de análisis, comunicación, secuencia de análisis y clases de diseño de CdU2.2 y CdU2.10. • Modelo arquitectónico de la aplicación web. • Prototipo de la interfaz gráfica de usuario de la herramienta de configuración de perfiles. • Entorno de desarrollo configurado tanto para la ejecución de pruebas unitarias y funcionales como para la generación de informes. • Informe de pruebas unitarias y funcionales de CdU2.1, CdU2.9, CdU2.11 y CdU2.7. 										
Grado de participación	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">Requisitos</td> <td style="width: 70%;"></td> </tr> <tr> <td>Análisis</td> <td></td> </tr> <tr> <td>Diseño</td> <td></td> </tr> <tr> <td>Implementación</td> <td></td> </tr> <tr> <td>Pruebas</td> <td></td> </tr> </table>	Requisitos		Análisis		Diseño		Implementación		Pruebas	
Requisitos											
Análisis											
Diseño											
Implementación											
Pruebas											

Tabla 5.15: Ficha de la iteración 7

Número de iteración	8										
Fase a la que pertenece	Construcción										
Comentarios											
Objetivos	<ul style="list-style-type: none"> • Análisis y diseño de CdU2.3, CdU2.4, CdU2.5 y CdU2.6. • Implementación y pruebas de CdU2.2 y CdU2.10. 										
Hitos	<ul style="list-style-type: none"> • Diagramas de clases de análisis de CdU2.3, CdU2.4, CdU2.5 y CdU2.6. • Diagramas de comunicación de CdU2.3, CdU2.4, CdU2.5 y CdU2.6. • Diagramas de secuencia de análisis de CdU2.3, CdU2.4, CdU2.5 y CdU2.6. • Diagramas de clases de diseño de CdU2.3, CdU2.4, CdU2.5 y CdU2.6. • Prototipo de la interfaz gráfica de usuario de las herramienta de gestión de compañías, factorías, proyectos y subproyectos. • Informe de pruebas unitarias y funcionales de CdU2.2 y CdU2.10. 										
Grado de participación	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 40%;">Requisitos</td> <td style="width: 10%;"></td> </tr> <tr> <td>Análisis</td> <td></td> </tr> <tr> <td>Diseño</td> <td></td> </tr> <tr> <td>Implementación</td> <td></td> </tr> <tr> <td>Pruebas</td> <td></td> </tr> </table>	Requisitos		Análisis		Diseño		Implementación		Pruebas	
Requisitos											
Análisis											
Diseño											
Implementación											
Pruebas											

Tabla 5.16: Ficha de la iteración 8

Número de iteración	9										
Fase a la que pertenece	Construcción										
Comentarios											
Objetivos	<ul style="list-style-type: none"> • Implementación y pruebas de CdU2.3, CdU2.4, CdU2.5 y CdU2.6. 										
Hitos	<ul style="list-style-type: none"> • Informe de pruebas unitarias y funcionales de CdU2.3, CdU2.4, CdU2.5 y CdU2.6. 										
Grado de participación	<table> <tr> <td>Requisitos</td> <td>-</td> </tr> <tr> <td>Análisis</td> <td></td> </tr> <tr> <td>Diseño</td> <td></td> </tr> <tr> <td>Implementación</td> <td></td> </tr> <tr> <td>Pruebas</td> <td></td> </tr> </table>	Requisitos	-	Análisis		Diseño		Implementación		Pruebas	
Requisitos	-										
Análisis											
Diseño											
Implementación											
Pruebas											

Tabla 5.17: Ficha de la iteración 9

Número de iteración	10										
Fase a la que pertenece	Transición										
Comentarios	<p>La fase de transición estará formada únicamente por esta iteración. En esta fase, que produce como resultado una versión entregable del software, se corregirán los fallos que hayan podido suceder al ejecutar las distintas pruebas unitarias y funcionales.</p>										
Objetivos	<ul style="list-style-type: none"> • Corregir cualquier <i>bug</i> que haya podido detectarse en las disciplinas de pruebas. • Elaborar el manual de usuario y la documentación asociada al proyecto. • Obtener la primera versión entregable del software. 										
Hitos	<ul style="list-style-type: none"> • Manual de instalación y despliegue. • Manual de usuario. • Versión entregable del software. 										
Grado de participación	<table> <tr> <td>Requisitos</td> <td>-</td> </tr> <tr> <td>Análisis</td> <td>-</td> </tr> <tr> <td>Diseño</td> <td>-</td> </tr> <tr> <td>Implementación</td> <td></td> </tr> <tr> <td>Pruebas</td> <td></td> </tr> </table>	Requisitos	-	Análisis	-	Diseño	-	Implementación		Pruebas	
Requisitos	-										
Análisis	-										
Diseño	-										
Implementación											
Pruebas											

Tabla 5.18: Ficha de la iteración 10

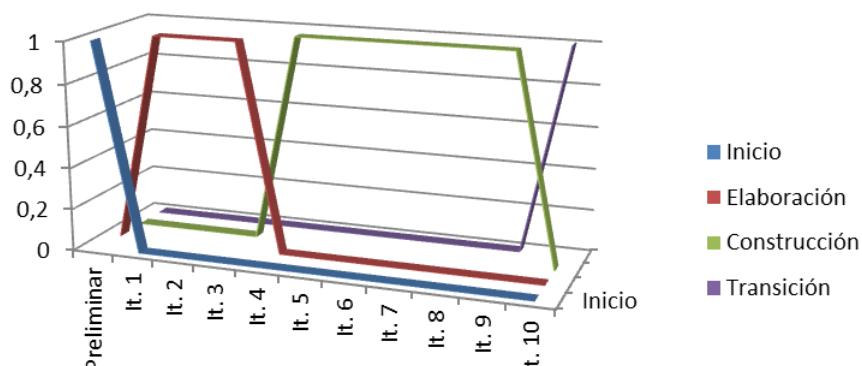


Figura 5.8: Gráfica de líneas que representa la duración de cada fase en iteraciones

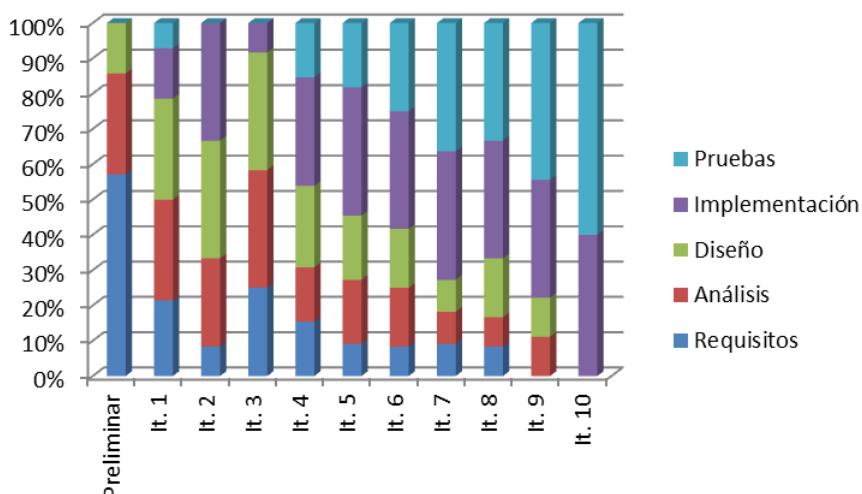


Figura 5.9: Gráfica de barras que representa el grado de participación de las iteraciones en cada disciplina

5.2 Fase de elaboración

La fase de elaboración consta de tres iteraciones centradas fundamentalmente en las disciplinas de análisis y diseño de los dos componentes que conformarán el sistema final. No obstante, también se abordan cuestiones de implementación, aunque la mayor parte será tratada en la fase de construcción.

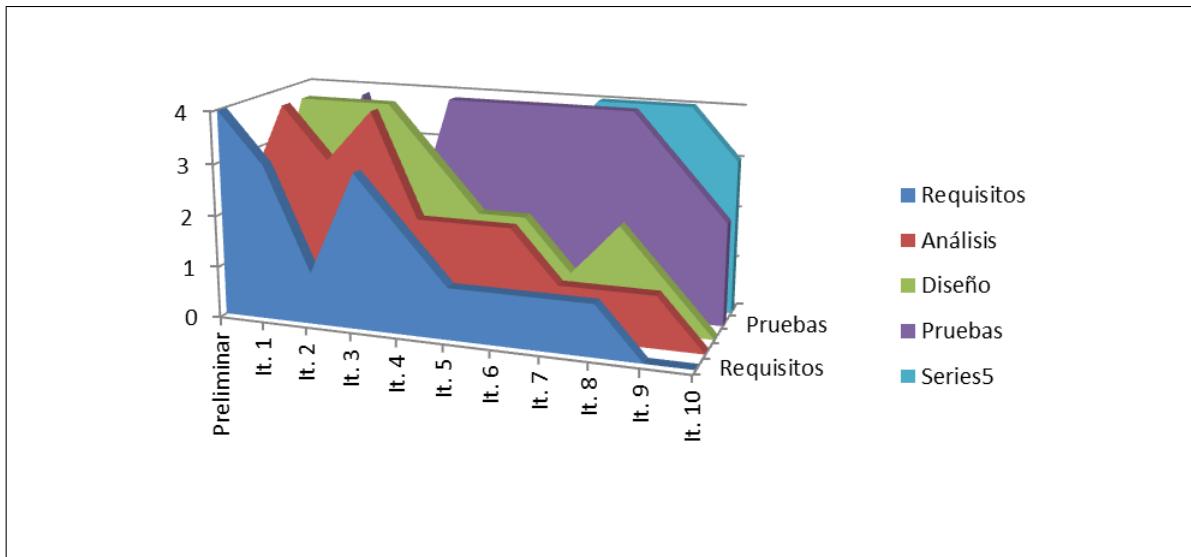


Figura 5.10: Gráfica de áreas que representa el grado de participación de las iteraciones en cada disciplina

5.2.1 Iteración 1

En esta iteración (Tabla 5.9) se aborda:

- Análisis y diseño de un pequeño proyecto utilizando la tecnología JOGL.
- Preparación del entorno de desarrollo y creación del proyecto mediante Maven.
- Implementación de un mini-proyecto de funcionalidad reducida empleando JOGL.

5.2.1.1 Disciplina de requisitos

Se deberá elaborar un pequeña aplicación de funcionalidad reducida utilizando la librería Java OpenGL. Este mini-proyecto, que servirá como base para todo el desarrollo del motor gráfico, deberá mostrar un poliedro de seis caras en tres dimensiones y permitir que el usuario lo rote para examinarlo desde cualquier punto de vista.

5.2.1.2 Disciplinas de análisis

Para poder abordar el análisis de un proyecto de estas características, se ha comenzado por estudiar la librería de gráficos OpenGL y su implementación en Java: JOGL. Ha sido necesario estudiar ambas librerías ya que Java OpenGL (JOGL) es un *binding* que da acceso a OpenGL pero tiene ciertas limitaciones y particularidades. La principal característica es la forma en

la que JOGL implementa el hilo de ejecución sobre el cual se realizan las operaciones de *pintado* de la escena.

En la figura 5.11 se muestra el diagrama de casos de uso que corresponde en este punto del ciclo de vida del motor gráfico. Como se puede observar, se trata de un escenario muy sencillo en el que el hilo de ejecución de JOGL se encarga de pintar un cubo y el usuario tiene la posibilidad de rotarlo.

Dado el nivel de abstracción en el que se encuentra el proyecto, se ha modelado el hilo de ejecución de JOGL como un actor del sistema. Este *pseudo-actor* desaparecerá en futuros diagramas de casos de uso y, de hecho, no se ha considerado en el diagrama general de casos de uso del motor gráfico (Fig. 5.7) que se obtuvo en la fase de inicio (Sección 5.1.3). Su representación en este nivel tiene como único objetivo modelar un productor de información (el usuario que realiza acciones para desencadenar el cálculo de determinados parámetros de rotación del cubo) y un consumidor (el hilo de ejecución de JOGL que pinta el cubo a partir de los parámetros calculados).

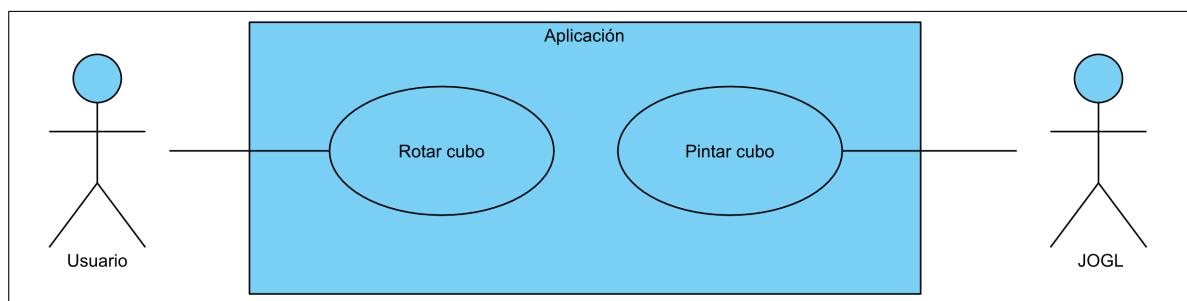


Figura 5.11: Diagrama de casos de uso para un mini-proyecto de funcionalidad reducida, basado en JOGL

El siguiente paso consiste en modelar el diagrama de clases de análisis que permitirá comprender el funcionamiento del sistema con un poco más de detalle (Fig. 5.12). En él se observa que el usuario interactúa con el contenedor de gráficos, cuyas características serán definidas en un futuro, que a su vez se encarga de iniciar el hilo de ejecución de JOGL. Este hilo es el responsable de pintar los cubos y poner a la escucha un *listener* de eventos de teclado que permitirá que el usuario rote los poliedros. Esta misma explicación se proporciona gráficamente en el diagrama de comunicación asociado (Fig. 5.13), donde se indica el paso de los distintos mensajes y señales.

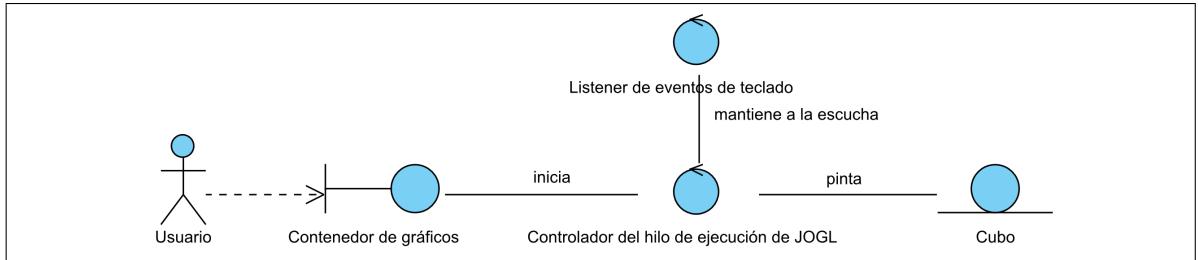


Figura 5.12: Diagrama de clases de análisis para un mini-proyecto de funcionalidad reducida, basado en JOGL

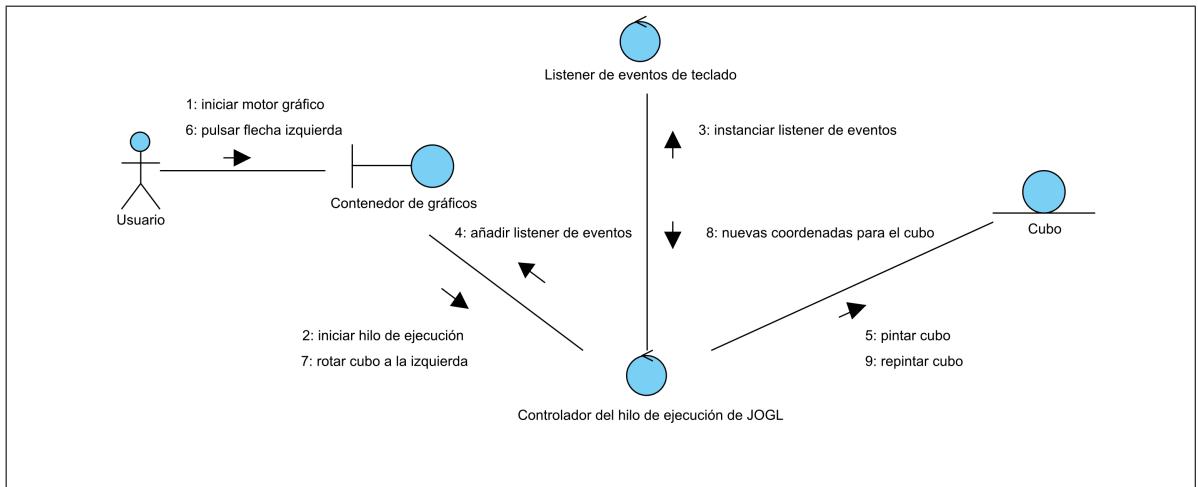


Figura 5.13: Diagrama de comunicación para un mini-proyecto de funcionalidad reducida, basado en JOGL

5.2.1.3 Disciplinas de diseño

Como fruto del estudio de la librería JOGL y de los diagramas de análisis elaborados en la disciplina de análisis de esta misma iteración, se obtiene el primer diagrama de clases de diseño que servirá como base para todo el motor gráfico (Fig. 5.14). En este diagrama, que hasta el momento sólo goza de las capas de presentación y dominio, es importante destacar los paquetes en los que están agrupados sus diferentes clases:

- **javax.media.opengl.** Este espacio de nombres está contenido en la librería de JOGL y nos proporciona la clase GLCanvas, que hereda de la clase Canvas de AWT (Abstract Window Toolkit) de Java, y la interfaz GLEventListener, que declara los métodos que se deberán implementar para poder pintar gráficos en pantalla.
- **com.sun.opengl.util.** Este espacio de nombres, también contenido en la librería de JOGL, proporciona la clase Animator que será la encargada de crear el hilo de ejecución de JOGL.

- **presentación.** Paquete propio que contendrá una clase de tipo *Window* que se encargará de configurar un *GLCanvas* y un *Animator*.
- **dominio.** Paquete propio con dos subpaquetes para una mejor organización de las clases del proyecto: **control**, que contendrá distintas clases de control, y **conocimiento**, que proporcionará las distintas clases de conocimiento.

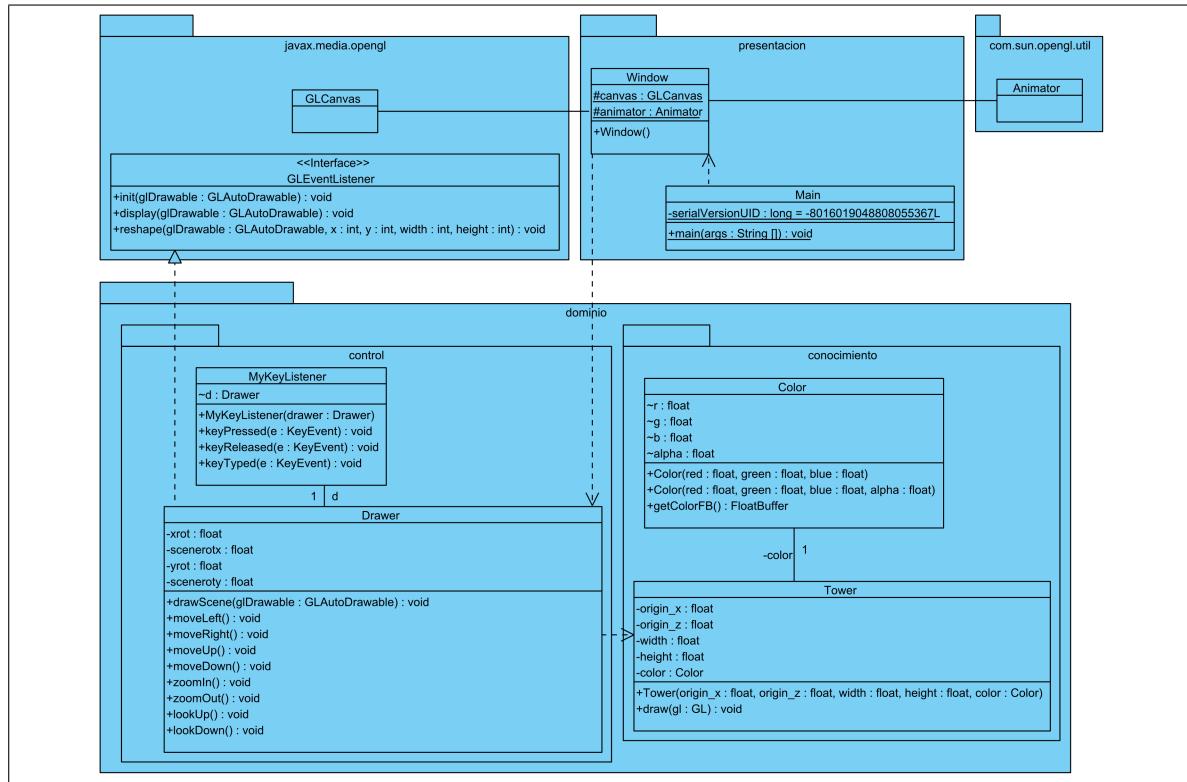


Figura 5.14: Diagrama de clases de diseño para un mini-proyecto de funcionalidad reducida, basado en JOGL

El diagrama de clases de diseño describe una vista estática del sistema, pero para comprender el funcionamiento del mismo es necesario especificar una vista dinámica que muestre cómo se crean los distintos objetos y cómo interactúan entre ellos. Para una mejor comprensión, esta vista se ha dividido en dos diagramas de secuencia: inicialización y ejecución.

En el diagrama de secuencia de inicialización (Fig. 5.15) se observa qué llamadas se producen entre los distintos objetos hasta que el hilo de ejecución de JOGL entra en el bucle que le permite repintar la escena constantemente.

En el diagrama de secuencia de ejecución (Fig. 5.16) se asume la parte de inicialización y se centra en la interacción entre objetos que se produce para pintar la escena. Ello conlleva omitir la parte que se corresponde con la figura 5.15 y el actor que inicia la acción.

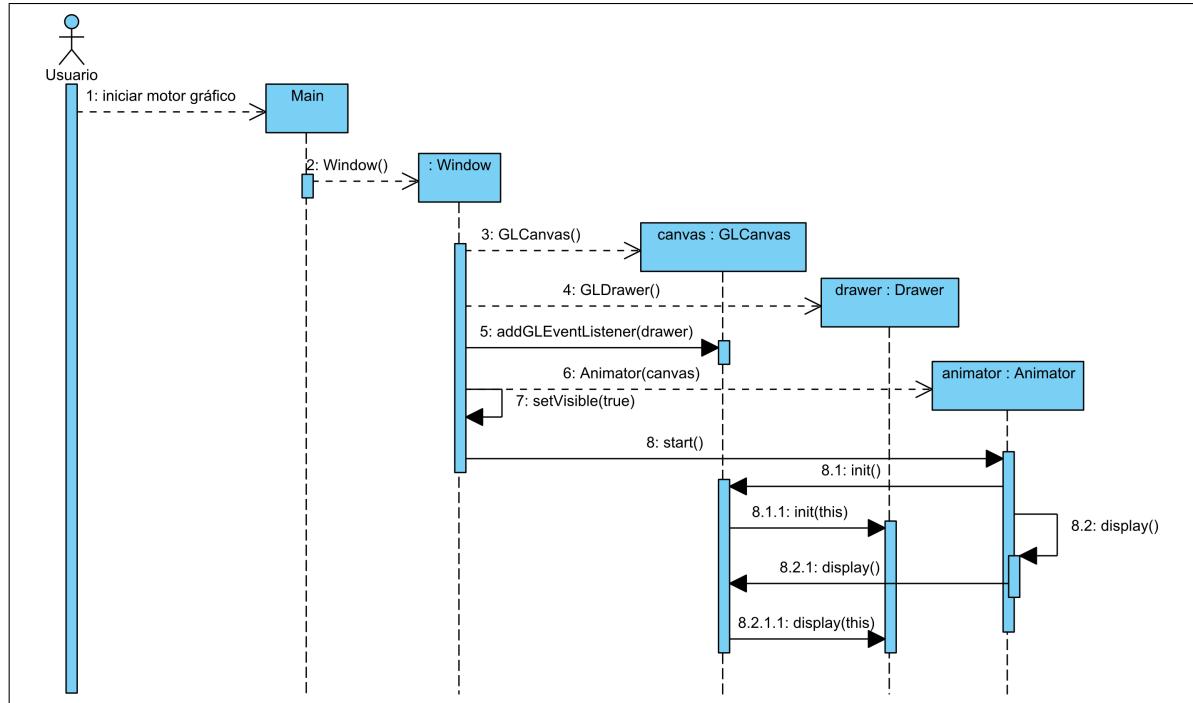


Figura 5.15: Diagrama de secuencia de la inicialización del hilo de JOGL

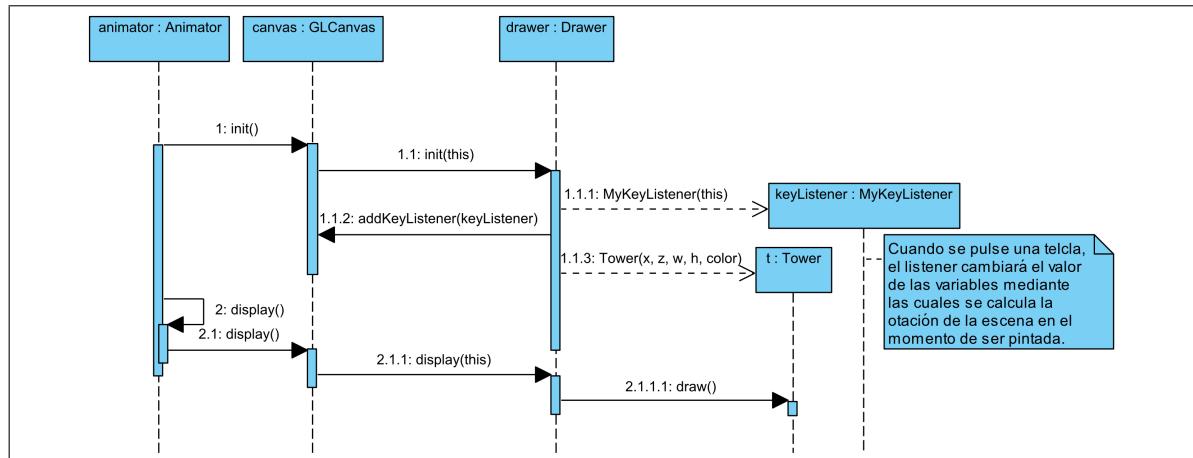


Figura 5.16: Diagrama de secuencia de ejecución del hilo de JOGL

5.2.1.4 Disciplinas de implementación

La disciplina de implementación del mini-proyecto que se aborda en esta iteración comienza con la configuración del entorno y la creación del proyecto.

En esta primera parte se ha instalado y configurado Eclipse IDE, Maven y el plugin m2eclipse en sus versiones especificadas en la sección 4.4. A continuación se ha creado un proyecto Maven de tipo **maven-archetype-quickstart** y se han añadido las dependencias de JOGL necesarias (Listado 5.1). Puesto que estas dependencias no se encuentran alojadas en los repositorios generales de Maven, es necesario añadirlas manualmente al repositorio local. Para

evitar conflictos de versiones, las dependencias que se han utilizado en este PFC se proporcionan en el DVD adjunto bajo el directorio *desglosa\builds\jogamp\deployment\webstart\archive*, junto con un pequeño fichero de texto que detalla cómo realizar la operación.

```

1 <dependencies>
2 ...
3   <dependency>
4     <groupId>com.jogamp.jogl</groupId>
5     <artifactId>jogl.all</artifactId>
6     <version>2.0-b23-20110303</version>
7     <scope>compile</scope>
8   </dependency>
9   <dependency>
10    <groupId>com.jogamp.gluegen</groupId>
11    <artifactId>gluegen-rt</artifactId>
12    <version>2.0-b23-20110303</version>
13    <scope>compile</scope>
14  </dependency>
15  <dependency>
16    <groupId>com.jogamp.newt</groupId>
17    <artifactId>newt.all</artifactId>
18    <version>2.0-b23-20110303</version>
19    <scope>compile</scope>
20  </dependency>
21  <dependency>
22    <groupId>com.jogamp.nativewindow</groupId>
23    <artifactId>nativewindow.all</artifactId>
24    <version>2.0-b23-20110303</version>
25    <scope>compile</scope>
26  </dependency>
27 ...
28 </dependencies>
```

Listado 5.1: Configuración de pom.xml para añadir JOGL como dependencia del proyecto

A continuación se implementan las clases diseñadas en esta misma iteración. El resultado es una pequeña ventana que pinta, de un modo muy básico y sin apenas calidad gráfica, una torre de color rojo y una torre de color verde sobre un suelo gris (Fig. 5.17).

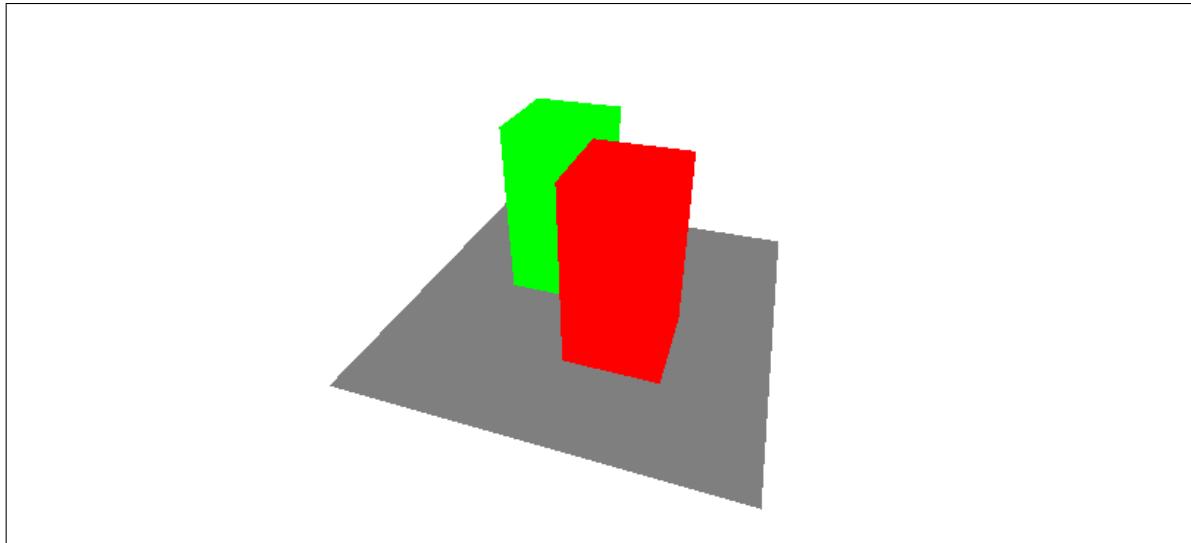


Figura 5.17: Resultado de la disciplina de implementación de la iteración 1

5.2.2 Iteración 2

En esta iteración (Tabla 5.9) se aborda:

- Análisis, diseño e implementación de la funcionalidad que permite que el usuario se desplace a través de la escena.
- Diseño arquitectónico de una estructura que permita generar nuevas escenas y vistas en función de la metáfora de visualización que quiera mostrarse.
- Implementación de un foco de luz que otorgue más realismo a la escena mostrada.

5.2.2.1 Disciplina de requisitos

Inicialmente, el motor gráfico dispondrá de un conjunto de modelos gráficos que serán emplazados en un entorno virtual tratando de representar la metáfora de un polígono industrial o de una ciudad. Por razones de extensibilidad, la arquitectura del proyecto deberá permitir que el equipo encargado del mantenimiento de la aplicación pueda añadir nuevos modelos gráficos con el fin de generar nuevas metáforas.

El aspecto visual de un modelo gráfico estará condicionado por los valores que toman sus dimensiones. Se entiende como **dimensión** de un modelo a aquel atributo cuyo valor afecta directamente en su representación visual. Las dimensiones se emplearán a la hora de crear los perfiles de visualización para poder asociarlos con los atributos de las entidades de

dominio del DGS. Así pues, se debe encontrar un mecanismo que permita diferenciar qué atributos de los elementos de una metáfora actúan como dimensiones y cuáles no.

Por otro lado, el motor gráfico no debe tener ningún tipo de dependencia con la herramienta contenedora. Esta característica implica que se establezca algún mecanismo de comunicación con el motor gráfico para que éste genere los gráficos correspondientes.

5.2.2.2 Disciplina de análisis

El usuario debe ser capaz de manipular la cámara a través de la cual visualiza la escena. Mediante un diagrama de casos de uso (Fig. 5.18) es posible especificar de un modo general qué movimientos y operaciones se encuentran disponibles: desplazamiento y rotación (en distintas direcciones y ejes de coordenadas). Para concretar los distintos tipos de desplazamiento y rotación se adjunta la descripción de ambos casos de uso en las tablas 5.19 y 5.20.

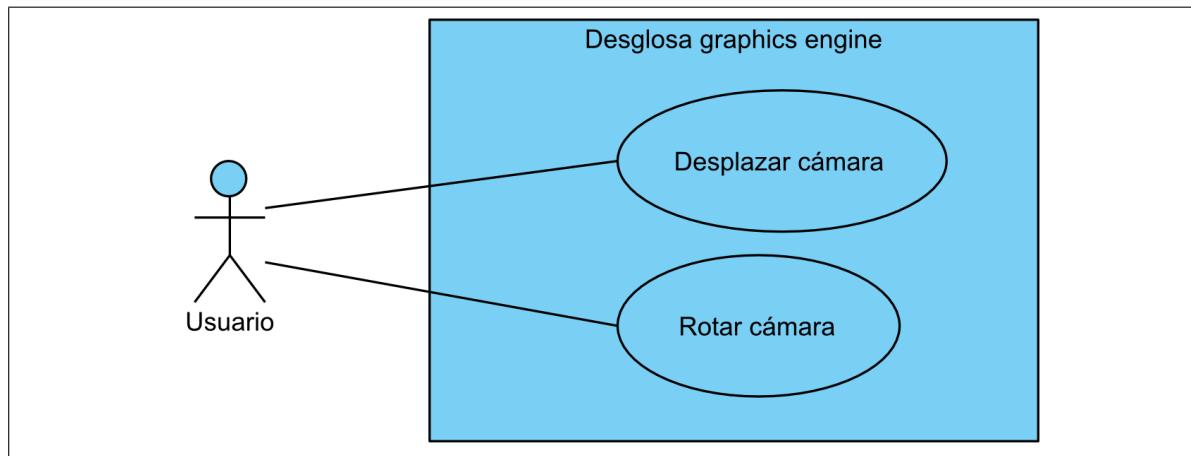


Figura 5.18: Diagrama de casos de uso para CdU1.1 y CdU1.2

Para tratar la cámara como un objeto que se puede situar en cualquier lugar de la escena y enfocar a cualquier punto de la misma, se propone una solución basada en vectores. Esta solución requiere un conjunto de vectores en un espacio de coordenadas (x , y , z) de las siguientes características:

- Un vector que indique la posición de la cámara en la escena.
- Un vector director unitario que indique la dirección de enfoque de la cámara.
- Tres vectores directores unitarios que indiquen la orientación de la cámara: arriba, derecha y frente (Fig. 5.19).

Identificador: CdU1.1
Nombre: Desplazar (Cámara)
Descripción: Funcionalidad para que un usuario pueda desplazar la cámara y moverse a través de un entorno 3D.
Precondiciones: Ninguna.
Post-condiciones: La cámara se desplaza.
Escenario 1: <ol style="list-style-type: none"> 1. El usuario pulsa la tecla W del teclado o hace clic izquierdo con el ratón mientras lo arrastra hacia delante. 2. La cámara se desplaza hacia delante.
Escenario 2: <ol style="list-style-type: none"> 1. El usuario pulsa la tecla S del teclado o hace clic izquierdo con el ratón mientras lo arrastra hacia atrás. 2. La cámara se desplaza hacia atrás.
Escenario 3: <ol style="list-style-type: none"> 1. El usuario pulsa la tecla Q del teclado o hace clic izquierdo con el ratón mientras lo arrastra hacia la izquierda. 2. La cámara se desplaza en dirección lateral hacia la izquierda.
Escenario 4: <ol style="list-style-type: none"> 1. El usuario pulsa la tecla E del teclado o hace clic izquierdo con el ratón mientras lo arrastra hacia la derecha. 2. La cámara se desplaza en dirección lateral hacia la derecha.
Escenario 5: <ol style="list-style-type: none"> 1. El usuario pulsa la tecla C del teclado. 2. La cámara se desplaza en sentido vertical hacia abajo.
Escenario 6: <ol style="list-style-type: none"> 1. El usuario pulsa la barra de espacio del teclado. 2. La cámara se desplaza en sentido vertical hacia arriba.

Tabla 5.19: Descripción del caso de uso CdU1.1

Identificador: CdU1.2
Nombre: Rotar (Cámara)
Descripción: Funcionalidad para que un usuario pueda rotar la cámara y cambiar la dirección de enfoque en un entorno 3D.
Precondiciones: Ninguna.
Post-condiciones: La cámara cambia la dirección de enfoque.
Escenario 1:
1. El usuario pulsa la tecla <i>RePag</i> del teclado o hace clic derecho con el ratón mientras lo arrastra hacia arriba. 2. La cámara rota cambiando su dirección de enfoque hacia arriba.
Escenario 2:
1. El usuario pulsa la tecla <i>AvPag</i> del teclado o hace clic derecho con el ratón mientras lo arrastra hacia abajo. 2. La cámara rota cambiando su dirección de enfoque hacia abajo.
Escenario 3:
1. El usuario pulsa la tecla A del teclado o hace clic derecho con el ratón mientras lo arrastra hacia la izquierda. 2. La cámara rota cambiando su dirección de enfoque hacia la izquierda.
Escenario 4:
1. El usuario pulsa la tecla D del teclado o hace clic derecho con el ratón mientras lo arrastra hacia la derecha. 2. La cámara rota cambiando su dirección de enfoque hacia la derecha.

Tabla 5.20: Descripción del caso de uso CdU1.2

De este modo, mediante sencillas operaciones vectoriales -tales como sumas, restas, multiplicaciones, productos vectoriales y productos escalares- es posible calcular la nueva posición y orientación de la cámara. Por ejemplo, para la operación de *elevar* (desplazar la cámara lo largo del eje Y) bastaría con sumar al vector que indica la posición de la cámara, un vector con la misma dirección que el vector director *arriba* que se sitúa sobre el eje Y.

Si se desea rotar la cámara sobre el eje Y (Fig. 5.20), es necesario definir a priori el ángulo de rotación (α). Nótese que la rotación se efectúa sobre el eje Y, por lo tanto, la dirección del

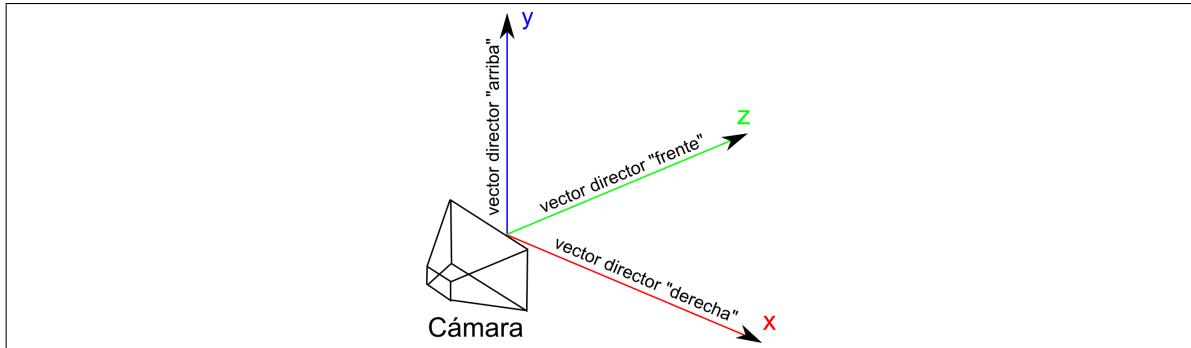


Figura 5.19: Esquema de los vectores directores que mantienen la orientación de la cámara

vector director *arriba*, que es perpendicular y saliente al papel, no varía. A partir de ahí, como se conocen los tres vectores directores que la orientan ($frente_0$, $derecha_0$ y $arriba_0$), se puede calcular la nueva dirección de enfoque $frente_1$ mediante la suma de sus componentes $frente_{1y}$ y $frente_{1x}$, donde $frente_{1y} = frente_0 * \cos\alpha$ y $frente_{1x} = derecha_0 * \sin\alpha$. A continuación, se recalcula el vector director $derecha_1$ a partir del producto vectorial de $arriba_0$ y $frente_1$.

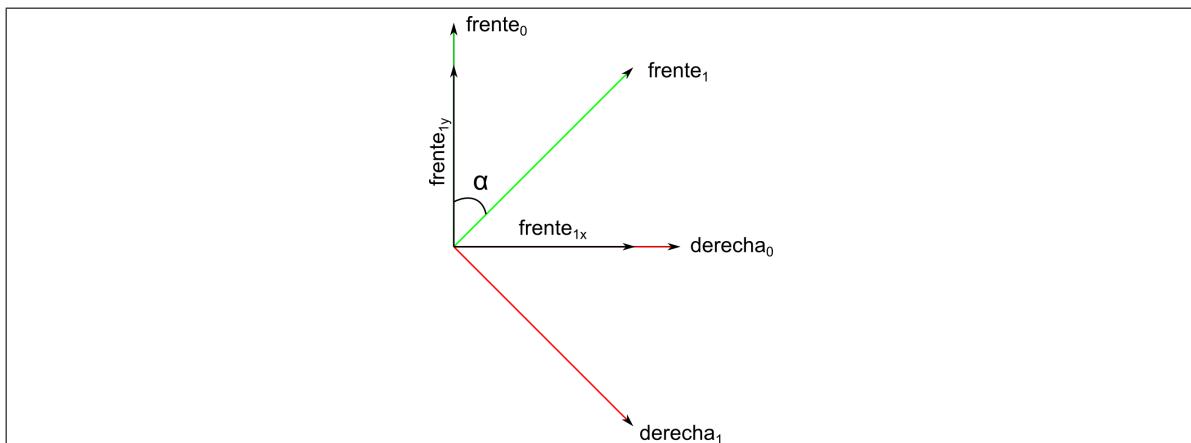


Figura 5.20: Cálculo de los vectores directores de orientación de la cámara después de una operación de rotación. Los vectores $arriba_0$ y $arriba_1$ no se visualizan ya que se consideran perpendiculares y en sentido saliente al papel

5.2.2.3 Disciplinas de diseño

De la disciplina de análisis de esta iteración se entiende que para abordar el diseño de los CdU1.1 y CdU1.2 es necesario diseñar una clase Java para la cámara y una clase auxiliar que represente un vector en un espacio de coordenadas (x , y , z) (Fig. 5.21). Las responsabilidades de cada una de estas clases es clara: la clase Vector3f se encargará de realizar operaciones

con vectores y la clase `GLCamera` se beneficiará de las mismas para calcular su posición y orientación al realizar las operaciones de desplazamiento y rotación.

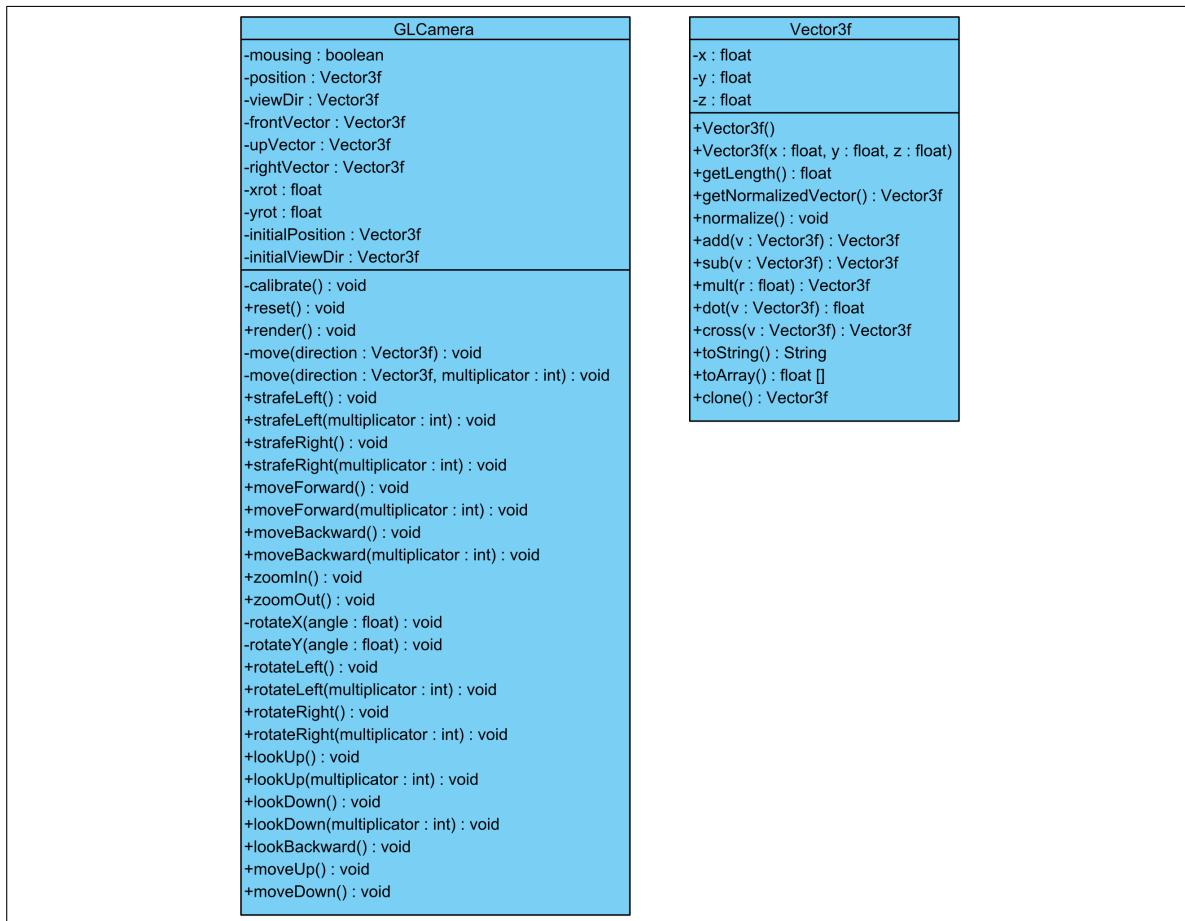


Figura 5.21: Diseño de las clases `GLCamera` y `Vector3f`

Los *listeners* de teclado y ratón podrán manipular el estado y comportamiento de la cámara gracias a la referencia que mantienen del objeto de tipo `GLDrawer`, que a su vez tiene una referencia a la cámara y es el encargado de pintar la escena (Fig. 5.22).

En cuanto al diseño de CdU1.5 se propone una arquitectura de vistas basada en herencia y polimorfismo. Así pues, se define una clase abstracta de tipo `GLViewManager` (Gestor de Vistas GL) que implementará un conjunto de operaciones comunes a todas las vistas (por ejemplo, dibujar el suelo de la escena) y definirá una plantilla con métodos que deberán ser implementados por cada vista especializada, es decir, las clases que la extiendan. Además, por medio del patrón de diseño **Factory Method** -formado por la clase `IViewManagerFactoryImpl` y la interfaz `IViewManagerFactory`- se abstraerá a la clase `GLDrawer` de la creación específica de cada vista (Fig. 5.23). Por otro lado, para evitar crear múltiples instancias de un mismo gestor de vistas, se utilizará el patrón **Singleton**.

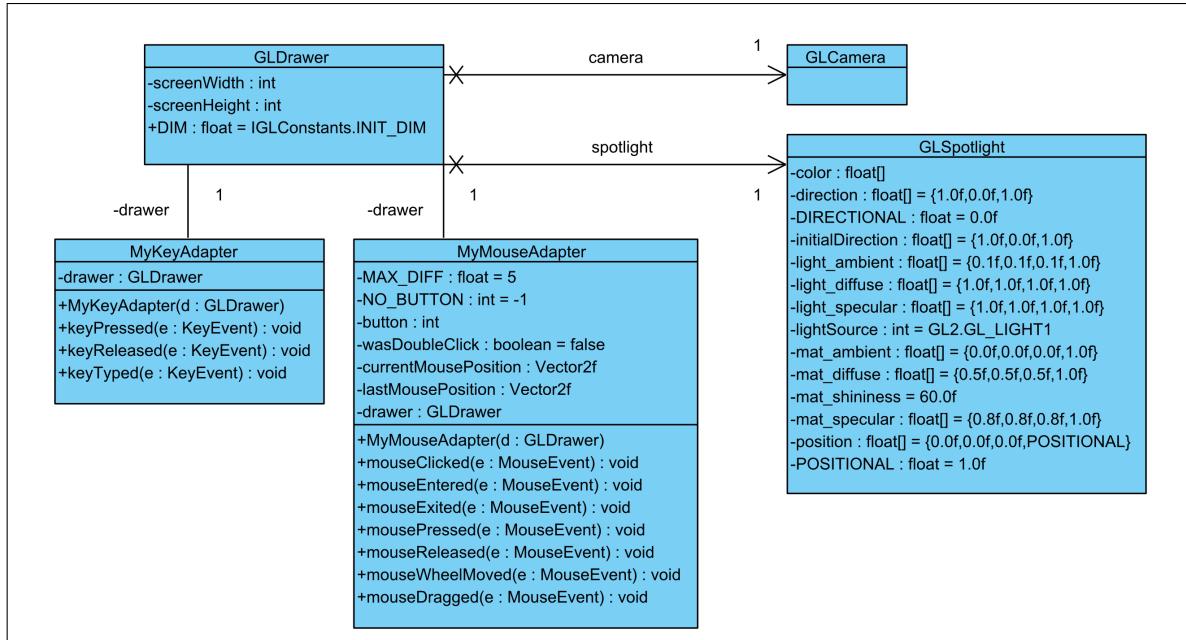


Figura 5.22: Diagrama de clases de diseño para CdU1.1 y CdU1.2

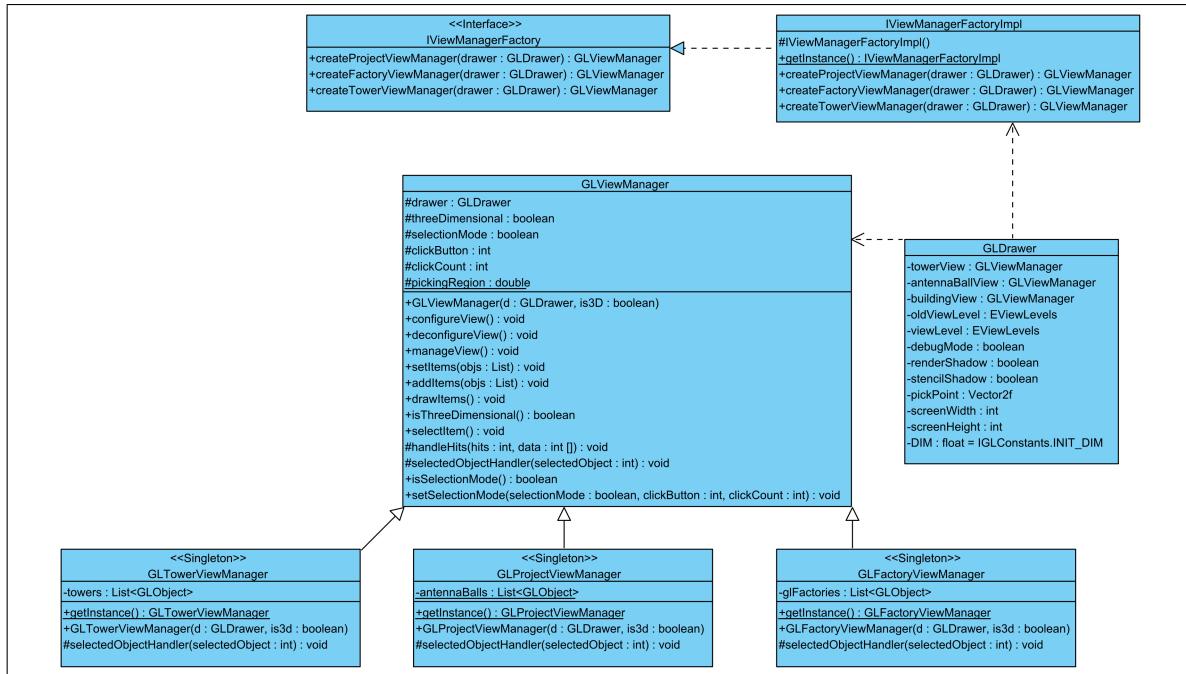


Figura 5.23: Diagrama de clases de diseño para CdU1.5

Para terminar, debido a los requisitos que se han identificado en esta iteración, es necesario plantear y tomar una serie de decisiones que condicionan el diseño, implementación y funcionamiento del motor gráfico en iteraciones posteriores.

Dimensiones de los modelos gráficos

Se plantea diseñar cada modelo gráfico como una clase Java. Como es habitual, una clase

Java contiene un conjunto de atributos que definen el estado de los objetos que se instancian. Para poder diferenciar qué atributos pueden ser empleados como dimensiones, así como para dotar la dimensión de la semántica que se crea necesaria, se propone anotar los atributos de la clase con **anotaciones personalizadas de Java**.

El uso de anotaciones personalizadas también conlleva la implementación de un pequeño **analizador de anotaciones** basado en **Java API Reflection**. Gracias a este API se puede utilizar el método de **introspección** para analizar en tiempo de ejecución la clase anotada y proporcionar la información necesaria a la interfaz gráfica de usuario para que ésta se configure automáticamente. De este modo se satisfará el requisito de interfaz gráfica de usuario adaptable.

En la figura 5.24 se muestra un breve diagrama de clases en la que se define la estructura de la anotación y su correspondiente analizador.

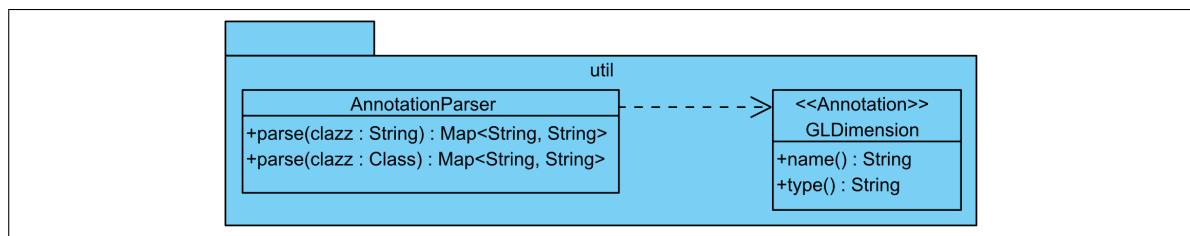


Figura 5.24: Diagrama de clases de la anotación GLDimension y su correspondiente analizador

Motor gráfico independiente de la aplicación contenedora

Persiguiendo el objetivo por el cual el motor gráfico podrá ser empleado en cualquier tipo de aplicación, se plantea diseñar un mecanismo de generación de gráficos a partir de una cadena de texto. Esta cadena de texto, de estructura predefinida, será utilizada como entrada de datos al motor gráfico. El motor gráfico la procesará según proceda y, finalmente, mostrará los resultados en forma de gráficos 3D.

Para establecer el formato de la cadena de texto de entrada, se propone **JSON (JavaScript Object Notation)** (Listado 5.2), cuya estructura de nodos será definida y detallada en sucesivas iteraciones.

```

1 { "menu": {
2   "id": "file",
3   "value": "File",
4   "popup": {
5     "menuitem": [

```

```

6      {"value": "New", "onclick": "CreateNewDoc()"},  

7      {"value": "Open", "onclick": "OpenDoc()"},  

8      {"value": "Close", "onclick": "CloseDoc()"}  

9    ]  

10 }  

11 } }

```

Listado 5.2: Ejemplo de texto en formato JSON (<http://json.org/example.html>)

5.2.2.4 Disciplinas de implementación

En esta disciplina se lleva a cabo la implementación de las clases que se han diseñado a lo largo del ciclo de vida para satisfacer los requisitos funcionales de desplazamiento y rotación de la cámara y el requisito no funcional de la iluminación de la escena.

En este documento sólo se incluyen dos aspectos relativos a la implementación la cámara. Se trata de las funciones *render()* y *calibrate()* de la clase **GLCamera** (Listado 5.3).

- *render()*. Esta función, que será ejecutada cada vez que se produzca cualquier tipo de modificación en la posición de la cámara o su dirección de enfoque, se encarga de hacer efectivos los nuevos cálculos.
- *calibrate()*. Un movimiento de rotación implica que la dirección a la que apunta el objetivo de la cámara cambie. Así pues, los vectores directores -*frente* y *derecha*- que establecen la orientación de la cámara quedan desorientados y deben ser “re-calibrados”.

Esta operación de *re-calibrado* se lleva a cabo mediante la *regla o ley de la mano derecha* y productos vectoriales.

```

1 public class GLCamera {  

2     private Vector3f position; // Camera position  

3     private Vector3f viewDir; // View direction of camera eye  

4     private Vector3f frontVector; // Front unit-director vector  

5     private Vector3f upVector; // Up unit-director vector  

6     private Vector3f rightVector; // Right unit-director vector  

7  

8     private void calibrate () {  

9         upVector = new Vector3f(0.0f, 1.0f, 0.0f);  

10        rightVector = viewDir.cross(upVector);  

11        frontVector = rightVector.cross(upVector).mult(-1);

```

```

12     xrot = yrot = 0.0f;
13 }
14
15 public void render() throws GLSingletonNotInitializedException
16 {
17     Vector3f viewPoint = this.getViewPoint();
18
19     GLSingleton.getGLU().gluLookAt(position.getX(), position.
20         getY(), position.getZ(),
21         viewPoint.getX(), viewPoint.getY(), viewPoint.getZ(),
22         upVector.getX(), upVector.getY(), upVector.getZ());
23 }

```

Listado 5.3: Fragmento de código Java de la clase GLCamera

5.2.3 Iteración 3

En esta iteración (Tabla 5.11), que se centra en el componente correspondiente a la aplicación web, se aborda:

- Especificación detallada de los requisitos funcionales más importantes de la herramienta: configuración de perfiles y visualización.
- Análisis y abstracción del mundo real en el contexto del DSG.
- Diseño del modelo de datos que dará lugar a la base de datos utilizada por la herramienta para la gestión del conocimiento.
- Investigación acerca de las tecnologías de desarrollo y diseño web que serán utilizadas.
- Preparación del entorno de desarrollo y creación del proyecto del componente web mediante Maven.

5.2.3.1 Disciplina de requisitos

La aplicación web debe ser diseñada de forma que de lugar a una herramienta profesional y de calidad. Para lograrlo se deberán utilizar tecnologías y *frameworks* (Sección 4.4) que

actualmente se utilicen en el mundo empresarial para fomentar un desarrollo y mantenimiento fácil y poco costoso.

A continuación se especifican detalladamente los requisitos funcionales más importantes del componente web de visualización: configuración de perfiles de visualización y visualización de medidas e indicadores del DGS.

Configuración de perfiles de visualización

La configuración de un perfil de visualización deberá comenzar con la selección de una entidad del contexto del desarrollo global -compañía, factoría, proyecto o subproyecto- y el modelo gráfico sobre el que se desea visualizar sus medidas e indicadores. Cuando se hayan seleccionado ambos elementos, se mostrarán dos listas: una incluirá las medidas e indicadores software de la entidad y la otra contendrá las dimensiones mapeables del modelo. Acto seguido, el usuario establecerá mediante un formulario los mapeos y configuraciones que estime oportunos, respetando la compatibilidad de los datos, y podrá aportar información adicional para mostrar una leyenda informativa dentro de la escena 3D. Además deberá indicar el nombre del perfil y una breve descripción. Para finalizar, el perfil de visualización deberá hacerse persistente para su posterior recuperación durante las operaciones de visualización.

Visualización de medidas e indicadores del DGS

Esta funcionalidad proporciona dos vistas: la primera de ellas geolocalizará las factorías de software que proceda, según un criterio de filtro o selección, en un mapa mundial; la segunda mostrará un conjunto de gráficos 3D generados con el motor gráfico. En esta segunda vista el usuario podrá interactuar con los elementos de la escena para navegar por los distintos niveles de abstracción de la visualización y desglosar información cada vez más específica.

5.2.3.2 Disciplina de análisis

De las decisiones de diseño que se han tomado en la iteración 2 (Sección 5.2.2.3), en este momento cabe destacar la que determina que el motor gráfico generará gráficos 3D a partir de una cadena de texto en formato JSON. Esto implica que el motor gráfico no maneje información relativa al contexto del desarrollo global. Por ello, toda la lógica de dominio relativa al desarrollo global, así como su control y persistencia, está contenida en la aplicación web.

Por otro lado, durante esta disciplina también se realiza un estudio conceptual del mundo real en el contexto del desarrollo global para modelar el diagrama entidad-relación extendido (EER) de los datos.

5.2.3.3 Disciplina de diseño

En esta disciplina se resuelven dos cuestiones importantes. Por un lado, se obtiene el modelo entidad-relación extendido (EER) que dará lugar a la base de datos y, por otro, se toman algunas decisiones de diseño relacionadas con los requisitos identificados en la disciplina de requisitos de esta misma iteración (Sección 5.2.3.1).

En el modelo EER (Fig. 5.25), que ha sido elaborado con la herramienta MySQL Workbench, se observan las distintas tablas que formarán la base de datos. Este modelo se ha diseñado a partir del diagrama de clases que representa estructura de la lógica de dominio (Fig. 5.26) ya que marca las clases persistentes con el estereotipo *ORM Persistable*. Conceptualmente expresa que una compañía posee un conjunto de factorías; cada factoría puede tener asignado un conjunto de proyectos; y cada proyecto, a su vez, se puede fragmentar en una serie de subproyectos. Este mismo concepto es una generalización de la figura 2.1 que se facilitó en la sección 2.1.

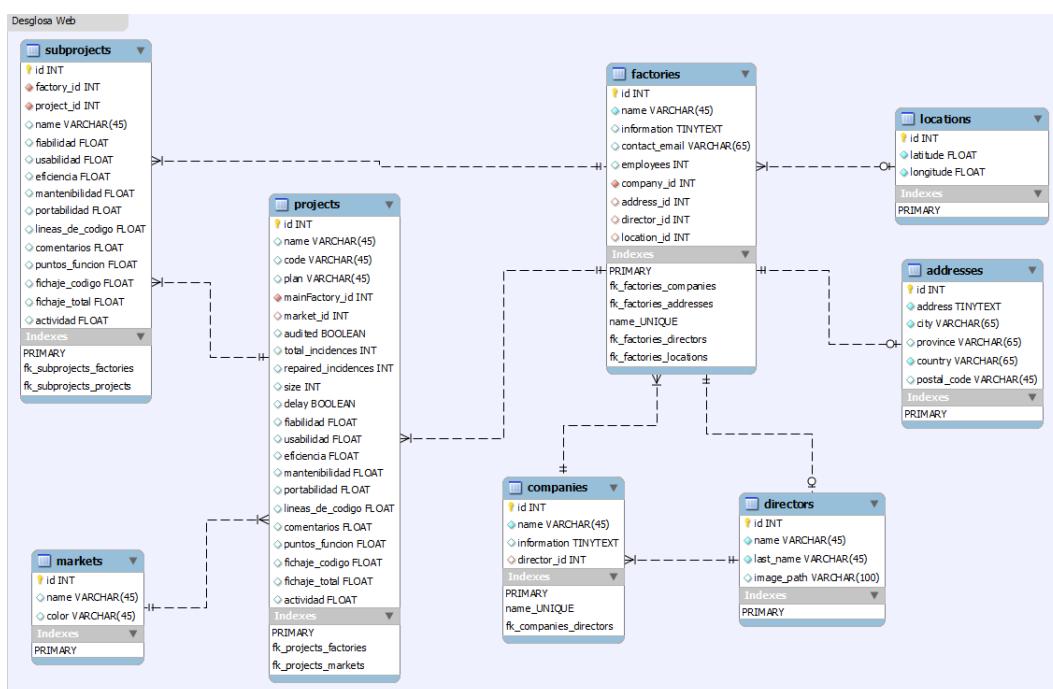


Figura 5.25: Modelo Entidad-Relación Extendido

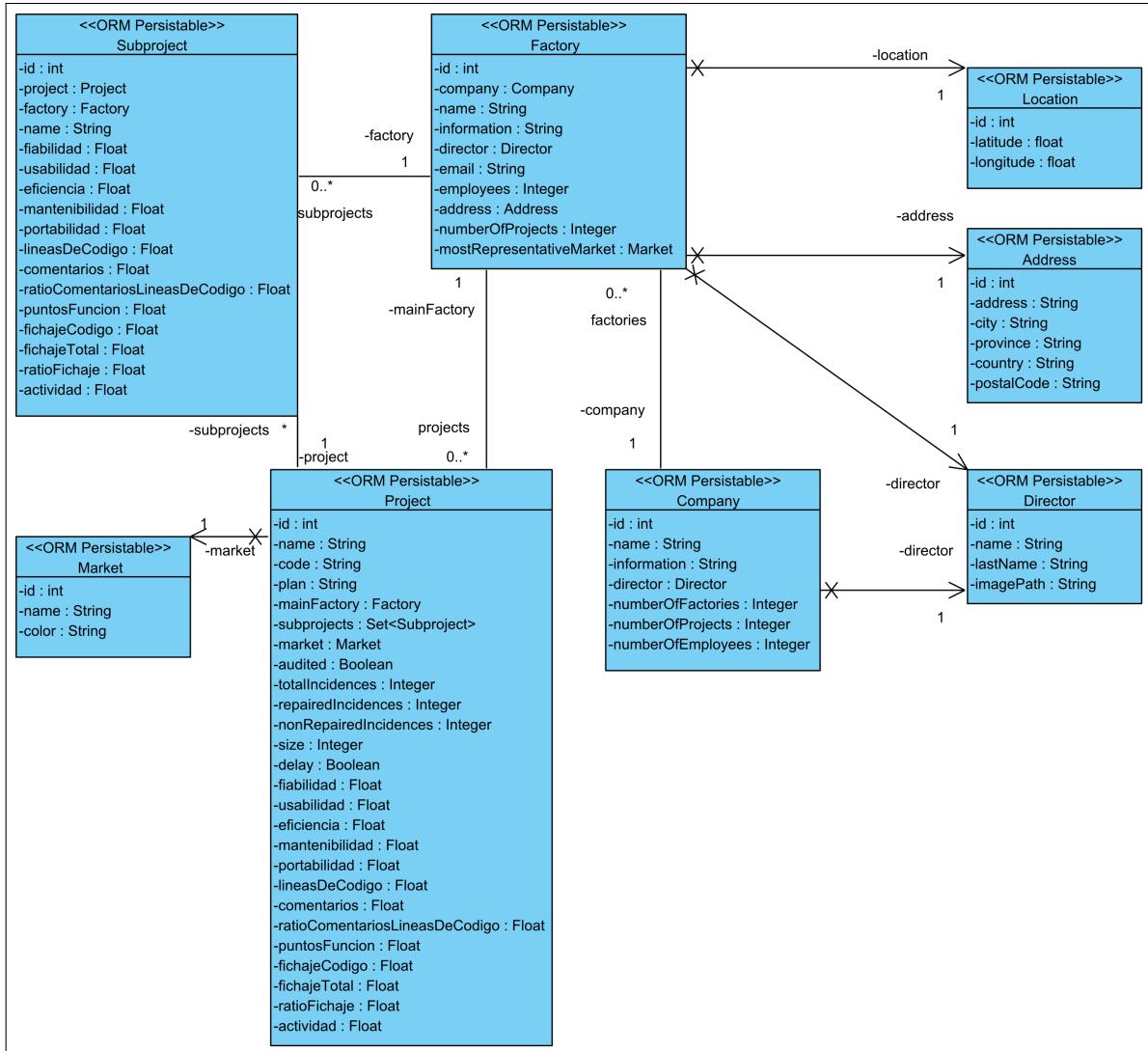


Figura 5.26: Diagrama de clases de conocimiento para el contexto del desarrollo global

A continuación se detallan las decisiones de diseño que se han tomado para satisfacer los requisitos identificados en esta misma iteración.

Configuración de perfiles de visualización

Las entidades que modelan el contexto del desarrollo global (compañía, factoría, proyecto y subproyecto) serán diseñadas como clases Java. Como es habitual, una clase Java tiene un conjunto de atributos que definen el estado de los objetos que se instancian. Para determinar cuáles de estos atributos son susceptibles de representación visual, se propone una solución basada en **anotaciones personalizadas de Java** similar a la de detección de dimensiones de los modelos gráficos. Por ello, será necesario definir una nueva anotación y su correspondiente **analizador de anotaciones** basado en **Java API Reflection** (Fig. 5.27).

No obstante, en este caso surge una ligera diferencia ya que existen asociaciones entre las diferentes clases del dominio. Esta característica se observa perfectamente en el modelo EER (Fig. 5.25). De modo que puede resultar interesante la posibilidad de incluir información acerca de las medidas o indicadores de las entidades asociadas a la entidad para la cual se está configurando el perfil. Para lograr esta funcionalidad se añade a la anotación *PropertyAnnotation* el atributo *embedded*, que indicará al analizador si debe emplear nuevamente el método de introspección sobre la clase tipo del atributo anotado. Esta solución permite, nuevamente, analizar las entidades en tiempo de ejecución y adaptar la interfaz gráfica de usuario a sus necesidades.

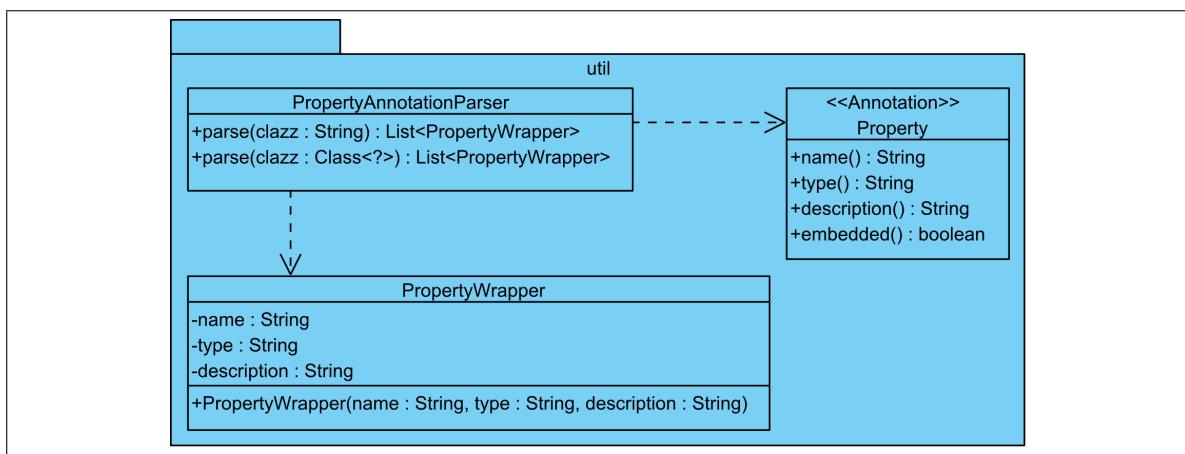


Figura 5.27: Diagrama de clases de la anotación *Property* y su correspondiente analizador

Por otro lado, los perfiles de visualización se harán persistentes mediante ficheros en formato XML (Extensible Markup Language), alojados en el servidor de la aplicación web. Se utilizará este formato dada la gran facilidad de *serialización* y *deserialización* que proporciona JAXB (Sección 4.4).

Visualización

La herramienta debe disponer de una vista que muestre un mapa del mundo sobre el cual se geolocalizarán distintas factorías en base a ciertos criterios de filtrado o selección. Para satisfacer este requisito, se propone el uso de alguno de los múltiples **Sistema de Información Geográfica** (SIG o GIS, en su acrónimo inglés Geographic Information System) que ofrecen un API para desarrolladores basada en JavaScript, como Bing Maps o Google Maps.

Por otro lado, para generar los gráficos 3D en base al perfil de visualización seleccionado por el usuario, será necesario examinar y modificar el estado y comportamiento de los objetos en tiempo de ejecución. Esta característica se puede llevar a cabo gracias al **Java API**

Reflection y el método de **reflexión**.

5.2.3.4 Disciplina de implementación

Como el entorno de desarrollo ya se instaló y configuró al comenzar la implementación del motor gráfico (Sección 5.2.1.4), se pasa directamente a la creación del proyecto.

En este caso la arquitectura Maven que se utilizará es **struts2-archetype-starter** [60]. Este arquetipo crea un proyecto de una aplicación web, cuya estructura se muestra en la figura 5.28, basada en tecnologías que se emplean actualmente en aplicaciones de producción. Entre sus características destacan la integración por defecto de Sitemesh, Spring, Struts 2 y mecanismos de internacionalización (i18N).

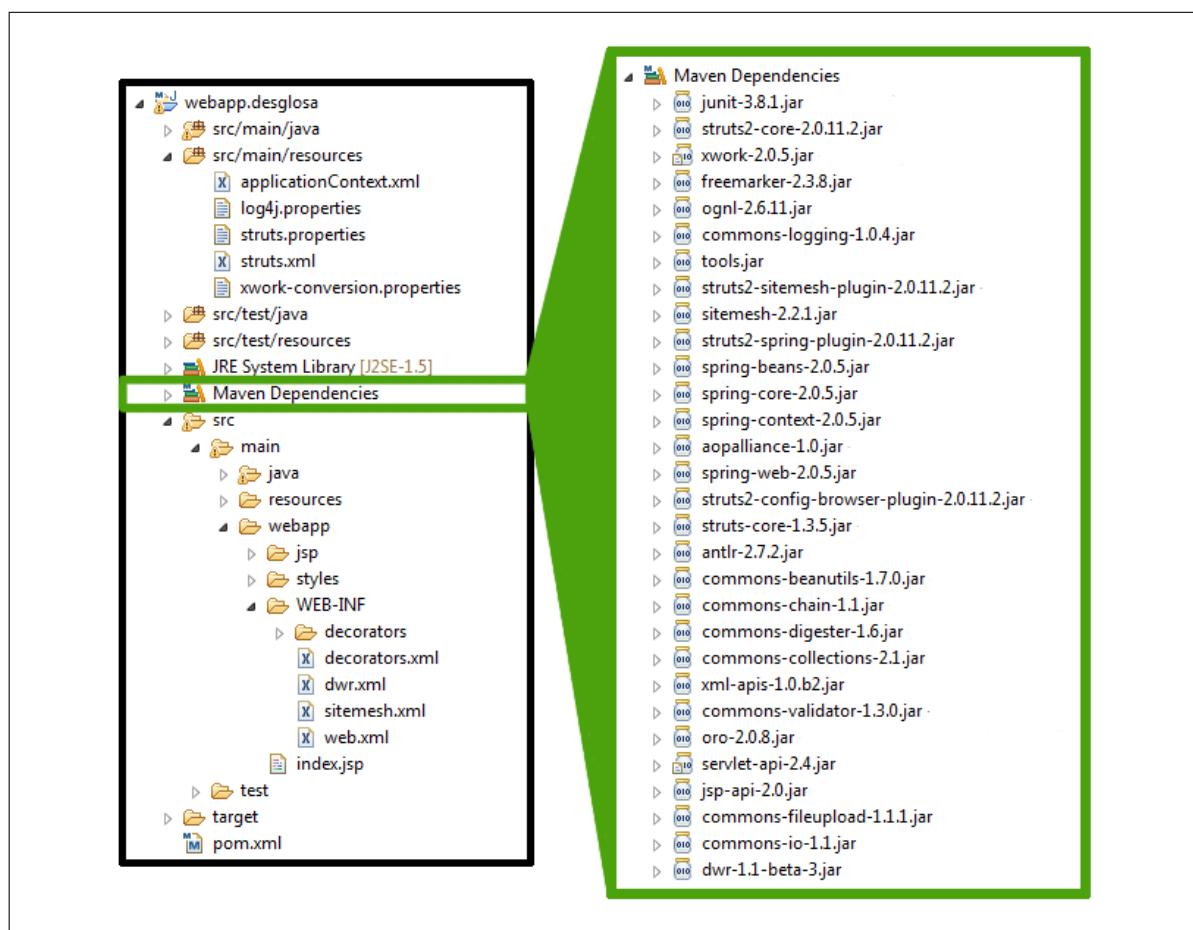


Figura 5.28: Estructura de un proyecto Maven con arquetipo *struts2-archetype-starter*

Para finalizar, se han implementado la anotación y el analizador de anotaciones descrito en la disciplina de diseño (Fig. 5.27), además las clases de dominio han sido debidamente anotadas.

De la implementación de la anotación personalizada **Property** (Listado 5.4) cabe destacar

las anotaciones `@Target` y `@Runtime` que establecen que la anotación `@Property` únicamente es aplicable a atributos de clase que se desea anotar y que debe mantenerse en el *bytecode* del fichero `.class` generado por el compilador hasta el tiempo de ejecución de la aplicación en la JVM (Java Virtual Machine), respectivamente.

```

1 package es.uclm.inf_cr.alarcos.desglosa_web.model.util;
2
3 import java.lang.annotation.ElementType;
4 import java.lang.annotation.Retention;
5 import java.lang.annotation.RetentionPolicy;
6 import java.lang.annotation.Target;
7
8 @Target(ElementType.FIELD)
9 @Retention(RetentionPolicy.RUNTIME)
10 public @interface Property {
11     String name() default "";
12
13     String type() default "";
14
15     String description() default "No description has been set.";
16
17     boolean embedded() default false;
18 }
```

Listado 5.4: Código fuente de la anotación personalizada `Property`

5.3 Fase de construcción

La fase de construcción, conformada por un total de 6 iteraciones, es la más extensa del ciclo de desarrollo de este PFC. En ella se abordan la mayor parte de las disciplinas de implementación y pruebas del sistema final.

5.3.1 Iteración 4

En esta iteración (Tabla 5.12) se aborda:

- Análisis y diseño para la configuración de escenas 3D.
- Diseño de una arquitectura extensible de metáforas de visualización.

- Implementación de la configuración y pintado de escenas.
- Pruebas y verificación del correcto funcionamiento de la cámara.

5.3.1.1 Disciplina de análisis

La configuración de escenas plantea una problemática referente al posicionamiento de los elementos de las metáforas de visualización en el entorno 3D. En esta disciplina se trabaja en un algoritmo de posicionamiento que calcule los emplazamientos de los elementos de la escena para que no se solapen unos con otros.

El emplazamiento de los elementos de la escena se realizará en tres niveles (Fig. 5.29):

- El primer nivel representa la metáfora de una ciudad o un polígono industrial, dependiendo de los elementos empleados en la escena.
- El segundo nivel representa los barrios o agrupaciones de elementos que conforman el primer nivel.
- El tercer nivel se corresponde con los elementos que se encuentran dentro de un barrio del segundo nivel. Estos elementos son los modelos gráficos que se emplean en la escena.

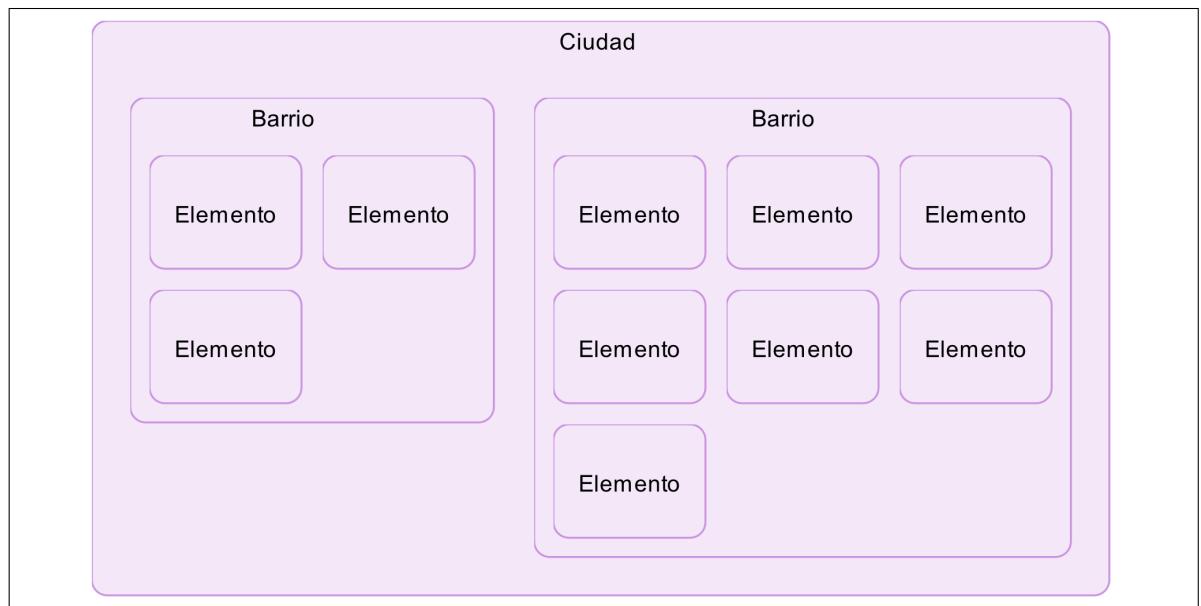


Figura 5.29: Esquema conceptual para el posicionamiento de los modelos gráficos en CdU1.5

Con el enfoque planteado, es posible calcular las posiciones relativas de los edificios en su propio barrio y, acto seguido, posicionar los barrios en la ciudad. Así pues, la aplicación

contenedora del motor gráfico deberá proporcionar una cadena de texto en formato JSON respetando la estructura de tres niveles que se ha descrito.

5.3.1.2 Disciplina de diseño

Durante esta disciplina, y a partir del análisis realizado, se elaboran los diagramas de diseño relativos al CdU1.6.

En la figura 5.30 se muestra el diagrama de clases que representa la estructura de una ciudad tal y como se ha sugerido en la disciplina de análisis. Cabe destacar que los *elementos* de un *barrio* son de tipo **GLObject**. Esta decisión de diseño se ha llevado a cabo para abstraer la generación de escenas del tipo de modelo gráfico que se vaya a emplear. De este modo, todos los modelos gráficos que se implementen deben ser especializaciones de *GLObject*.

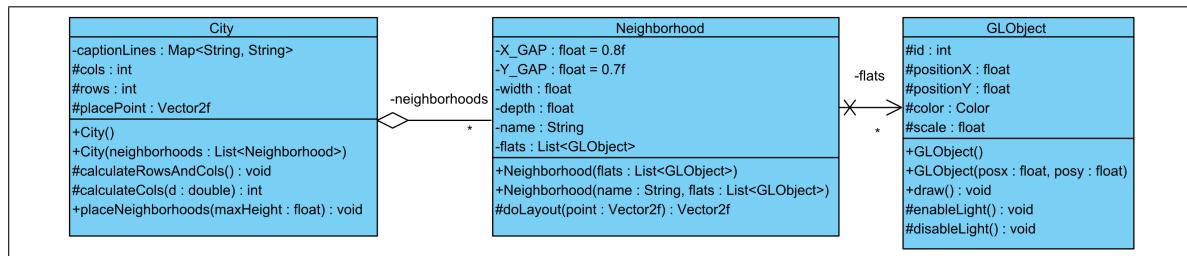


Figura 5.30: Diagrama de clases para CdU1.5 (1/2)

En la figura 5.31 se puede analizar cómo se han diseñado los modelos gráficos. En primer lugar se modela una clase **GLObject** con los atributos básicos que debe tener cualquier modelo gráfico, ya sea en dos o tres dimensiones: un identificador, su posición en un plano de dos dimensiones, el color y la escala. En segundo lugar se crea una clase denominada **GLObject3D** que hereda de *GLObject*. Esta especialización añade conceptos relativos con la altura o tercera dimensión del modelo y es a partir de la cual heredaran todos los modelos que se deseen implementar.

5.3.1.3 Disciplina de implementación

Durante esta disciplina se implementan los casos de uso CdU1.5 y CdU1.6, tal y como se estableció en el plan de iteraciones.

Cabe destacar que dada la jerarquía establecida para los modelos gráficos (Fig. 5.31), las dimensiones pueden encontrarse presentes en cualquier nivel (padres o hijos). Por lo tanto, el analizador de anotaciones de dimensión se ha implementado de forma que pueda recorrer

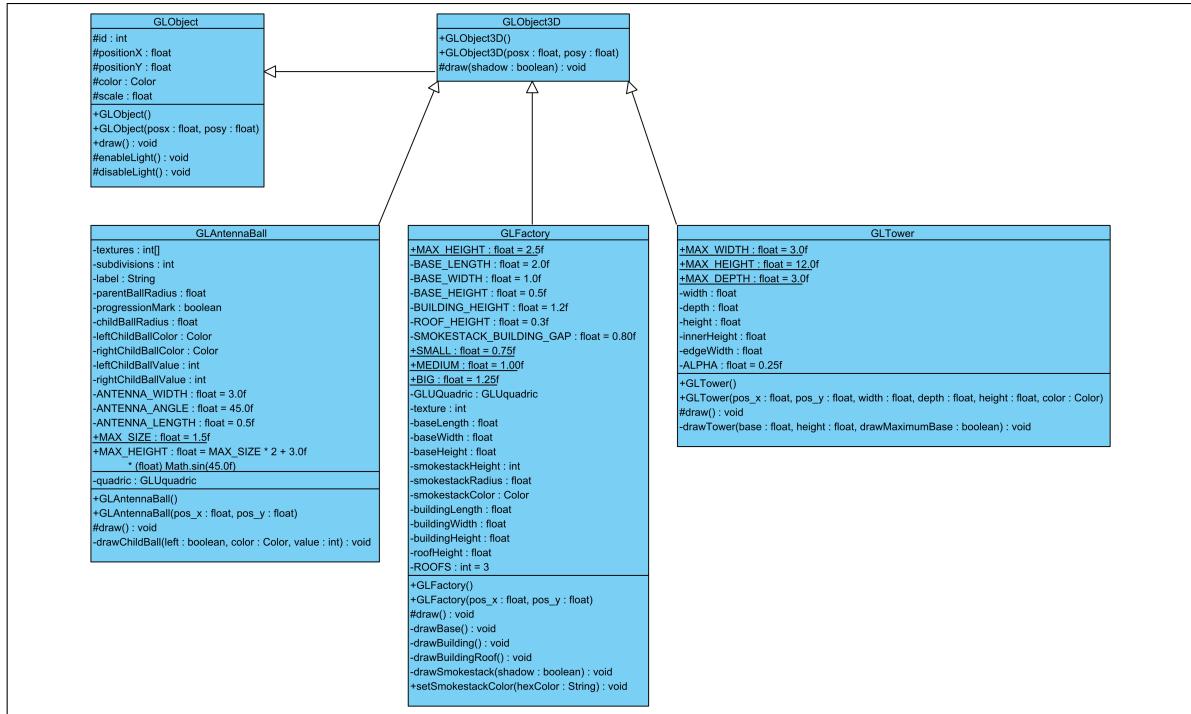


Figura 5.31: Diagrama de clases para CdU1.5 (2/2)

toda la jerarquía de herencia desde el modelo indicado hasta el nodo raíz de la misma. En el listado 5.5 se muestra un ejemplo de una clase anotada mediante la anotación *GLDimension*.

```

1 public abstract class GLObject {
2     @GLDimension(name = "id", type = "int")
3     protected int id;
4     protected float maxWidth = 0.0f;
5     protected float maxDepth = 0.0f;
6     protected float positionX;
7     protected float positionY;
8     @GLDimension(name = "color", type = "color")
9     protected Color color;
10    @GLDimension(name = "scale", type = "float_range")
11    protected float scale;
12    ...
13}

```

Listado 5.5: Ejemplo de clase Java anotada con la anotación *GLDimension*

Para finalizar, se incluye una captura de pantalla que muestra la evolución del motor gráfico (Fig. 5.32).

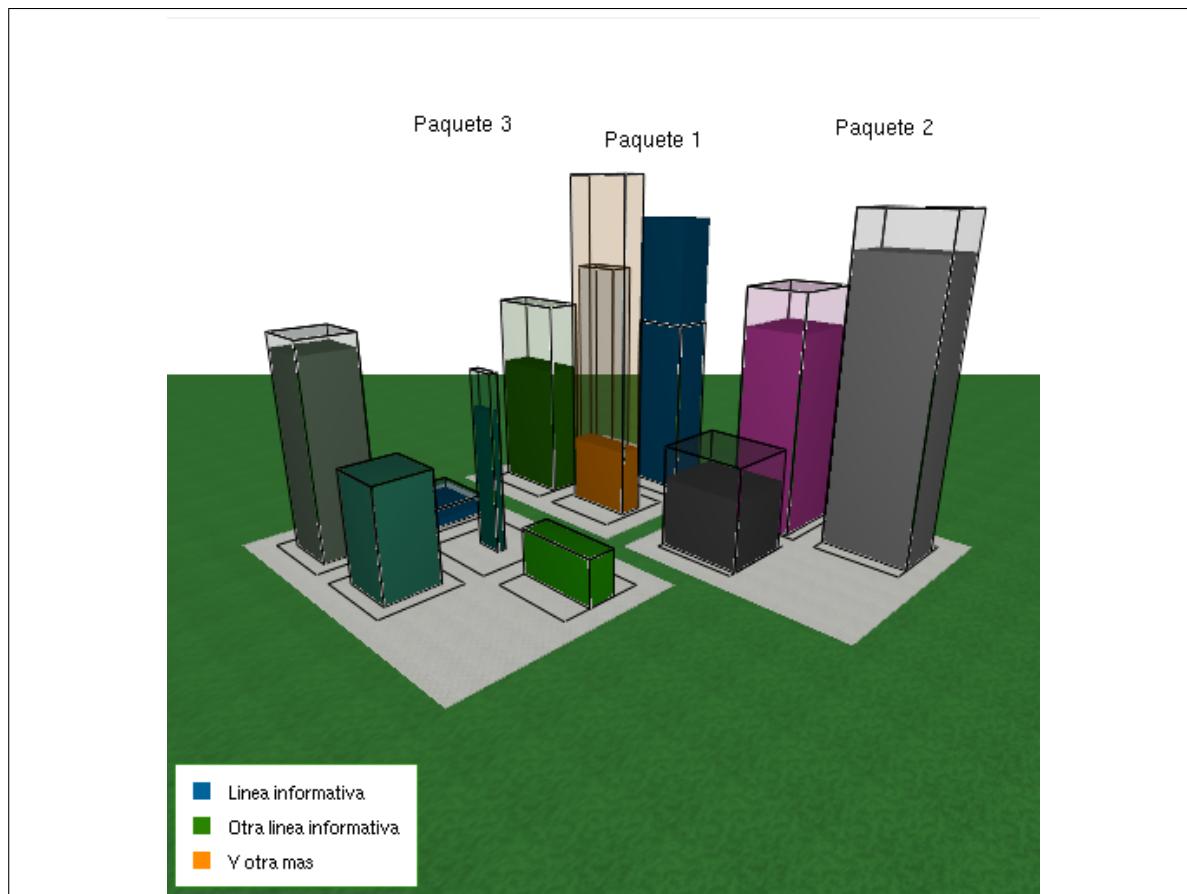


Figura 5.32: Gráficos generados después de la disciplina de implementación de la cuarta iteración

5.3.1.4 Disciplina de pruebas

Las pruebas de los casos de uso correspondientes al movimiento y rotación de la cámara del motor gráfico (CdU1.1 y CdU1.2) se han realizado con la colaboración de un usuario experto.

El proceso que se ha seguido ha sido el siguiente:

1. Diseño de los casos de prueba utilizando como base la descripción de los CdU1.1 y CdU1.2 (Tabla 5.19).
2. Elaboración de un *checklist* con los casos de prueba diseñados en el punto 1. En cada entrada del checklist se ha incluido una casilla en la que el usuario deberá anotar si el resultado de la operación ha sido correcto o no.
3. Planificación de una reunión con el usuario experto, en la cual se le facilitará un ordenador con una instancia del motor gráfico y el checklist elaborado en el punto 2.
4. Ejecución de los casos de prueba diseñados y cumplimentado del checklist.

El informe obtenido como resultado de esta disciplina indica que todas las pruebas han sido ejecutadas correctamente. En este documento sólo se adjunta el informe correspondiente a las pruebas del CdU1.1 (Tabla 5.21).

Identificador de CDU: CdU1.1		
Caso de prueba	Descripción	Resultado
Desplazar hacia delante	El usuario percibe la sensación de desplazamiento hacia delante cuando pulsa la tecla W del teclado o hace clic izquierdo con el ratón mientras lo arrastra hacia delante.	<input checked="" type="checkbox"/> Sí <input type="checkbox"/> No
Desplazar hacia atrás	El usuario percibe la sensación de desplazamiento hacia atrás cuando pulsa la tecla S del teclado o hace clic izquierdo con el ratón mientras lo arrastra hacia atrás.	<input checked="" type="checkbox"/> Sí <input type="checkbox"/> No
Desplazar hacia la izquierda	El usuario percibe la sensación de desplazamiento lateral hacia la izquierda cuando pulsa la tecla Q del teclado o hace clic izquierdo con el ratón mientras lo arrastra hacia la izquierda.	<input checked="" type="checkbox"/> Sí <input type="checkbox"/> No
Desplazar hacia la derecha	El usuario percibe la sensación de desplazamiento lateral hacia la derecha cuando pulsa la tecla E del teclado o hace clic izquierdo con el ratón mientras lo arrastra hacia la derecha.	<input checked="" type="checkbox"/> Sí <input type="checkbox"/> No
Descender	El usuario percibe la sensación de descenso vertical cuando pulsa la tecla C del teclado.	<input checked="" type="checkbox"/> Sí <input type="checkbox"/> No
Ascender/Elevar	El usuario percibe la sensación de ascenso vertical cuando pulsa la barra espaciadora del teclado.	<input checked="" type="checkbox"/> Sí <input type="checkbox"/> No

Tabla 5.21: Informe de pruebas para CdU1.1

5.3.2 Iteración 5

En esta iteración (Tabla 5.13) se aborda:

- Análisis, diseño e implementación de las funcionalidades que permiten seleccionar objetos de una escena 3D, identificar el objeto seleccionado y cambiar entre distintos niveles de visualización.
- Análisis y diseño de la integración del motor gráfico en la web y del SIG.
- Pruebas y verificación de la correcta configuración y visualización de escenas.

5.3.2.1 Disciplina de análisis

Dada la estrecha relación que existe entre los casos de uso CdU1.3 y CdU1.7, el análisis de ambos se ha realizado de manera conjunta.

En la figura 5.33 se muestra un diagrama de comunicación que representa el paso de mensajes que se produce entre los distintos controladores del motor gráfico cuando el usuario efectúa un *clic* sobre un modelo gráfico. Entre ellos destaca un nuevo elemento llamado **observador**. Este elemento, que se ha representado con una clase de control, en realidad consistirá en un **patrón observador**. Por medio de este patrón, será posible notificar el resultado de una operación de selección a la interfaz gráfica de usuario sin acoplar la capa de dominio con la capa de presentación.

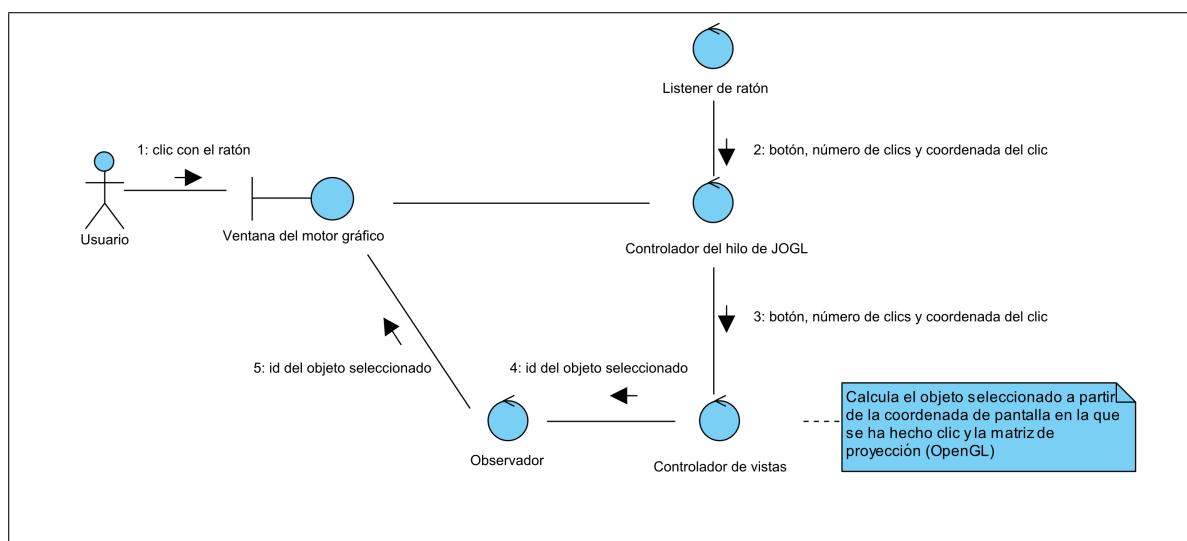


Figura 5.33: Diagrama de comunicación para CdU1.3 y 1.7

Por otro lado, el cambio de vistas o niveles de visualización no es una tarea trivial debido a las limitaciones de implementación que presenta JOGL en lo que a múltiples hilos de ejecución se refiere. Como Java implementa los *listeners* de teclado y ratón como hilos separados del hilo principal, no es posible cambiar la vista directamente desde ellos. Así pues,

dichos hilos cambiarán una variable de control que será examinada por el hilo principal de ejecución que será el encargado de hacer el cambio de vista efectivo. Para comprender mejor esta idea se proporciona un diagrama de flujo (Fig. 5.34).

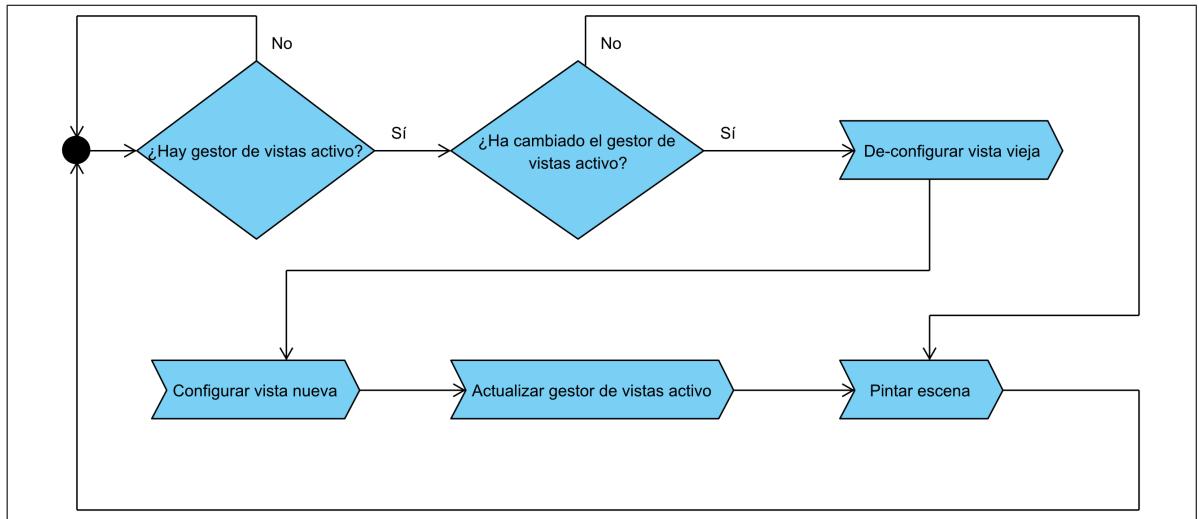


Figura 5.34: Diagrama de flujo para CdU1.4

5.3.2.2 Disciplina de diseño

Durante esta disciplina se abordan cuestiones de diseño tanto del motor gráfico como de la aplicación web.

Motor gráfico

La selección de objetos es una funcionalidad similar en todas las vistas. Por ello, su implementación se hará una sola vez en la clase *GLViewManager* a partir de la cual surgen todas las vistas especializadas (Fig. 5.35). Por otro lado, cada vista -*GLTowerViewManager*, *GLProjectViewManager* y *GLFactoryViewManager*- deberá implementar de manera específica cómo tratar el objeto seleccionado. No obstante, las acciones que se llevarán a cabo con el objeto seleccionado no se han delegado en el motor gráfico, sino que se dejarán a merced de la aplicación contenedora que, en este caso, es una aplicación web. Esta decisión se fundamenta en el deseo de maximizar la capacidad de reutilización del motor gráfico en otros proyectos.

Para que el motor gráfico pueda notificar el objeto seleccionado a la aplicación contenedora, cualesquiera que sea la tecnología en la que se implemente su interfaz gráfica de usuario (jsp, swing, awt, etcétera), se ha diseñado el sistema de notificación mediante un **patrón observador** (Fig. 5.35). El uso de este patrón permite que la notificación de selección de

objetos no tenga ningún tipo de dependencia con la capa de presentación del motor gráfico ni con la aplicación contenedora. La única dependencia de uso se da entre la capa de presentación del motor gráfico con la capa de presentación de la aplicación contenedora. Aún así, esta dependencia puede reducirse casi al completo mediante un patrón **Adapter**.

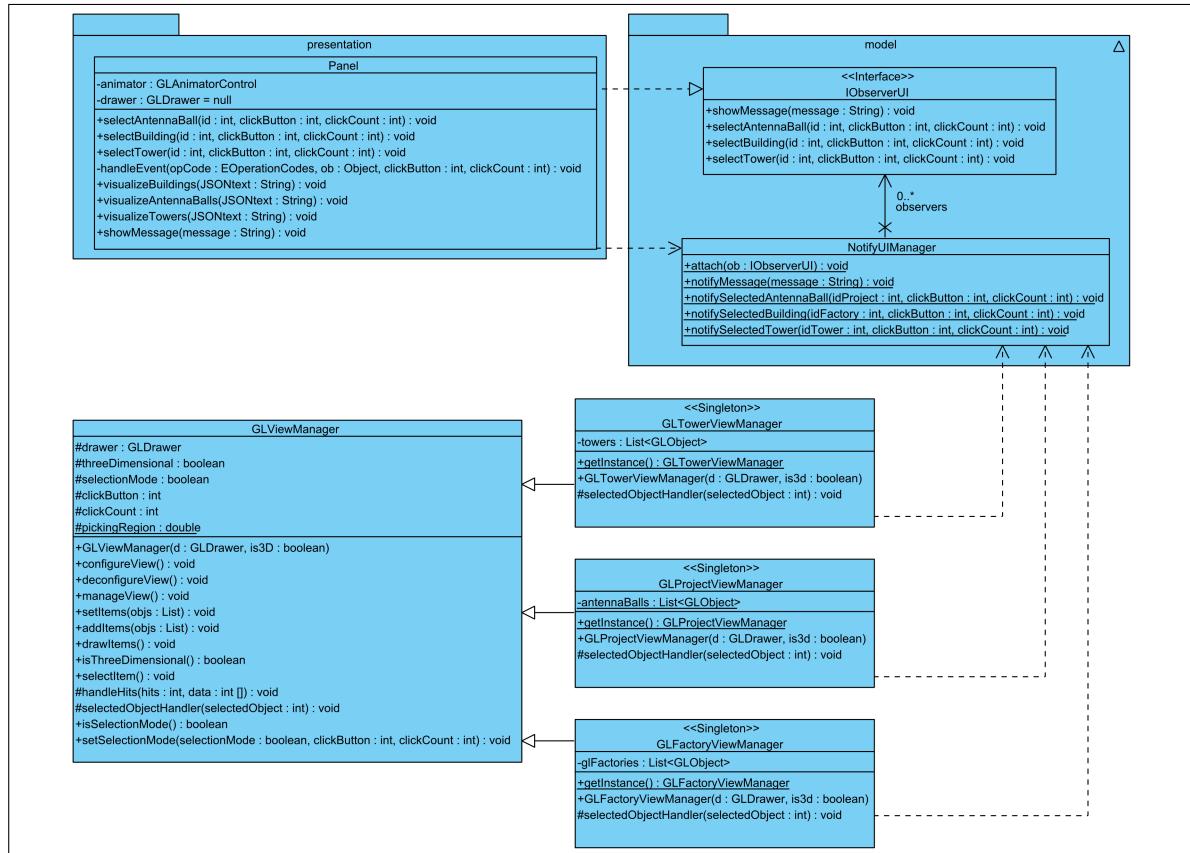


Figura 5.35: Diagrama de clases de diseño para CdU1.3 y CdU1.7

El diagrama de secuencia que se muestra en la figura 5.36, representa una vista dinámica del funcionamiento del mecanismo de selección de objetos. Para empezar, es necesario que el contenedor del motor gráfico se añada a la lista de observadores para que se le puedan notificar las operaciones de selección cuando sea requerido. A continuación, cuando el usuario efectúa un clic sobre un elemento de la escena, se desencadena una serie de eventos y paso de mensajes. Finalmente, el gestor de vista activo **notificará al controlador NotifyUIManager** qué objeto ha sido seleccionado para que éste lo notifique a todos los observadores, entre los que se encuentra el contenedor del motor gráfico.

Aplicación web

Tal y como se propuso en la disciplina de diseño de la tercera iteración (Sección 5.2.3.3), el CdU2.7 y CdU2.8 se satisfará mediante alguno de los múltiples **Sistema de Información**

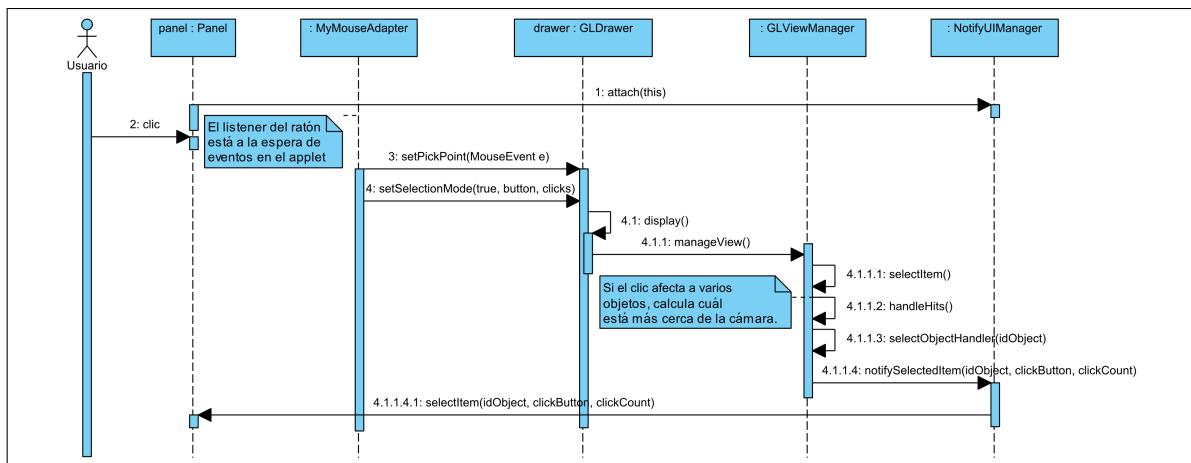


Figura 5.36: Diagrama de secuencia para CdU1.3 y 1.7

Geográfica (SIG o GIS, en su acrónimo inglés Geographic Information System) que ofrecen un API para desarrolladores basada en JavaScript, como Bing Maps o Google Maps. Para concretar, en este PFC se utilizará Google Maps ya que dispone de un API bien documentada y su licencia se adapta a los requisitos del mismo.

Por otro lado, surge la necesidad de integrar el motor gráfico en la aplicación web. Esta operación se llevará a cabo mediante un *applet* ya que, desde la versión 1.6.0_10 del JDK (Java Development Kit), su eficiencia se ha mejorado notablemente y hace muy factible el despliegue de grandes aplicaciones contenidas en los mismos. No obstante, persiste la problemática relacionada con la dependencia que posee el motor gráfico con múltiples librerías de propósito general -como *jogl.jar* y *gluegen-rt.jar*- y librerías nativas, es decir, que dependen de la arquitectura y del sistema operativo sobre el que se ejecuta la aplicación.

Persiguiendo el objetivo de desplegar el motor gráfico mediante algún mecanismo que abstraiga al usuario de la distribución e instalación del motor gráfico y sus dependencias, se propone una solución basada en el componente **JNLPAppletLauncher**, que se encargará de llevar a cabo dicha tarea (Fig. 5.37).

JNLPAppletLauncher [50] es un componente de propósito general basado en la tecnología JNLP², utilizado para desplegar applets que utilizan extensiones que contienen código nativo

²**JNLP (Java Network Launching Protocol)** es un protocolo en formato XML que define el procedimiento a seguir para ejecutar una aplicación Java. Este protocolo incluye información sobre el paquete JAR que se distribuye, el nombre de la clase principal de la aplicación y parámetros adicionales. Con esta información JNLP hace posible la ejecución de aplicaciones Java directamente desde Internet mediante un navegador web, sin la necesidad de instalarlas en el sistema del usuario, y asegura que el cliente siempre ejecute la última versión de la aplicación.

como Java3D, JOGL (Java binding for OpenGL -Graphics Library- API) o JOAL (Java binding for OpenAL -Audio Library- API).

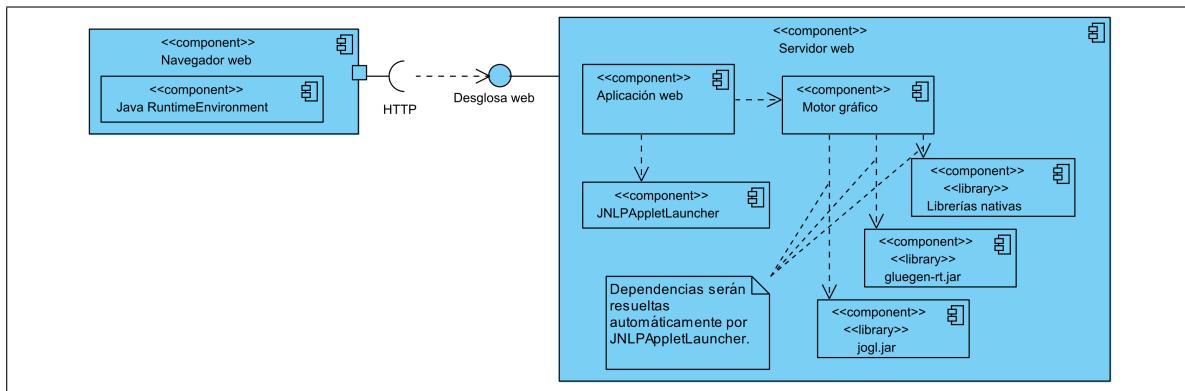


Figura 5.37: Diagrama de componentes del sistema

Para finalizar, y debido a la integración de ambos componentes mediante un applet, cabe destacar que el paso de mensajes, en ambas direcciones, se realizará mediante JavaScript.

5.3.2.3 Disciplina de implementación

Durante esta disciplina se ha llevado a cabo la implementación de los casos de uso que se había previsto. Con el fin de completar el diagrama de flujo facilitado en la figura 5.34, se adjunta una parte de su implementación (Listado 5.6).

```

1 public void display(GLAutoDrawable glDrawable) {
2     if (!viewLevel.equals(EViewLevels.UnSetLevel)) {
3         try {
4             // This state machine prevents calls to updateProjection
5             // every
6             // time display() is called
7             // Due to JOGL multi-threading issues, this is used to
8             // change to
9             // 2D or 3D projection within the main thread (not
10            // keyboard or
11            // mouse inputs)
12            // http://staff.www.ltu.se/~mjt/ComputerGraphics/jogl-doc/
13            // jogl_usersguide/index.html
14            if (!oldViewLevel.equals(viewLevel)) {
15                this.updateProjection();
16                if (!oldViewLevel.equals(EViewLevels.UnSetLevel))
17                    getViewManager(oldViewLevel).deconfigureView();
18            }
19        } catch (Exception e) {
20            System.out.println("Error in display()");
21        }
22    }
23 }
```

```
14         getViewManager(viewLevel).configureView();  
15         oldViewLevel = viewLevel;  
16     }  
17     ...  
18     ...  
19 }
```

Listado 5.6: Fragmento de código Java para la implementación de CdU1.4

5.3.2.4 Disciplina de pruebas

Las pruebas planificadas en esta iteración se han realizado nuevamente con la ayuda de un usuario experto. En este caso, el usuario debe validar la correcta configuración y visualización de las metáforas de visualización y los elementos que las conforman.

Al igual que en las pruebas de la disciplina anterior (Sección 5.3.1.4), en esta disciplina se ha vuelto a emplear el método basado en *checklists*. No obstante, en este caso el usuario ha detectado errores en la visualización a través de las transparencias de los elementos de la metáfora de la ciudad. Esto es debido a que las transparencias sólo se visualizan correctamente si los elementos son pintados en pantalla de manera ordenada desde las posiciones más lejanas hasta las más cercanas, respecto de la cámara. Además, ha hecho constar la falta de percepción de profundidad.

En conclusión, las pruebas planificadas para esta iteración **no** han sido superada. Esto supone, tal y como indica el Proceso Unificado de Desarrollo, la repetición de esta iteración antes de pasar a la siguiente.

5.3.3 Iteración 5 (repetición)

En esta iteración se resuelven los problemas detectados en la disciplina de pruebas de la iteración número 5 (Sección 5.3.2.4).

5.3.3.1 Disciplina de requisitos

El usuario experto que ha llevado a cabo las pruebas de la quinta iteración afirma que no percibe sensación de profundidad cuando visualiza la escena. Para resolver este problema se

propone la generación de sombras en tiempo real y la adición de un *skybox*³.

5.3.3.2 Disciplina de análisis

Durante esta disciplina se analiza cómo resolver dos de las cuestiones que han surgido en referencia al motor gráfico: errores de visualización en las transparencias de las torres de la metáfora de ciudad y cálculo de sombras en tiempo real. Ambos casos tienen una estrecha relación con la posición de la cámara en la escena.

En primer lugar, para que el usuario perciba las transparencias correctamente, es necesario pintar los objetos de la escena desde los más lejanos a la cámara hasta los más cercanos. Por ello, es necesario diseñar algún algoritmo que, proporcionándole la lista de elementos -con sus coordenadas en la escena- y la posición de la cámara, calcule las distancias y las ordene de forma descendente. Este algoritmo deberá ser aplicado cada vez que se ejecute cualquier tipo de movimiento sobre la cámara ya que las distancias pueden variar.

En segundo lugar, para calcular las sombras en tiempo real será necesario realizar algunos cálculos en un espacio vectorial de tres dimensiones. Estos cálculos consisten en hallar el producto escalar entre el vector que indica la posición del foco de iluminación -que usualmente está junto a la cámara- y el vector normal del plano que representa el suelo de la escena. Este vector se calcula mediante el producto vectorial de dos vectores cualesquiera, teniendo en cuenta que el plano se encuentra en $y = 0$. A partir de estos datos se calculará la matriz de proyección de las sombras, sobre la cual se volverán a pintar los elementos que conforman la escena, pero esta vez en color negro semi-transparente.

5.3.3.3 Disciplina de diseño

En esta disciplina se llevan a cabo los diseños de los algoritmos propuestos en la disciplina de análisis de esta misma iteración.

5.3.3.4 Disciplina de implementación

Durante esta disciplina se lleva cabo la implementación del pintado de sombras en tiempo real, el *skybox* que otorgará mayor sensación de profundidad y el algoritmo ordena los modelos

³Método empleado en el área de los videojuegos que se basa en la adición de imágenes de fondo para conseguir que una escena parezca más grande de lo que realmente es.

gráficos en función de la distancia con respecto de la cámara para que las transparencias sean dibujadas correctamente.

En las figuras 5.38 y 5.39 se muestran dos capturas de pantalla en las que se aprecia la evolución y el resultado de los gráficos generados.

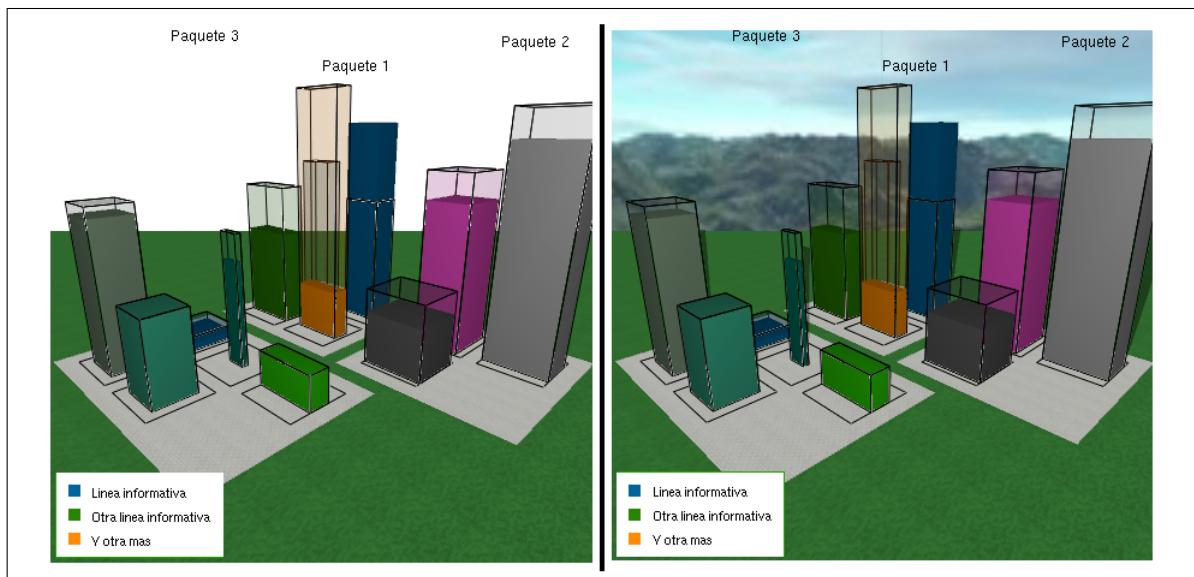


Figura 5.38: Gráficos generados después de la disciplina de implementación de la sexta iteración:
a la izquierda sin sombras ni *skybox*; a la derecha con ambas características

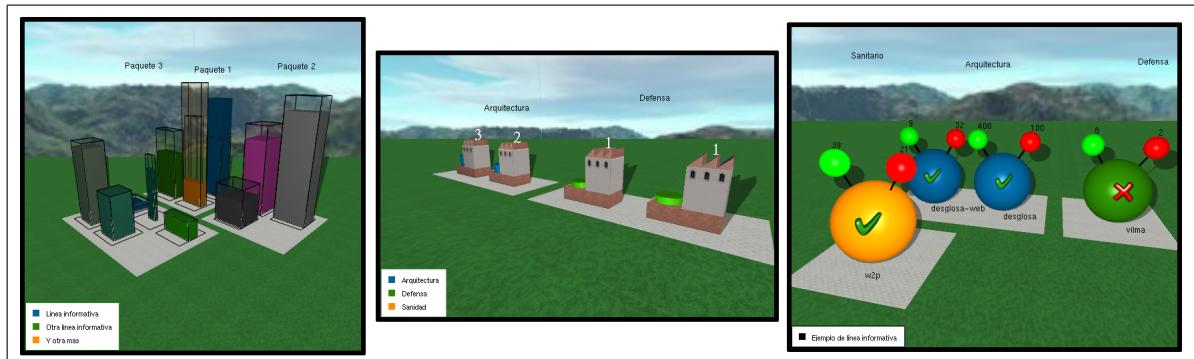


Figura 5.39: Gráficos generados después de la disciplina de implementación de la sexta iteración. De izquierda a derecha se muestran las metáforas de ciudad y polígono industrial y una representación de proyectos software

5.3.3.5 Disciplina de pruebas

En esta disciplina se han repetido los casos de prueba de la quinta iteración (Sección 5.3.2.4) y se ha obtenido un resultado positivo. Por lo tanto, se procede a abordar la siguiente iteración.

5.3.4 Iteración 6

En esta iteración (Tabla 5.14) se aborda:

- Análisis y diseño de los casos de uso relativos al control de acceso de la aplicación web.
- Implementación de las funcionalidades que permiten ejecutar la herramienta de visualización, notificar qué objeto 3D ha sido seleccionado en el motor gráfico y ver el mapa mundial.
- Pruebas de la integración del motor gráfico en la aplicación web y mecanismos de selección de objetos 3D.

5.3.4.1 Disciplina de requisitos

Antes de abordar el desarrollo del sistema de control de acceso, se propone al usuario final la posibilidad de dos alternativas de **política de acceso**, que define qué usuarios o sistemas pueden acceder a determinados recursos o información. La primera de ellas consiste en un **control de acceso basado en roles**; la segunda, en un **control de acceso basado en grupos**.

- Control de Acceso basado en **roles** (en inglés, *Role-based access control* o RBAC).
Consiste en una política de seguridad en la que el sistema otorga privilegios al usuario en función de su rol. Un usuario puede tener uno o varios roles.
- Control de Acceso basado en **grupos** (en inglés, *Group-based access control* o GBAC).
Este modelo extiende al RBAC y ofrece más flexibilidad para modelar políticas de acceso más complejas y exigentes. En este caso, los privilegios se otorgan al usuario en función de los grupos a los que pertenezca. Un usuario puede pertenecer a uno o varios grupos, y un grupo puede tener asignados uno o varios roles.

El usuario final, bajo la recomendación de un experto, ha seleccionado la segunda opción dada la gran flexibilidad y opciones de extensibilidad y mantenibilidad que proporciona en cuanto a la adición de futuras políticas de acceso.

5.3.4.2 Disciplina de análisis

Las funcionalidades relativas a autenticación, seguridad y control de acceso son muy comunes en aplicaciones web. Gracias a ellas es posible establecer políticas que indiquen quiénes

tienen la capacidad de acceder a un conjunto determinado de recursos o información. Por tanto, para no reinventar la rueda en cada ocasión que es necesario implementar características de esta índole, se recomienda hacer uso de frameworks especialmente diseñados para estos propósitos.

Spring Security [58] es el estándar *de-facto* para la seguridad de aplicaciones basadas en Spring. Consiste en framework potente y configurable mediante ficheros XML, que proporciona métodos de autenticación y control de acceso en aplicaciones basadas en Spring. Su publicación bajo una licencia Apache 2.0 y su uso extendido por agencias gubernamentales, aplicaciones militares y bancos, hacen de él una alternativa muy competente y confiable.

En la figura 5.40 se muestra un pequeño sub-diagrama de casos de uso. En él se observa que el usuario puede realizar las operaciones de iniciar sesión o finalizar sesión. Ambas operaciones realizarán una configuración de los menús que se muestran al usuario en la interfaz gráfica. Esta configuración se delega en la librería *struts-menu*, por lo que a este nivel no es necesario especificarla con más detalle.

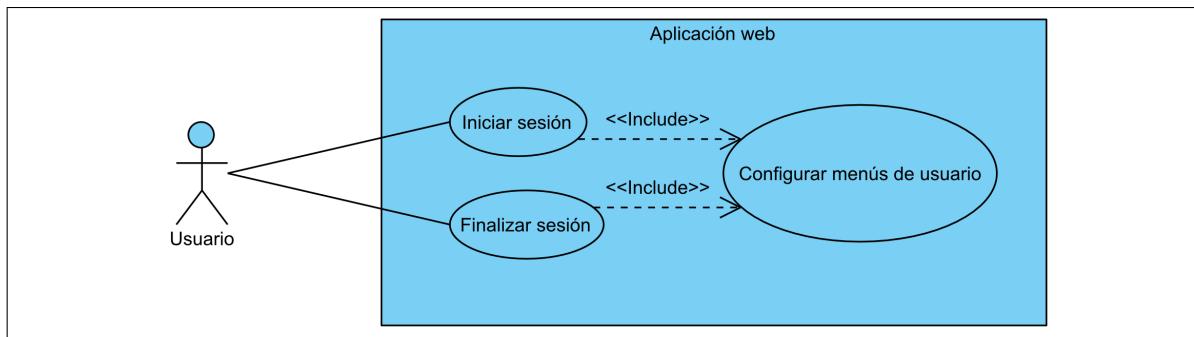


Figura 5.40: Diagrama de casos de uso para CdU2.1 (Iniciar sesión)

Llegados a este punto, se procede a la descripción de los casos de uso que se están tratando. Para no extender demasiado el capítulo, sólo se incluye la descripción de la funcionalidad de *inicio de sesión*, tanto en el escenario general como algunos de sus escenarios alternativos (Tabla 5.22).

A continuación, se elaboran los diagramas asociados a la disciplina de análisis. En este documento se proporcionan aquellos relacionados con la funcionalidad de *iniciar sesión* (Fig. 5.41, 5.42 y 5.43). Nótese como en el paso de uno a otro se va perdiendo abstracción y se va obteniendo un mayor grado de detalle, estando cada vez más cerca de la disciplina de diseño.

Para poder continuar con el diseño e implementación del control de acceso mediante este framework, es necesario haber definido la política de acceso (Sección 5.3.4.1) y cual será el método de conexión con la base de datos. En este caso, ya se definieron las tecnologías que se

Identificador: CdU2.1
Nombre: Acceder al sistema (Iniciar sesión)
Descripción: Funcionalidad para que un usuario pueda autenticarse y acceder al sistema
Precondiciones: El usuario debe conocer el nombre de usuario y la contraseña, y estar dado de alta en el sistema.
Post-condiciones: El usuario accede al sistema.
<p>Escenario general</p> <ol style="list-style-type: none"> 1. El usuario introduce usuario y contraseña. 2. Se autentica al usuario. 3. Se crea una sesión para el usuario. 4. Se informa al usuario del éxito de la operación. 5. Se redirige al usuario a la página principal.
<p>Escenario alternativo 1: usuario o contraseña incorrectos</p> <ol style="list-style-type: none"> 1. El usuario introduce usuario y contraseña. 2. Se produce un error de autenticación. 3. Se informa al usuario que se ha producido un error.
<p>Escenario alternativo 2: usuario inexistente</p> <ol style="list-style-type: none"> 1. (Ver flujo alternativo 1).
<p>Escenario alternativo 3: cuenta de usuario bloqueada, caducada o deshabilitada</p> <ol style="list-style-type: none"> 1. El usuario introduce usuario y contraseña. 2. Se autentica al usuario. 3. La cuenta de usuario está bloqueada, caducada o deshabilitada. 4. Se informa al usuario que se ha producido un error.

Tabla 5.22: Descripción del caso de uso CdU2.1 (Iniciar sesión)

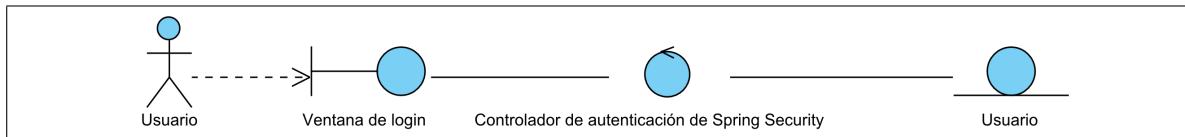


Figura 5.41: Diagrama clases de análisis para CdU2.1 (Iniciar sesión)

emplearían en el diseño y desarrollo web en la iteración 3, y se llegó a la conclusión de que la conexión con la base de datos se realizaría mediante Hibernate.

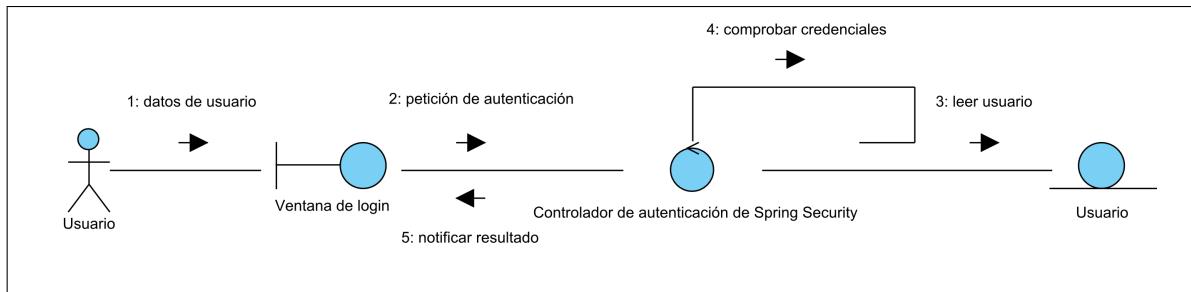


Figura 5.42: Diagrama de comunicación para CdU2.1 (Iniciar sesión)

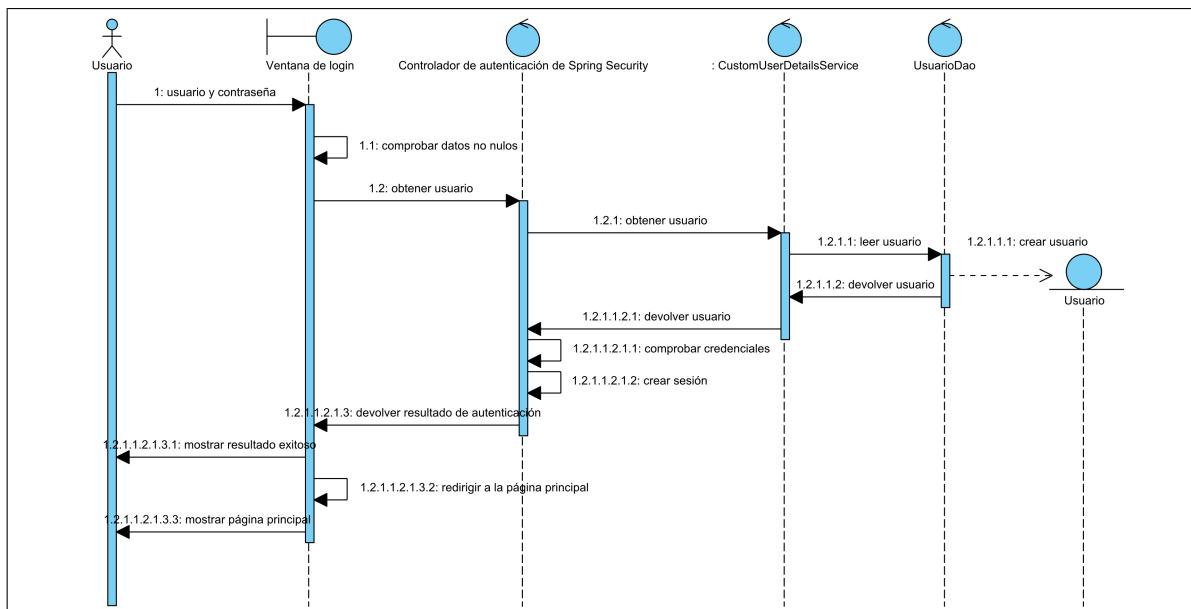


Figura 5.43: Diagrama de secuencia para CdU2.1 (Iniciar sesión)

5.3.4.3 Disciplina de diseño

En la figura 5.44 se muestra un diagrama de clases que representa las clases e interfaces que intervienen en la implementación de un GBAC mediante Spring, Spring Security e Hibernate. Los espacios de nombre relativos a dichas tecnologías se representan en la parte superior de la imagen como paquetes y subpaquetes UML, donde se han representado las interfaces y clases relevantes para la resolución de este problema. En la parte inferior de la imagen se representa el conocimiento que es necesario implementar en la aplicación. De todas las clases representadas, únicamente las clases *User*, *Group* y *Role* serán persistentes.

- **Clase User.** Implementa el interfaz *UserDetails* de Spring Security. Este interfaz obliga que la clase *User* implemente los métodos que define y, por tanto, se añadan sus respectivos atributos (*username*, *password*, *enabled*, *accountNonExpired*, *credentialsExpired* y *accountNonLocked*). Además, presenta una asociación unidireccional múltiple con

la clase *Group* que será implementada mediante un *Set* de Java. Por último, se mapea mediante JPA (Java Persistence API) con la tabla *users* del esquema de bases de datos descrito en (Fig. 5.45). Cabe destacar la implementación del método *getAuthorities* que devuelve todos los roles, encapsulados en un array de tipo *GrantedAuthority* -definido por Spring Security-, de cada uno de los grupos a los que pertenece el usuario.

- **Clase Group.** De un modo similar a la clase *User*, la clase *Group* presenta una asociación unidireccional múltiple con la clase *Role*, que será implementada mediante un *Set* de Java. El mapeo ORM (Object-Relational Mapping) se realizará mediante anotaciones JPA con la tabla *groups*. Una vez más, en este caso cabe destacar la implementación del método *getAuthorities*.
- **Clase Role.** En este caso no se presenta ningún tipo de asociación, por lo que en esta clase sólo será necesario implementar el *interface* *GrantedAuthority* de Spring Security y realizar su correspondiente mapeo con la tabla *roles* mediante JPA.
- **Clase CustomUserDetailsService.** Esta clase implementa el *interface* *UserDetailsService* y su responsabilidad se basa en establecer el mecanismo de búsqueda y materialización de los datos del usuario. Este proceso se realiza mediante un DAO (Data Access Object) que hereda las funcionalidades proporcionadas por Spring mediante la clase **HibernateDaoSupport**.

Para finalizar con la disciplina de diseño, en la figura 5.45 se aprecia un esquema básico de la base de datos que es necesario diseñar para implementar un GBAC mediante Spring Security. Tal y como se ha indicado, únicamente serán persistentes los usuarios, los grupos y los roles (y las relaciones muchos-a-muchos que los asocian). Si se desea añadir información adicional, es posible añadir nuevas columnas a las tablas del esquema propuesto.

- **Tabla users.** Contiene la información relativa a los usuarios de la aplicación web. Spring Security exige que los usuarios tengan un nombre de usuario, una contraseña y una serie de atributos que indiquen si la cuenta está habilitada, caducada, bloqueada o si las credenciales han caducado. Esto es debido a que la clase Java perteneciente a la entidad *Usuario* deberá implementar un interfaz de Spring denominado *UserDetails* (Fig. 5.44).

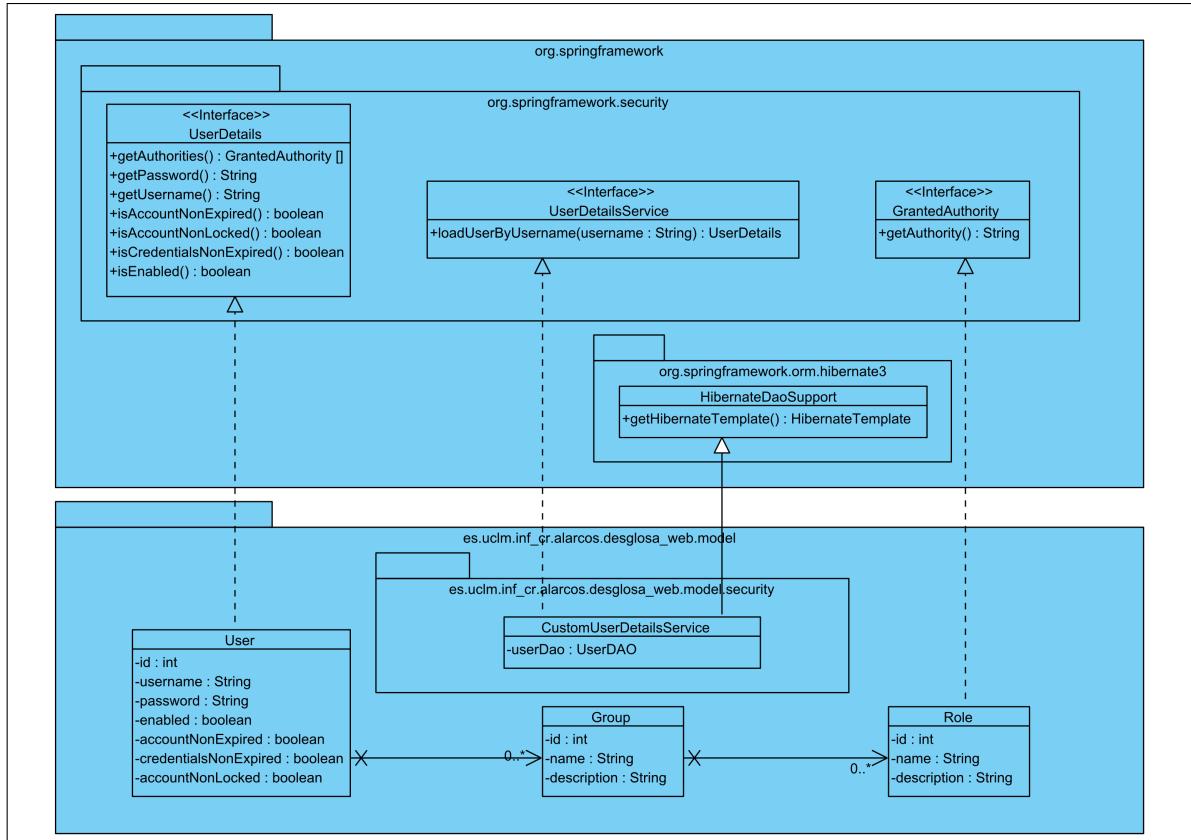


Figura 5.44: Diagrama de clases de diseño para CdU2.1 (GBAC mediante Spring Security)

- **Tabla groups.** Contiene el nombre y la descripción de los grupos de usuarios que se desean crear. Se establecerá una relación *muchos-a-muchos* entre los usuarios y los grupos gracias a la tabla *users_groups*.
- **Tabla roles.** Contiene el nombre y la descripción de los roles disponibles en el sistema. En Spring Security, por defecto aunque configurable, el nombre de los roles debe comenzar con el prefijo “*ROLE_*”. Se establecerá una relación *muchos-a-muchos* entre los grupos y los roles gracias a la tabla *groups_roles*.

5.3.4.4 Disciplina de implementación

En esta parte de la disciplina se han implementado aquellos casos de uso que estaban planificados. De aquí en adelante, la aplicación web proporciona las siguientes funcionalidades:

- Muestra un mapa mundial basado en Google Maps y permite que el usuario navegue a través de él (Fig. 5.46).
- Integra el motor gráfico mediante JNLPAppletLauncher y permite que el usuario ejecute la última versión del motor gráfico sin la necesidad de realizar procesos de

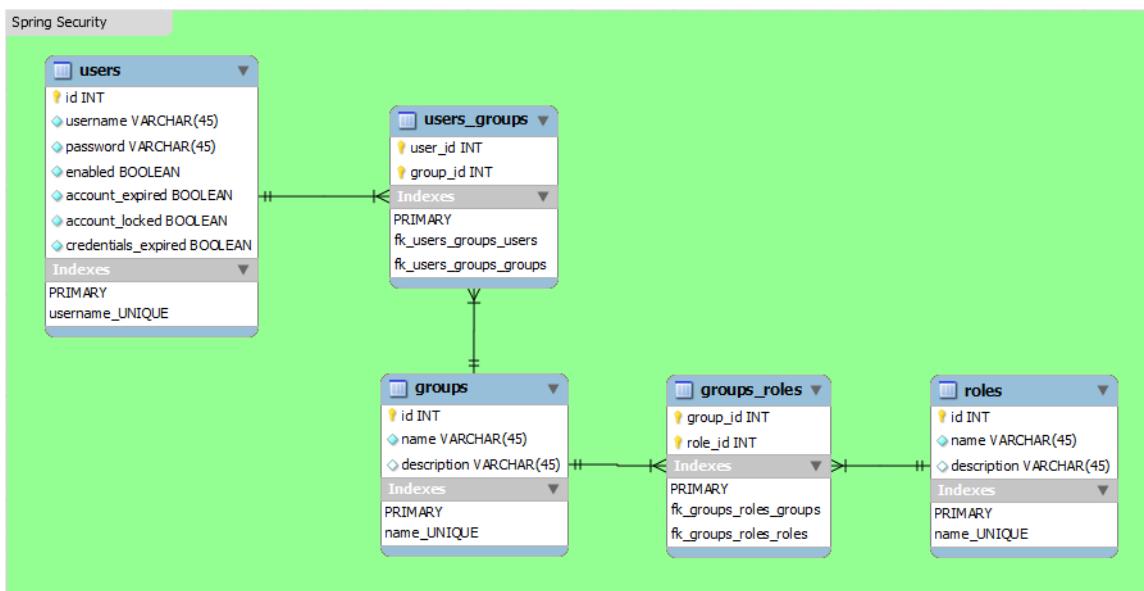


Figura 5.45: Modelo Entidad-Relación Extendido para CdU2.1 (GBAC mediante Spring Security)

instalación o configuración complejos. La descarga de extensiones y librerías nativas se realiza automáticamente gracias a un fichero de configuración *.jnlp* que indica dónde se encuentran disponibles los distintos recursos nativos.

- El usuario puede seleccionar los distintos modelos que conforman las metáforas que genera el motor gráfico y obtiene un mensaje en el cual se indica el identificador del elemento seleccionado.

5.3.4.5 Disciplina de pruebas

Dada la naturaleza de los casos de uso a probar, esta disciplina de pruebas se ha llevado a cabo con la ayuda de un usuario experto y el método basado en *checklists*. En ella se ha comprobado que el usuario es capaz de seleccionar los distintos modelos de una escena generada mediante el motor gráfico, y que obtiene sus correspondientes identificadores.

Además, se ha comprobado que el applet en el que se distribuye el motor gráfico carga correctamente en los siguientes navegadores: Google Chrome 15.0 o superior, Mozilla Firefox 4.0 o superior, Internet Explorer 7 o superior.

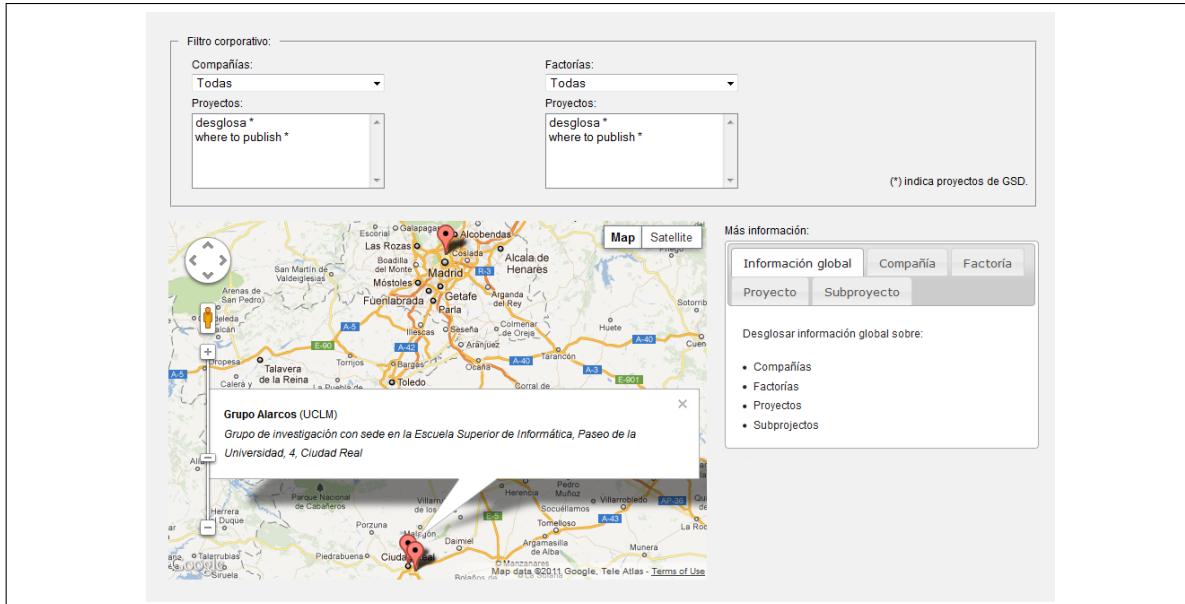


Figura 5.46: Captura de pantalla de la integración de Google Maps en la aplicación web

5.3.5 Iteración 7

En esta iteración (Tabla 5.15) se aborda:

- Análisis y diseño de gestión y selección de perfiles de visualización.
- Implementación del control de acceso a la aplicación web mediante Spring Security.
- Configuración del entorno de ejecución de pruebas integrado en Maven.
- Diseño e implementación de pruebas unitarias y funcionales relativas al control de acceso, ejecución de la herramienta de visualización, notificación de selección y visualización del mapa mundial.

5.3.5.1 Disciplina de análisis

El caso de uso CdU2.2 es una funcionalidad con cierta complejidad e importancia en la herramienta. Por esta razón se ha elaborado un diagrama de casos de uso más detallado (Fig. 5.47). Además, se proporciona la descripción del escenario general para facilitar el estudio del diagrama (Tabla 5.23).

El diagrama de casos de uso y la descripción de sus escenarios no proporcionan una vista detallada de la funcionalidad, sino que muestran la interacción entre el usuario y el sistema. Para comenzar a obtener vistas más específicas, se elabora el diagrama de comunicación

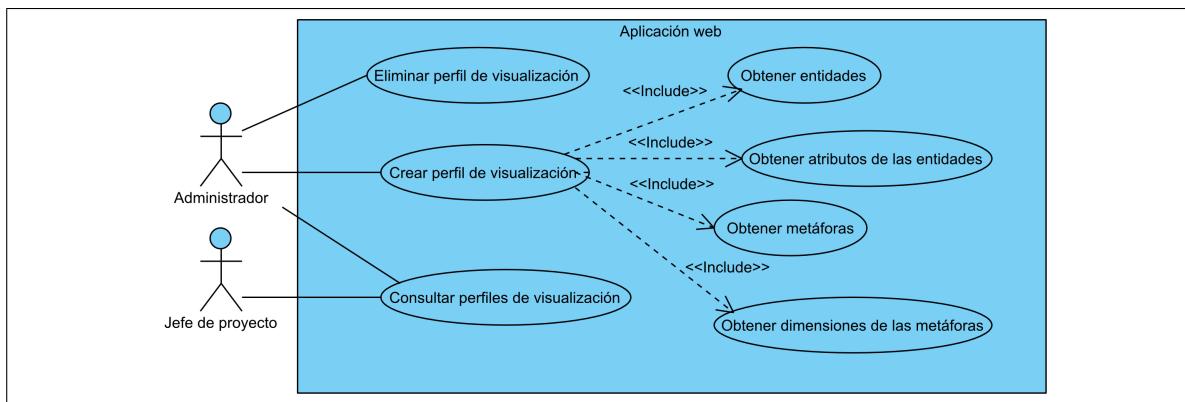


Figura 5.47: Diagrama de casos de uso para CdU2.2

Identificador: CdU2.2
Nombre: Gestión de perfiles (Crear perfiles)
Descripción: Funcionalidad para que el administrador del sistema pueda configurar y crear perfiles de visualización.
Precondiciones: El usuario debe haber iniciado sesión en el sistema como <i>administrador</i> .
Post-condiciones: Se crea un perfil de visualización en formato XML.
Escenario general <ol style="list-style-type: none"> 1. El sistema muestra al usuario una lista de entidades y una lista de modelos gráficos. 2. El usuario selecciona una entidad. 3. El sistema muestra los atributos mapeables de la entidad. 4. El usuario selecciona un modelo gráficos. 5. El sistema muestra las dimensiones mapeables de la metáfora. 6. El usuario configura los mapeos entre los atributos de la entidad y las dimensiones del modelo, y datos adicionales del perfil. 7. El sistema guarda el perfil en formato XML.

Tabla 5.23: Descripción del caso de uso CdU2.2 (Crear perfil de visualización)

a partir de ambos modelos. Este diagrama ofrece una vista más detallada en cuanto a los controladores y entidades que intervienen en la ejecución de la operación.

En la figura 5.48 se muestra el diagrama de comunicación del caso de uso CdU2.2. En él observa que la clase de control *ProfileAction* actúa de fachada entre la interfaz gráfica de usuario y el resto de controladores y entidades. Esta estructura se da en todos los diagramas de comunicación de la aplicación web porque la fachada es una acción de Struts 2 y este es el fundamento del patrón MVC que implementa dicho framework.

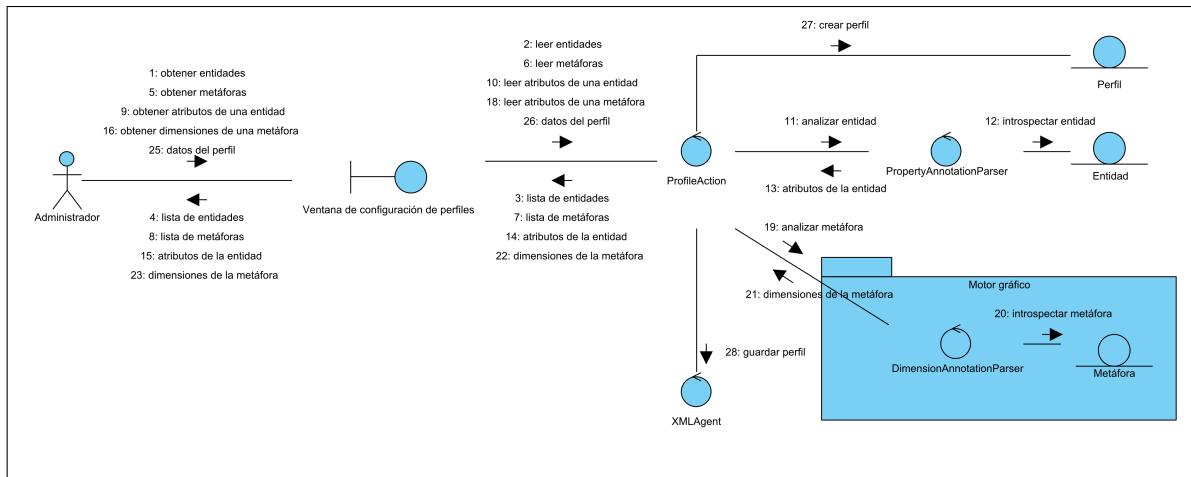


Figura 5.48: Diagrama de comunicación para CdU2.2 (Crear perfil de visualización)

También destaca la interacción con las clases de un paquete denominado *motor gráfico*. Esto sucede porque la aplicación web, que actúa como contenedora del motor gráfico, debe tener cierto conocimiento sobre el mismo y, como consecuencia, dependencia. En este caso, la dependencia se encuentra sobre la clase *DimensionAnnotationParser*, que proporcionará a la aplicación web la última versión de las dimensiones de los modelos gráficos.

Con el estudio de análisis que se ha realizado durante esta disciplina, se modela el primer prototipo de la interfaz gráfica de usuario (Fig. 5.49). Esta interfaz se ha diseñado siguiendo la idea del típico *asistente de instalación* que permite aportar los datos de configuración paso a paso.

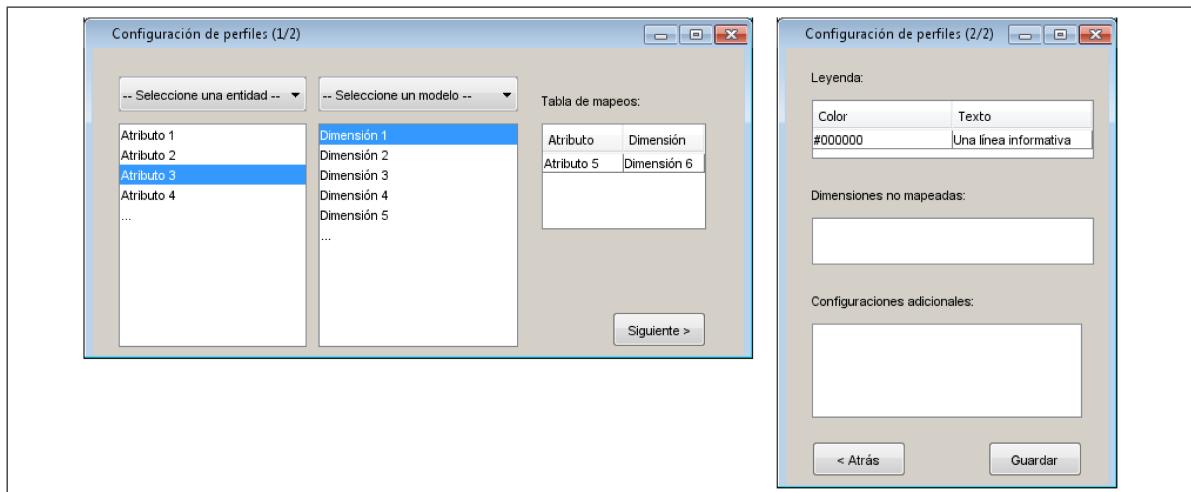


Figura 5.49: Prototipo de la interfaz gráfica para CdU2.2 (Crear perfil de visualización)

5.3.5.2 Disciplina de diseño

Se debe diseñar una arquitectura adecuada para que la aplicación tenga una organización estructural adecuada que haga posible la reutilización de código, la aplicación de patrones estructurales y de diseño, y reduzca el acoplamiento entre capas y clases.

En la figura 5.50 se muestra un diagrama de paquetes que cumple con las expectativas mencionadas. A grandes rasgos se observan tres paquetes -presentación, dominio y persistencia- que se corresponden con las tres capas del **patrón MVC** (Modelo-Vista-Controlador). Dentro de cada uno de estos grandes paquetes se han modelado distintos subpaquetes que agrupan ficheros y clases de características similares. Por ejemplo, en el paquete *jsp* se incluirán todas las páginas *jsp*, en el paquete *js* contendrá las librerías de JavaScript, etcétera.

De entre todos los subpaquetes destacan los siguientes:

- **dao**, que a su vez contiene al subpaquete *hibernate*. Esto indica que existirá una especificación genérica de la estructura que debe tener un fichero que implemente el **patrón DAO** (Data Access Object) y, además, existirá una implementación específica basada en Hibernate.
- **applet**, que presenta una asociación especial con el paquete *Desglosa Graphics Engine*. Esta asociación indica que el paquete *applet* contiene el componente del motor gráfico.
- **actions**, que contiene las acciones de Struts 2 que satisfarán los requisitos funcionales del sistema. Nótese la dependencia de este paquete con los paquetes de control, modelo, utilidades, etcétera.
- **interceptor**, que contiene interceptores que se implementarán para extender la funcionalidad de Struts 2 a las necesidades del sistema.
- **control**, en el que se incluirán todas las clases de utilidades que permiten operar con las clases de dominio.
- **model**, en el que se incluirán las clases de dominio.

Durante esta disciplina también se ha abordado el diseño de cuestiones más específicas, como la estructura de los perfiles de visualización. Para comenzar, se ha estudiado el tipo de información que se desea hacer persistente y la estructura del fichero XML que se escribirá

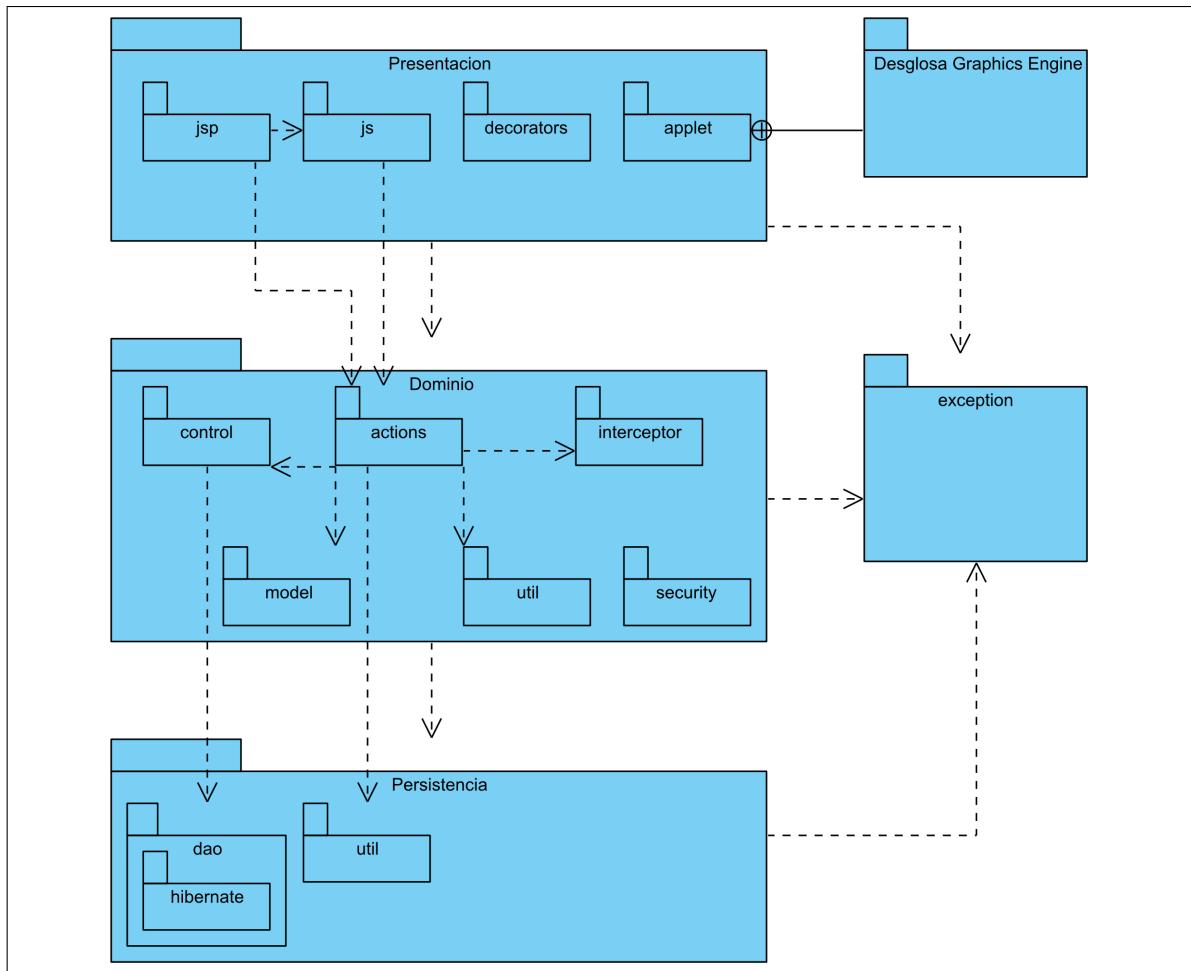


Figura 5.50: Diagrama de paquetes de la aplicación web

cuando se cree un perfil. A continuación se detallan los nodos y atributos que conforman dicha estructura:

- **profile:** es el nodo raíz de la estructura y contiene dos atributos: *model*, que referencia al modelo gráfico, y *entity*, que referencia a la entidad que se quiere visualizar.
- **name:** nombre del perfil.
- **description:** cadena de texto que describe el tipo de información que podrá visualizar el usuario por medio de este perfil.
- **mappings:** lista de asociaciones que contiene elementos de tipo *mapping*.
- **mapping:** elemento que contiene la asociación entre un atributo de la entidad -*entityAttr*- , una dimensión del modelo -*modelAttr*-, y una lista de reglas -*rule*- si procede.
- **constants:** lista de nodos *constant* que contiene la configuración de las dimensiones

del modelo que no se han asociado con ningún atributo de la entidad. El valor constante de una dimensión es tomado como su valor por defecto.

- **caption:** lista de nodos *entry* que configura la leyenda informativa que será mostrada en la escena.

```

1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <profile model="model.gl.knowledge.GLFactory" entity="es.uclm.
   inf_cr.alarcos.desglosa_web.model.Factory">
3    <name>Profile snippet</name>
4    <description>snippet of a visualization profile</description>
5    <mappings>
6      <mapping>
7        <entityAttr name="id" type="int"/>
8        <modelAttr name="id" type="int"/>
9      </mapping>
10     <mapping>
11       <entityAttr name="employees" type="int"/>
12       <modelAttr name="scale" type="float_range"/>
13       <rule>
14         <low xsi:type="xs:int">0</low>
15         <high xsi:type="xs:int">20</high>
16         <value xsi:type="xs:float">0.0</value>
17       </rule>
18     </mapping>
19   </mappings>
20   <constants>
21     <constant name="color" type="color">
22       <value xmlns:xsi="http://www.w3.org/2001/XMLSchema-
          instance" xmlns:xs="http://www.w3.org/2001/
          XMLSchema" xsi:type="xs:string">ffffff</value>
23     </constant>
24   </constants>
25   <caption>
26     <entry color="00649a" label="Linea informativa"/>
27   </caption>
28 </profile>
```

Listado 5.7: Ejemplo de un perfil de visualización en formato XML

En el listado 5.7 se propone un ejemplo que cumple con la estructura expuesta anteriormente. Para poder obtener un fichero de este tipo mediante la tecnología JAXB, es necesario diseñar una jerarquía de clases que represente dicha estructura (Fig 5.51). Acto seguido, es necesario anotar debidamente los atributos de las clases mediante anotaciones de JAXB para que sea posible su *serialización* a formato XML y su posterior escritura en un fichero..

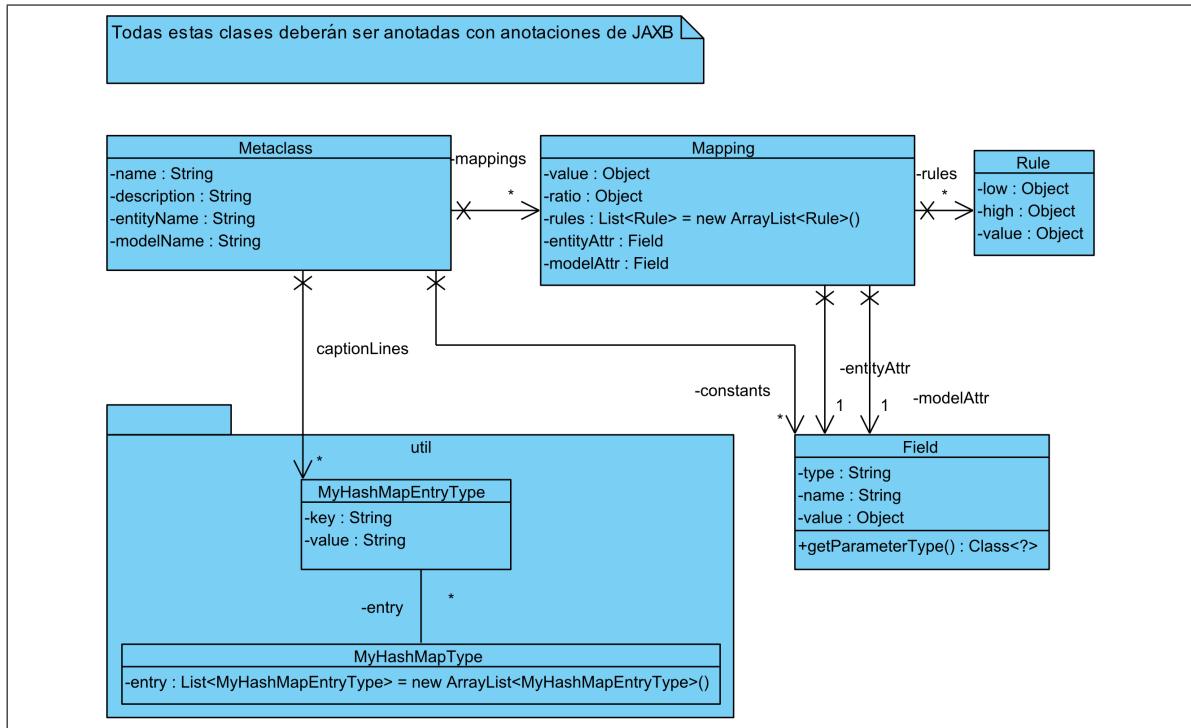


Figura 5.51: Diagrama de clases de dominio para CdU2.2

Para finalizar, cabe destacar que cuando se desee configurar una escena a partir de un perfil de visualización, primero habrá que *deserializarlo*, es decir, instanciar la jerarquía de objetos definida por la estructura de clases de la figura 5.51. Acto seguido, habrá que operar con dichos objetos e instanciar los modelos gráficos por medio del método de **reflexión**. Por último, se construirá una cadena con formato JSON que será proporcionada al motor gráfico como dato de entrada para que éste genere los gráficos.

5.3.5.3 Disciplina de implementación

Para llevar a cabo la implementación del control de acceso, es necesario configurar Spring Security mediante un fichero de configuración. Se trata de un fichero XML que por convenio suele llamarse *applicationContext-security.xml* (Véase Anexo D.1). Dada la gran cantidad de parámetros que se pueden incluir en este fichero, se recomienda consultar su manual de referencia para profundizar en el tema y realizar una configuración precisa [59]. En él destaca

el bean *userDetailsService*, que hace referencia a nuestra clase *CustomUserDetailsService*. Además, es referenciado como *proveedor de autenticación* en el bean *authentication-provider*, donde se especifica la codificación de las contraseñas de usuario (md5 en este caso).

A continuación se implementan las clases correspondientes según el diseño elaborado en 5.3.4.3 y el formulario de *login* (Listado 5.8). Éste será un formulario tradicional en el que el usuario pueda introducir su nombre y su contraseña pero, en este caso, para delegar el proceso de autenticación a Spring Security, el *action* al que se enviarán los datos será *j_spring_security_check*. Si se produce algún error durante la autenticación, el mensaje de error será lanzado en forma de excepción y su valor será accesible por medio de la variable *SPRING_SECURITY_LAST_EXCEPTION.message*.

```
1 <c:if test="${param.result == 'failed'}">
2     <div id="errorMessage" class='messageBox error' style="float
3         :left;"><c:out value="${SPRING_SECURITY_LAST_EXCEPTION.
4             message}" />.</div>
5
6 <form class="form" method="post" id="loginForm" action="
```

```
22      </li>
23  </ul>
24
25  <s:submit id="submit" name="login" value="#{getText(labelLogin)}" tabindex="3"></s:submit>
26  </fieldset>
27 </form>
```

Listado 5.8: Código fuente del fichero login.jsp para autenticación mediante Spring Security

Por otro lado, los menús que se muestran al usuario en la interfaz gráfica son renderizados de forma automática en función de los privilegios que posee. Esta característica se ha implementado mediante la librería *struts-menu*, cuya configuración se lleva a cabo con un fichero XML (Listado 5.9). En este fichero se especifican los distintos menús y los elementos que los componen. Además, en el atributo *roles* se indica qué roles debe tener el usuario autenticado para que dicho menú sea renderizado y mostrado. Cabe destacar que este framework es completamente compatible con Spring Security y, por tanto, los roles asignados al usuario mediante el sistema de autenticación serán los que se empleen para determinar qué menús serán mostrados al usuario.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <MenuConfig>
3   <Displayers>
4     <Displayer name="Velocity" type="net.sf.navigator.
5       displayer.VelocityMenuDisplayer" />
6     <Displayer name="Simple" type="net.sf.navigator.
7       displayer.SimpleMenuDisplayer" />
8   </Displayers>
9   <Menus>
10    <Menu name="Home" title="menu.home" description="Home Menu
11      " page="/" />
12    <Menu name="AdminMenu" title="menu.admin" description="
13      Admin Menu" roles="ROLE_ADMIN">
14      <Item name="ManageCompanies" title="menu.manage.
15        companies" page="/listCompanies" />
16      <Item name="ManageFactories" title="menu.manage.
17        factories" page="/listFactories" />
18      <Item name="ManageProfiles" title="menu.manage.
```

```

    profiles" page="/listProfiles" />
13   </Menu>
14
15   <Menu name="ManagerMenu" title="menu.manager" description=
16     "Manager Menu" roles="ROLE_ADMIN,ROLE_MANAGER">
17     <Item name="ManageProjects" title="menu.manage.projects" 
18       page="/listProjects" />
19     <Item name="ManageSubprojects" title="menu.manage.
20       subprojects" page="/listSubprojects" />
21   </Menu>
22
23   <Menu name="Visualization" title="menu.visualization"
24     description="Visualization Menu" page="/visualization"
25     roles="ROLE_ADMIN,ROLE_MANAGER,ROLE_USER" />
26
27   <Menu name="Logout" title="menu.logout" description="
28     Logout" page="/logout" roles="ROLE_ADMIN,ROLE_MANAGER,
29     ROLE_USER" />
30
31   <Menu name="Login" title="menu.login" page="/login" />
32
33   <Menu name="Contact" title="menu.contact" description="
34     Contact Menu" page="/contactUs" />
35
36 </Menus>
37
38 </MenuConfig>

```

Listado 5.9: Fichero de configuración de struts-menu (menu-config.xml)

5.3.5.4 Disciplina de pruebas

A partir de esta iteración es cuando se comienza a abordar las pruebas de la aplicación web. Así pues, el primer paso consiste en diseñar una arquitectura de pruebas y configurar Maven para poder implementarlas, ejecutarlas y obtener los informes correspondientes.

La arquitectura de pruebas debe permitir implementar, por un lado, las pruebas unitarias de las acciones de struts y, por otro, las pruebas relativas al resto de clases del sistema: controladores de la lógica de negocio, clases DAO, clases de dominio y clases de utilidades. Es necesario diseñar la arquitectura de este modo ya que las pruebas de las acciones de struts requieren de ciertas funcionalidades, como el acceso a los objetos *HttpServletRequest* y *HttpServletResponse*, que no son necesarias en otros tipos de pruebas.

En la figura 5.52 se muestra el diagrama de clases de la arquitectura de pruebas propuesta. En la parte superior del diagrama se muestran las librerías de terceros que serán utilizadas. En la parte central, enmarcado por el paquete *test.util*, se muestran las clases que deberán

ser implementadas para obtener los servicios requeridos de dichas librerías. Y en la parte inferior se muestran las clases que se corresponden con los *test cases* de acciones, dominio, conocimiento y DAO que contienen los casos de prueba a implementar.

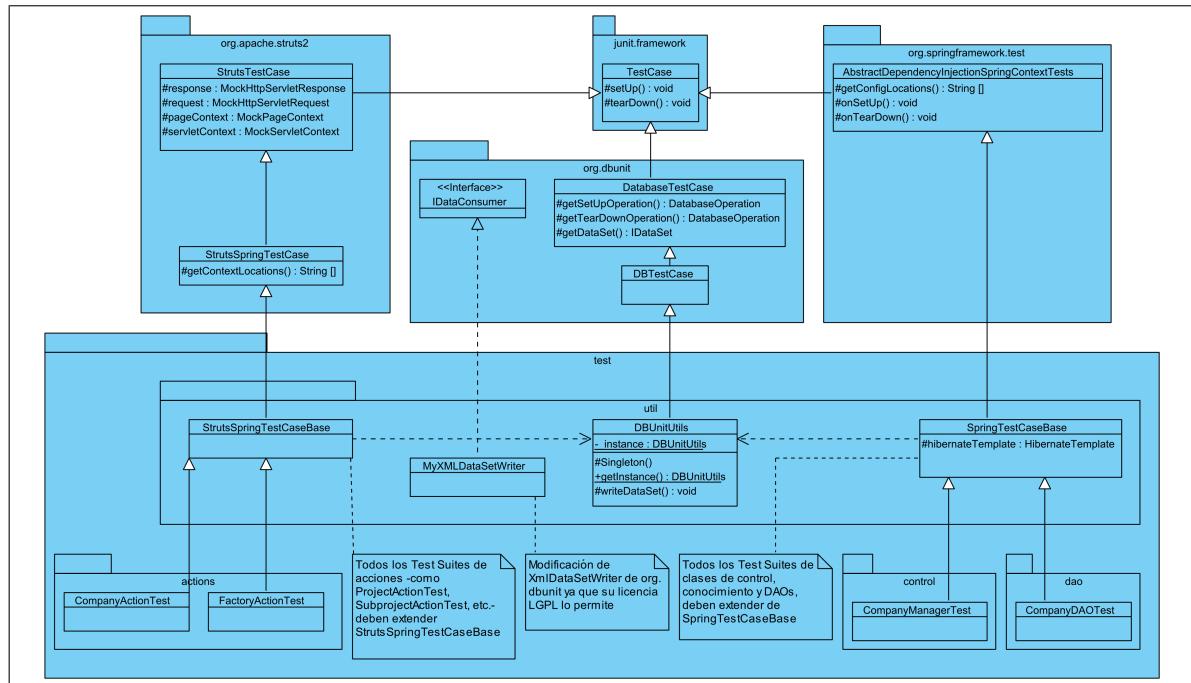


Figura 5.52: Diagrama de clases de la arquitectura de pruebas unitarias

A continuación, se realizará la configuración para las pruebas unitarias y, posteriormente, la configuración para las pruebas funcionales.

Configuración de Maven para la ejecución de pruebas unitarias

Para llevar a cabo la implementación de pruebas es necesario editar el fichero de configuración de Maven y añadir como dependencia del proyecto la librería *JUnit* (Listado 5.10).

```

1 <dependency>
2   <groupId>junit</groupId>
3   <artifactId>junit</artifactId>
4   <version>4.8.2</version>
5   <scope>test</scope>
6 </dependency>
```

Listado 5.10: Configuración de pom.xml para añadir JUnit como dependencia del proyecto

A continuación y siguiendo el mismo proceso, se añadirán al proyecto el resto de dependencias reflejadas en el diagrama de clases de la arquitectura de pruebas (Fig. 5.52):

- **spring-test.** Gracias a esta librería, la implementación de pruebas se beneficiará de las ventajas que aporta *Spring Framework*, tales como la *inyección de dependencias* y el acceso a los *beans* creados por el **Contenedor de Inversión de Control** (IoC, en su acrónimo inglés Inversion of Control). Por tanto, de aquí en adelante, cada vez que sea necesario implementar un *test case*, éste debería heredar de la clase *AbstractDependencyInjectionSpringContextTests* y sobrescribir el método *getConfigLocations*, el cual devolverá un vector de cadenas en la que cada elemento se corresponde con la ruta los ficheros de configuración de Spring. No obstante, para no duplicar código y fomentar la reutilización del mismo, se propone crear una capa intermedia mediante la clase **SpringTestCaseBase** que adoptará las características mencionadas. Así pues, todos los *test cases* que se implementen serán especializaciones de esta clase. De este modo, el programador podrá acceder al objeto *applicationContext* desde todos y cada uno de los *test cases* y recuperar cualquier bean creado mediante el contenedor IoC de Spring.
- **struts2-junit-plugin.** Esta librería permite que los *test cases* de las acciones de struts puedan acceder a un conjunto de *mocks* o *dummies* que simulan la funcionalidad de los objetos *HttpServletRequest*, *HttpServletResponse*, *PageContext* y *ServletContext*, utilizados en servlets y páginas JSP. De igual modo que en el caso anterior, se incorporará una capa intermedia mediante la clase **StrutsSpringTestCaseBase** que heredará de *StrutsSpringTestCase* para sobrescribir el método que establece la configuración de la librería. Así pues, todos los *test cases* que se implementen serán especializaciones de esta clase. De este modo, el programador podrá acceder a los objetos mencionados desde todos y cada uno de los *test cases*.
- **dbunit.** Esta librería permite establecer el estado de una base de datos a partir de un fichero XML y viceversa, es decir, generar un fichero XML a partir de los datos de una base de datos. De este modo, mediante la clase **DBUnitUtils**, es posible reestablecer el estado de la base de datos siempre que sea necesario.

Cabe destacar las dependencias de las clases *StrutsSpringTestCaseBase* y *SpringTestCaseBase* con la clase *DBUnitUtils*. Estas dependencias se deben a que ambas clases, que actúan como capas intermedias que abstraen al programador de ciertas tareas, restablecen el

estado de la base de datos antes de ejecutar cada *test case*. Esto permite que el programador de los casos de prueba siempre tenga conciencia de los datos que va a manejar y no se produzcan situaciones de error inesperadas. Por lo tanto, el programador no debe implementar los métodos *setUp* y *tearDown* que usualmente se encargan de inicializar y eliminar los datos de la base de datos con los que operaran los casos de prueba.

Una vez que se han implementado los *test cases*, Maven, mediante el plugin de JUnit, construirá los *test suites* y los ejecutará automáticamente durante el proceso de construcción del proyecto. Si algún test produce un error, el proceso de construcción será cancelado. El informe de pruebas, que permite conocer diversas estadísticas y el resultado de ejecución de las mismas, se genera mediante un plugin de Maven denominado *maven-surefire-report-plugin*.

Otro aspecto importante de la etapa de pruebas consiste en analizar cuánto código ha sido probado mediante la implementación de las mismas. Para ello, es posible generar informes de cobertura de código mediante el plugin *cobertura-maven-plugin*. Este plugin no requiere configuración y deberá ser añadido en la sección *reporting* del fichero pom.xml (Listado 5.11).

```
1 <reporting>
2 ...
3   <plugins>
4     <plugin>
5       <groupId>org.codehaus.mojo</groupId>
6       <artifactId>cobertura-maven-plugin</artifactId>
7       <version>2.5.1</version>
8     </plugin>
9   </plugins>
10 </reporting>
```

Listado 5.11: Configuración de pom.xml para añadir Cobertura como plugin de generación de informes

Los informes de pruebas y cobertura obtenidos al finalizar la elaboración de este PFC se pueden consultar en anexo A.

Configuración de Maven para la ejecución de pruebas funcionales

Las pruebas funcionales son aquellas que verifican que se satisfacen los requisitos funcionales del sistema. En este caso, se comprueba que el usuario pueda llevar a cabo las tareas para las que se desarrolla la herramienta y obtenga los resultados que corresponda en cada

momento, tanto si la operación se está llevando a cabo con normalidad como si se produce algún error.

Existen varias soluciones para realizar las pruebas funcionales de una aplicación web de un modo automatizado. En este PFC se utilizará Canoo Webtest [47], que permite simular la interacción del usuario con la interfaz gráfica de la aplicación y comprobar la respuesta del sistema.

La implementación de los *test cases* de Canoo Webtest se puede realizar mediante clases Java o ficheros XML. En caso de emplear ficheros XML, la automatización de su ejecución se realizará mediante tareas de **Ant**. No obstante, este método tiene una dificultad añadida ya que es necesario que la aplicación web se encuentre en un contenedor de servlets para que Canoo Webtest pueda comportarse como un usuario y analizar la respuesta del servidor. Para ello, será necesario realizar una compleja configuración de Maven (Véase Anexo D.2) que instancie un contenedor de servlets (*cargo-maven2-plugin*) y ejecute, mediante el plugin **maven-antrun-plugin**, las tareas que contienen las pruebas funcionales.

Para la generación de informes de las pruebas funcionales se hará uso del plugin **webtest-maven-plugin**. Este plugin, del mismo modo que el empleado para la generación de informes de las pruebas unitarias, no requiere configuración y deberá ser añadido en la sección *reporting* del fichero pom.xml.

Diseño e implementación de pruebas unitarias y funcionales

Una vez que se ha configurado el entorno tal y como se ha descrito anteriormente, se procede a diseñar e implementar las pruebas unitarias y funcionales.

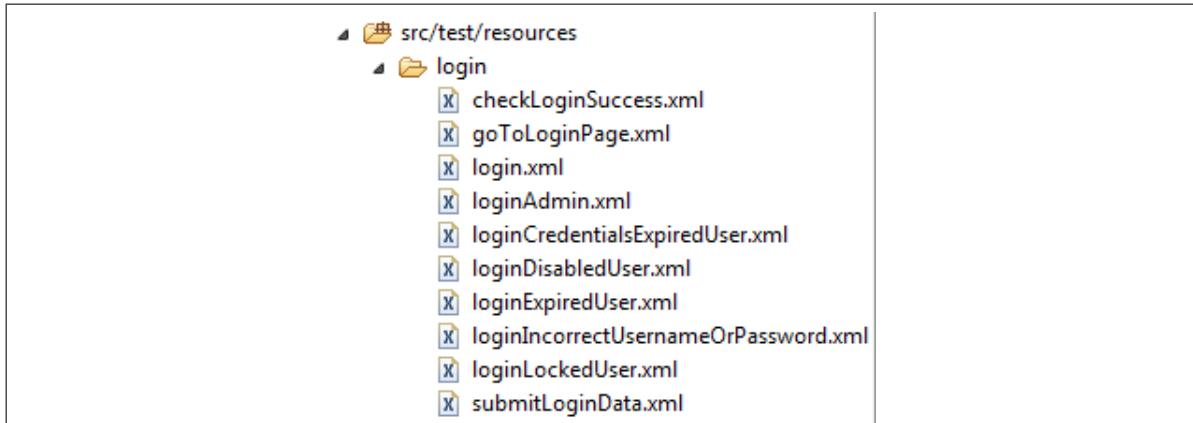
Según la ficha de iteración (Tabla 5.15), procede elaborar las pruebas de los CdU2.1, CdU2.9, CdU2.11 y CdU2.7. Para evitar que la sección se extienda demasiado, en este documento se sólo se incluye el proceso desarrollado en términos del CdU2.1 (control de acceso), que parte desde el diseño de los casos de prueba. En las tablas 5.24, 5.25, 5.26 y 5.27 se incluyen algunos de estos diseños, que se corresponden con los distintos escenarios de casos de uso definidos en la tabla 5.22.

Los casos de prueba diseñados han sido implementados mediante ficheros XML de Canoo Webtest y su ejecución automatizada mediante tareas Ant. En la figura 5.53 se observa una captura de pantalla de los mismos. En ella, cada fichero XML se corresponde con un *test case* e implementa cada uno de los casos de prueba diseñados.

Caso de uso: CdU2.1 (Control de acceso)
Caso de prueba: 1
Descripción: El usuario desea iniciar sesión en el sistema.
Precondiciones: Ninguna.
Resultado: El usuario accede al sistema y es redirigido a la página principal.

Tabla 5.24: Descripción del caso de prueba 1 para CdU2.1

Caso de uso: CdU2.1 (Control de acceso)
Caso de prueba: 2
Descripción: El usuario desea finalizar la sesión en el sistema.
Precondiciones: El usuario debe tener una sesión activa.
Resultado: El usuario es desconectado del sistema y redirigido a la página principal.

Tabla 5.25: Descripción del caso de prueba 2 para CdU2.1**Figura 5.53:** Estructura de los casos de prueba para CdU2.1

Después de la ejecución de los casos de prueba, Webtest genera un informe en formato HTML. En él se pueden analizar detalladamente los casos de prueba que se han ejecutado y los resultados de su ejecución. Además, en caso de producirse algún error, se indica exactamente en qué paso.

En la figura 5.54 se facilita el informe de pruebas generado por Canoo Webtest para CdU2.1: en la parte superior se indica que se han ejecutado 7 casos de prueba; en la parte inferior se muestran detalles individuales de la ejecución de cada uno de ellos. El nombre de cada caso de prueba es un enlace que lleva al analista a una nueva página en la que puede

Caso de uso: CdU2.1 (Control de acceso)
Caso de prueba: 3
Descripción: La usuario introduce un nombre de usuario y/o contraseña incorrecto.
Precondiciones: Ninguna.
Resultado: El usuario no inicia sesión y se le muestra un mensaje de error indicando que el nombre de usuario o la contraseña son incorrectos.

Tabla 5.26: Descripción del caso de prueba 3 para CdU2.1

Caso de uso: CdU2.1 (Control de acceso)
Caso de prueba: 4
Descripción: La cuenta de usuario está bloqueada
Precondiciones: Ninguna.
Resultado: El usuario no inicia sesión y se le muestra un mensaje de error indicando que la cuenta de usuario se encuentra bloqueada.

Tabla 5.27: Descripción del caso de prueba 4 para CdU2.1

estudiar detalladamente la ejecución del caso de prueba (Fig. 5.55 y 5.56).

5.3.6 Iteración 8

En esta iteración (Tabla 5.16) se aborda:

- Análisis y diseño de las funcionalidades relativas a la gestión de compañías, factorías, proyectos y subproyectos.
- Implementación de la gestión y selección de perfiles de visualización.
- Pruebas unitarias y funcionales relativas a la gestión y selección de perfiles de visualización.

5.3.6.1 Disciplina de análisis

Durante esta disciplina se aborda el análisis de las funcionalidades que permiten gestionar el conocimiento del sistema. Se trata de los requisitos de gestión de compañías, factorías, proyectos y subproyectos. El objetivo de estas funcionalidades es habilitar las tareas de creación,



Figura 5.54: Informe de pruebas para CdU2.1 (Vista global)

modificación y consulta de las distintas entidades que conforman la estructura organizacional del desarrollo global de software. Para comprender esta estructura se recomienda mirar la figura 2.1, incluída en la sección 2.2.1.

Para comenzar, se modela un diagrama de casos de uso algo más detallado que el que se ofreció en un primer momento. En este diagrama (Fig. 5.57) se muestran las tareas asociadas a cada caso de uso. Cabe destacar que sólo se han representado las tareas relativas a la gestión de compañías y factorías por no alargar la extensión de este documento.

A partir del diagrama de casos de uso y de la descripción de sus escenarios, se han modelado los diagramas de comunicación. En las figuras 5.58 y 5.59 se muestran los diagramas de comunicación de los escenarios generales para las funcionalidades de *crear compañía* y *ver o consultar compañía*. De estos diagramas cabe destacar dos detalles: por un lado, la clase *CompanyAction* que actúa de **fachada** entre la interfaz gráfica de usuario y el resto de

#	Result	Name	Parameter
✓ normal			
Test started at Thu Dec 01 13:34:38 CET 2011, lasting 00:00:33 (32717 ms).			
Source: D:\Repositorios\uclm-esi-alarcos\desglosa-web\src\test\resources\web-tests.xml:28:			
Base URL (used by invoke steps with a relative URL): http://localhost:8081/desglosa-web/			
1	✓	storeProperty name username value admin => value admin	
2	✓	storeProperty name password value admin => value admin	
3	✓	invoke get Login Page Resulting page -> complete url http://localhost:8081/desglosa-web/login url /login	
4	✓	verifytitle regex true we should see the login title text .*Inicio de sesión.*	
5	✓	setinputfield name j_username set user name value admin	
6	✓	setinputfield name j_password set password value admin	
7	✓	clickbutton label Iniciar Sesión Click the submit button Resulting page	
8	✓	verifytext Login is ok text .*Ha iniciado sesión correctamente..*	
9	✓	verifytitle regex true Home Page follows if login ok text .*:: DESGLOSA :: Soporte para proyectos de Desarrollo Global de Software.*	
10	✓	verifytext regex true Logged in as desired user text .*Sesión iniciada como: admin.*	

[Back to Test Report Overview](#)

Figura 5.55: Informe de pruebas para CdU2.1 (Caso de prueba 1 -Tabla 5.24-)

controladores y entidades (**patrón MVC**); y por otro lado, la clase **HibernateDaoSupport** que es una de las herramientas que proporciona Spring para acceder a bases de datos mediante Hibernate.

5.3.6.2 Disciplina de diseño

Durante esta disciplina se aborda el diseño de las funcionalidades que permiten gestionar el conocimiento del sistema: gestión de compañías, factorías, proyectos y subproyectos.

Inicialmente se han modelado los diagramas de secuencia que se corresponden con los diagramas de comunicación de la disciplina de análisis (Fig. 5.60 y 5.61). Como se puede

✓ locked user		
#	Result	Name
1	✓	storeProperty name username value locked => value locked
2	✓	storeProperty name password value locked => value locked
3	✓	invoke get Login Page Resulting page -> complete url http://localhost:8081/desglosa-web/login url /login
4	✓	verifytitle regex true we should see the login title text .*Inicio de sesión.*
5	✓	setinputfield name j_username set user name value locked
6	✓	setinputfield name j_password set password value locked
7	✓	clickbutton label Iniciar Sesión Click the submit button Resulting page
8	✓	verifytitle regex true we should see the login title yet text .*Inicio de sesión.*
9	✓	verifytext regex true User account must be locked text .*La cuenta del usuario está bloqueada.*

[Back to Test Report Overview](#)

Figura 5.56: Informe de pruebas para CdU2.1 (Caso de prueba 4 -Tabla 5.27-)

apreciar, estos diagramas también han sido modelados con más nivel de detalle, ya que se incluyen las llamadas específicas a los métodos de las clases y se representa la llamada a los métodos *validate* de Struts 2 antes de ejecutar una acción.

Por la misma razón que en la disciplina de análisis, en este documento sólo se incluyen los diagramas relativos al escenario general de las funcionalidades de *crear compañía* y *ver compañía*.

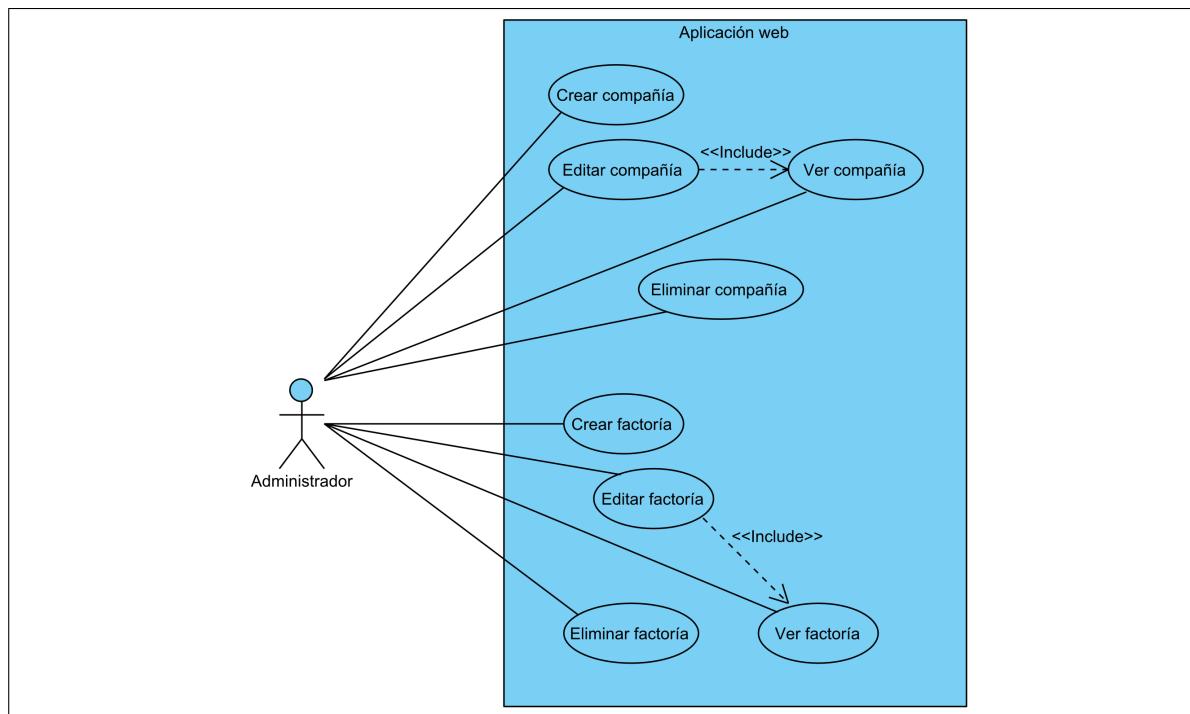


Figura 5.57: Diagrama de casos de uso para CdU2.3 y CdU2.4

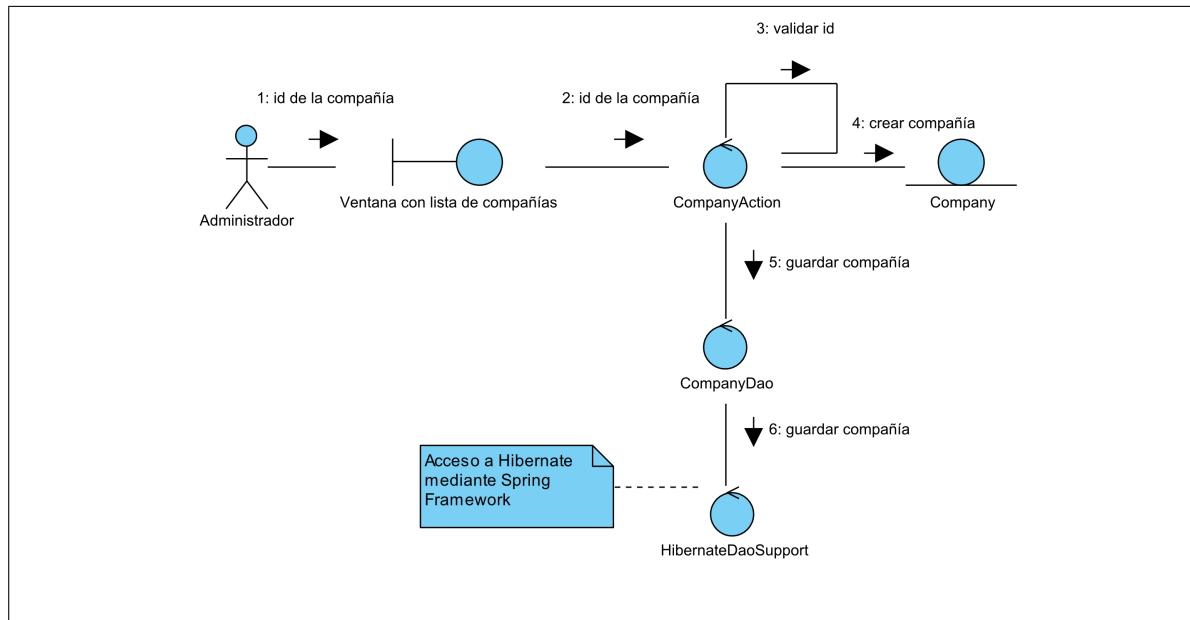


Figura 5.58: Diagrama de comunicación para CdU2.3 (Crear compañía)

5.3.6.3 Disciplina de implementación

En esta disciplina se lleva a cabo la implementación de una clase **GLObjectManager**. Esta clase utiliza métodos de **reflexión** para generar las objetos de las metáforas de visualización en tiempo real y asignar los valores correspondientes a sus dimensiones. Gracias a la solución propuesta, es posible agregar nuevas metáforas de visualización al motor gráfico y

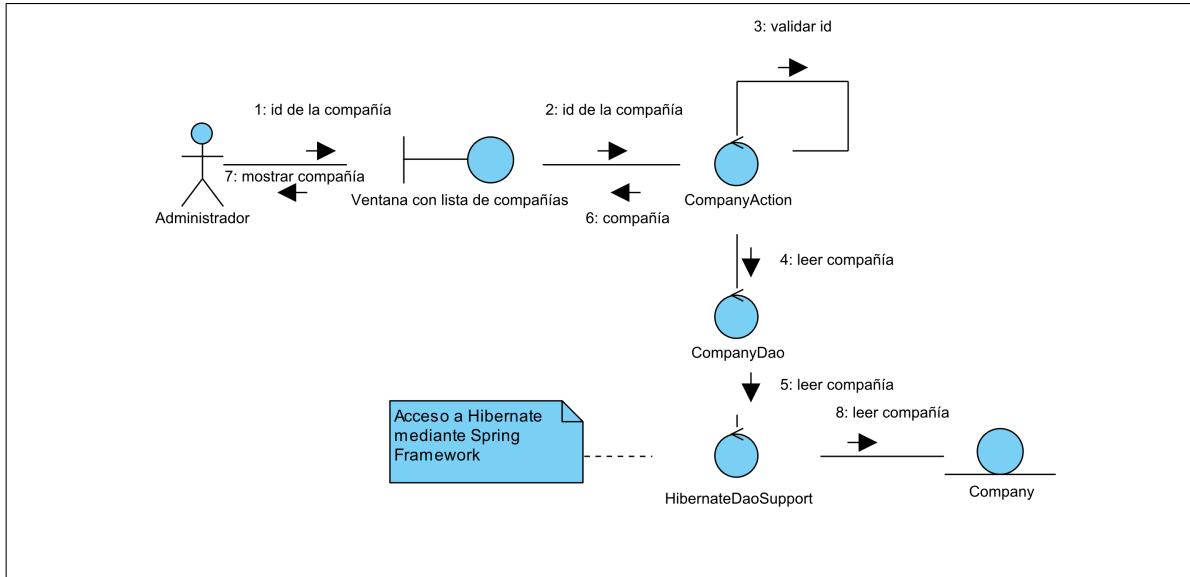


Figura 5.59: Diagrama de comunicación para CdU2.3 (Ver compañía)

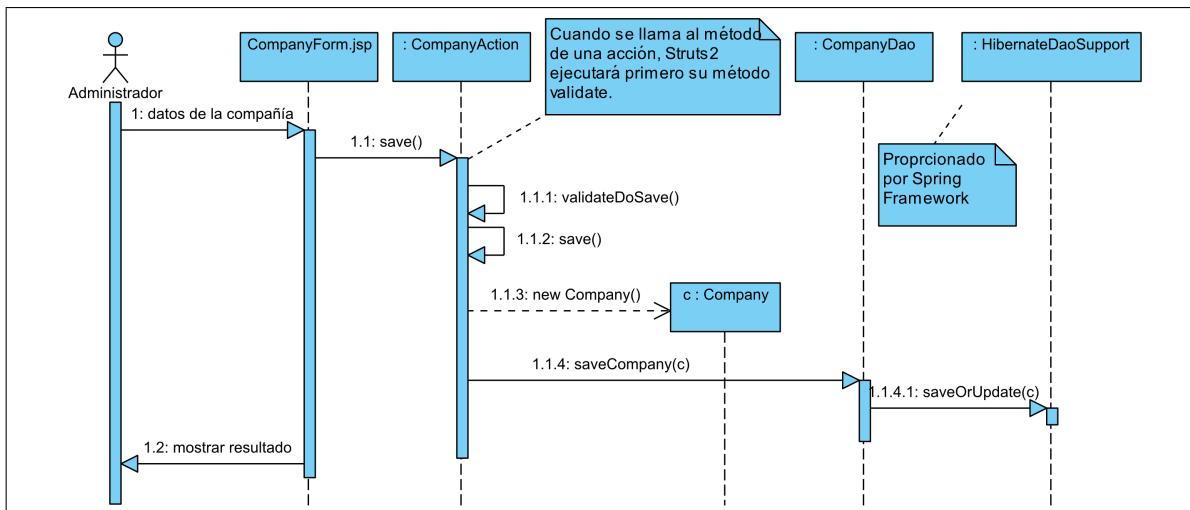


Figura 5.60: Diagrama de secuencia para CdU2.3 (Crear compañía)

utilizarlas desde la aplicación web sin la necesidad de hacer cambios en la clase.

El procedimiento se basa en *deserializar* el fichero XML del perfil seleccionado para obtener su jerarquía de objetos (Fig. 5.51). Una vez conocidos la entidad y el modelo gráfico que asocia el perfil, se recupera de la base de datos todos los objetos del tipo de la instancia que se está tratando. Acto seguido, se construyen los métodos *setters* y *getters* de ambas instancias conforme se leen los datos del perfil. Finalmente, se crea una instancia de la clase **City**, que está contenida en el motor gráfico y es la única dependencia de la aplicación web con el mismo (junto con el analizador de dimensiones), a la cual se le añaden los elementos de la escena con los valores asignados a sus dimensiones.

Una vez que se tiene en memoria la ciudad completa, se obtiene su representación en

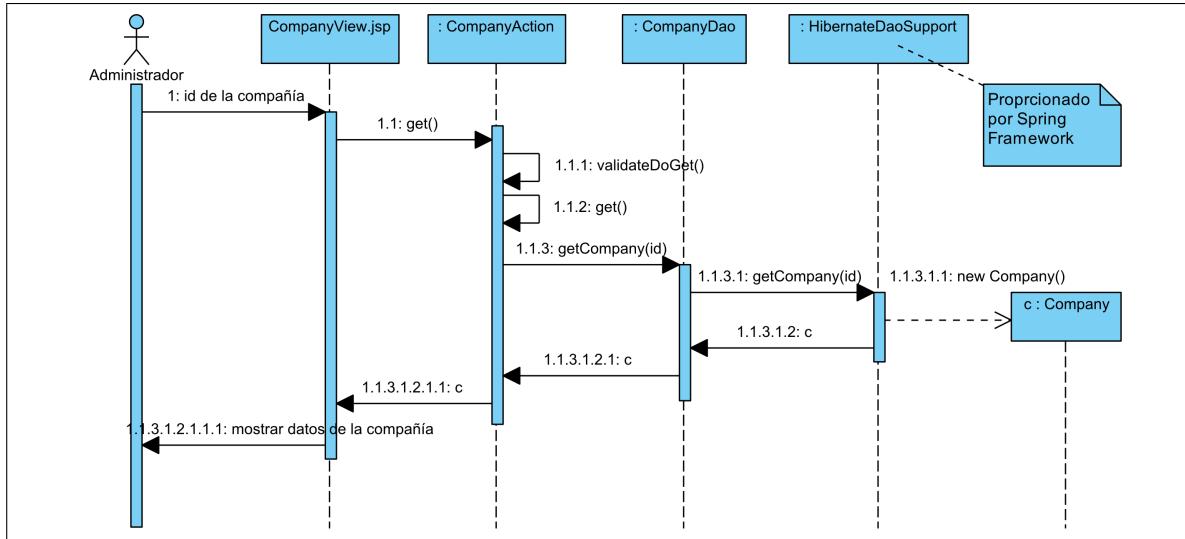


Figura 5.61: Diagrama de secuencia para CdU2.3 (Ver compañía)

formato JSON. La cadena de texto resultante será la que se le proporcione al motor gráfico como dato de entrada y a partir de la cual se generarán los gráficos en tres dimensiones.

Como resultado de esta disciplina, se facilita una cadena de texto en formato JSON (Listado 5.12) y el resultado que produce el motor gráfico cuando la procesa (Fig. 5.62).

```

1 {
2   "captionLines": [
3     {
4       "Línea informativa": "00649a",
5       "Otra línea informativa": "2a8400",
6       "Y otra más": "ff8c00"
7     },
8   "model": "model.gl.knowledge.GLTower",
9   "neighborhoods": [
10    {
11      "flats": [
12        {
13          "color": {"alpha": 1, "b": 0.6039216, "g": 0.39215687, "r": 0},
14          "depth": 1,
15          "height": 6.9,
16          "id": 4,
17          "innerHeight": 5.5,
18          "maxDepth": 3,
19          "maxHeight": 12,
20        }
21      ]
22    }
23  ]
24 }

```

```
20      \\"maxWidth\\":3,  
21      \\"positionX\\":0,  
22      \\"positionZ\\":0,  
23      \\"scale\\":1,  
24      \\"width\\":2  
25    }, ...  
26  } ...  
27 } ...  
28 }
```

Listado 5.12: Fragmento de la representación en formato JSON de una instancia de la clase City

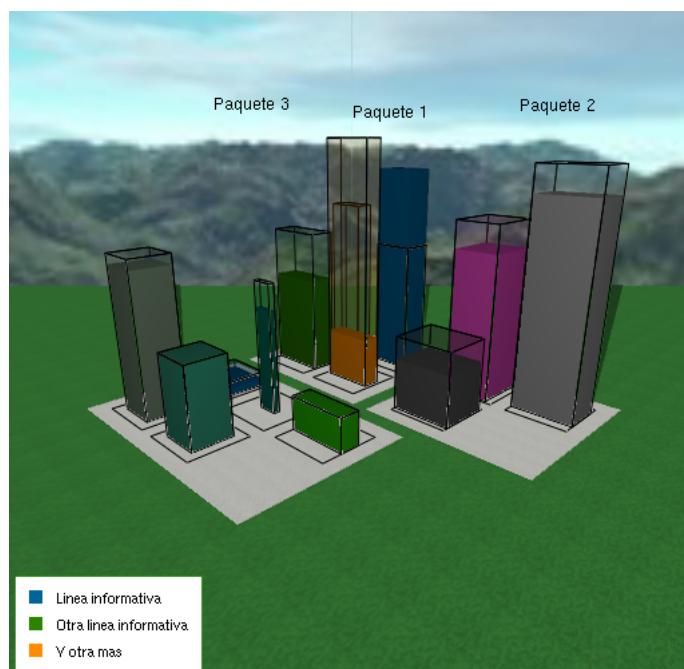


Figura 5.62: Resultado de procesar la cadena de texto en formato JSON (Listado 5.12)

5.3.6.4 Disciplina de pruebas

Durante esta disciplina se construyen los casos de prueba que sirven para comprobar la funcionalidad de los requisitos implementados la disciplina de implementación de esta misma iteración.

Cabe destacar que uno de los casos de prueba consiste verificar que los tipos de datos asociados entre los atributos de las entidades y las dimensiones de los modelos son compatibles.

No se incorporan más detalles al documento para no alargar la extensión del mismo.

5.3.7 Iteración 9

En esta iteración (Tabla 5.17) se aborda:

- Implementación de las funcionalidades de gestión de compañías, factorías, proyectos y subproyectos.
- Pruebas unitarias y funcionales de las funcionalidades implementadas en esta misma iteración.

5.3.7.1 Disciplina de implementación

Durante esta disciplina se implementan las funcionalidades relativas a la gestión de compañías, factorías, proyectos y subproyectos, y se corrigen algunos bugs que se han detectado en el sistema. Además se obtienen los diagramas de componentes y de despliegue correspondientes.

En la figura 5.63 se muestra un fragmento de la página que permite consultar los datos de una factoría de software, así como realizar operaciones con los proyectos que lidera y los subproyectos que está desarrollando.

5.3.7.2 Disciplina de pruebas

Durante esta disciplina se construyen los casos de prueba que sirven para comprobar la funcionalidad de los requisitos implementados la disciplina de implementación de esta misma iteración.

Al tratarse de funcionalidades típicas de gestión de información no se requiere un estudio más detallado en este documento.

5.4 Fase de transición

Esta fase consta de una única iteración, donde se realizarán las tareas necesarias para preparar el producto final y elaborar un entregable con documentación para el cliente.

Datos del director

Nombre	test director name 3
Apellidos	test director last name 3
Foto	

Proyectos liderados

Los proyectos que se muestran a continuación son aquellos cuya factoría principal o líder es la que está consultando.

Se ha encontrado un proyecto.

1	Proyecto	Código	Plan	Mercado	Subproyectos	2
<input checked="" type="radio"/> where to publish	w2p	1112	<input checked="" type="checkbox"/> Arquitectura			

Ver proyecto **Editar proyecto** **Eliminar proyecto** **Crear proyecto**

Subproyectos en desarrollo

Los subproyectos que se muestran a continuación son aquellos cuyo desarrollo lo desempeña la factoría que está consultando.

Se han encontrado 2 subproyectos. A continuación se muestran todos los subproyectos.

1	Subproyecto	Proyecto	Código del proyecto	Plan del proyecto	Mercado del proyecto	Subproyectos	Factoría	proyecto
<input checked="" type="radio"/> dgs-webapp	desglosa	dgs	1112	<input checked="" type="checkbox"/> Defensa	3		test factory name 2	
<input checked="" type="radio"/> w2p-webapp	where to publish	w2p	1112	<input checked="" type="checkbox"/> Arquitectura	2		test factory name 2	

Ver subproyecto **Editar subproyecto** **Eliminar subproyecto** **Crear subproyecto**

Figura 5.63: Captura de pantalla de un fragmento de la página que permite consultar información de una factoría de software

5.4.1 Iteración 10

En esta última iteración del ciclo se han corregido algunos *bugs* que se han detectado a lo largo del ciclo de desarrollo. Además, tal y como se indica en la ficha de iteración 5.18 se elaboran los documentos asociados al proyecto:

- **Manual de instalación y despliegue.** Documento que indica cómo construir los proyectos y pasar la herramienta a producción. será vital para el equipo que se encargue del mantenimiento del sistema (Véase Anexo B).
- **Manual de usuario.** Documento que explica las principales funcionalidades de las que dispone el sistema desarrollado (Véase Anexo C).

A continuación se procede la creación del paquete de distribución y su instalación en la empresa. Este proceso consta de dos partes, una para cada componente:

- **Motor gráfico.** Este componente será distribuido en un paquete JAR (Java ARchive). Gracias al uso de Maven para la gestión del ciclo de vida del proyecto, basta con ejecutar

el comando *mvn package* para obtener el correspondiente archivo. A continuación, se colocará el fichero JAR obtenido bajo el directorio *src\main\webapp\applet* con el nombre *desglosa.jar*.

- **Aplicación web.** Este componente será distribuido en formato WAR (Web application ARchive), que contiene todo el código ejecutable de la aplicación web. Una vez más, se ejecutará el comando *mvn package* para obtener el fichero correspondiente, que deberá ser desplegado en un contenedor de servlets y páginas JSP.

Para finalizar, cabe destacar la posibilidad de la generación automática de páginas web para la distribución de cada uno de los componentes que conforman este PFC. Esto es posible, una vez más, gracias al uso de Maven. Para generar dichas páginas web bastará con ejecutar el comando *mvn site* para cada uno de los componentes.

Capítulo 6

Conclusiones y Propuestas

6.1 Aspectos destacables de la solución propuesta

Tras haber finalizado la elaboración de este PFC se ha obtenido un sistema software compuesto por dos componentes -una aplicación web y un motor de gráficos 3D- cuyos ciclos de vida pueden ser llevados a cabo de forma independiente. Ambos componentes integrados conforman un sistema potente, flexible y extensible que facilita las distintas tareas de gestión *organizacional* en escenarios de desarrollo global, así como el seguimiento de los proyectos globalizados mediante diversas técnicas de visualización.

Algunos de los problemas derivados del desarrollo global surgen a partir de la distribución de recursos e información. Mediante la herramienta elaborada en este PFC se permite centralizar información acerca de compañías y factorías de software dispersas por todo el mundo, que colaboran en el desarrollo de proyectos. La información acerca de estos proyectos y los subproyectos que los conforman también queda centralizada. De este modo, los profesionales del sector pueden disponer de ella cuando la necesiten para llevar a cabo sus actividades. Cuestiones como la ubicación de las factorías de las con las que se puede contar para realizar un desarrollo o el tipo de mercado en el que se encuentran especializadas, son de vital importancia a la hora de decidir dónde se desarrollarán los distintos proyectos que se desean llevar a cabo.

Estos aspectos junto con los datos relativos a la calidad y productividad empleados en este tipo de desarrollos suponen una gran cantidad de información. Pero todas las actividades no requieren el mismo tipo de información, por lo tanto, manejar la información relevante en cada momento y analizarla de forma adecuada es crucial. Esta característica permite, entre otros, detectar anomalías, prevenir situaciones de riesgo, asegurar la calidad del software desarrollado y ofrecer soporte en la toma de decisiones en este tipo de actividades.

Para satisfacer este tipo de necesidades, *Desglosa* goza de un alto grado de personalización que permite filtrar la información que se desea analizar y decidir cuál es el mejor modo para representarla gráficamente mediante metáforas de visualización. Así, *Desglosa* es capaz de generar una serie de gráficos en 3D que facilitan la interpretación de datos complejos por medio de la exploración visual.

Todas estas funcionalidades son fruto de la consecución de un conjunto de tareas que se han ido abordando a lo largo del ciclo de desarrollo. Entre ellas destacan las siguientes:

- Modelado de una abstracción del mundo real relativo al contexto del DGS.
- Análisis de técnicas de visualización.
- Selección de medidas e indicadores de calidad y productividad utilizados en el contexto empresarial bajo el paradigma del DGS.
- Definición de metáforas de visualización que permitan representar aquella información seleccionada.
- Diseño de un método que permite configurar y personalizar la visualización de la información, con el objetivo de filtrar aquella que sea relevante para cada tipo de actividad.
- Diseño de mecanismos que permiten adaptar la herramienta de un modo sencillo y rápido a nuevos requisitos de visualización o información a visualizar.

De esta forma se puede apreciar que los diferentes objetivos que fueron identificados en la sección 2.2 han sido alcanzados satisfactoriamente.

Por otro lado, la arquitectura de ambos componentes ha sido diseñada en base a jerarquías de herencia e interfaces. De este modo, se facilita notablemente el mantenimiento del sistema en lo que a futuros requisitos se refiere, tales como la adición de nuevas metáforas de visualización, nuevos modelos gráficos, o una estructura organizacional del desarrollo global más compleja. Además, destacar que se han aplicado patrones de diseño y buenas prácticas de programación en todos aquellos lugares en los que ha sido necesario, tratando siempre de lograr un compromiso entre la complejidad de la arquitectura y los principios de alta cohesión y bajo acoplamiento entre clases.

Otro punto importante es la naturaleza multiplataforma e independiente de la arquitectura de la que goza el motor gráfico. En este aspecto se ha invertido mucho esfuerzo para abstraer,

tanto al programador como al usuario final, de la dependencia de librerías nativas para poder generar gráficos en tres dimensiones. Así pues, se han provisto mecanismos que automatizan las labores de descarga e instalación de este tipo de librerías tanto en el proceso de construcción del proyecto como en la puesta en producción y ejecución del mismo, haciendo posible su portabilidad a sistemas basados en Microsoft Windows, GNU/Linux y MacOS, en sus versiones de 32 y 64 bits.

También es conveniente puntualizar que el motor gráfico es completamente independiente de la aplicación que lo contiene. Por esta razón, todo tipo de aplicación, cualquiera que sea su índole -escritorio, web o móvil- y área de aplicación -no sólo desarrollo global-, puede incorporarlo y beneficiarse de todas su cualidades, entre las que destacan la generación de gráficos 3D, la navegación interactiva a través de la escena y la selección de los elementos que se encuentran en la misma.

Finalmente, destacar el sistema de autenticación y control de acceso que se ha incorporado en la aplicación web. Se trata de un sistema basado tecnologías que actualmente se implementa en sistemas gubernamentales, aplicaciones militares y bancos. Por tanto, dado que este sistema será implantado en empresas, la seguridad es un aspecto que se ha tenido muy en cuenta.

6.2 Trabajo actual y futuras mejoras

El sistema que se ha obtenido durante la elaboración de este PFC es un desarrollo real para empresa con un alto grado de complejidad. Por lo tanto, se trata de un proyecto que sigue vivo y que actualmente está recibiendo un conjunto de modificaciones y ampliaciones que quedan fuera del objetivo de este PFC. Entre ellas destacan las siguientes características:

- Gestión de usuarios, grupos y roles.
- Incorporación de distintos niveles de topografía en la escena. Esto permite agregar una nueva dimensión a las metáforas, con el objetivo de configurar la superficie del terreno y elevar determinados elementos o barrios.
- Posibilidad de elegir entre distintos métodos de normalización en la generación de gráficos 3D. Esta característica permitirá que no haya elementos excesivamente grandes o excesivamente pequeños en la escena.

- Capacidad de definir nuevas entidades empleadas en el paradigma del desarrollo global mediante ficheros XML. Estos ficheros XML se emplearán para construir clases y objetos en tiempo de ejecución, que serán utilizados en las tareas de visualización y configuración de perfiles.

Por último, destacar que actualmente ya se encuentra publicada la primera versión de la especificación del API de WebGL. Por tanto, se realizará un estudio de viabilidad y eficiencia para comprobar las ventajas e inconvenientes que acarrearía migrar el motor gráfico a dicha tecnología.

6.3 Conocimientos adquiridos

Los conocimientos adquiridos son el mejor sabor de boca que se queda después de finalizar el desarrollo de este PFC. Cuestiones como la aplicación y seguimiento de una metodología de desarrollo en un caso real, así como abordar todas las fases del ciclo de vida de un proyecto, han sido experiencias muy importantes. No obstante, destaca el perfil profesional, basado en tecnologías Java que actualmente se utilizan en empresa, que se ha forjado durante todo este proceso.

Las características más importantes de dicho perfil son las siguientes:

- Gestión del ciclo de vida de proyectos mediante **Maven**.
- Uso de **Spring Framework** como capa de comunicación entre múltiples librerías basadas en Java, inyección de dependencias y creación de *beans* mediante el contenedor IoC (Inversion of Control).
- Implementación de un sistema de autenticación y control de acceso mediante **Spring Security**, antes conocido como Acegi Security.
- Manejo de la persistencia mediante **Hibernate**, **JPA** y las facilidades que ofrece Spring.
- Desarrollo de aplicaciones web basadas en **Struts2** con todo el conocimiento adicional que ello conlleva (interceptores, validadores, acciones, etcétera).
- Diseño e implementación de una arquitectura de pruebas extensible y reutilizable mediante **DBUnit**, **JUnit**, **Spring-Test** y **Canoo WebTest**.

- Experiencia en el diseño y desarrollo de páginas web dinámicas mediante **AJAX** y **JavaScript**.
- Generación de gráficos en tres dimensiones mediante **OpenGL**.

Por último, denotar los conocimientos docentes adquiridos en diferentes disciplinas como informática gráfica, ingeniería del software y bases de datos, entre otras.

Bibliografía

- [1] *FCD 25010, Software engineering-Software product Quality Requirements and Evaluation (SQuaRE) Quality mode.*
- [2] *ISO/IEC 14598. Information technology. Software product evaluation. Part 1: General overview.*
- [3] *ISO/IEC 9126. Software engineering. Product quality. Part 1: Quality model.*
- [4] Advantech global software services. <http://www.advantech.com.tw/embcore/globalpresent.aspx>, Accedido el 30 de Septiembre de 2011.
- [5] Alexander, Christopher: *The Timeless Way of Building*. Oxford University Press, New York, 1979, ISBN 0195024028.
- [6] Alexander, Christopher, Sara Ishikawa, and Murray Silverstein: *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, 1977, ISBN 0195019199.
- [7] Aurum, Aybüke *et al.*: *Managing Software Engineering Knowledge*. Springer, 2003.
- [8] Calero, Coral, Marian Moraga, and Mario G. Piattini: *Calidad del producto y proceso software*. Editorial Ra-Ma, 2010, ISBN 9788478979615.
- [9] Cho, Juyun: *Globalization and global software development*. Issues in information systems, III (2):287–290, 2007.
- [10] Croucher, Sheila: *Globalization and belonging: the politics of identity in a changing world*. page 10, 2004.
- [11] Damian, Daniela and Deependra Moitra: *Guest editors' introduction: Global software development: How far have we come?* IEEE Software, 5:17–19, 2006.

- [12] Diehl, Stephan: *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, Berlin, 2007, ISBN 978-3-540-46504-1.
- [13] Ducasse, Stéphane *et al.*: *Visualization of Practices and Metrics*, March 2010.
- [14] Engel, W.F.: *Beginning Direct3D game programming*. Game Programming. Premier Press, 2003, ISBN 9781931841399.
- [15] Espinoza, José Martín Olguín: *El proceso unificado de desarrollo de software*. <http://yaqui.mx1.uabc.mx/~molguin/as/RUP.htm>, Accedido el 5 de Julio de 2011.
- [16] Feijjs, Loe and Roel De Jong: *3d visualization of software architectures*. Communication of the ACM, 41(12):73–78, 1998.
- [17] Fenton, Norman E. and Shari Lawrence Pfleeger: *Software metrics: A rigorous approach*. 1997.
- [18] Foley, J.D.: *Computer graphics: principles and practice*. The Systems Programming Series. Addison-Wesley, 1996, ISBN 9780201848403.
- [19] Freeman, R.E.: *Strategic Management: A Stakeholder Approach*. Cambridge University Press, 2010, ISBN 9780521151740.
- [20] Friedman, Thomas L: *The world is flat: A brief history of the twenty-first century*. Farrar, Straus and Giroux, New York, 2005.
- [21] Gamma, Erich *et al.*: *Design Patterns*. Addison-Wesley, Boston, MA, 1995, ISBN 0201633612.
- [22] García, Félix *et al.*: *Towards a consistent terminology for software measurement*. Information and Software Technology, 48(8):631–644, 2006.
- [23] Govil-Pai, S.: *Principles of computer graphics: theory and practice using OpenGL and Maya*. Springer, 2004, ISBN 9780387955049.
- [24] Healey, Christopher G., Kellogg S. Booth, and James T. Ennis: *Visualizing real-time multivariate data using preattentive processing*. ACM TOMACS, 5, 1995.

- [25] Herbsleb, James D. and Deependra Moitra: *Global software development*. IEEE Software, March/April:16–20, 2001.
- [26] Hofstede, G., G.H. Hofstede, and G.J. Hofstede: *Cultures and organizations: software of the mind*. Successful strategist series. McGraw-Hill, 2005, ISBN 9780071439596.
- [27] Holten, Danny, Roel Vliegen, and Jarke J. van Wijk: *Visual realism for the visualization of software metrics*. VISSOFT, pages 27–32, 2005.
- [28] Hoyer, R. W. and B. B. Y. Hoyer: *What is quality*. Quality Progress, July 2001.
- [29] Jacobson, Ivar, Grady Booch y James Rumbaugh: *El proceso unificado de desarrollo de software*. Addison Wesley, 2000.
- [30] Katzy, Bernhard, Roberto Evaristo, and Ilze Zigurs: *Knowledge management in virtual projects: A research agenda*. 1999.
- [31] Kilgard, Mark: *Shadows, lighting, reflections, and textures all at once*. <http://www.opengl.org/resources/code/samples/mjktips/TexShadowReflectLight.html>, Accedido el 8 de Septiembre de 2011.
- [32] Kobitzsch, Werner, Dieter Rombach, and Raimund L. Feldmann: *Outsourcing in india*. IEEE Software, 18(2):78–86, March/April 2001.
- [33] Krishna, Srinivas, Sundeep Sahay, and Geoff Walsham: *Managing cross-cultural issues in global software outsourcing*. Commun. ACM, pages 62–66, 2004.
- [34] Lanza, Michele: *Polymetric views: A lightweight visual approach to reverse engineering*. IEEE Transactions on Software Engineering, 29(9):782–795, September 2003.
- [35] Layman, Lucas *et al.*: *Essential communication practices for extreme programming in a global software development team*. Information and Software Technology, 48(9):781–794, 2006.
- [36] Luna, F.D. and R. Lopez: *Introduction to 3D game programming with DirectX 9.0*. Wordware game developer's library. Wordware Pub., 2003, ISBN 9781556229138.
- [37] Pfleeger, Shari Lawrence: *Assessing software measurement*. IEEE Software, pages 25–26, March/April 1997.

- [38] Piattini, Mario: *Medición y Estimación del Software: Técnicas y Métodos para Mejorar la Calidad y la Productividad*. Ra-Ma, 2008.
- [39] Piattini, Mario G. and Félix O. García: *Calidad en el desarrollo y mantenimiento del software*. Editorial Ra-Ma, 2003.
- [40] Piattini, Mario G. *et al.*: *Calidad de Sistemas de Información*. Editorial Ra-Ma, 2011.
- [41] Prikladnicki, Rafael *et al.*: *Distributed software development: Practices and challenges in different business strategies of offshoring and onshoring*. Global Software Engineering 2007 ICGSE 2007 Second IEEE International Conference, pages 262–274, 2007.
- [42] Richardson, Ita *et al.*: *Global software development: The challenges*. Technical Report. Muncie, Indiana, U.S.A.: Ball State University, 2005.
- [43] Sahay, Sundeep, Brian Nicholson, and Srinivas Krishna: *Global IT Outsourcing: Software Development across Borders*. Cambridge University Press, November 2003.
- [44] Sangwan, Raghvinder *et al.*: *Global Software Development Handbook*. Auerbach Publications, 2007.
- [45] Schuh, Peter: *Integrating Agile Development in the Real World*. Charles River Media, 2004.
- [46] Shreiner, D. and OpenGL Architecture Review Board: *OpenGL programming guide: the official guide to learning OpenGL, version 2*. OpenGL Series. Addison-Wesley, 2006, ISBN 9780321335739.
- [47] Sitio Web Oficial de Canoo WebTest. <http://webtest.canoo.com/webtest/manual/WebTestHome.html>, Accedido el 25 de Septiembre de 2011.
- [48] Sitio Web Oficial de Indra: *Indra lidera el proyecto de I+D ORIGIN para optimizar el desarrollo de software en las compañías globales*. <http://www.indracompany.com/sites/default/files/Indra-ORIGIN.doc>, Accedido el 12 de Diciembre de 2011.
- [49] Sitio Web Oficial de Java OpenGL. <http://jogamp.org/jogl/www/>, Accedido el 2 de Diciembre de 2011.

- [50] Sitio Web Oficial de JogAmp: *JNLPAppletLauncher Documentation*. <http://jogamp.org/applet-launcher/www/>, Accedido el 10 de Enero de 2011.
- [51] Sitio Web Oficial de m2eclipse. <http://eclipse.org/m2e/>, Accedido el 12 de Diciembre de 2011.
- [52] Sitio Web Oficial de Maven: *Maven Getting Started Guide*. <http://maven.apache.org/guides/getting-started/index.html>, Accedido el 12 de Diciembre de 2011.
- [53] Sitio Web Oficial de OpenGL. <http://www.opengl.org/>, Accedido el 12 de Diciembre de 2011.
- [54] Sitio Web Oficial de ORIGIN: *ORganizaciones Inteligentes Globales INnovadoras*. <http://innovation-labs.com/origin/index.html>, Accedido el 12 de Julio de 2011.
- [55] Sitio Web Oficial de SGI: *OpenGL Sample Implementation*. <http://oss.sgi.com/projects/ogl-sample/>, Accedido el 2 de Diciembre de 2011.
- [56] Sitio Web Oficial de SGI: *The OpenGL API*. <http://www.sgi.com/products/software/opengl/>, Accedido el 2 de Diciembre de 2011.
- [57] Sitio Web Oficial de Spring Framework. <http://www.springsource.org/>, Accedido el 12 de Diciembre de 2011.
- [58] Sitio Web Oficial de Spring Security. <http://static.springsource.org/spring-security/site/>, Accedido el 12 de Agosto de 2011.
- [59] Sitio Web Oficial de Spring Security: *Spring Security Documentation*. <http://static.springsource.org/spring-security/site/reference.html>, Accedido el 12 de Agosto de 2011.
- [60] Sitio Web Oficial de Struts: *Struts 2 Maven Archetypes*. <http://struts.apache.org/2.2.3/docs/struts-2-maven-archetypes.html>, Accedido el 2 de Diciembre de 2011.
- [61] Sitio Web Oficial del Grupo Alarcos. <http://alarcos.inf-cr.uclm.es/>, Accedido el 1 de Noviembre de 2011.

- [62] Torossi, Gustavo: *El Proceso Unificado de Desarrollo de Software*.
- [63] Vorwerk, Raymund: *3DJVis: Explore large software systems in 3D*. <http://www.v-research.net/3DJVis/default.htm>, Accedido el 15 de Septiembre de 2011.
- [64] Walnum, C.: *Microsoft Direct3D programming: kick start*. Kick Start Series. Sams, 2003, ISBN 9780672324987.
- [65] Ware, Colin: *Information Visualization. Perception for design*. Morgan Kaufmann, February 2004.
- [66] Wettel, Richard and Michele Lanza: *Visualizing software systems as cities*. In Proceedings of ICPC 2007 (4th IEEE International Workshop of Visualizing Software for Understanding and Analysis), pages 92–99, 2007.
- [67] Zubrow, Dave: *Software Quality Requirements and Evaluation, the ISO 25000 Series*. PSM Technical Working Group, 2004.

Apéndice A

Informes de pruebas y cobertura de código

En este anexo se muestran pequeños fragmentos de los informes de pruebas y cobertura de código que se han generado en la última iteración del ciclo de desarrollo. La colección completa de todos los informes puede encontrarse en el sitio web generado mediante Maven que se encuentra bajo el directorio *documentacion\site* (Fig. A.1).

Desglosa: GSD Visualization System

Last Published: 2012-01-10 | Version: 0.0.1-SNAPSHOT

Desglosa: GSD Visualization System

Project Documentation

- ▶ Project Information
- ▼ Project Reports
 - Surefire Report
 - Source Xref
 - Scm ChangeLog
 - Canoo WebTest Report
 - Tag List
 - Cobertura Test Coverage
 - CPD Report
 - Checkstyle
 - JavaDocs
 - PMD Report

Built by

Generated Reports

This document provides an overview of the various reports that are automatically generated by Maven. Each report is briefly described below.

Overview

Document	Description
Surefire Report	Report on the test results of the project.
Source Xref	HTML based, cross-reference version of Java source code.
Scm ChangeLog	Generated change log report from SCM.
Canoo WebTest Report	Canoo WebTest Report.
Tag List	Report on various tags found in the code.
Cobertura Test Coverage	Cobertura Test Coverage Report.
CPD Report	Duplicate code detection.
Checkstyle	Report on coding style conventions.
JavaDocs	JavaDoc API documentation.
PMD Report	Verification of coding rules.

Copyright © 2012. All Rights Reserved.

Figura A.1: Informes generados mediante Maven

En la figura A.2 se muestra una parte del informe que muestra los resultados de ejecución de los casos de prueba. En él se aprecia que se han implementado un total de 143 tests y que todos se han ejecutado correctamente.

En las figuras A.3, A.4 y A.5 se muestran algunos ejemplos de los informes de cobertura

obtenidos. En concreto se muestran los informes de cobertura de los paquetes *action*, *control* y *model.util*. Como se puede observar se ha cubierto en torno al 80 % de código, lo cual es una cantidad de código importante. Gran parte del código no cubierto se debe a constructores, excepciones y métodos *setters&getters* no utilizados.

Surefire Report																				
Summary																				
[Summary] [Package List] [Test Cases]																				
<table border="1"> <thead> <tr> <th>Tests</th><th>Errors</th><th>Failures</th><th>Skipped</th><th>Success Rate</th><th>Time</th><th></th></tr> </thead> <tbody> <tr> <td>143</td><td>0</td><td>0</td><td>0</td><td>100%</td><td>528.36</td><td></td></tr> </tbody> </table>							Tests	Errors	Failures	Skipped	Success Rate	Time		143	0	0	0	100%	528.36	
Tests	Errors	Failures	Skipped	Success Rate	Time															
143	0	0	0	100%	528.36															
Note: failures are anticipated and checked for with assertions while errors are unanticipated.																				
Package List																				
[Summary] [Package List] [Test Cases]																				
Package		Tests	Errors	Failures	Skipped	Success Rate														
es.uclm.inf_cr.alarcos.desglosa_web.actions		85	0	0	0	100%														
es.uclm.inf_cr.alarcos.desglosa_web.util		2	0	0	0	100%														
es.uclm.inf_cr.alarcos.desglosa_web.control		36	0	0	0	100%														
es.uclm.inf_cr.alarcos.desglosa_web.dao		20	0	0	0	100%														
Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.																				

Figura A.2: Fragmento del informe de resultados de la ejecución de pruebas

Coverage Report - es.uclm.inf_cr.alarcos.desglosa_web.actions					
Package /	# Classes	Line Coverage	Branch Coverage	Complexity	
es.uclm.inf_cr.alarcos.desglosa_web.actions	10	73% 660/898	80% 228/282	2,542	
Classes in this Package /	Line Coverage	Branch Coverage	Complexity		
CompanyAction	72% 96/132	91% 33/36	2,44		
FactoryAction	69% 121/174	73% 65/88	3,37		
GenericActionInterface	N/A N/A	N/A N/A	1		
ProfileAction	79% 86/108	77% 14/18	1,885		
ProfileAction\$1	100% 6/6	N/A N/A	1,885		
ProfileAction\$2	100% 5/5	N/A N/A	1,885		
ProjectAction	76% 96/126	82% 38/46	3		
SubprojectAction	74% 97/130	85% 36/42	2,583		
VisualizationAction	78% 153/196	80% 42/52	2,71		

Report generated by [Cobertura 1.9.4.1](#) on 10/01/12 1:32.

Figura A.3: Informe de cobertura de código del paquete *action*

Coverage Report - es.uclm.inf_cr.alarcos.desglosa_web.control					
Package /	# Classes	Line Coverage	Branch Coverage	Complexity	
es.uclm.inf_cr.alarcos.desglosa_web.control	7	83% 435/521	66% 218/330	3,727	
Classes in this Package /	Line Coverage	Branch Coverage	Complexity		
CompanyManager	100% 29/29	N/A N/A	1,2		
FactoryManager	100% 32/32	N/A N/A	1,182		
GObjectManager	75% 223/297	67% 180/266	24,167		
MarketManager	100% 5/5	N/A N/A	1		
ProfileManager	87% 87/99	55% 32/58	2,933		
ProjectManager	100% 32/32	100% 4/4	1,455		
SubprojectManager	100% 27/27	100% 2/2	1,3		

Report generated by [Cobertura 1.9.4.1](#) on 10/01/12 1:32.

Figura A.4: Informe de cobertura de código del paquete *control*

Coverage Report - es.uclm.inf_cr.alarcos.desglosa_web.model.util					
Package /	# Classes	Line Coverage	Branch Coverage	Complexity	
es.uclm.inf_cr.alarcos.desglosa_web.model.util	8	74% 74/100	90% 29/32	1,485	
Classes in this Package /	Line Coverage	Branch Coverage	Complexity		
Measure	N/A N/A	N/A N/A	0		
MeasureAnnotationParser	76% 10/13	80% 8/10	2,25		
MeasureWrapper	44% 8/18	N/A N/A	1		
MyHashMapEntryType	66% 8/12	N/A N/A	1		
MyHashMapType	80% 8/10	100% 2/2	1,25		
Property	N/A N/A	N/A N/A	0		
PropertyAnnotationParser	90% 30/33	95% 19/20	4,333		
PropertyWrapper	71% 10/14	N/A N/A	1		

Report generated by [Cobertura 1.9.4.1](#) on 10/01/12 1:32.

Figura A.5: Informe de cobertura de código del paquete *model.util*

Apéndice B

Manual de Instalación y Despliegue

El sistema de visualización que se ha desarrollado con la elaboración de este PFC consta de dos componentes cuyo código fuente se puede encontrar en el DVD adjunto a esta documentación. El ciclo de vida de ambos ha sido gestionado mediante Maven 2.2.1, lo que facilita el proceso de construcción y despliegue de los mismos. No obstante, si no desea lidiar con Maven, puede instalar una base de datos de prueba mediante el script *install.sql* y desplegar el fichero *desglosa-web.war* en un servidor web tipo Apache Tomcat. Ambos ficheros se encuentran bajo el directorio *dist*.

Si por el contrario prefiere construir ambos proyectos, siga los pasos que se detallan a continuación.

B.1 Instalación de las librerías de JOGL en el repositorio local de Maven

La versión de JOGL empleada en este PFC es la 2.0. Esta versión no se encuentra disponible en los repositorios oficiales de Maven, por lo que deberá ser descargada de la página oficial de JOGL [49] (también se proporciona en el DVD adjunto a este PFC) e instalada en el repositorio local de Maven.

Para comenzar el proceso abra un terminal, diríjase a la ruta en la que se encuentre el proyecto **desglosa** (motor gráfico) y navegue a través de la ruta de directorios *builds - jogamp - deployment - webstart - archive*. En este directorio encontrará numerosos archivos cuyos nombres se corresponden con la estructura *librería-versión-plataforma-arquitectura.7z*. Por ejemplo, el fichero *jogl-2.0-b20-20110302-windows-amd64.7z* se corresponde con la librería jogl, versión 2.0, plataforma Microsoft Windows y arquitectura de 64 bits.

Descomprima el fichero *jogl-2.0-b23-20110303-x-y.7z*, donde *x* se corresponde con su sistema operativo e *y* con la arquitectura de su máquina. Sitúese en el directorio *jar* y ejecute los comandos del listado B.1 en un terminal. Esto añadirá las librerías a su repositorio local de Maven.

```

1 mvn install:install-file -Dfile=gluegen-rt.jar -DgroupId=com.
    jogamp.gluegen -DartifactId=gluegen-rt -Dversion=2.0-b23
    -20110303 -Dpackaging=jar
2
3 mvn install:install-file -Dfile=jogl.all.jar -DgroupId=com.
    jogamp.jogl -DartifactId=jogl.all -Dversion=2.0-b23-20110303
    -Dpackaging=jar
4
5 mvn install:install-file -Dfile=nativewindow.all.jar -DgroupId=
    com.jogamp.nativewindow -DartifactId=nativewindow.all -
    Dversion=2.0-b23-20110303 -Dpackaging=jar
6
7 mvn install:install-file -Dfile=newt.all.jar -DgroupId=com.
    jogamp.newt -DartifactId=newt.all -Dversion=2.0-b23-20110303
    -Dpackaging=jar

```

Listado B.1: Comandos para la instalación de librerías en el repositorio local de Maven

B.2 Construcción de los proyectos

En el DVD adjunto a este PFC se encuentran dos directorios llamados *desglosa* y *desglosa-web*. El primero de ellos contiene todos los ficheros relativos al motor gráfico y el segundo a la aplicación web.

Como ya se ha argumentado a lo largo de esta documentación, la aplicación web contiene al motor gráfico. La ubicación del motor gráfico dentro de la aplicación web se encuentra en el directorio *desglosa-web\src\main\webapp\applet\desglosa.jar*. Actualmente incluye la última versión de éste, por lo que no es necesario volver a construir el fichero JAR. No obstante, generar una nueva versión del mismo es tan sencillo como ejecutar el comando *mvn package* en el directorio del proyecto *desglosa* y sustituir el viejo fichero *jar* por el nuevo.

A continuación, si ejecuta el comando “*mvn -DskipTests -Pprod package*” bajo el directo-

rio *desglosa-web*, creará la base de datos con el *dataset* de prueba y generará un nuevo fichero WAR que podrá ser desplegado en cualquier contenedor de servlets como Apache Tomcat. Es posible que si el servidor de bases de datos no se encuentra en la máquina en la que esté operando deba editar el fichero *pom.xml*, ya que por defecto se especifica éste se encuentra en *localhost@3306*.

También existe la posibilidad de desplegar la aplicación mediante un plugin de Maven llamado Jetty. Jetty es un contenedor de servlets completamente funcional y puede emplearse para comprobar el funcionamiento de la aplicación. Para desplegar la aplicación automáticamente en este servidor ejecute el comando “*mvn jetty:run -Dmaven.skip.test=true -Pprod*” o “*mvn jetty:run-war -DskipTests -pProd*”.

Apéndice C

Manual de Usuario

DESGLOSA es una herramienta de visualización cuyo objetivo es facilitar la gestión organizacional, en el contexto del desarrollo global, y el seguimiento de los proyectos globalizados. Para lograrlo, trata de dar soporte a las distintas actividades que se llevan a cabo en este tipo de escenarios y se ofrece distintos mecanismos de representación gráfica para la información que se considera relevante, como pueden ser las medidas e indicadores de calidad y productividad utilizados en este paradigma. Gracias a esta característica es posible predecir y detectar distintos riesgos y anomalías, así como ofrecer ayuda en la toma de decisiones de las actividades de la organización, con el objetivo de incrementar la competitividad de las organizaciones a nivel internacional.

Cuando el usuario escribe la dirección URL del sistema en su navegador web, obtiene una página tal y como la que se muestra en la figura C.1.

Por razones de confidencialidad, todos los datos que se muestran a lo largo de este manual, tales como ubicaciones, nombres propios, nombres de entidades y datos de medidas e indicadores, son ficticios y con propósito meramente ilustrativo. Cualquier parecido con la realidad es pura coincidencia.

C.1 Acceso al sistema

Las funcionalidades de la aplicación están condicionadas en función del tipo de usuario que va a utilizarla. Por defecto, el sistema reconoce tres tipos de usuarios: administrador, jefe de proyecto y usuario estándar. Según el *dataset* que se incorpora por defecto, los usuarios que pueden emplearse son los siguientes:

- Usuario perteneciente al grupo de administradores: usuario *admin* y contraseña *admin*.



Figura C.1: Página de inicio de la aplicación web

- Usuario perteneciente al grupo de jefes de proyecto: usuario *manager* y contraseña *manager*.
- Usuario perteneciente al grupo de usuarios usuario *user* y contraseña *user*.

Para llevar a cabo el proceso de autenticación, el usuario deberá acceder a la funcionalidad por medio del botón “*iniciar sesión*” que se encuentra en la parte superior derecha de la página. En el formulario que se proporciona (Fig. C.2) deberá introducir su nombre de usuario y su contraseña.

El resultado del proceso de autenticación será notificado al usuario, tanto si es satisfactorio como si se ha producido algún error. En este último los distintos tipos de error son los siguientes (Fig. C.3):

- Usuario y/o contraseña incorrectos.
- La cuenta del usuario está bloqueada.
- El usuario está deshabilitado.
- La cuenta del usuario ha caducado.

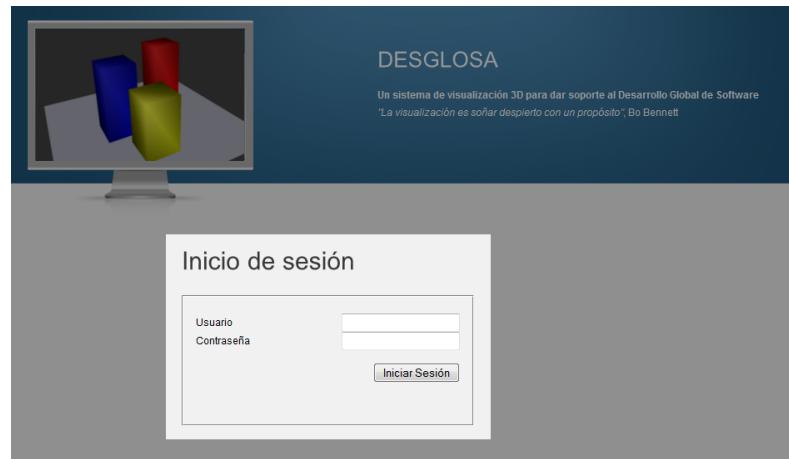


Figura C.2: Formulario de autenticación

 (a)	 (b)
 (c)	 (d)

Figura C.3: Errores obtenidos en el proceso de autenticación: (a) El usuario o la contraseña introducidos son incorrectos (b) La cuenta de usuario ha sido deshabilitada (c) La cuenta de usuario ha sido bloqueada (d) La cuenta de usuario ha caducado

Si el proceso de autenticación se lleva a cabo con normalidad, el usuario será redirigido a la página principal. En ella podrá observar que el nombre de usuario con el que se ha autenticado aparece en la parte inferior izquierda y que los menús de la parte superior izquierda se han configurado automáticamente para permitir el acceso a las funcionalidades que puede desempeñar (Fig. C.4). En este ejemplo se ha utilizado el usuario *admin*.



Figura C.4: Página de inicio para un usuario autenticado. En la parte inferior izquierda se muestra su nombre de usuario y en la parte superior derecha los menús configurados para su rol

C.2 Visualización

La funcionalidad de visualización consta de dos partes: una basada en un **Sistema de Información Geográfica** (SIG o GIS, en su acrónimo inglés Geographic Information System) -en concreto Google Maps-, y otra basada en un generador de gráficos en 3D.

C.2.1 Visualización mediante el Sistema de Información Geográfica

El **sistema de información geográfica** permite conocer las ubicaciones de las factorías que participan en diferentes escenarios. El escenario por defecto muestra las ubicaciones de todas las factorías que se encuentran registradas en el sistema. Los distintos escenarios que se pueden obtener se configuran mediante el *filtro organizacional* que se muestra en la parte superior del mapa (Fig. C.5). En dicho filtro, dividido en tres columnas, se muestra la siguiente información:

- La primera columna representa las compañías dadas de alta y los proyectos en los que colaboran. Cuando se selecciona una compañía, la lista de proyectos se actualiza y muestra únicamente los proyectos en los que colabora dicha compañía. Además, se actualiza la lista de factorías y se emplazan en el mapa todas sus ubicaciones.
- La segunda columna muestra las factorías que pertenecen a la compañía seleccionada en la primera columna y los proyectos en los que colaboran dichas factorías. Cuando se selecciona una factoría, la lista de proyectos es actualizada y su ubicación emplazada en el mapa.
- La tercera columna muestra los proyectos y subproyectos que hay registrados en el sistema. Si el usuario selecciona un proyecto, se actualiza la lista de subproyectos con los subproyectos que lo conforman. Al seleccionar un subproyecto, se muestra en el mapa la ubicación de la factoría que lo desarrolla.

De modo común a las tres columnas, cuando el usuario selecciona un proyecto se ubican en el mapa todas las factorías que participan en su desarrollo.

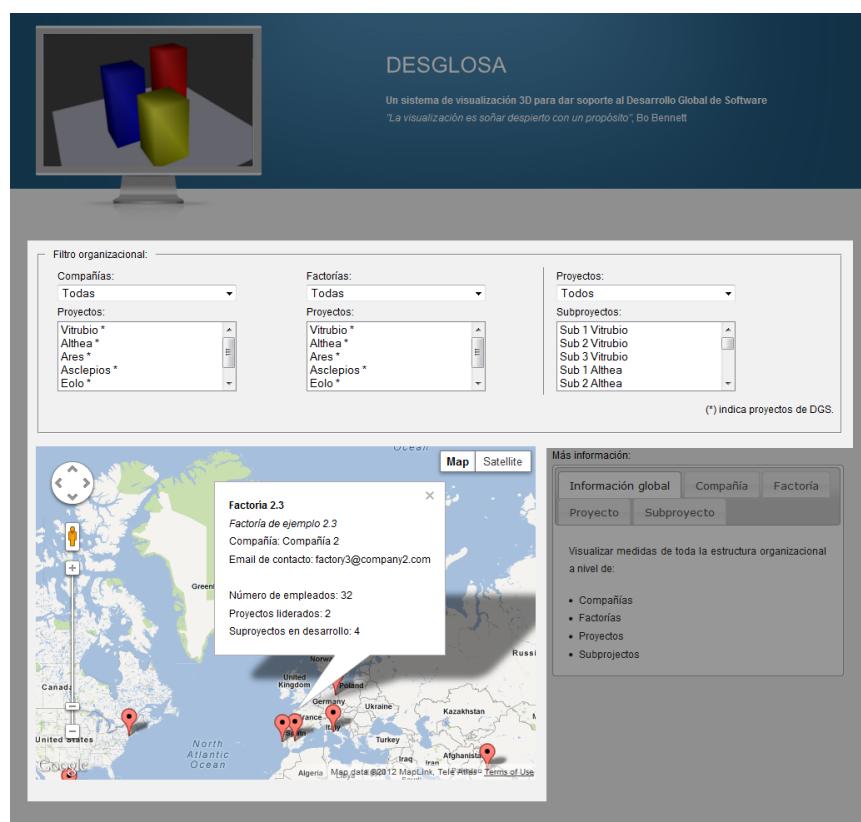


Figura C.5: Página de visualización basada en Google Maps

Por ejemplo, si se selecciona la denominada *Compañía 1*, se observa que participa en dos proyectos (Vitrubio y Valdelvira) y que dispone de cuatro factorías en Argentina, Brasil, México y Portugal (Fig. C.6).

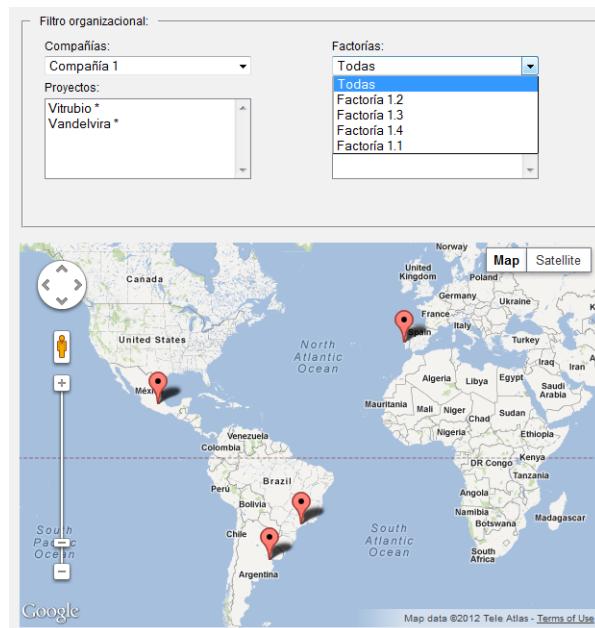


Figura C.6: Resultado de seleccionar la compañía *Compañía 1* en el filtro organizacional

Por otro lado, en el panel “*más información*” que se encuentra a la derecha del mapa se muestra información detallada del elemento seleccionado en el *filtro organizacional*. En la figura C.7 se muestra información detallada acerca del proyecto *Althea*, previamente seleccionado en el filtro de proyectos de la tercera columna. Otros datos como las medidas software del proyecto y sus subproyectos asociados también son mostrados, pero no se incluyen en la figura por no extender el manual en exceso. Nótese que esta funcionalidad también está disponible si se selecciona una compañía, una factoría o un subproyecto.

El panel de *más información* es muy útil ya que no sólo proporciona información detallada acerca del elemento que se esté analizando, sino que también es una puerta de acceso a la visualización de gráficos 3D sobre las medidas e indicadores del elemento.

C.2.2 Visualización de medidas mediante gráficos en 3D

El acceso a la funcionalidad de visualización de medidas e indicadores mediante gráficos 3D se realiza mediante el panel *más información* que se encuentra a la derecha del mapa del mundo. Este panel consta de 5 pestañas: información global, compañía, factoría, proyecto y

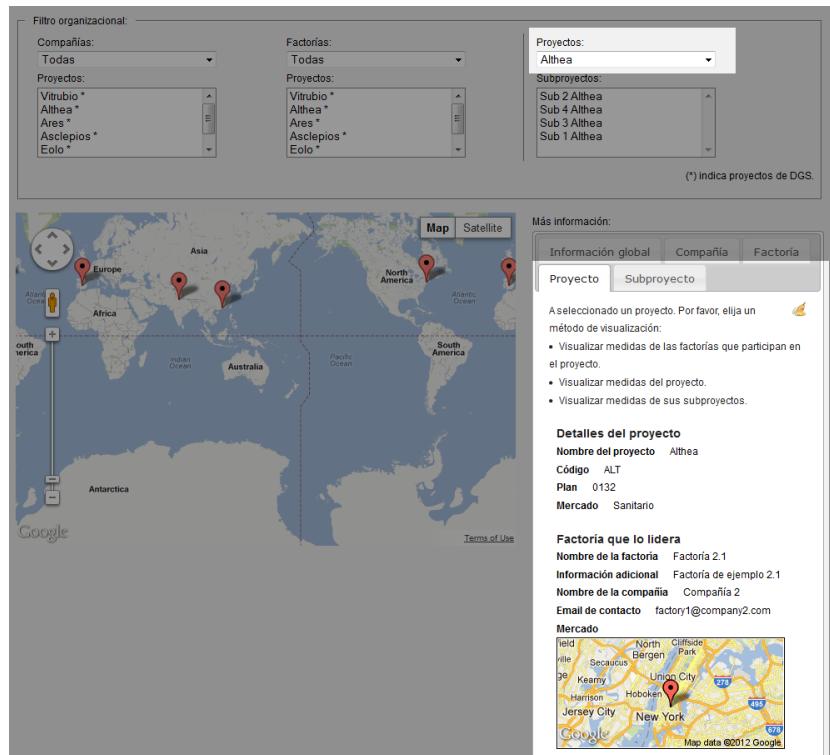


Figura C.7: Panel que muestra información detallada del proyecto seleccionado

subproyecto. Desde la pestaña *información global* se facilita la visualización de medidas de toda la estructura organizacional, comenzando en el nivel de abstracción que se desee (Fig. C.8). En el resto de pestañas se permite consultar información detallada acerca del elemento seleccionado, entre la que se incluyen los valores de sus distintas medidas e indicadores. Además permite acceder a la visualización de medidas basada en gráficos 3D en distintos niveles de abstracción (Fig. C.9).

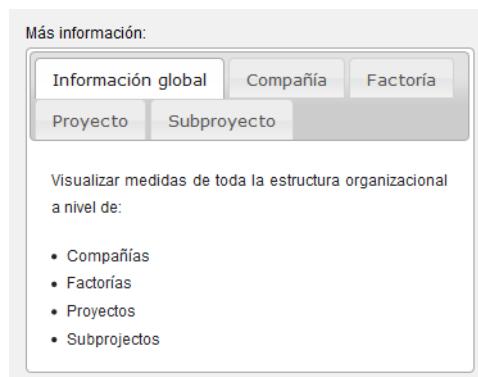


Figura C.8: Pestaña de *información global* que ofrece acceso a la visualización de datos globalizados



Figura C.9: Pestaña *compañía* del panel *más información* que ofrece acceso a la visualización de datos globalizados

La primera vez que el usuario trate de lanzar el motor gráfico, el navegador web tratará de instalar las librerías necesarias para la generación de gráficos 3D. En la figura C.10 se muestra en mensaje emergente que debe ser confirmado por el usuario.



Figura C.10: Mensaje de confirmación para la instalación de librerías de JOGL y librerías nativas

Si el usuario decide visualizar las medidas de un elemento mediante las distintas opciones de visualización que se muestran en el panel *más información*, se abrirá una nueva ventana en la que se deberá seleccionar un perfil de visualización y un método de agrupación.

- Los perfiles de visualización, cuyo proceso de configuración se mostrará en la sección C.3.1, definen determinadas características de configuración que condicionan la forma en la que los gráficos 3D que van a ser generados. Cualidades como el modelo gráfico

que se utilizará en la representación y la relación entre los distintos atributos de las entidades y las dimensiones de los modelos son especificados en dicho perfil.

- El método de agrupación define bajo qué criterio se agruparán los distintos elementos que se representarán en la escena 3D, es decir, cómo se formarán los barrios de la escena. De este modo, será posible agrupar proyectos pertenecientes al mismo tipo de mercado o mostrar qué factorías colaboran para desarrollar los subproyectos que conforman un proyecto.

En la figura C.11 se observa la ventana de selección de perfil y método de agrupación que se muestra al usuario cuando éste desea visualizar las medidas de toda la estructura organizacional a nivel de *compañía*. Al seleccionar ambas opciones y hacer *clic* sobre el botón *OK*, se lanzará el motor gráfico y éste generará la escena en base al perfil seleccionado y la información que se disponga acerca de las compañías.

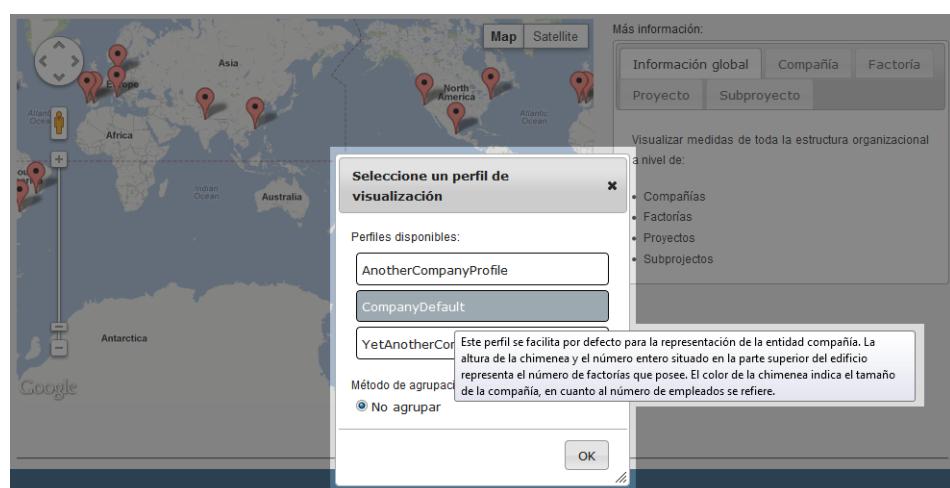


Figura C.11: Ventana de selección de perfil y método de agrupación para la visualización de medidas e indicadores a nivel de compañía

En la figura C.12 se muestra la escena generada a partir de un perfil predefinido. En ella, tal y como está configurado dicho perfil, se muestran tres elementos que se corresponden con cada una de las compañías dadas de alta en el sistema. En cada elemento se representan las siguientes características:

- La altura de la chimenea y el número sobre el edificio indican el número de factorías que posee la compañía.

- El color de la chimenea indica el número de empleados de la compañía: menos de 100 empleados (rojo), entre 100 y 200 empleados (azul), y más de 200 empleados (verde).

A continuación el usuario puede seleccionar un edificio -mediante un clic de ratón- para obtener información detallada acerca de la compañía que representa o descender al nivel de factoría de dicha compañía -mediante doble clic de ratón-.



Figura C.12: Visualización de compañías mediante la metáfora de un polígono industrial. Cada industria representa una compañía

Otros elementos a destacar de la figura C.12 son las etiquetas que se muestran bajo la escena 3D. Estas etiquetas indican qué perfil de visualización se ha seleccionado y la descripción del mismo. Por otro lado, sobre la escena 3D se encuentra un elemento titulado "*controles de navegación*". Al seleccionar este elemento se muestra una tabla informativa que contiene las distintas operaciones que puede realizar el usuario para interaccionar con la escena 3D (Fig. C.13). Preste especial atención a las teclas del teclado y a las acciones del ratón que permiten desplazar y rotar la cámara, ya que así es posible maximizar la eficiencia de la visualización.

Para continuar con el ejemplo se hará doble clic sobre la compañía con más de 200 empleados (la de la chimenea verde). Esta operación permitirá descender al nivel de factoría de dicha compañía. Para ello será necesario seleccionar el perfil de visualización mediante el

▼ Controles de navegación		
Teclado	Ratón	Efecto
W	Presionar clic izquierdo + Arrastrar arriba	Mover adelante
E	Presionar clic izquierdo + Arrastrar derecha	Mover derecha
S	Presionar clic izquierdo + Arrastrar abajo	Mover atrás
Q	Presionar clic izquierdo + Arrastrar izquierda	Mover izquierda
Re Pág	Presionar clic derecho + Arrastrar arriba	Rotar arriba
D	Presionar clic derecho + Arrastrar derecha	Rotar derecha
Av Pág	Presionar clic derecho + Arrastrar abajo	Rotar abajo
A	Presionar clic derecho + Arrastrar izquierda	Rotar izquierda
C		Bajar cámara
Barra espaciadora		Izar cámara
F11		Activar/desactivar sombras
F12		Mostrar/ocultar depurador
Clic izquierdo		Seleccionar objeto
Doble click izquierdo		Navegar al siguiente nivel

Figura C.13: Componente que muestra los controles que pueden emplearse para interaccionar con la escena 3D

cual se configurarán los elementos que representan las factorías y el método de agrupación de las mismas.

En la figura C.14 se muestra el resultado de utilizar un perfil predefinido de factoría y la agrupación por proyectos. Esta vista permite conocer qué factorías de la compañía *Compañía I* están colaborando en un mismo proyecto, ya que cada elemento representa una factoría y cada barrio un proyecto. Según la descripción del perfil, en esta ocasión se representan las siguientes características:

- La altura de la chimenea y el número sobre el edificio representa el número de subproyectos que la factoría está desarrollando.
- El color de la chimenea indica el mercado de especialización de la factoría.
- El tamaño del modelo indica el número de empleados de la factoría.

Como se puede apreciar, en este nivel, aunque se emplea la misma metáfora de visualización que en el nivel de compañía, se está utilizando una dimensión adicional (tamaño del modelo). Además, en la figura C.14 destaca que la factoría con menos empleados es la que tiene asignados más subproyectos y que el tipo mercado de especialización que prevalece sobre todas las factorías es *sanitario*.



Figura C.14: Visualización, mediante la metáfora de un polígono industrial, de los proyectos en los que trabajan las factorías de la compañía *Compañía I*

De manera análoga al nivel de compañía, en este nivel se puede obtener información detallada de una factoría -mediante un clic de ratón sobre el elemento deseado- o visualizar las medidas e indicadores de los proyectos en los que una factoría participa. Si se efectúa un clic sobre ella, se obtendrá el panel que aparece a la derecha de la figura C.14 y se podrá analizar, entre otros, qué proyectos lidera (2) y qué subproyectos tiene en desarrollo (4).

Para continuar con el ejemplo, se hará doble clic sobre esta misma factoría, se seleccionará el perfil predefinido para proyectos y el método de agrupación por *mercado*. En la figura C.15 se muestra el resultado de esta operación. En la escena se observan los cuatro proyectos en los que participa la factoría seleccionada y un conjunto de características definidas por el perfil:

- El número de la esfera superior verde indica las incidencias que se han resuelto en el proyecto.
- El número de la esfera superior roja indica las incidencias que quedan por resolver.
- El ícono de la esfera inferior representa si el proyecto ha superado la auditoría.
- El tamaño de la esfera inferior indica el tamaño del proyecto en número de trabajadores.
- El color de la esfera inferior indica el tipo de mercado al que pertenece el proyecto.

Si además se efectúa un clic sobre el proyecto *Vandelvira*, se obtiene la información que aparece en el panel de la derecha. En él se pueden analizar, entre otros, las medidas e indicadores del proyecto y sus subproyectos asociados (Fig. C.15).

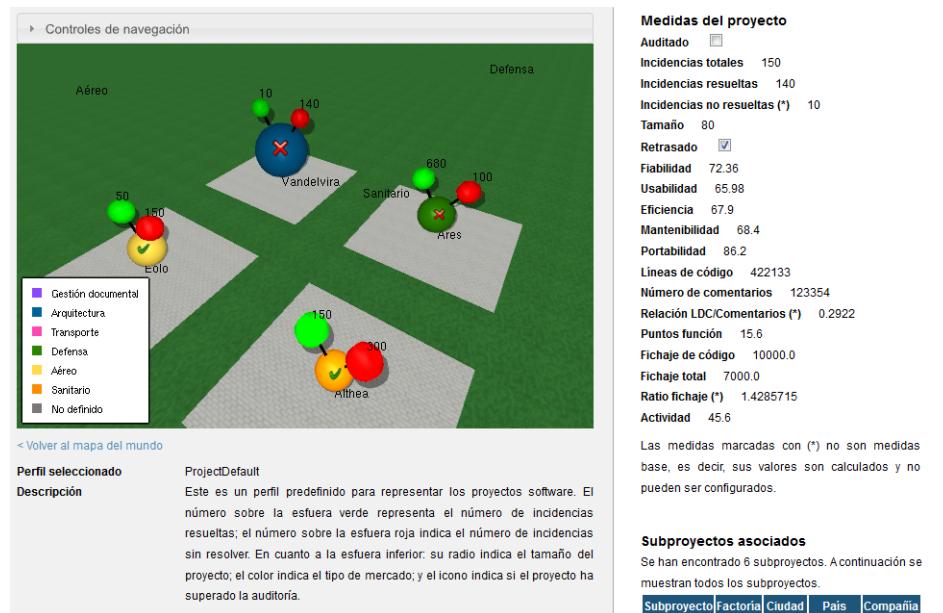


Figura C.15: Visualización, mediante un modelo diseñado para representar proyectos software, de los proyectos en los que participa una factoría determinada

Una vez más, y con un funcionamiento similar al de todos los niveles anteriores, es posible analizar los subproyectos que conforman un proyecto. Para continuar con el ejemplo se efectuará un doble clic sobre el proyecto *Vandelvira*, se seleccionará el perfil predefinido y el método de agrupamiento por factoría. El resultado se muestra en la figura C.16. En ella, y gracias al método de agrupamiento, se puede comprobar qué factorías desarrollan cada uno de los subproyectos que conforman *Vandelvira*. Destaca que la factoría llamada *Factoría 1.3* está llevando desarrollando dos subproyectos de *Vandelvira* y uno de ellos parece que muestra alguna anomalía.

Primero analizaremos las características del perfil:

- El color de la torre indica el rango de valores en el que se encuentra el ratio de fichaje de código y fichaje total estimado: aceptable (verde), marginal (azul) y no aceptable (rojo).
- La altura de la torre indica la fiabilidad.
- El ancho de la torre indica la usabilidad.

- El relleno de la torre indica la portabilidad.
- La profundidad de la torre indica la mantenibilidad.

Con todo ello se deduce que el ratio de fichaje y la portabilidad de uno de los subproyectos que desarrolla la factoría 1.3 presenta algún tipo de anomalía. Si se selecciona esa torre, se corroboran estas deducciones ya que la portabilidad del subproyecto goza de un valor muy bajo (9.8 sobre 100.0) y el ratio de fichaje roza con su límite (0.9 sobre 1.0).

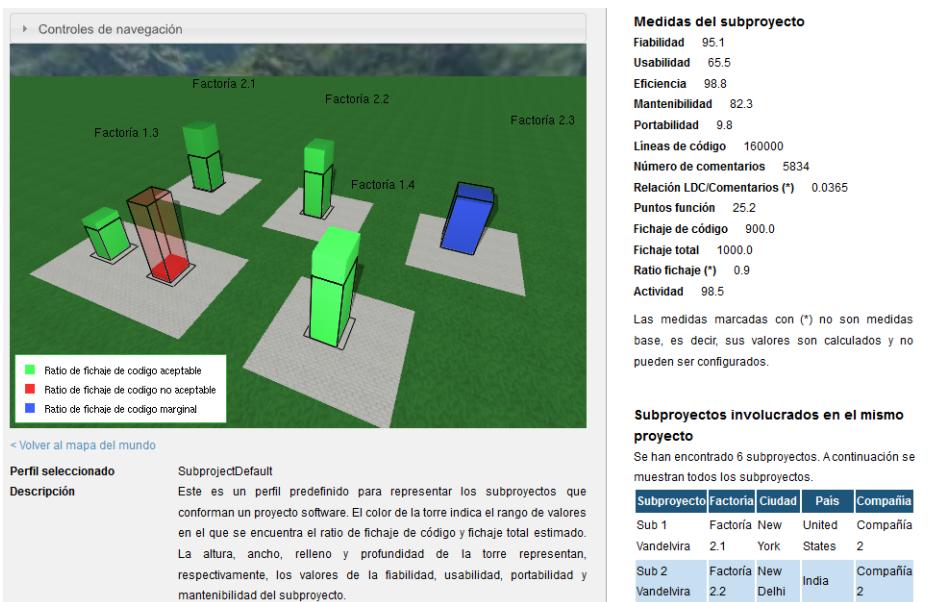


Figura C.16: Visualización, mediante la metáfora de ciudad, de los subproyectos que conforma un proyecto. La agrupación se muestra en base a la factoría que desarrolla cada subproyecto

En el ejemplo que se ha seguido a lo largo de esta sección se ha mostrado cómo descender a través de los distintos elementos de la escena para mostrar información cada vez más específica. En otras palabras, se ha mostrado cómo acceder a las factorías de una compañía específica, a los proyectos en los que participa una factoría determinada y a los subproyectos que conforman un proyecto concreto. No obstante, mediante las opciones facilitadas en la pestaña *información global* es posible acceder a todo este tipo de información comenzando a nivel global de toda la estructura organizacional. En la figura C.17 se muestran, agrupados por proyecto, todos los subproyectos dados de alta en el sistema.

Para finalizar, destacar que se pueden definir tantos perfiles de visualización como sea necesario para cada tipo de entidad (compañía, factoría, proyecto o subproyecto) y emplear en cada situación el modelo gráfico que se estime más oportuno.



Figura C.17: Visualización, mediante la metáfora de ciudad, de todos los subproyectos dados de alta en el sistema. La agrupación se muestra en base a al proyecto al que pertenecen

C.3 Perfiles de visualización

Los perfiles de configuración son ficheros en formato XML que definen qué modelo gráfico será empleado para representar cada entidad. Además, establecen un conjunto de asociaciones entre los atributos de la entidad y las dimensiones del modelo que condicionan la información que se percibirá en el proceso de visualización.

Para gestionar los perfiles de visualización el usuario deberá autenticarse y pertenecer al grupo de administradores. Sólo así, podrá acceder al *panel de control de perfiles de visualización* por medio del menú que se encuentra en la parte superior izquierda de la aplicación web. Este panel de control se muestra en la figura C.18. En él se observa la lista de perfiles de visualización que se encuentran registrados en el sistema. En dicha lista se especifica el nombre del perfil, una descripción del mismo, y la entidad y el modelo gráfico que asocia. Por defecto se ofrece un perfil predefinido para cada entidad:

- Representación gráfica de **compañías** mediante la metáfora de polígono industrial.
- Representación gráfica de **factorías** mediante la metáfora de polígono industrial.
- Representación gráfica de **proyectos** mediante modelos formados por esferas.

- Representación gráfica de **subproyectos** mediante la metáfora de ciudad.

En la parte inferior se observan tres botones que permiten consultar el fichero XML del perfil, eliminarlo o crear un nuevo perfil.

Panel de control: Perfiles de visualización

Se han encontrado 4 perfiles de visualización. A continuación se muestran todos los perfiles de visualización.				
	Nombre del perfil	Descripción del perfil	Entidad asociada	Modelo gráfico asociado
○	CompanyDefault	<p>Este perfil se facilita por defecto para la representación de la entidad compañía. La altura de la chimenea y el número entero situado en la parte superior del edificio representa el número de factorías que posee. El color de la chimenea indica el tamaño de la compañía, en cuanto al número de empleados se refiere.</p>	Compañía	Edificios de factoría
○	FactoryDefault	<p>Este es un perfil predefinido para representar las factorías de software. La altura de la chimenea y el número entero situado en la parte superior del edificio representa el número de subproyectos que está desarrollando. El color de la chimenea indica el tipo de mercado en el que dicha factoría se encuentra especializada. El tamaño del edificio indica el número de empleados de la factoría: 0-50 (pequeño), 51-100 (mediano) y más de 101 (grande)</p>	Factoría	Edificios de factoría
○	ProjectDefault	<p>Este es un perfil predefinido para representar los proyectos software. El número sobre la esfera verde representa el número de incidencias resueltas; el número sobre la esfera roja indica el número de incidencias sin resolver. En cuanto a la esfera inferior: su radio indica el tamaño del proyecto; el color indica el tipo de mercado; y el ícono indica si el proyecto ha superado la auditoría.</p>	Proyecto	Bolas con antenas
○	SubprojectDefault	<p>Este es un perfil predefinido para representar los subproyectos que conforman un proyecto software. El color de la torre indica el rango de valores en el que se encuentra el ratio de fichaje de código y fichaje total estimado. La altura, ancho, relleno y profundidad de la torre representan, respectivamente, los valores de la fiabilidad, usabilidad, portabilidad y mantenibilidad del subproyecto.</p>	Subproyecto	Torres genéricas

[Ver perfil](#) |
 [Eliminar perfil](#) |
 [Crear perfil](#)

Figura C.18: Panel de control para la gestión de perfiles de visualización

C.3.1 Configuración de perfiles de visualización

La configuración de perfiles de visualización se lleva a cabo mediante un asistente compuesto por tres pasos: elección y configuración de la entidad y el modelo gráfico de representación, configuración de dimensiones constantes y leyenda informativa y, por último, nombre y descripción del perfil de visualización.

C.3.1.1 Primer paso: configuración de las asociaciones del perfil

En este primer paso se selecciona qué entidad se desea representar -compañía, factoría, proyecto o subproyecto- y qué modelo gráfico se utilizará para su representación -torres, fábricas o bolas con antenas-. Acto seguido, se deberá configurar las asociaciones entre los atributos de la entidad y las dimensiones del modelo gráfico (Fig. C.19).

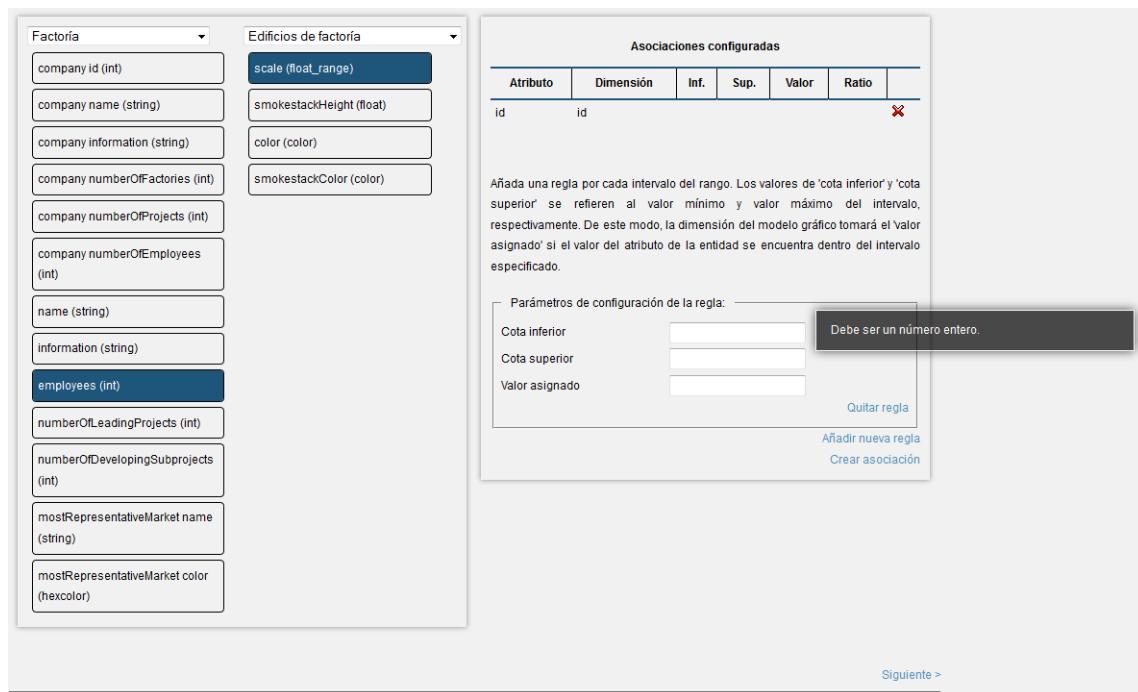


Figura C.19: Primer paso del asistente de configuración de perfiles de visualización

Los tipos de datos que se manejan en los atributos de las entidades y las dimensiones de los modelos son números enteros (*int*), números en punto flotante (*float*), intervalos en punto flotante (*float_range*), cadenas de texto (*string*), valores booleanos (*boolean*) y colores (*color* y *hexcolor*). En la tabla C.1 se muestran las compatibilidades entre tipos de datos. De manera análoga, en la tabla C.2 se muestran las asociaciones que son incompatibles y, por lo tanto, no se pueden establecer.

La flexibilidad que ofrecen todas estas compatibilidades permiten múltiples representaciones gráficas para cualquier tipo de información. Por ejemplo, según la tabla C.1, una dimensión de tipo *color* puede ser asociada con un atributo de tipo *entero*, *punto flotante*, *booleano*, *cadena de texto* o *color*. De este modo es posible asignar un color al modelo en función de las siguientes posibilidades:

- Si el atributo asociado es de tipo *entero* y representa el número de incidencias no resueltas en un proyecto, el color del modelo podría ser verde, si el número de incidencias no resueltas se encuentra bajo un determinado umbral; o rojo, si el número de incidencias no resueltas supera dicho umbral.
- Si el atributo asociado es de tipo *punto flotante* y representa la mantenibilidad del sistema, el color del modelo podría ser verde (aceptable), si la mantenibilidad tiene un

valor entre 90.0 y 100.0; azul (marginal), si el valor de la mantenibilidad se encuentra entre 70.0 y 90.0; y rojo (no aceptable), si el valor de la mantenibilidad es inferior a 70.0.

- Si el atributo asociado es de tipo *booleano* y representa si el proyecto ha superado la auditoria, el color del modelo podría ser verde, si la ha superado satisfactoriamente; o rojo, si no la ha superado (Fig. C.20).
- Si el atributo asociado es de tipo *cadena de texto* y representa el tipo de mercado de un proyecto, el color del modelo podría variar en función del tipo de mercado.

The screenshot shows a user interface for configuring visualization rules. On the left, there are two columns of dropdown menus. The first column under 'Proyecto' includes: id (int), name (string), code (string), plan (string), market name (string), market color (hexcolor), audited (boolean), totalincidences (int), repairedincidences (int), nonRepairedincidences (int), size (int), delay (boolean), fiabilidad (float), usabilidad (float), eficiencia (float), mantenibilidad (float), portabilidad (float), and lineasDeCodigo (int). The second column under 'Torres genéricas' includes: id (int), scale (float_range), height (float), and color (color). The 'color (color)' field is highlighted with a blue background. On the right, the 'Asociaciones configuradas' panel has a header 'Atributo' and columns 'Dimensión', 'Inf.', 'Sup.', 'Valor', and 'Ratio'. A message at the top says 'No se ha creado ninguna asociación.' Below this, a note says: 'Añada una regla por cada uno de los posibles valores que puede tomar el atributo de la entidad y configúrello como valor esperado'. It shows two rule configurations. Rule 1 (unchecked) has 'Valor esperado' set to 'false' and 'Color' set to red. Rule 2 (checked) has 'Valor esperado' set to 'true' and 'Color' set to green. A color picker shows a gradient from green to red, with the current selection being a dark green. Color parameters are also shown: R: 79, G: 219, B: 33, H: 105, S: 85, B: 88, and a hex code #4fdb21.

Figura C.20: Ejemplo para la configuración de reglas mediante las cuales se asignan colores a valores booleanos

Otro ejemplo consiste en configurar una dimensión de tipo *intervalos en punto flotante*. Según la tabla C.1, este tipo de dimensiones son compatibles con atributos de tipo *entero*, *punto flotante*, *booleano* y *cadena de texto*. Así, si la dimensión representa la escala o tamaño del modelo, se podrían establecer las siguientes asociaciones:

- Si el atributo asociado es de tipo *entero* y representa el número de empleados de una factoría, la escala del modelo podría ser 0.5 (más pequeña de lo habitual) si tiene entre

1 y 20 empleados; 1.0 (tamaño normal) si tiene entre 20 y 100 empleados; 1.5 (más grande de lo habitual) si tiene más de 100 empleados. En la figura C.21 se muestra un ejemplo similar.

- Si el atributo asociado es de tipo *punto flotante* y representa la relación entre el fichaje de horas realizadas y el fichaje de horas estimadas de un proyecto, la escala del modelo podría ser 0.75 (más pequeña de lo habitual) si el valor está entre 0.0 y 0.5, es decir, la holgura del fichaje de horas es amplia; 1.0 (tamaño normal) si el valor está entre 0.5 y 0.9, es decir, el fichaje de horas aún goza de holgura; 2.0 (más grande de lo habitual) si el valor es mayor de 0.9 lo cual supone que apenas goza de holgura o se han superado las horas estimadas. De este modo, se resaltan aquellos proyectos cuyo fichaje de horas no goza de holgura o ha superado el valor estimado.
- Si el atributo asociado es de tipo *booleano* y representa si el proyecto está retrasado o no, la escala del modelo podría ser 1.0 (tamaño normal) si no se encuentra retrasado; 1.75 (más grande de lo habitual) si va retrasado. De este modo los proyectos retrasados prevalecerán sobre los proyectos que no lo están.
- Si el atributo asociado es de tipo *cadena de texto* y representa el tipo de mercado de especialización de una factoría, la escala del modelo podría reflejar su valor de forma que destaque más las factorías de un mercado determinado.

Toda esta flexibilidad es posible por medio de una pequeña configuración que se lleva a cabo en el momento de crear la asociación. Cuando una asociación requiera configuración, se mostrará un mensaje de ayuda y se irá indicando con *tooltips* los tipos de datos que debe introducir en cada campo. De este modo, la configuración queda perfectamente guiada y asistida. Esta característica es muy efectiva a la hora de configurar las reglas de intervalos y valores.

Por último, destacar que para un correcto funcionamiento de la visualización, es necesario asociar los campos *id* de la entidad y del modelo gráfico, ya que sólo de este modo será posible identificar qué elemento de la escena representa a una determinada entidad.

Tipo del atributo	Tipo de la dimensión	Aclaración
Entero	Entero	No requiere configuración adicional
Entero	Intervalos en punto flotante	Requiere configurar reglas para especificar intervalos
Entero	Número en punto flotante	Requiere la configuración de un valor que será utilizado en el proceso de normalización (opcional)
Entero	Color	Requiere configurar reglas para especificar intervalos
Entero	Cadena de texto	No requiere configuración adicional
Número en punto flotante	Número en punto flotante	Requiere la configuración de un valor que será utilizado en el proceso de normalización (opcional)
Número en punto flotante	Intervalos en punto flotante	Requiere configurar reglas para especificar intervalos
Número en punto flotante	Color	Requiere configurar reglas para especificar intervalos
Número en punto flotante	Cadena de texto	No requiere configuración adicional
Booleano	Booleano	No requiere configuración adicional
Booleano	Intervalos en punto flotante	Requiere configurar reglas para especificar intervalos
Booleano	Color	Requiere configurar reglas
Cadena de texto	Cadena de texto	No requiere configuración adicional
Cadena de texto	Intervalos en punto flotante	Requiere configurar reglas
Cadena de texto	Color	Requiere configurar reglas
Color en formato hexadecimal	Color	No requiere configuración adicional

Cuadro C.1: Compatibilidades entre tipos de atributo de las entidades y tipos de dimensión de los modelos gráficos

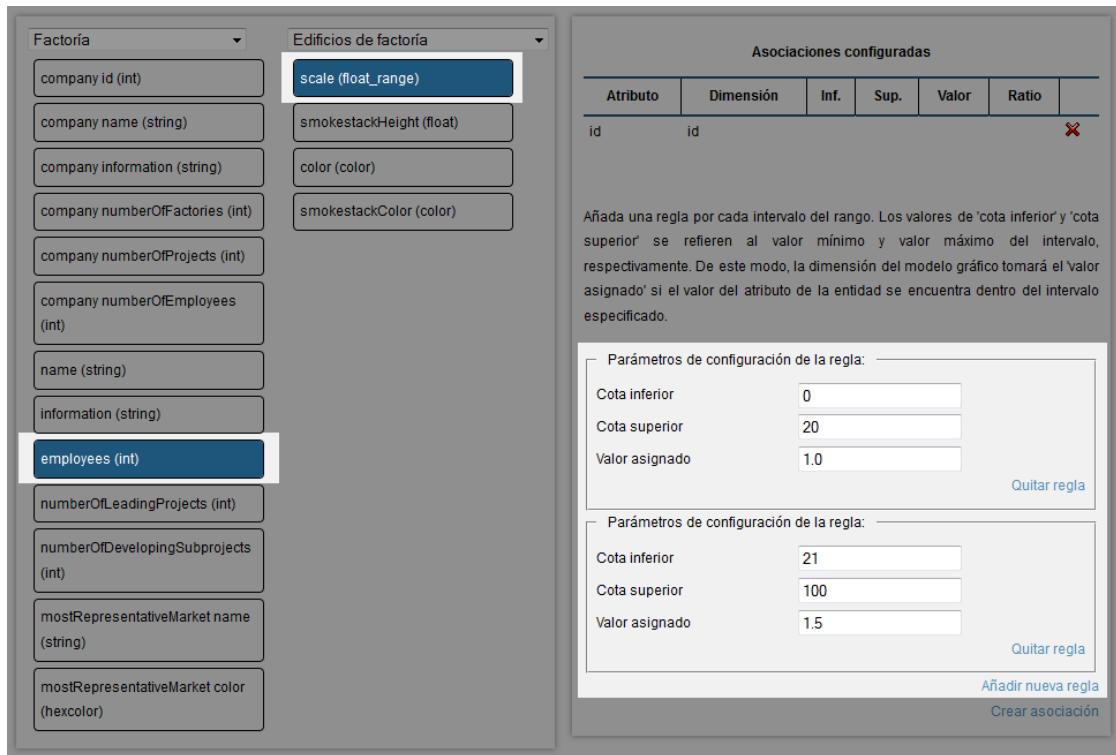


Figura C.21: Ejemplo para la configuración de reglas mediante las cuales se asignan valores en punto flotante a intervalos de números enteros

C.3.1.2 Segundo paso: configuración de las dimensiones no asociadas y la leyenda informativa

En el segundo paso del asistente se configuran dos características. En la figura C.22 se muestra una captura de pantalla.

Por un lado, las dimensiones del modelo que no han sido asociadas con ningún atributo de la entidad podrán tomar valores constantes. Si no se configura ningún valor, éstas tomarán por defecto un conjunto de valores predefinidos.

Por otro lado, se puede configurar una leyenda informativa que será mostrada en la escena. Esta característica es opcional y se pueden añadir tantas líneas informativas como se deseé. Cada línea se compone de un color y una cadena de texto. En la figura C.16 se puede observar un ejemplo de esta característica.

C.3.1.3 Tercer paso: datos básicos del perfil

En el tercer y último paso de configuración del perfil el usuario aportará el nombre del perfil y una descripción tan clara y precisa como sea posible. Sólo de este modo, los usuarios que

Tipo del atributo	Tipo de la dimensión
Entero	Booleano
Número en punto flotante	Entero
Número en punto flotante	Booleano
Booleano	Entero
Booleano	Número en punto flotante
Booleano	Cadena de texto
Cadena de texto	Entero
Cadena de texto	Número en punto flotante
Cadena de texto	Booleano
Color en formato hexadecimal	Entero
Color en formato hexadecimal	Número en punto flotante
Color en formato hexadecimal	Intervalos en punto flotante
Color en formato hexadecimal	Booleano
Color en formato hexadecimal	Cadena de texto

Cuadro C.2: Tipos de atributos y dimensiones incompatibles

Configuración de la leyenda y dimensiones no asociadas

En este paso se podrán añadir tantas líneas a la leyenda informativa como sean necesarias. Además, se permite la configuración de valores constantes a las dimensiones que no han sido asociadas. Las dimensiones a las que no se le asignen un valor constante, tomarán por defecto los valores configurados en el motor gráfico.

Configuración de dimensiones no asociadas

Dimensión	Valor
scale (float_range)	<input type="text"/>
height (float)	<input type="text"/>
color (color)	<input type="color"/>
width (float)	<input type="text"/>
innerHeight (float)	<input type="text"/>
depth (float)	<input type="text"/>

Debe ser un número en punto flotante. Si no se especifica un valor, se tomará 1.0 por defecto.

Configuración de la leyenda informativa

Color	Texto
■	Una línea informativa ×
■	Otra línea ×
■	Y otra más ×

[Nueva línea informativa](#)

[< Atrás](#) [Siguiente >](#)

Figura C.22: Segundo paso del asistente de configuración de perfiles de visualización

hagan uso de la funcionalidad de visualización podrán saber qué beneficios aporta un perfil determinado.

C.4 Gestión de la estructura organizacional del DGS

La gestión organizacional del Desarrollo Global de Software (DGS) se refiere a la gestión de las entidades que conforman su estructura, tales como las compañías, factorías, proyectos y subproyectos. Esta gestión incluye tanto las tareas de consulta, edición, eliminación y creación de este tipo de entidades, así como las operaciones de consulta y edición de diversas medidas e indicadores.

En este manual de usuario, con el propósito de no extenderlo en exceso, únicamente se mostrará la gestión de factorías software y la gestión de medidas e indicadores de proyectos software. Con todo ello, aunque cada entidad goza de sus particularidades, no es necesario entrar en detalle en todas las tareas de gestión ya que se ha seguido el mismo estilo y formato para que el usuario se adapte a la interfaz de un modo rápido y sin esfuerzo.

C.4.1 Gestión de factorías software

La gestión de factorías software, así como la gestión de compañías, sólo está disponible para aquellos usuarios que pertenezcan al grupo de administradores. Por esta razón, para acceder a esta funcionalidad a través del menú de la aplicación, el usuario deberá autenticarse previamente.

Cuando el usuario selecciona la opción de *gestión de factorías* obtiene una pantalla similar a la que se muestra en la figura C.23. En ella se pueden consultar las distintas factorías que se encuentran dadas de alta en el sistema así como diferentes características, tales como el tipo de mercado en el que se están especializadas, el número de empleados que alberga, los proyectos que lidera, etcétera. Además, si se pasar el cursor del ratón sobre los iconos del mundo se muestra un *tooltip* emergente con una imagen que indica su posición en el mapa.

Siguiendo el formato empleado en la gestión de perfiles de visualización, en la parte inferior del panel de control se muestra un conjunto de botones que permiten realizar distintas operaciones. En esta sección se abordan las funcionalidades de consulta, edición, eliminación y creación de factorías, de las cuales las tres primeras requieren la selección previa de una de las factorías de la tabla.

Al seleccionar una factoría y acceder a la funcionalidad *ver factoría* se obtiene una página en la que se muestra información detallada de la factoría, entre la que destaca la lista de proyectos que lidera y la lista de subproyectos que desarrolla (Fig. C.24).

Panel de gestión de factorías

Se han encontrado 11 factorías. A continuación se muestran de 1 a 10.

1 2 Siguiente >

Factoría	Información adicional	Mercado	Empleados	Contacto	Proyecto	Ubicación
Factoría 1.1	Factoría de ejemplo 1.1	-	10	factory1@company1.com		
Factoría 1.2	Factoría de ejemplo 1.2	-	30	factory2@company1.com		
Factoría 1.3	Factoría de ejemplo 1.3	Arquitectura	95	factory3@company1.com		
Factoría 1.4	Factoría de ejemplo 1.4	Arquitectura	63	factory4@company1.com	0	Compañía 1
Factoría 2.1	Factoría de ejemplo 2.1	Sanitario	57	factory1@company2.com	2	Compañía 2
Factoría 2.2	Factoría de ejemplo 2.2	Sanitario	44	factory2@company2.com	0	Compañía 2
Factoría 2.3	Factoría de ejemplo 2.3	Sanitario	32	factory3@company2.com	2	Compañía 2
Factoría 2.4	Factoría de ejemplo 2.4	Arquitectura	115	factory4@company2.com	0	Compañía 2
Factoría 3.1	Factoría de ejemplo 3.1	Defensa	7	factory1@company3.com	1	Compañía 3
Factoría 3.2	Factoría de ejemplo 3.2	Defensa	21	factory2@company3.com	0	Compañía 3

[Ver medidas](#) [Ver factoría](#) [Editar factoría](#) [Eliminar factoría](#) [Crear factoría](#)

Figura C.23: Panel de control para la gestión de factorías software

Proyectos liderados

Los proyectos que se muestran a continuación son aquellos cuya factoría principal o líder es la que está consultando.

Se han encontrado 2 proyectos. A continuación se muestran todos los proyectos.

Proyecto	Código	Plan	Mercado	Subproyectos
Althea	ALT	0132	Sanitario	4
Vandelvira	VDLV	0541	Arquitectura	6

[Ver proyecto](#) [Editar proyecto](#) [Eliminar proyecto](#) [Crear proyecto](#)

Subproyectos en desarrollo

Los subproyectos que se muestran a continuación son aquellos cuyo desarrollo lo desempeña la factoría que está consultando.

Se han encontrado 2 subproyectos. A continuación se muestran todos los subproyectos.

Subproyecto	Proyecto	Código del proyecto	Plan del proyecto	Mercado del proyecto	Subproyectos	Factoría
Sub 1 Vandelvira	Vandelvira	VDLV	0541	Arquitectura	6	Factoría 2.1
Sub 1 Althea	Althea	ALT	0132	Sanitario	4	Factoría 2.1

[Ver subproyecto](#) [Editar subproyecto](#) [Eliminar subproyecto](#) [Crear subproyecto](#)

Figura C.24: Fragmento de la página de consulta de una factoría software

El usuario también puede decidir eliminar una factoría, para lo cual deberá confirmar la operación a través del mensaje emergente que se muestra (Fig. C.25).

La edición y creación de nuevas factorías utilizan el mismo formulario, por lo tanto sólo es necesario abordar una de ellas en este manual. Este formulario se basa en un asistente compuesto por 3 pasos.

- Primer paso: se deberá seleccionar a qué compañía pertenece e introducir los datos

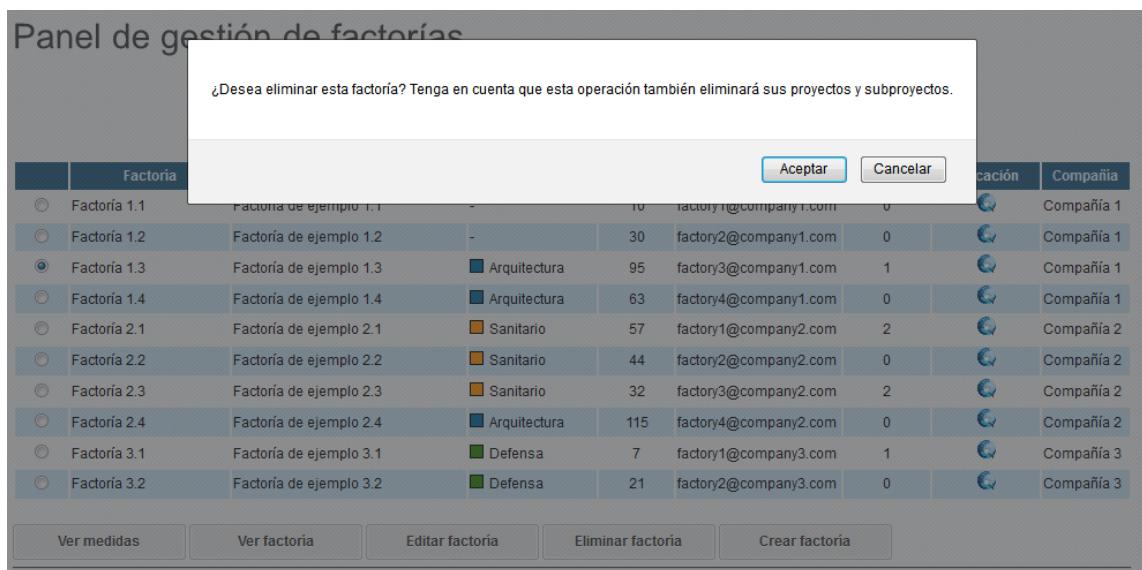


Figura C.25: Mensaje de confirmación para la eliminación de una factoría software

básicos de la factoría, tales como el nombre, información adicional, un email de contacto y el número de empleados que tiene (Fig. C.26).

Formulario de factoría

Este asistente le guiará a través de 3 pasos para configurar una nueva factoría. Deberá introducir la información de la factoría, a qué compañía pertenece, los datos de su director y su ubicación.

Los campos marcados con (*) son obligatorios.

Compañía a la que pertenece

(*) Seleccione a qué compañía pertenece la factoría que desea crear. Si no la encuentra en la siguiente lista, cree una nueva compañía [aqui](#) o contacte con el administrador del sistema.

Se han encontrado 3 compañías. A continuación se muestran todas las compañías.

Compañía	Información adicional
Compañía 1	Compañía de ejemplo número 1.
Compañía 2	Compañía de ejemplo número 2.
Compañía 3	Compañía de ejemplo número 3.

Detalles de la factoría

Nombre de la factoría (*)	Factoría de ejemplo
Información adicional	Descripción de la facto
Email de contacto	factoria@company.cor
Número de empleados	37

Siguiente >

Figura C.26: Primer paso del asistente de creación de una factoría software

- Segundo paso: se deberán aportar el nombre y apellidos del director de la factoría. De

forma opcional se podrá aportar una fotografía del mismo (Fig. C.27).

Este asistente le guiará a través de 3 pasos para configurar una nueva factoría. Deberá introducir la información de la factoría, a qué compañía pertenece, los datos de su director y su ubicación.

Los campos marcados con (*) son obligatorios.

Datos del director

Nombre (*)	Jose Domingo
Apellidos (*)	López López
Foto	D:\alarcos.jpg

< Atrás Siguiente >

Figura C.27: Segundo paso del asistente de creación de una factoría software

- Tercer paso: se deberá indicar la ubicación de la factoría en el mapa. Para realizar esta operación se puede escribir la dirección en los campos facilitados y hacer clic sobre el botón *localizar*, o buscar la ubicación directamente sobre el mapa y hacer clic sobre ella para que el sistema averigüe su dirección (Fig. C.28).

C.4.2 Gestión de medidas

Existen dos tipos de medidas: medidas base y medidas derivadas. Por un lado, las medidas base toman un valor que debe ser aportado por el usuario y, por otro, las medidas derivadas obtienen su valor realizando una serie de cálculos sobre las medidas base.

Si el usuario se encuentra autenticado como administrador o jefe de proyecto podrá acceder a la *gestión de proyectos* y obtener una página como la que se muestra en la figura C.29. Como se puede observar, el estilo y el formato es similar al utilizado en la gestión de factorías (Sección C.4.1). En esta página el usuario puede seleccionar un proyecto y acceder a la funcionalidad *ver medidas*.

En la página de *ver medidas* se obtiene una pantalla como la que se muestra en la figura C.30. En ella se pueden estudiar los valores que toman las medidas base y las medidas derivadas -estas últimas se marcan con un asterisco (*)-.

Cuando el usuario hace clic en el botón *configurar medidas base* que se encuentra en la parte inferior de la página, accede a un formulario que le permite introducir los valores correspondientes a dichas medidas (Fig. C.31).

Formulario de factoría

Este asistente le guiará a través de 3 pasos para configurar una nueva factoría. Deberá introducir la información de la factoría, a qué compañía pertenece, los datos de su director y su ubicación.

Los campos marcados con (*) son obligatorios.

Configure la ubicación de la factoría

Para establecer la ubicación de la factoría puede clickarla sobre el mapa o rellenar los campos que se muestran a continuación y seleccionar "localizar".

Dirección (*)	Paseo de la Universidad
Ciudad (*)	Ciudad Real
Provincia	Ciudad Real
País (*)	Spain
Código postal	13003
<input type="button" value="Localizar"/> <input type="button" value="Limpiar"/>	

Ubicación encontrada.

< Atrás

Figura C.28: Tercer paso del asistente de creación de una factoría software

Panel de gestión de proyectos

Se han encontrado 6 proyectos. Aquí se detallan los principales datos de los proyectos.

Proyecto	Código	Plan	Mercado	Subproyectos	Ciudad	País
Vitruvio	VTRB	0932	Arquitectura	3	Lomas de Zamora	Argentina
Althea	ALT	0132	Sanitario	4	New York	United States
Ares	ARS	0142	Defensa	5	Ciudad Real	Spain
Asclepios	ASCP	0421	Sanitario	2	Berlin	Germany
Eolo	EOL	0094	Aéreo	2	Ciudad Real	Spain
Vandelvira	VDLV	0541	Arquitectura	6	New York	United States

Figura C.29: Panel de control para la gestión de proyectos software

Medidas de proyecto configuradas

Nombre del proyecto	Eolo
Código	EOL
Plan	0094

A continuación puede consultar los valores de las medidas de este proyecto.

Auditado	<input checked="" type="checkbox"/>
Incidencias totales	200
Incidencias resueltas	150
Incidencias no resueltas (*)	50
Tamaño	40
Retrasado	<input type="checkbox"/>
Fiabilidad	81.45
Usabilidad	36.87
Eficiencia	81.23
Mantenibilidad	85.6
Portabilidad	74.1
Líneas de código	1214520
Número de comentarios	55466
Relación LDC/Comentarios	0.0457
(*)	
Puntos función	10.2
Fichaje de código	1000.0
Fichaje total	8790.0
Ratio fichaje (*)	0.11376564
Actividad	67.8

Las medidas marcadas con (*) no son medidas base, es decir, sus valores son calculados y no pueden ser configurados.

Configurar medidas base

Figura C.30: Página para la consulta de medidas de un proyecto software

Configuración de medidas del proyecto

Nombre del proyecto	Asclepios
Código	ASCP
Plan	0421

Por favor, introduzca los valores de las medidas que desea actualizar.

Auditado	<input checked="" type="checkbox"/>
Incidencias totales	300
Incidencias resueltas	0
Tamaño	60
Retrasado	<input type="checkbox"/>
Fiabilidad	97.8
Usabilidad	54.9
Eficiencia	99.7
Mantenibilidad	96.12
Portabilidad	83.45
Líneas de código	4592744
Número de comentarios	6654
Puntos función	30.5
Fichaje de código	5000.0
Fichaje total	16400.0
Actividad	99.9

Debe ser un número en punto flotante.

Actualizar medidas

Figura C.31: Formulario para la configuración de valores de las medidas de un proyecto software

Apéndice D

Código Fuente

D.1 Configuración de Spring Security

La configuración de Spring Security se lleva a cabo mediante el siguiente fichero XML. En él se indica qué recursos y páginas son accesibles por cada rol, así como el comportamiento de la aplicación web ante las distintas operaciones de autenticación, cierre de sesión o accesos denegados.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans:beans xmlns="http://www.springframework.org/schema/
   security"
3   xmlns:beans="http://www.springframework.org/schema/beans"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://www.springframework.org/schema/
   beans
6     http://www.springframework.org/schema/beans/spring-beans-
   2.0.xsd
7     http://www.springframework.org/schema/security
8     http://www.springframework.org/schema/security/spring-
   security-2.0.1.xsd">
9
10 <global-method-security secured-annotations="enabled" />
11
12 <http auto-config="true" access-denied-page="/401.jsp">
13   <intercept-url pattern="/index" filters="none"/>
14   <intercept-url pattern="/login" access="
      IS_AUTHENTICATED_ANONYMOUSLY"/>
15   <intercept-url pattern="/logout" access="ROLE_ADMIN,
      ROLE_MANAGER,ROLE_USER"/>
```

```
16      <intercept-url pattern="/profiles/*" access="ROLE_ADMIN,
17          ROLE_MANAGER,ROLE_USER"/>
18
19      <intercept-url pattern="/jsp/admin/*" access="ROLE_ADMIN
19          "/>
20
21      <!-- companies -->
22
23      <intercept-url pattern="/jsp/viewCompanies.jsp" access="
23          ROLE_ADMIN"/>
24
25      <intercept-url pattern="/jsp/viewCompany*.jsp" access="
25          ROLE_ADMIN"/>
26
27      <intercept-url pattern="/jsp/formCompany*.jsp" access="
27          ROLE_ADMIN"/>
28
29      <!-- factories -->
30
31      <intercept-url pattern="/jsp/viewFactories.jsp" access="
31          ROLE_ADMIN"/>
32
33      <intercept-url pattern="/jsp/viewFactory*.jsp" access="
33          ROLE_ADMIN"/>
34
35      <intercept-url pattern="/jsp/formFactory*.jsp" access="
35          ROLE_ADMIN"/>
36
37      <!-- projects -->
38
39      <intercept-url pattern="/jsp/viewProjects.jsp" access="
39          ROLE_ADMIN,ROLE_MANAGER"/>
40
41      <intercept-url pattern="/jsp/viewProject*.jsp" access="
41          ROLE_ADMIN,ROLE_MANAGER"/>
42
43      <intercept-url pattern="/jsp/formProject*.jsp" access="
43          ROLE_ADMIN,ROLE_MANAGER"/>
44
45      <!-- subprojects -->
46
47      <intercept-url pattern="/jsp/viewSubprojects.jsp" access
47          ="ROLE_ADMIN,ROLE_MANAGER"/>
48
49      <intercept-url pattern="/jsp/viewSubproject*.jsp" access
49          ="ROLE_ADMIN,ROLE_MANAGER"/>
50
51      <intercept-url pattern="/jsp/formSubproject*.jsp" access
51          ="ROLE_ADMIN,ROLE_MANAGER"/>
52
53
54      <intercept-url pattern="/jsp/generate*MeasureView.jsp"
54          access="ROLE_ADMIN,ROLE_MANAGER,ROLE_USER"/>
55
56      <intercept-url pattern="/jsp/generateCompanyMeasureForm.
56          jsp" access="ROLE_ADMIN"/>
57
58      <intercept-url pattern="/jsp/generateFactoryMeasureForm.
58          jsp" access="ROLE_ADMIN"/>
```

```
38      <intercept-url pattern="/jsp/generateProjectMeasureForm.jsp" access="ROLE_ADMIN,ROLE_MANAGER"/>
39      <intercept-url pattern="/jsp/generateSubprojectMeasureForm.jsp" access="ROLE_ADMIN,ROLE_MANAGER"/>
40
41      <intercept-url pattern="/jsp/visualization.jsp" access="ROLE_ADMIN,ROLE_MANAGER,ROLE_USER"/>
42      <intercept-url pattern="/jsp/build*" access="ROLE_ADMIN,ROLE_MANAGER,ROLE_USER"/>
43      <!-- <intercept-url pattern="/**" access="IS_AUTHENTICATED_ANONYMOUSLY"/> -->
44      <form-login login-page="/login" default-target-url="/login?result=success" always-use-default-target="true" authentication-failure-url="/login?result=failed" />
45  </http>
46
47  <authentication-provider user-service-ref='userDetailsService'>
48    <password-encoder hash="md5"/>
49  </authentication-provider>
50
51  <beans:bean id="userDetailsService" class="es.uclm.inf_cr.alarcos.desglosa_web.security.CustomUserDetailsService">
52    <!-- userDao bean is defined in applicationContext-dao.xml -->
53    <beans:property name="userDao" ref="userDao"/>
54  </beans:bean>
55 </beans:beans>
```

Listado D.1: Fichero de configuración de Spring Security: applicationContext-security.xml

D.2 Automatización de pruebas funcionales de una aplicación web mediante Maven

La automatización de las pruebas funcionales para una aplicación web requiere algún método que simule la interacción del usuario con el sistema. Esto precisa, entre otros, de un servidor web que mantenga alojada dicha aplicación. La puesta en marcha del servidor web puede realizarse de forma manual o automatizada. Si se desea automatizar, la complejidad de configuración de Maven aumenta considerablemente.

Para automatizar la puesta en marcha de un servidor web mediante Maven es necesario crear un perfil que indique, antes de la fase de pruebas, cómo descargar, instalar y ejecutar algún contenedor de servlets tipo Apache Tomcat. De este modo, cuando Maven ejecute las pruebas, la aplicación web estará disponible en *host* y puerto que se indique en dicha configuración.

Acto seguido, se indicará al sistema mediante el cual se ejecutan las pruebas funcionales -Canoo WebTest en este caso- en qué servidor se encuentra la aplicación web sobre la que tiene que lanzar los casos de prueba implementados.

La configuración que se ha llevado a cabo para lograr este comportamiento se muestra en el listado D.2.

```
1 <!-- == Testing Profile == -->
2 <!-- Use "mvn clean verify -Pintegration-test" when you want to
     run this
3   profile -->
4 <!-- ===== -->
5 <profile>
6   <id>integration-test</id>
7   <activation>
8     <property>
9       <name>!maven.test.skip</name>
10    </property>
11    </activation>
12    <build>
13      <plugins>
14        <plugin>
15          <groupId>org.codehaus.cargo</groupId>
```

```
16 <artifactId>cargo-maven2-plugin</artifactId>
17 <version>0.3</version>
18 <configuration>
19   <wait>${cargo.wait}</wait>
20   <container>
21     <containerId>${cargo.container}</containerId>
22     <zipUrlInstaller>
23       <url>${cargo.container.url}</url>
24       <installDir>${installDir}</installDir>
25     </zipUrlInstaller>
26   </container>
27   <configuration>
28     <home>${project.build.directory}/${cargo.container}/
29       container</home>
30     <properties>
31       <cargo.hostname>${cargo.host}</cargo.hostname>
32       <cargo.servlet.port>${cargo.port}</cargo.servlet.port>
33       <cargo.context>${project.build.finalName}</cargo.context
34         >
35     </properties>
36   </configuration>
37 </configuration>
38 <executions>
39   <execution>
40     <id>start-container</id>
41     <phase>pre-integration-test</phase>
42     <goals>
43       <goal>start</goal>
44     </goals>
45   </execution>
46   <execution>
47     <id>stop-container</id>
48     <phase>post-integration-test</phase>
49     <goals>
50       <goal>stop</goal>
51     </goals>
52   </execution>
</executions>
</plugin>
```

D.2. AUTOMATIZACIÓN DE PRUEBAS FUNCIONALES DE UNA APLICACIÓN WEB 210 MEDIANTE MAVEN

```
53 <plugin>
54   <artifactId>maven-antrun-plugin</artifactId>
55   <version>1.1</version>
56   <configuration>
57     <tasks>
58       <taskdef resource="webtest_base_relaxed.taskdef">
59         <classpath refid="maven.test.classpath" />
60       </taskdef>
61       <!-- Delete old results file if it exists -->
62       <delete dir="target/webtest/webtest-results" />
63       <mkdir dir="target/webtest/webtest-results" />
64       <!-- This is so the default will be used if no test case
           is specified -->
65       <property name="test" value="run-all-tests" />
66       <echo level="info">Testing '${project.build.finalName}'
           with
67           locale '${user.language}'</echo>
68       <ant antfile="src/test/resources/web-tests.xml" target="${
           test}">
69         <property name="user.language" value="${user.language}" />
70         <property name="webapp.name" value="${project.build.
           finalName}" />
71         <property name="host" value="${cargo.host}" />
72         <property name="port" value="${cargo.port}" />
73         <property name="haltonfailure" value="false" />
74         <property name="haltonerror" value="false" />
75         <property name="loglevel" value="error" />
76         <property name="summary" value="true" />
77         <property name="saveresponse" value="true" />
78         <property name="resultpath" value="target/webtest/
           webtest-results" />
79     </ant>
80   </tasks>
81 </configuration>
82 <executions>
83   <execution>
84     <phase>integration-test</phase>
85   <goals>
```

```
86      <goal>run</goal>
87    </goals>
88  </execution>
89</executions>
90<dependencies>
91  <dependency>
92    <groupId>com.canoo.webtest</groupId>
93    <artifactId>webtest</artifactId>
94    <version>${webtest.version}</version>
95    <!-- groovy-all does not have a pom in central repo
96        exclude groovy -->
97    <!!-- to prevent trying to fetch pom -->
98  <exclusions>
99    <exclusion>
100      <groupId>groovy</groupId>
101      <artifactId>groovy-all</artifactId>
102    </exclusion>
103  </exclusions>
104</dependency>
105</dependencies>
106</plugin>
107</plugins>
108</build>
109</profile>
```

Listado D.2: Snippet para la automatización de pruebas funcionales de la aplicación web mediante Maven

