ESCUELA SUPERIOR DE INGENIERÍA

GRADO EN INGENIERÍA INFORMÁTICA

LENGUAJE DE DOMINIO ESPECÍFICO PARA GENERACIÓN DE APLICACIONES WEB DE PROCESOS ADMINISTRATIVOS

Ismael Jerez Ibáñez

19 de abril de 2016

ESCUELA SUPERIOR DE INGENIERÍA

GRADO EN INGENIERÍA INFORMÁTICA

LENGUAJE DE DOMINIO ESPECÍFICO PARA GENERACIÓN DE APLICACIONES WEB DE PROCESOS ADMINISTRATIVOS

- Departamento: Departamento de Ingeniería Informática
- Directores del proyecto: Inmaculada Medina Bulo y Antonio García Domínguez
- Autor del proyecto: Ismael Jerez Ibáñez

Cádiz, 19 de abril de 2016

Fdo.: Ismael Jerez Ibáñez

Agradecimientos

- A Antonio García Domínguez, por haber sido mi tutor y por su ayuda incondicional cuando he tenido problemas o dudas durante la realización de este trabajo.

- A Francisco Palomo Lozano, por haberme facilitado la puesta en contacto con Antonio García Domínguez para la realización de este TFG.

- A Inmaculada Medina Bulo, por permitirme colaborar con el personal del Área de Informática de la UCA donde se ha podido poner en práctica este trabajo.

- A Juan José Caballero Muñoz, Jesús Franco Oliva, Gerardo Aburruzaga García y José Manuel Frías Carnero, como personal del Área de Informática de la UCA, y a José Luis Ortega Seijo, como desarrollador de la FUECA, que han hecho uso de mi trabajo, con la adaptación y el esfuerzo que eso conlleva.

- A mi familia y amigos, por el interés y la confianza mostrada en el trabajo que he realizado.

ÍNDICE GENERAL

1. Introducción **17**

1.1. Introducción y motivación 17

1.2. Objetivos 19

1.3. Conceptos básicos y tecnologías empleadas 21

1.3.1. Proceso administrativo 21

1.3.2. Lenguaje específico de dominio 22

1.3.3. Ingeniería dirigida por modelos 26

1.3.4. Xtext 30

1.3.5. Epsilon 34

1.3.6. Django 38

2. Planificación **41**

2.1. Metodología de desarrollo 41

2.2. Etapas del trabajo 46

2.2.1. Elicitación y análisis de requisitos 46

2.2.2. Definición del lenguaje específico de dominio 48

2.2.3. Aprendizaje de la tecnología Xtext 49

2.2.4. Desarrollo del lenguaje específico de dominio 49

2.2.5. Aprendizaje de la tecnología Epsilon 49
2.2.6. Aprendizaje de la tecnología Django 50
2.2.7. Desarrollo del generador de la aplicación Django . . . 50
2.2.8. Desarrollo de las pruebas para la aplicación generada . 52
2.2.9. Desarrollo del plugin de Eclipse 52
2.2.10. Documentación 52
2.3. Distribución temporal 53

3. Análisis 57
3.1. Requisitos 57
3.2. Análisis de las herramientas existentes 59
3.2.1. AuraPortal Helium 60
3.2.2. Eunomia Process Builder 60
3.2.3. Bonita BPM 60
3.2.4. Conclusión del análisis de las herramientas 66
3.3. Especificaciones de AdminDSL 67
3.3.1. Requisitos del lenguaje 67
3.3.2. Sintaxis abstracta 68
3.3.3. Sintaxis concreta 73
3.4. Especificaciones del entorno desarrollado para Eclipse 87
3.4.1. Modelo de casos de uso 87
3.4.2. Modelo de comportamiento del sistema 94
3.5. Especificaciones de la aplicación web generada 95
3.5.1. Modelo de casos de uso 95
3.5.2. Modelo conceptual de datos 111
3.5.3. Modelo de comportamiento del sistema 114

4. AdminDSL 121
4.1. Diseño del editor basado en Xtext 121
4.2. Desarrollo del lenguaje AdminDSL 123

4.2.1. Definición del lenguaje 123
4.2.2. Editor 130
4.2.3. Validador 131
4.3. Diseño y desarrollo del entorno para Eclipse 133
4.3.1. Extensión de menú contextual 134
4.3.2. Paquete de instalación/actualización 135
4.4. Herramientas utilizadas 136
4.4.1. Eclipse con Xtext y Epsilon 136
4.4.2. Control de versiones 137

5. Generador de aplicaciones Django 139
5.1. Desarrollo del generador de código 139
5.1.1. Transformador EGX 141
5.1.2. Plantilla principal EGL 142
5.1.3. Plantillas EGL 143
5.2. Diseño de la aplicación web Django generada 146
5.3. Generación de código: esqueleto base 148
5.3.1. Fichero de configuración “settings.py” 149
5.3.2. Fichero de configuración “urls.py” 149
5.3.3. Modelos autogenerados y predefinidos 150
5.3.4. Clases para el manejo y control de formularios 157
5.3.5. Métodos de vistas 162
5.3.6. Señales y receptores 165
5.3.7. Orden *update_states* 167
5.4. Generación de código: AdminDSL a Django 168
5.5. Herramientas utilizadas 195
5.5.1. Eclipse con Xtext y Epsilon 195
5.5.2. PyCharm 195
5.5.3. Control de versiones 196

6. Pruebas **197**
6.1. Plan de pruebas 197
6.1.1. Alcance 197
6.1.2. Tiempo y lugar 197
6.1.3. Naturaleza 198
6.2. Pruebas de AdminDSL 198
6.2.1. Pruebas de validaciones 199
6.2.2. Pruebas de navegabilidad y alcance 200
6.3. Pruebas de la aplicación Django con Selenium 201
6.3.1. Subsistemas y casos de prueba 203
6.3.2. Cobertura de las pruebas 213
6.3.3. Medidas de rendimiento 214

7. Validación **215**
7.1. Artículos 215
7.1.1. Fifth International Symposium on Business Modeling and Software Design (BMSD 2015) 215
7.1.2. Jornadas de Ingeniería del Software y Bases de Datos XX (JISBD 2015) 216
7.2. Aplicaciones 217
7.2.1. Planes de mejora de titulaciones 217
7.2.2. Seguimiento del Plan Estratégico de la Universidad de Cádiz 219
7.2.3. Encuestas sobre puestos de trabajo 219
7.2.4. Registro y seguimiento de presupuestos 221

8. Conclusiones y trabajo futuro **223**
8.1. Resultados obtenidos 223
8.2. Impresiones personales 223
8.3. Trabajo futuro 224

A. Gramática completa de AdminDSL **227**

B. Restricciones de AdminDSL con OCL **233**

B.1. Contexto de *Application* 233

B.2. Contexto de *Role* 233

B.3. Contexto de *Process* 234

B.4. Contexto de *Section* 235

B.5. Contexto de *Group* 235

B.6. Contexto de *Field* 236

B.7. Contexto de *State* 238

B.8. Contexto de *Transition* 239

B.9. Contexto de *StateRole* 240

B.10. Contexto de *PermissionWithTarget* 240

B.11. Contexto de *Permission* 241

B.12. Contexto de *Option* 241

C. Manual de instalación e implantación **247**

C.1. Requisitos de entorno 247

C.2. Entorno para AdminDSL y el generador de aplicaciones Django 248

C.2.1. Herramienta Eclipse Luna 249

C.2.2. Plugins a través de Eclipse 249

C.3. Aplicación web generada 255

D. Manual de usuario **257**

D.1. Cómo empezar: abriendo el editor 257

D.2. Trabajando con AdminDSL 259

D.2.1. Definir nombre de la aplicación 261

D.2.2. Definir roles participantes en la aplicación 261

D.2.3. Definir un proceso administrativo 261

D.3. Generando la aplicación web Django 269

E. Manual del desarrollador **271**

E.1. Crear una feature en Eclipse 271

E.2. Crear extensión de menú contextual en Eclipse 271

E.3. Pruebas con Django + Selenium 274

E.3.1. Desarrollo de pruebas 274

E.3.2. Ejecución de pruebas 279

Bibliografía **281**

ÍNDICE DE FIGURAS

1.1. Ejemplo de flujo de estados de un proceso administrativo. . . 22
1.2. Representación de la arquitectura lógica de Epsilon para los lenguajes y drivers de tareas específicas. 36
1.3. Ejemplo de configuraciones de Django cada vez a mayor escala. 40

2.1. Diagrama de Gantt: parte 1. 54
2.2. Diagrama de Gantt: parte 2. 55

3.1. Bonita BPM: Añadiendo atributos en el proceso. 61
3.2. Bonita BPM: Añadiendo conjunto de datos. 62
3.3. Bonita BPM: Declarando variables o campos globales. 63
3.4. Bonita BPM: Primera ejecución del proceso con formulario asociado. 63
3.5. Bonita BPM: Editando formulario de una tarea. 64
3.6. Bonita BPM: Posicionando campos del formulario y añadiendo validaciones. 65
3.7. Bonita BPM: Añadiendo una bifurcación. 65
3.8. Diagrama de clases: Aplicación. 69
3.9. Diagrama de clases: Procesos. 69
3.10. Diagrama de clases: secciones y entidades. 70

3.11. Diagrama de clases: Relaciones. 70
3.12. Diagrama de clases: Estados. 71
3.13. Diagrama de casos de uso: Entorno desarrollado para Eclipse. 89
3.14. Diagrama de secuencia para el caso de uso: "Crear fichero APDSL". 96
3.15. Diagrama de secuencia para el caso de uso: "Abrir fichero APDSL". 97
3.16. Diagrama de secuencia para el caso de uso: "Guardar fichero APDSL". 98
3.17. Diagrama de secuencia para el caso de uso: "Generar aplicación web". 98
3.18. Diagrama de casos de uso. 99
3.19. Diagrama de casos de uso. 100
3.20. Diagrama conceptual de datos asociado al módulo base de la aplicación web generada. 111
3.21. Diagrama conceptual de datos asociado a cada módulo de proceso administrativo de la aplicación web generada. 112
3.22. Diagrama de secuencia para el caso de uso: "Comenzar proceso". 114
3.23. Diagrama de secuencia para el caso de uso: "Editar proceso". 115
3.24. Diagrama de secuencia para el caso de uso: "Ver proceso". . . 116
3.25. Diagrama de secuencia para el caso de uso: "Listar procesos". 116
3.26. Diagrama de secuencia para el caso de uso: "Eliminar proceso". 117
3.27. Diagrama de secuencia para el caso de uso: "Cambiar Estado proceso" con cambio automático. Véase figura 3.23 para cambio de estado por decisión. 118
3.28. Diagrama de secuencia para el caso de uso: "Guardar borrador". 119
3.29. Diagrama de secuencia para el caso de uso: "Eliminar borrador". 120

4.1. Diagrama de componentes del entorno Xtext para el desarrollo de AdminDSL 122
4.2. Diagrama de componentes del entorno desarrollado en Eclipse. 134

5.1. Diagrama de componentes del entorno Epsilon para el desarrollo del generador de código 140
5.2. Diagrama de componentes de la aplicación Django generada. 147
5.3. Diagrama de clases de los modelos autogenerados y predefinidos. 151
5.4. Diagrama de clases asociadas al manejo y control de formularios. 158
5.5. Diagrama de clases de los modelos generados en cada app asociada a un proceso administrativo. 173

C.1. Marketplace en Eclipse para la instalación de ANTLR IDE 4. . 250
C.2. Ventana de instalación en Eclipse para Epsilon. 252
C.3. Ventana de instalación para OCL Examples and Editors. 254
C.4. Ventana de instalación para AdminDSL. 255

D.1. Ventana de selección de tipo de proyecto. 258
D.2. Ventana para especificar el nombre del proyecto. 258
D.3. Ventana para especificar el nombre del fichero y crearlo. . . . 260
D.4. Ventana para especificar el nombre del proyecto. 270

E.1. Añadiendo plugins a la feature 272
E.2. Estructura de la extensión del menú contextual para Eclipse . 273

CAPÍTULO 1

INTRODUCCIÓN

En este capítulo del documento se presenta la motivación a través de una introducción, así como los objetivos que se tratan de satisfacer con este Trabajo de Fin de Grado (TFG). Además, se ofrece una descripción de conceptos básicos y tecnologías empleadas en el desarrollo del mismo.

1.1. Introducción y motivación

En el Área de Informática [1] de la Universidad de Cádiz se desarrollan múltiples aplicaciones con el fin de gestionar, controlar y mantener procesos administrativos. Estos procesos administrativos se pueden definir como documentos formales a rellenar por un usuario y cuyo ciclo de vida viene dado por diferentes estados, dentro de los cuales ciertos tipos de usuario podrán ver o editar dicho documento.

Este proceso básico se repite continuamente en las aplicaciones que se han ido desarrollando en el Área de Informática, por lo que es muy común la reimplementación de aplicaciones con funcionalidades muy similares. Todo esto resulta en una falta de estándares y pérdida de esfuerzo por parte

del personal de desarrollo, reflejándose en cómo muchas aplicaciones muy parecidas han sido desarrolladas de forma muy diferente, lo cual dificulta el mantenimiento a largo plazo. Dichas aplicaciones se han ido desarrollando en diferentes marcos de trabajo y esto, sumado a la continua actualización de los mismos, hace difícil la integración entre las mismas. También es importante saber que ciertas versiones o marcos de trabajo pueden pasar a estar obsoletos, por lo que puede ser probable una reimplementación completa de las aplicaciones que han hecho uso del mismo. Un ejemplo claro, lo tenemos en el cambio de versión de Symfony 1.x a Symfony 2.x donde no basta con una serie de cambios sino que se requiere la reimplementación total de la aplicación (esta reimplementación ha sido ofertada como propuesta de trabajo de fin de grado para una determinada aplicación en el Área de Informática).

No está de más decir que es evidente que una aplicación desarrollada bajo una correcta documentación y a partir de una descripción de procesos es, sin duda, más fácil de mantener. Esto último es muy importante, ya que durante la fase de producción y ejecución de las aplicaciones que se demandan, se suelen tener que introducir cambios a petición de los clientes y, en muchos casos, estos cambios se tienen que hacer por personal que no ha participado durante la fase de desarrollo y que no tiene disponible una buena documentación. Además, si nos remontamos a una fase más temprana al desarrollo, ocurre que, en la mayor parte de los casos, los requisitos no han quedado lo suficientemente claros antes de comenzar a desarrollar la aplicación y esto implica un mayor número de cambios posteriores durante la misma fase de desarrollo o la fase de producción.

En general, las descripciones de los requisitos pueden no estar lo suficientemente detalladas o no tener en cuenta restricciones que se ven reflejadas posteriormente debido a las tecnologías que se van a utilizar, por lo que poseer una descripción situada entre los requisitos, el desarrollo y el

producto final puede ser de gran utilidad.

Dados los problemas que se han mencionado anteriormente, lo ideal sería poder trabajar con un marco que permitiese definir estos procesos administrativos de una forma sencilla, clara y ordenada, aportando una estructura base bien documentada y estandarizada de una aplicación que permita gestionar, controlar y mantener dichos procesos. Esta idea principal es la que ha impulsado a la realización de este TFG que consiste en el desarrollo de un lenguaje específico de dominio («*Domain Specific Language*» o DSL) denominado AdminDSL que permita definir procesos administrativos con todo lo que eso conlleva (formulario representativo, roles de usuarios participantes, estados del proceso, permisos de visibilidad y edición, etc.).

A partir del código definido a través de este DSL, se generará la aplicación web en el marco de trabajo correspondiente. No obstante, el DSL es independiente de cualquier marco de trabajo para aplicaciones web, pero debe ser lo suficientemente flexible y extensible para permitir su transformación en cualquiera de estos, como pueden ser Symfony [2] y Django [3] por ejemplo, sin perder la correcta representación del proceso que ha sido definido a través del mismo.

El desarrollo del DSL no es el único elemento que abarcará este TFG, sino que además se ha añadido el desarrollo de un generador de aplicaciones en el marco de trabajo Django a partir de procesos definidos mediante este DSL.

1.2. Objetivos

El objetivo principal de este trabajo, como el de cualquier proyecto de ingeniería, es elaborar y llevar a acabo una solución que resuelva los problemas que se han presentado anteriormente de la mejor forma posible. Esta solución consiste principalmente en el desarrollo de un lenguaje específico

de dominio que permita ofrecer una descripción completa de procesos administrativos y de un generador que elabore una aplicación web de forma automatizada a partir de la descripción aportada por el código que utiliza este DSL.

De esta forma, podemos enumerar los siguientes objetivos principales:

1. Aportar una herramienta para describir procesos administrativos de la forma más completa, clara y sencilla posible. Objetivo que impulsa el desarrollo de un lenguaje específico de dominio.

2. Facilitar la generación de aplicaciones web con una estructura base estándar y fácil de mantener a partir de la descripción de procesos administrativos. Objetivo que impulsa el desarrollo de un generador de código capaz de construir una aplicación web en el marco de trabajo Django que permita gestionar, controlar y mantener los procesos administrativos que se describan.

Estos dos objetivos principales cubren con los siguientes objetivos, en el contexto de aplicaciones para la gestión de procesos administrativos:

- Mejorar la comprensión y el cumplimiento de los requisitos.
- Disminuir el esfuerzo del personal de desarrollo.
    - Evitar implementación desde cero de aplicaciones similares.
    - Facilitar el desarrollo de las aplicaciones.
    - Evitar la reimplementación de aplicaciones web ya desarrolladas debido a cambios de versiones o tecnologías obsoletas.
- Estandarizar los procedimientos de desarrollo y la estructura base de las aplicaciones desarrolladas, con el fin de:
    - Facilitar el mantenimiento.

- Facilitar la integración entre las diferentes aplicaciones desarrolladas.

Cabe decir que este TFG será publicado bajo la licencia Eclipse Public License 1.0, una licencia de software de código abierto utilizada por la Fundación Eclipse para su software que sustituye a la Licencia Pública Común (CPL) y elimina ciertas condiciones relativas a los litigios sobre patentes. Además vendrá acompañado de una correcta documentación en español donde se presentará, entre otras cosas, un manual de instalación e implantación, así como un manual de usuario que permita facilitar tanto el uso del DSL como el uso a nivel de desarrollador y de usuario final de las aplicaciones web Django generadas.

1.3. Conceptos básicos y tecnologías empleadas

A continuación, se describen conceptos importantes para poder comprender mejor el contexto y la finalidad de este TFG tal y como se ha explicado anteriormente. Además se describen las tecnologías que se han empleado para el desarrollo del mismo.

1.3.1. Proceso administrativo

Un proceso administrativo es una secuencia de pasos en los cuales se rellena un documento formal por una o varias personas. A la vista de una aplicación web, se puede entender perfectamente como un formulario en línea a rellenar por uno o varios usuarios en diferentes fases.

Un proceso administrativo debe transitar por diferentes estados según la decisión de uno de sus participantes o bien, según una determinada fecha de inicio o una determinada fecha de finalización o ambas. Durante los distintos estados o fases por las que pasa un proceso administrativo, ciertos usuarios podrán ver o editar partes del documento según sea conveniente.

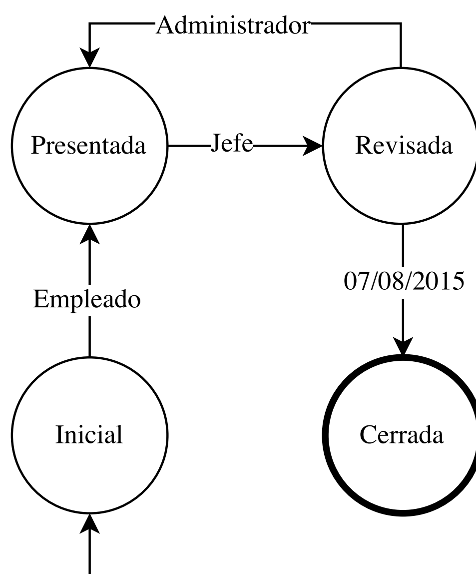


Figura 1.1: Ejemplo de flujo de estados de un proceso administrativo.

Generalmente, los procesos administrativos acaban en un estado cerrado, es decir, un estado a partir del cual no existe ninguna transición a otro estado, y suele ser de aceptación o rechazo.

Este flujo de estados puede contener una secuencia circular de transiciones que permita, para un mismo proceso administrativo, visitar de nuevo ciertos estados. Un autómata puede representar perfectamente el flujo de estados de un proceso administrativo, como podemos observar en el ejemplo de la figura 1.1, donde se representa el flujo de estados de un proceso administrativo con cuatro fases y cuatro transiciones posibles, tres de ellas por decisión de un usuario y una de ellas por superación de una fecha límite.

1.3.2. Lenguaje específico de dominio

Un lenguaje específico de dominio o DSL [4] es un lenguaje de programación computacional de expresividad limitada centrado en un dominio particular.

Hay cuatro elementos clave en esta definición:

- Lenguaje de programación computacional: Un DSL es utilizado por personas para dar instrucciones a un ordenador con el fin de que haga algo. Como con cualquier lenguaje de programación moderno, su estructura está diseñada para facilitar su comprensión por parte de las personas, pero también debería ser algo ejecutable por un ordenador.

- Naturaleza del lenguaje: Un DSL es un lenguaje de programación y, como tal, debería tener un sentido de la fluidez donde la expresividad viene no sólo de expresiones individuales sino también de la manera en la que pueden ser compuestas.

- Expresividad limitada: Un propósito general en los lenguajes de programación aporta muchas capacidades: permitir variedad en los datos, control y estructuras de abstracción. Todos estas capacidades son útiles pero hacen que el lenguaje sea más difícil de aprender y usar. Un DSL supone un mínimo de características necesarias para soportar su dominio. No se puede construir un sistema software completo en un DSL. En su lugar, se usa un DSL para un aspecto particular de un sistema.

- Centrado en un dominio: Un lenguaje limitado solamente es útil si está centrado claramente en un dominio pequeño. Este dominio, en el que se centra el lenguaje, es lo que hace que la limitación del lenguaje merezca la pena.

Lo contrario de un DSL recibe el nombre de GPL o lenguaje de propósito general, como por ejemplo Java u otro lenguaje común de programación. Con un GPL se puede resolver cualquier problema que pueda resolverse por un ordenador, pero no siempre de la mejor forma posible.

Los lenguajes específicos de dominio pueden dividirse en tres categorías principales: DSL externos, DSL internos y lenguajes de bancos de trabajo («*workbenches*»).

- Un DSL externo es un lenguaje separado del lenguaje principal de la aplicación con la que se trabaja. Generalmente, un DSL externo tiene una sintaxis personalizada, pero también los hay que utilizan una sintaxis de otro lenguaje como XML. Ejemplos de DSL externos pueden ser expresiones regulares, SQL, Awk, y ficheros de configuración XML para sistemas como Struts e Hibernate.

- Un DSL interno es una manera particular de usar un lenguaje de propósito general. Se puede entender como un subconjunto de un lenguaje de propósito general ya que sólo utiliza una parte de sus características para poder manejar un aspecto pequeño de la totalidad de un sistema. Un ejemplo clásico lo podemos encontrar en Lisp. Los programadores hablan a menudo sobre la programación en Lisp como la creación y el uso de DSL.

- Un lenguaje de banco de trabajo o lenguaje «*workbench*» es un IDE especializado para la definición y construcción de DSL. En particular, un lenguaje «*workbench*» es utilizado no sólo para determinadas estructuras de un DSL sino también para ediciones personalizadas de entornos enfocado a personas que escriben scripts de DSL.

Los DSL son cada vez más usados y, por lo tanto, el desarrollo de los mismos es cada vez más frecuente. Esta popularidad viene dada principalmente por dos razones:

- La mejora de la productividad para los desarrolladores: Un DSL bien escogido puede facilitar mucho la comprensión de un complicado bloque de código, además de mejorar la productividad de aquellos que trabajan con el mismo.

- La mejora de la comunicación con expertos del dominio: Ofrece un texto común que actúa tanto de software ejecutable como de descripción que puede ser leída por expertos en el dominio permitiéndoles representar sus ideas sobre un sistema.

En este TFG, el DSL está enfocado a describir procesos administrativos y por lo tanto trata de ser lo más representativo posible sin entrar demasiado en detalles de desarrollo web u otros tipos de transformaciones. Sin embargo, se ha querido colocar elementos opcionales que permitan definir aspectos propios del marco de trabajo al que se va a transformar, sin que esto afecte a las transformaciones en otros marcos.

El desarrollo de un DSL viene dado generalmente por las siguientes tres fases [5]:

1. Diseño del metamodelo.

2. Desarrollo del modelo de presentación.

3. Desarrollo de transformaciones.

Diseño del metamodelo

Un metamodelo es un conjunto de aspectos y conceptos del dominio a modelar y las relaciones entre los mismos. Por lo tanto, para el diseño de un metamodelo es muy importante conocer en profundidad el dominio que se desea modelar y ver qué elementos del mismo se van a poder representar a través del DSL. Además, es importante diseñar el metamodelo de la forma más abierta posible, es decir, diseñarlos de forma que ofrezcan la mayor adaptabilidad frente a cambios en el dominio, por ejemplo, generalizando elementos concretos en un elemento padre que sirva para poder añadir nuevos elementos de ese mismo tipo en caso de que se requieran en un futuro.

Desarrollo del modelo de presentación

Escoger el modelo de presentación del DSL es muy importante de cara al usuario que lo va a utilizar y del problema que se va a resolver mediante el uso del mismo. La facilidad de adaptación y uso del lenguaje para el usuario, así como la claridad y el detalle en la representación del dominio vienen dados, en gran parte, por cómo se ha desarrollado el modelo de presentación.

Dado el modelo de presentación, existen dos tipos de DSL:

- Textuales: Basados en una gramática, suelen ser más expresivos y menos limitados.
- Visuales: Basados en elementos gráficos, suelen ser más fáciles de entender, pero a su vez suelen restringir más el metamodelo a los elementos que se hayan definido para su presentación.

En AdminDSL, se ha utilizado un modelo de presentación textual similar a algunos de los lenguajes de programación más habituales, como pueden ser C, C++ o Java. Este modelo se especificará con más detalle a lo largo de este documento.

Desarrollo de transformaciones

Generalmente, un DSL no está enfocado a ofrecer una visión meramente descriptiva del dominio, si no que suele tener asociado un generador que permita transformar el código que utiliza este lenguaje en otro modelo o texto.

1.3.3. Ingeniería dirigida por modelos

La ingeniería dirigida por modelos («*Model Driven Engineering*» o MDE) [6] es una práctica general enfocada en la creación de modelos, o abstracciones

de un dominio, centrándose más en algunos conceptos del mismo y dejando de lado conceptos de la informática. Un modelo es una descripción de un sistema escrito en un lenguaje bien definido. Un lenguaje bien definido es un lenguaje con formas (sintaxis) y significados (semántica) bien definidos, los cuales pueden ser interpretados automáticamente por un ordenador.

Todo modelo posee las siguientes características:

- Representa de forma más sencilla algo más complejo.
- Sirve para un propósito concreto: describir y especificar cómo es un objeto (no tiene por qué ser un objeto software).
- Es más fácil trabajar con un modelo que con el objeto que representa.
- Un metamodelo puede entenderse como un modelo de un modelo. Por lo tanto, un modelo es una abstracción y un metamodelo es una abstracción de mayor nivel que permite describir las propiedades del modelo en sí mismo.

Los metamodelos pueden ser utilizados de las siguientes formas:

- Como un esquema de datos semánticos que necesitan ser intercambiados o almacenados.
- Como un lenguaje para soportar un proceso particular.
- Como un lenguaje para expresar una semántica adicional de información existente.

Enfoque Dirigido por Modelos

En un enfoque Dirigido por modelos (MDE) el sistema de modelos tiene suficiente detalle como para permitir la generación de un sistema completo de aplicación de los modelos propios. De hecho, “el modelo es el código”, es

decir, la atención se centra en el modelado y el código es generado mecánicamente a partir de modelos. Sin embargo, MDE no sólo se centra en las transformaciones o generaciones de código, sino que los modelos pueden ser utilizados para otros tipos de análisis no relacionados con el software o como soporte para tomar decisiones.

Si bien, el enfoque escogido para este TFG es el que viene directamente relacionado a las transformaciones. Respecto a estas, existen dos tipos que se presentan a continuación:

- Modelo a Modelo (M2M): Tiene sus principios en la creación automática de m modelos objetivos a partir de n modelos fuentes.

- Modelo a Texto (M2T): Sus principios parten de la generación automática de texto, generalmente código de un lenguaje de programación a partir de uno o varios modelos.

Para este TFG, el enfoque utilizado ha sido el de transformación M2T. Es decir, a partir del metamodelo definido se genera código escrito conformándose, como resultado final, una aplicación web. Además, no sólo se genera código asociado a Django, sino también, por ejemplo, a "shell scripts" para la instalación de dependencias iniciales de la aplicación.

Continuando con el enfoque general y realizando una comparación entre MDE y la orientación a objetos: en MDE "todo es un modelo", y en la orientación a objetos "todo es un objeto". MDE es un enfoque abierto e integrador que abarca muchos otros espacios tecnológicos de manera uniforme. Un espacio tecnológico de trabajo es un contexto con una serie de conceptos relacionados, el cuerpo de conocimientos, herramientas, conocimientos necesarios y posibilidades.

La tecnología de Ingeniería Dirigida por Modelos ofrece un enfoque prometedor para hacer frente a la incapacidad de los lenguajes de tercera generación en aliviar la complejidad de las plataformas y expresar conceptos

de dominio de manera eficaz.

Objetivos de MDE

El principal objetivo de este enfoque surge por la necesidad de separar de manera muy clara la lógica de negocio y la tecnología utilizada. Es decir, se habla de un principio fundamental en la ingeniería del software denominado como la separación de las preocupaciones (en inglés, «*separation of concerns*»). Este principio afirma que un determinado problema implica diferentes tipos de preocupaciones, que deben ser identificadas y separadas para hacer frente a la complejidad, y para lograr los factores de calidad de ingeniería tales como la robustez, adaptabilidad, mantenibilidad y reutilización.

Cumpliendo con este objetivo, el enfoque dirigido por modelos permite, entre otras cosas:

- Generar software nuevo a partir de modelos.
- Apoyar a los desarrolladores en su productividad.
- Realizar el proceso de construcción de software a través de modelos durante todo del ciclo de vida del software.
- Generar los cambios en las partes del modelo en lugar de generarlos en el sistema que representa.

Comparando estos objetivos con los objetivos iniciales especificados en este documento, vemos como existe varias coincidencias, por lo que convierte a este enfoque en un buen candidato como tecnología de este TFG.

Beneficios de MDE

El MDE está destinado a aumentar la productividad al máximo, la compatibilidad entre sistemas, simplificando el proceso de diseño, y promovien-

do la comunicación entre los individuos y los equipos que trabajan en el sistema. Este aumento de productividad se lleva a cabo tanto a corto plazo (facilitando el análisis, especificación y desarrollo del sistema) como a largo plazo (facilitando el mantenimiento del sistema).

Un segundo aspecto del MDE, y estratégicamente más importante, es la reducción de la sensibilidad de los primeros objetos de arte al cambio. A continuación, se indican cuatro formas fundamentales de los cambios que son de particular importancia:

- Personal: El desarrollo del conocimiento referente al sistema no debe ser sólo almacenado en la mente de los desarrolladores, esta información se perderá siempre que se produzcan cambios de personal o se asignen individuos a mantener sistemas que no han sido creados por ellos mismos.

- Requisitos: Los cambios de los requisitos es un problema conocido en la ingeniería del software. Todos estos cambios deben tener un bajo impacto sobre los sistemas existentes en términos de mantenimiento y no perturbar los sistemas que ya se encuentran en producción.

- Plataformas de desarrollo: Están en un estado de constante evolución. Para disociar la vida de un software de la herramienta de desarrollo utilizada para su creación inicial, es necesario disociar el objeto o el modelo que representa el objeto de la herramienta de desarrollo.

- El despliegue de plataformas. Para aumentar la vida útil de los objetos de software es necesario protegerlos contra los cambios en el despliegue de la plataforma.

1.3.4. Xtext

Xtext [7] es un marco de trabajo para el desarrollo de lenguajes de programación y lenguajes específicos de dominio. Cubre todos los aspectos de

una infraestructura de lenguaje completa, desde analizadores, pasando por enlazadores, compiladores o intérpretes hasta su integración en Eclipse de forma totalmente automatizada, aportando un editor, un validador, una vista fuera de línea, etc.

Xtext aporta un conjunto de lenguajes específicos de dominio y APIs modernas para describir los diferentes aspectos del lenguaje a desarrollar. Basado en esa información, ofrece una implementación completa del lenguaje ejecutándose en una máquina virtual de Java. Los componentes del compilador del lenguaje a desarrollar son independientes de Eclipse y pueden ser usados en cualquier entorno Java. Incluye tales cosas como el analizador, el árbol de sintaxis abstracta (AST), el serializador y el formateador de código, el marco de trabajo del ámbito y el enlazador, pruebas de compilación y analizador estático también conocido como validador y, por último, un generador de código o intérprete. Estos componentes vienen integrados y están basados en el marco de trabajo de modelado de Eclipse [8] («*Eclipse Modeling Framework*» o EMF), lo cual permite usar Xtext junto con otros marcos de trabajo EMF, como por ejemplo, el proyecto de modelado gráfico o GMF.

En este TFG, se ha empleado Xtext exclusivamente para el desarrollo del lenguaje específico de dominio AdminDSL, sin utilizar su componente generador de código o intérprete (tarea reservada para Epsilon [9]). Esta selección se debe a las múltiples ventajas que ofrece Xtext con respecto a otros marcos de trabajo de desarrollo de lenguajes específicos de dominio y que se presentan a continuación:

- Notación clara y fácil de aprender: Muy parecida a la notación de Bison [10] para definir reglas gramaticales. También se utilizan notaciones en Xtend (un lenguaje heredado de Java, más abstracto y enfocado en el dominio de Xtext) o el mismo lenguaje Java para editar el resto de componentes que se ofrecen.

- Analizadores léxico y gramatical van de la mano: En Xtext se incluyen ambos analizadores, de forma que se puede comenzar a definir un lenguaje directamente sin necesidad de separar el análisis léxico del gramatical. Además, provee definiciones por defecto como la introducción de líneas de comentario con las palabras reservadas « // » (para una sola línea de comentario), « /* » y « */ » (varias líneas de comentario).

- Editor de texto del lenguaje: Xtext provee de forma totalmente automatizada un editor de texto integrado en Eclipse que viene provisto de un analizador estático que obliga a respetar las reglas que se definen a través del metamodelo, mostrando errores, avisos y resaltado de líneas o palabras reservadas, entre otras cosas.

- Estructura bien definida y fácil de controlar: De forma general, ofrece dos directorios diferentes cada uno con un archivo por cada componente. Uno de los directorios contiene archivos en formato de lenguaje Xtend y otro en formato de lenguaje Java, de forma que se permite elegir en qué lenguaje se desea programar cada componente, por ejemplo, el proveedor del ámbito del lenguaje se puede programar en Xtend mientra que el validador puede ser programado en Java para mejorar su eficiencia.

Alternativas a Xtext

Una de las alternativas más conocidas es el uso de Flex y Bison [10]. Estas dos tecnologías juntas permiten definir de forma completa un compilador para un lenguaje específico de dominio.

Flex es un una herramienta que permite generar analizadores léxicos. A partir de un conjunto de expresiones regulares, busca concordancias en un fichero de entrada y ejecuta acciones asociadas a estas expresiones. Es

compatible casi al 100 % con Lex, una herramienta clásica de Unix para la generación de analizadores léxicos, pero es un desarrollo diferente realizado por GNU bajo licencia GPL. Bison, por otro lado, es un generador de analizadores sintácticos de propósito general que convierte una descripción para una gramática independiente del contexto (en realidad de una subclase de éstas, las LALR) en un programa en C que analiza esa gramática. Es compatible al 100 % con Yacc, una herramienta clásica de Unix para la generación de analizadores léxicos, pero es un desarrollo diferente realizado por GNU bajo licencia GPL. Todas la gramáticas escritas apropiadamente para Yacc deberían funcionar con Bison sin ningún cambio. Usándolo junto a Flex esta herramienta permite construir compiladores de lenguajes.

A diferencia de Xtext, estas dos tecnologías quedan ya algo primitivas y además no aportan ninguna ventaja extra que no sea el propio compilador del lenguaje. Por ejemplo, para construir un editor de texto que integre un validador y un resaltado de líneas para el DSL definido tendríamos que emplear otra tecnología diferente. Además, si se desea mejorar la gramática con aspectos como el ámbito de una variable, tendríamos que entrar en código C más difícil de comprender.

Como se ha visto, también se han mencionado las tecnologías Lex y Yacc. Pero dado que Flex y Bison heredan sus propiedades y pueden entenderse como una evolución de estas, se puede concluir que Lex y Yacc no aportarían ninguna ventaja extra significativa en comparación a Flex y Bison.

Por otro lado, Xtext se basa en ANTLR [11] para su análisis sintáctico. ANTLR es un analizador y generador que puede ser usado para leer, procesar, ejecutar o traducir texto estructurado o ficheros binarios. Es ampliamente utilizado para la construcción de lenguajes, herramientas y marcos de trabajo. Desde una descripción de un lenguaje formal (lo que se denomina como la gramática), ANTLR genera un analizador sintáctico para dicho lenguaje que automáticamente construye árboles de sintaxis abstracta, re-

presentativos de cómo una gramática se corresponde con la entrada.

Con Xtext, a diferencia de con ANTLR, no se pueden especificar predicados semánticos en una gramática. Además, no es posible incluir acciones arbitrarias y la única plataforma soportada para el objetivo es Java o Xtend (que hereda de Java). Debido a esto, se podría pensar que el uso de Xtext es menos beneficioso, sin embargo, son estas limitaciones las que hacen que Xtext pueda ser una mejor opción frente a ANTLR ya que aporta una mayor simplicidad. Por ejemplo, si se quiere obtener un «*unparser*» que serialice de forma arbitraria modelos / grafos de sintaxis que corresponden con la gramática.

Dicho esto, Xtext ha sido seleccionada como la mejor opción debido a la cantidad de componentes útiles en relación al desarrollo de un DSL que ofrece de forma automatizada, lo cual permite reducir el número de tecnologías a usar y el aprendizaje y adaptación que eso conlleva, y a la facilidad de uso en comparación a las alternativas mencionadas.

1.3.5. Epsilon

Epsilon [9] es una familia de lenguajes y herramientas para la generación de código, transformaciones modelo a modelo, validación de modelos, comparación, migración y refactorización que trabaja directamente con EMF y otros tipos de modelos.

Como núcleo principal de Epsilon se encuentra «Epsilon Object Language» (EOL), un lenguaje imperativo orientado a modelos que combina el estilo procedimental de JavaScript con las capacidades de consulta de modelos de OCL.

Lenguajes para tareas específicas

Epsilon aporta una serie de lenguajes para tareas específicas, los cuales usan EOL como lenguaje de expresión. Véase la figura 1.2 donde se mues-

tra una representación de esta arquitectura lógica. Cada lenguaje de tarea específica ofrece construcciones y sintaxis que son adaptados a la tarea específica. Estos lenguajes son:

- «Epsilon Transformation Language» (ETL): Un lenguaje de transformación basado en reglas de modelo a modelo que permite transformaciones de múltiples modelos de entradas en múltiples modelos de salida.

- «Epsilon Validation Language» (EVL): Un lenguaje de validación de modelos que ofrece pruebas de consistencia tanto para modelos como para metamodelos, gestión de dependencias de restricciones y correcciones específicas que los usuarios pueden invocar para reparar inconsistencias identificadas.

- «Epsilon Generation Language» (EGL): Lenguaje de transformación basado en plantillas de modelo a texto para la generación de código, documentación y otros artefactos textuales de modelos. Ofrece desacoplamiento entre el contenido y el destino, regiones protegidas para mezclar código generado con código escrito a mano y coordinación de plantillas.

- «Epsilon Wizard Language» (EWL): Un lenguaje adaptado a transformaciones interactivas específicas de modelos sobre elementos seleccionados por el usuario.

- «Epsilon Comparison Language» (ECL): Un lenguaje basado en reglas para encontrar correspondencias entre elementos de modelos de diversos metamodelos.

- «Epsilon Merging Language» (EML): Un lenguaje basado en reglas para unir modelos de diversos metamodelos, después de haber identificado sus correspondencias con ECL (u otro lenguaje).

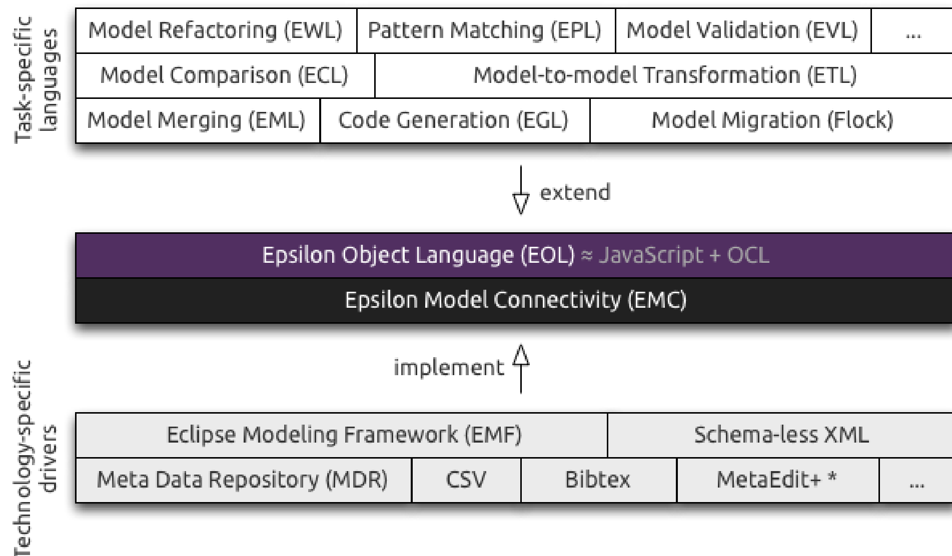


Figura 1.2: Representación de la arquitectura lógica de Epsilon para los lenguajes y drivers de tareas específicas.

- «Epsilon Flock»: Un lenguaje de transformación basado en reglas para actualizar modelos en respuesta a los cambios del metamodelo.

Herramientas

Además de los lenguajes descritos anteriormente, Epsilon también ofrece una serie de herramientas y utilidades para trabajar con modelos. Entre otras herramientas y utilidades que se ofrecen, se presentan las siguientes:

- EuGENia: es un generador de modelos GMF. Su objetivo es acelerar el proceso de desarrollo de un editor GMF y disminuir la barrera de entrada para nuevos desarrolladores. Además, EuGENia permite a los desarrolladores generar editores GMF totalmente funcionales necesitando sólo que se especifiquen algunas notaciones de alto nivel en el metamodelo Ecore.

- Exeed: es una versión mejorada del editor EMF integrado basado en árboles jerárquicos que permite a los desarrolladores personalizar las etiquetas y los iconos de los elementos de un modelo simplemente colocando ciertas anotaciones sencillas en las respectivas «*EClasses*» en el metamodelo Ecore. Exeed también permite establecer valores de referencia utilizando la técnica de arrastrar y soltar en lugar de usando un combinado de cajas en la vista de propiedades.
- ModeLink: es un editor que consiste en dos o tres editores basado en árboles jerárquicos, y que es muy conveniente para establecer enlaces entre diferentes modelos utilizando la técnica de arrastrar y soltar.
- EUnit: es un marco de trabajo de pruebas unitarias especializado en pruebas para tareas de gestión de modelos, tales como transformaciones modelo a modelo, transformaciones modelo a texto o validación de modelos. Está basado en Epsilon pero puede ser usado como tecnología externa a Epsilon. Las pruebas vienen escritas por combinación de un script en EOL y un buildfile en ANT.

En este TFG, Epsilon se ha utilizado con el fin de desarrollar el generador de código a partir del código escrito en AdminDSL. Por lo tanto, los lenguajes empleados han sido: EOL (como lenguaje básico y primordial de Epsilon) y EGL. Por otro lado, la herramienta ofrecida por Epsilon que se ha utilizado ha sido Exeed, ya que ofrece una representación muy clara del metamodelo y ha sido de gran ayuda para depurar errores o mejorar la estructura del mismo.

Otras alternativas a Epsilon

Como se había mencionado antes, Xtext ofrece un componente para interpretar el modelo definido, es decir, un generador de código. Este generador viene ya integrado con la estructura que ofrece Xtext y puede ser usado

de una forma relativamente sencilla. Sin embargo, ofrece ciertas limitaciones con respecto a Epsilon. En primer lugar, Epsilon posee una documentación más amplia que Xtext, permitiendo así facilitar su uso. En segundo lugar, con Epsilon se permiten establecer regiones protegidas. Estas regiones permiten al usuario introducir código escrito a mano sobre el código generado, de forma que en generaciones posteriores sobre el mismo código, todo código situado dentro de estas regiones no se verá afectado, lo cual aporta mayor flexibilidad y un aumento del tiempo de uso del DSL, permitiendo añadir cambios en el código del DSL sin afectar a cambios en el código generado propios de la plataforma. No obstante, puede resultar más incómodo integrar las funcionalidades de Epsilon con los componentes de Xtext, pero puesto que Epsilon trabaja directamente con el metamodelo (Ecore) y Xtext genera de forma automática uno, la conexión entre ambas ha sido directa y sencilla.

Por lo mencionado anteriormente y bajo mi punto de vista personal dado que mi tutor, Antonio García Domínguez, participa en el proyecto Epsilon y, por lo tanto, me ha permitido familiarizarme más con este entorno gracias a su ayuda, se ha seleccionado Epsilon y en concreto EGL como tecnología para sustituir a Xtext en la parte de generación de código.

1.3.6. Django

Django [3] es un marco de trabajo de alto nivel para aplicaciones web en Python que fomenta el desarrollo rápido y un diseño claro y práctico. Elaborado por desarrolladores expertos, se encarga de reducir gran parte de los problemas en el desarrollo web. De esta forma, el desarrollador puede centrarse en escribir su aplicación sin necesidad de reinventar la rueda.

Algunas de sus características principales son:

- Rapidez: Diseñado para ayudar a los desarrolladores a trabajar con aplicaciones desde su idea inicial hasta su finalización de la forma

más rápida posible.

- Seguridad: Django toma seriamente la seguridad y ayuda a los desarrolladores evitando muchos errores comunes de seguridad.

- Escalabilidad: Algunos de los sitios más activos en la Web se aprovechan de la capacidad de Django para escalar con rapidez y flexibilidad. Véase ejemplo de escalabilidad en la figura 1.3 [12].

Django es un marco de trabajo gratuito y de código abierto, lo cual amplía mucho la libertad del desarrollador que hace uso del mismo ya que puede personalizar ciertas funcionalidades internas acomodándolas a sus necesidades o incluso comprender mejor el funcionamiento del sistema, lo cual facilita en gran medida la depuración de errores. Además, ofrece una amplia documentación bien estructurada y fácil de comprender, así como tutoriales básicos, por lo que facilita mucho el proceso de familiarización en el desarrollo de aplicaciones web haciendo uso del mismo.

En este TFG, Django es el marco de trabajo donde se sitúan los resultados del generador de código asociado a AdminDSL.

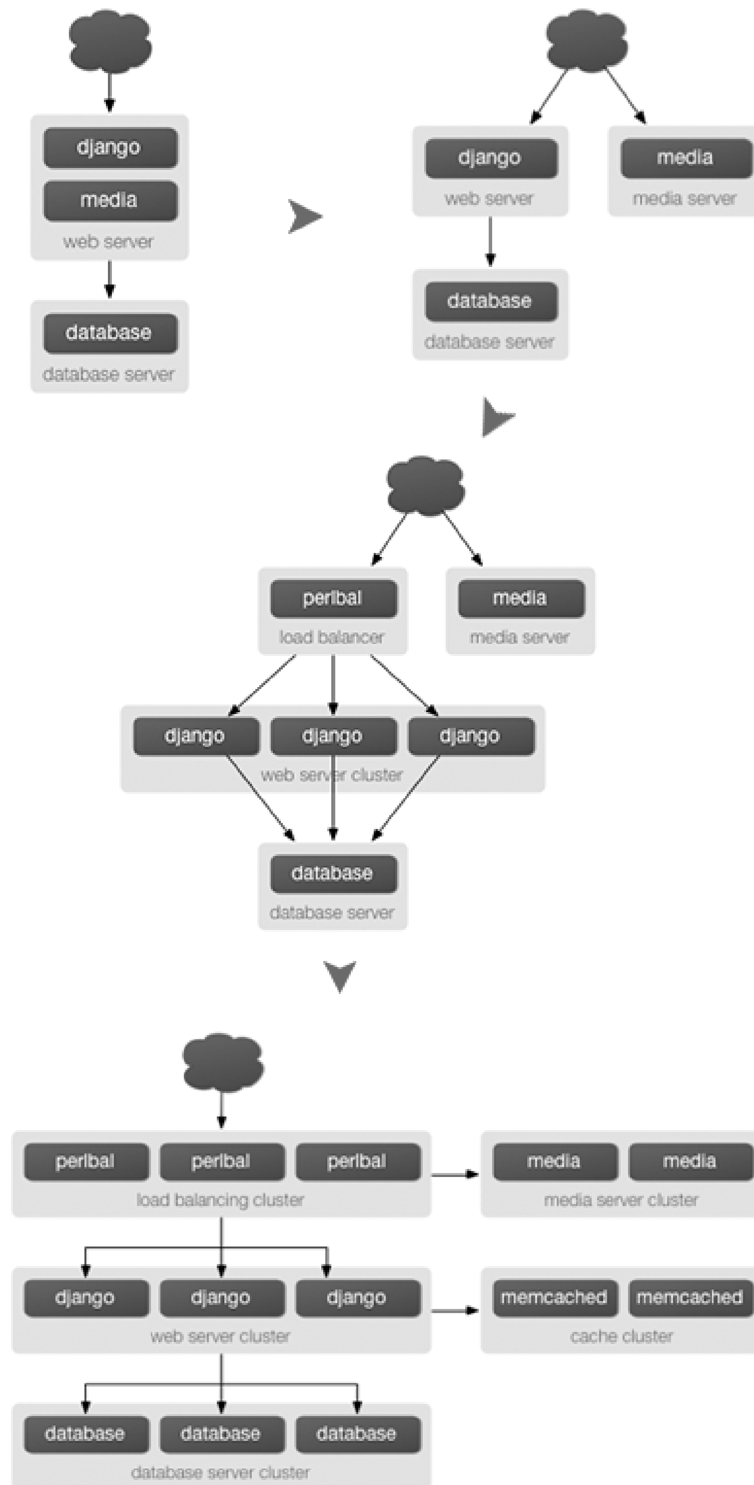


Figura 1.3: Ejemplo de configuraciones de Django cada vez a mayor escala.

CAPÍTULO 2

PLANIFICACIÓN

En este capítulo del documento se detallan las fases en las que se ha elaborado el TFG y la metodología que ha sido utilizada durante el desarrollo del mismo.

2.1. Metodología de desarrollo

Para el desarrollo de este trabajo de fin de grado se ha llevado a cabo una metodología ágil, en concreto, la cada vez más conocida *Extreme Programming* (XP) o programación extrema. Esta metodología está pensada para mejorar la calidad y la respuesta frente a cambios en los requisitos. Como un tipo de desarrollo ágil del software, se trata de liberar versiones en cortos ciclos de desarrollo, con el fin de mejorar la productividad e introducir puntos de prueba en los cuales nuevos requisitos pueden ser añadidos o modificados.

A continuación se explicará más a fondo esta metodología, presentando brevemente su historia, sus principales objetivos y actividades, así como los valores que esta reconoce.

Programación extrema o XP

XP [13] es un estilo de desarrollo software centrado en excelentes técnicas de programación, comunicación clara y trabajo en equipo. Este estilo de desarrollo incluye:

- Una filosofía de desarrollo de software basada en los valores de comunicación, retroalimentación, simplicidad, coraje y respeto.
- Un cuerpo de prácticas que mejoran el desarrollo del software.
- Un conjunto de principios complementarios, técnicas intelectuales para la traducción de valores en la práctica, útil cuando no hay una práctica fácilmente manejable para un problema particular.
- Una comunidad que comparte estos valores y muchas de las mismas prácticas.

XP se distingue de otras metodología por:

- Sus cortos ciclos de desarrollos, resultando en una retroalimentación temprana, concreta y continuada.
- Su enfoque de plan incremental, lo cual rápidamente propone un plan que irá evolucionando durante la vida del proyecto.
- Su habilidad para programar (a nivel temporal) de forma flexible la implementación de las funcionalidades, respondiendo a cambios en los requisitos de negocio.
- Su dependencia con las pruebas automatizadas escritas por programadores, clientes y examinadores («*testers*») para monitorizar el progreso de desarrollo, permitir la evolución del sistema y detectar defectos de forma temprana.

- Su dependencia con la comunicación oral, pruebas y código fuente para presentar la estructura y el propósito del sistema.
- Su dependencia con un proceso de diseño evolutivo que dura tanto tiempo como dura el sistema.
- Su dependencia con colaboraciones estrechas de participantes activos con un talento ordinario.
- Su dependencia con las prácticas que funcionan tanto con los instintos a corto plazo de los miembros del equipo como con los intereses a largo plazo del proyecto.

Historia

La programación extrema fue creada por Kent Beck durante su trabajo en el «*Chrysler Comprehensive Compensation System*» (C3) del proyecto payroll. Beck se convirtió en el líder del proyecto C3 en marzo de 1996 y comenzó a refinar la metodología de desarrollo utilizada en dicho proyecto. Escribió un libro sobre la metodología en octubre de 1999 publicado bajo el nombre de «*Extreme Programming Explained*» [13]. Dicho libro describía lo que ahora se conoce por programación extrema, sin embargo, una serie de ediciones, empezando por ediciones del libro previamente mencionado, fueron realizadas por Beck con el fin de propagar la idea a una audiencia más extensa.

XP generó un significativo interés alrededor de las comunidades del software a finales de los 90 y a principios del siglo XXI, viendo adopciones en entornos radicalmente diferentes a su entorno original.

Valores

Los valores que propone XP son los siguientes:

- Comunicación: Es considerado lo más importante en el desarrollo de software en equipo. Cuando surge un problema en la mayoría de los casos algunas personas saben cómo solucionarlo, sin embargo, este conocimiento no siempre llega a alguien con la capacidad de realizar los cambios, por lo que es muy importante mantener una comunicación entre los miembros del equipo.
- Simplicidad: es el valor con más saber en XP. Hacer un sistema lo suficientemente simple como para solucionar sólo problemas de hoy es un trabajo difícil. Soluciones simples de ayer pueden estar bien hoy o parecer simplistas o complejas. Cuando se necesita cambiar para obtener simplicidad, se debe encontrar un camino desde el lugar donde uno se encuentra hasta el lugar donde se quiere estar.
- Retroalimentación: Una dirección no fija permanece válida por mucho tiempo si se habla sobre detalles en el desarrollo del software, los requisitos del sistema o la arquitectura del sistema. Las direcciones fijadas en función de la expierencia tienen una vida media especialmente corta. El cambio es inevitable y esto crea la necesidad de retroalimentación.
- Coraje: es una acción efectiva de cara al miedo. No cabe duda de que las personas involucradas en un desarrollo software pueden sentir miedo. El cómo ellos enfrentan sus miedos es lo que decide si están trabajando como una parte efectiva del equipo de desarrollo o no. Algunas veces, el coraje se manifiesta como una vía de acción. Si sabes cuál es el problema, haces algo al respecto. Otras veces, el coraje se manifiesta como paciencia. Si sabes que hay un problema pero no sabes cuál es, hay que tener el coraje para esperar a que el verdadero problema aparezca de manera distinguida.
- Respeto: Los cuatro valores anteriores apuntan a uno que se encuentra

situado bajo los mismos: el respeto. Si los miembros de un equipo no se preocupan los unos de los otros ni siquiera de lo que están haciendo, XP no funcionará. Si los miembros de un equipo no se preocupan por un proyecto, nada podrá salvarlo. Cada persona inmersa de alguna forma en el desarrollo del software tiene los mismos derechos humanos que cualquier otra. Nadie merece más la pena que otro. Por lo tanto, para el desarrollo del software, las contribuciones de cada persona en el equipo necesitan ser respetadas con el fin de mejorar la humanidad y la productividad.

Principios

Los valores mencionados son demasiado abstractos para directamente poder guiar el comportamiento. Para este objetivo, XP define una serie de principios, de los cuales se han seleccionado los más importantes de cara a este TFG.

- Humanidad: Las personas desarrollan software. Este simple hecho invalida la mayoría de consejos metodológicos disponibles. En XP se trata de poner de acuerdo tanto las necesidades de negocio como las necesidades personales, aunque esto no implique todas ellas.

- Beneficio mutuo: Cada actividad debería beneficiar a todo interesado. El beneficio mutuo es el más importante de los principios de XP y el más difícil de alcanzar. Siempre hay soluciones a cualquier problema que beneficia a unos pero les cuesta a otros. Si se quieren personas que acepten tus consejos, se necesita resolver más problemas que los que has creado. El beneficio mutuo en XP está buscando prácticas que nos beneficien ahora, después y, del mismo modo, a nuestros clientes.

- Mejora: En el desarrollo del software, no existen procesos perfectos o

diseños perfectos. Sin embargo, estos elementos, entre otros, se pueden perfeccionar.

- Reflexión: Un buen equipo o desarrollador no sólo realiza su trabajo sino que también piensa sobre cómo ha trabajado y por qué está trabajando. Analiza por qué ha tenido éxito o por qué ha fracasado. No trata de esconder sus fallos sino que los expone y aprende de ellos.
- Oportunidad: Aprender a ver problemas como oportunidades para el cambio. Se trata de no resolver problemas simplemente para "sobrevivir" en el desarrollo del software sino convertirlos en oportunidades para el aprendizaje y la mejora.
- Fracaso: A pesar de que el proyecto no tenga éxito, no significa una derrota para el equipo de desarrollo o el desarrollador. De cada proyecto, siempre se obtiene algo de valor.

2.2. Etapas del trabajo

En este apartado, se describen las distintas etapas por las que se ha pasado para la elaboración de este TFG. Si bien, estas etapas no siguen un orden estricto. En ocasiones, ciertas etapas se han realizado de forma paralela, en otras ocasiones se ha tenido que volver a visitar algunas de ellas o estas se mantienen activas durante todo el desarrollo con el fin de solucionar problemas o añadir mejoras.

2.2.1. Elicitación y análisis de requisitos

La elicitación de los requisitos ha sido realizada a través de reuniones. En principio y a lo largo de gran parte de la vida de este trabajo como TFG, las reuniones eran establecidas solamente con Antonio García Domínguez, como tutor del TFG, quién me propuso la idea inicial del desarrollo del

proyecto. Durante estas reuniones, generalmente, se presentaban una serie de problemas o de funcionalidades que debían ser cubiertas por este trabajo y se dedicaba un cierto tiempo a tratar de encontrar la solución más óptima o mejorar soluciones ya implementadas. Dicho esto, las reuniones tenían la siguiente estructura:

- Presentación del problema y contexto: Definición del problema y situación donde se encuentra con el fin de facilitar que la solución sea lo más óptima posible.
- Búsqueda de una solución: Se trata de encontrar múltiples soluciones disponibles para hacer frente al problema y seleccionar la que se crea más óptima.
- Resolución de dudas: Tiempo para consejos o resolución de dudas que habían sido previamente preparadas.
- Revisión y aspectos a mejorar: Señalándose aquellas partes que no eran del todo correctas y mejoras tanto a nivel del DSL como a nivel del código generado.

Estas reuniones se fueron realizando una vez por semana en las salas de reuniones del edificio del CITI.

Tras la marcha de Antonio García Domínguez, por asuntos laborales, las reuniones fueron realizadas directamente con el personal del Área de Informática. El proyecto ya estaba suficientemente avanzado como para permitir su aplicación, por lo que la idea de dichas reuniones era ponerlo en marcha para el desarrollo de las aplicaciones que habían sido demandadas y que se adaptaban al concepto de proceso administrativo. Durante esta puesta en marcha, los miembros del personal daban su opinión respecto a qué aspectos se podían mejorar y qué aspectos debían permanecer inalterables, todo esto bajo su punto de vista personal. Estos nuevos requisitos eran registrados y analizados con el fin de conocer su viabilidad, en algunas ocasiones

bajo aprobación de Antonio García Domínguez a través del correo institucional. De esta forma, la recogida de requisitos ha sido continua durante todo el proyecto, y no sólo al principio.

Estás últimas reuniones fueron realizadas en una media de una o dos por semana también en las salas de reuniones del edificio del CITI o, de forma menos frecuente, en el área de desarrollo del edificio de la FUECA.

2.2.2. Definición del lenguaje específico de dominio

Esta etapa consiste en definir la gramática del lenguaje. Lógicamente, esta definición se ha visto alterada a lo largo del proyecto, debido a la llegada de nuevos requisitos. No obstante la definición de la base del lenguaje, lo cuál ha permanecido prácticamente inalterable tras su definición, fue realizada durante las primeras reuniones.

Este definición se realizó de forma iterativa, llevándose a cabo versiones cada vez más complejas y partiendo de una básica.

La dinámica de trabajo fue la siguiente:

- Descripción de un lenguaje ideal: Describir el perfecto lenguaje que todo lo hace y de la forma más sencilla posible.

- Refinamiento del lenguaje ideal: Acotar sus características con el fin de acercarlo lo más posible al dominio de los procesos administrativos.

- Estudio de viabilidad de las características del lenguaje: Comprobar si la gramática resultante puede ser implementada o no.

La definición del DSL ha sido realizada de forma paralela a la etapa de desarrollo del mismo. Es importante saber que en esta etapa de definición del DSL se ha ido definiendo qué se quiere desarrollar como lenguaje pero no cómo se va a desarrollar.

2.2.3. Aprendizaje de la tecnología Xtext

Una vez fue seleccionada la tecnología Xtext para desarrollar el lenguaje AdminDSL, se realizó un estudio de la herramienta mediante la lectura de su documentación oficial y mediante la realización de tutoriales facilitados en el sitio web de la misma [7].

2.2.4. Desarrollo del lenguaje específico de dominio

Una vez conocida la herramienta, se realizaban las siguientes tareas por cada versión obtenida en la etapa de definición del DSL.

1. Desarrollo de la gramática.
2. Generación del plugin que soporta dicha gramática.
3. Puesta en marcha del plugin.
4. Uso y prueba del editor: Comprobando que las validaciones se ajustan a la gramática definida.
5. Corrección de errores en la gramática.
6. Repetición del proceso empezando por el paso número 2 hasta que todo funcione correctamente.

2.2.5. Aprendizaje de la tecnología Epsilon

Para el aprendizaje de la tecnología Epsilon se ha utilizado principalmente su libro como referencia [14]. En concreto, se ha trabajado con los capítulos:

- 1: Capítulo de introducción para conocer la tecnología.
- 3: Capítulo donde se describe el lenguaje base EOL.

- 7: Capítulo donde se describe el lenguaje EGL que se ha utilizado para la generación de código a partir del modelo del DSL obtenido con Xtext.

2.2.6. Aprendizaje de la tecnología Django

Antes de empezar a desarrollar un generador de código, lógicamente, es necesario conocer a fondo qué es lo que se va a generar. Este aprendizaje se ha realizado a través de la documentación y de los tutoriales facilitados en la página oficial de Django [3]. Además se ha aprendido mucho mediante la visualización de proyectos Django activos en el gestor de proyectos Redmine del Área de Informática.

2.2.7. Desarrollo del generador de la aplicación Django

Una vez obtenidas las versiones de AdminDSL, se daba comienzo al desarrollo del generador de código. La idea es que el generador de código cubriese todas las características de la versión del DSL existente hasta el momento. De esta forma, cuando se obtenía una nueva versión, se ampliaba el generador de forma oportuna a los cambios realizados en el DSL. Aunque esto no era siempre así ya que, en ciertos cambios de versión del DSL, el generador no se veía afectado.

A continuación, se describen las distintas subetapas de esta fase, las cuáles sí siguen un orden lógico aunque, como se ha mencionado anteriormente, muchas de estas han tenido que ser visitadas de nuevo para añadir cambios y corregir errores.

Generación de modelos

Durante esta subetapa se desarrolla la primera versión del generador de código. Con esta versión, se genera un proyecto Django básico que contiene

simplemente definidos los modelos según el código de AdminDSL del cuál genera.

Generación inicial de vistas y formularios

Se amplía el generador de código para que se elaboren vistas y formularios. Las vistas iniciales fueron tres, una para comenzar un nuevo proceso administrativo (visualización de su formulario), otra para editarlo y otra para ver los procesos administrativos ya comenzados o los que se pueden comenzar.

Generación de migraciones

Se trataba de pasar la creación de grupos de usuarios a las migraciones en Django en lugar de hacerlo en cualquier otra parte del código generado. Estas migraciones se encargan de definir las tablas en la base de datos y permite, entre otras cosas, poder realizar modificaciones en la estructura de la misma sin necesidad de tener que destruirla y, por lo tanto, tener que realizar un volcado completo de esta con el fin de no perder ninguna información.

Generación de lógica de estados

Añadir en la generación de código toda la lógica de estados por las que pasan los procesos administrativos.

Generación de lógica de permisos

Añadir todos los permisos asociados a los estados definidos en AdminDSL, de forma que se realicen los filtrados de procesos administrativos correspondiente según el usuario y el rol, y la visibilidad o edición de los campos de estos por el mismo.

Refactorización en el código a generar

Dado a que de una subetapa a otra se van produciendo ampliaciones en el código a generar, y dado los múltiples cambios realizados en AdminDSL que han ido afectando a esta generación, era primordial realizar una refactorización de todo el código generado con el fin de hacerlo más fácil de leer y comprender.

Generación de vistas finales

Subetapa en la cual se añadieron nuevas vistas, de forma que por cada proceso administrativo definido se genere una interfaz separada que permita ver los procesos activos, en borrador o cerrados y que permita, además, poder comenzar un nuevo proceso administrativo.

2.2.8. Desarrollo de las pruebas para la aplicación generada

Tras la realización de cada versión generadora, se han ido realizando pruebas para comprobar que todo funciona correctamente. Muchas de estas de forma manual, otras definidas y realizadas mediante la integración de Selenium en Django.

2.2.9. Desarrollo del plugin de Eclipse

En esta etapa se desarrolla el plugin de Eclipse que permita su instalación e implantación en dicho entorno de forma sencilla de cara al usuario final de este trabajo.

2.2.10. Documentación

Etapa en la que se ha desarrollado la memoria de este TFG. Para ello, se ha empleado el lenguaje LaTeX [15, 16].

2.3. Distribución temporal

Este trabajo tuvo su comienzo a durante el mes de diciembre de 2014. El ritmo de trabajo ha sido constante aunque ha sido más intenso a partir del mes de febrero, con una dedicación media de 5 horas diarias con uno o dos días de descanso cada semana y una reunión semanal.

Dado que he actuado como participante en los distintos proyectos de aplicaciones que han hecho uso de este trabajo (en algunos como desarrollador, en otros como supervisor y ayudante), la dedicación en los últimos meses se ha visto reducida en cuanto a la ampliación de funcionalidades y enfocada más a la corrección de errores o mejoras específicas que se han ido localizando a través del uso del mismo.

En el mes de junio de 2015 se puso fin a las ampliaciones de funcionalidades y características de AdminDSL, con el fin de poder ser presentado como TFG. No obstante, tal y como se indicará más adelante en este documento, se pretende continuar ampliando y mejorando este sistema en un futuro.

Gracias al uso del gestor de proyectos Redmine del Área de Informática, se ha podido observar y planificar en cierta medida el desarrollo de este trabajo. Para ello, se ha hecho uso del sistema de peticiones que ofrece esta plataforma. De este modo, por cada acción a realizar sobre este TFG se le ha sido asociada una petición, gestionándose el número de horas dedicadas y el porcentaje de trabajo realizado, entre otras opciones.

En las figuras 2.1 y 2.2 se puede visualizar un diagrama de Gantt donde aparecen las distintas etapas que se han ido llevando a cabo a lo largo de este TFG y el tiempo que se ha reservado para su realización.

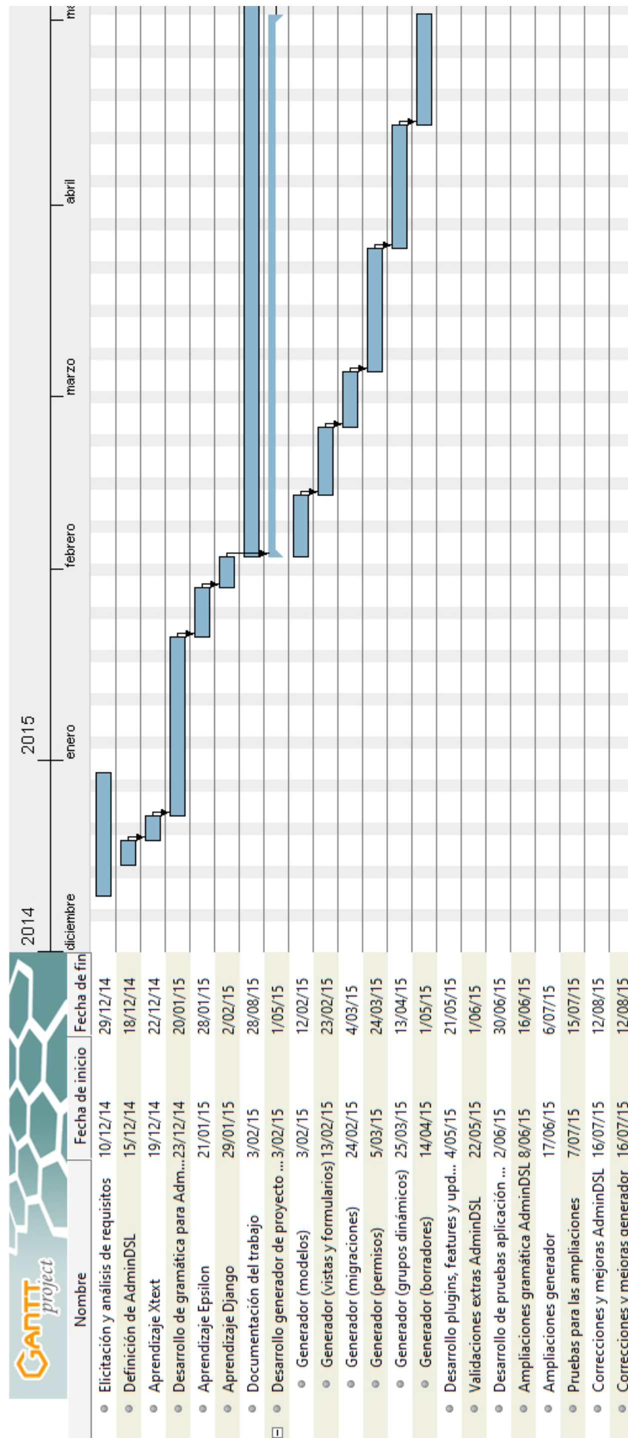


Figura 2.1: Diagrama de Gantt: parte 1.

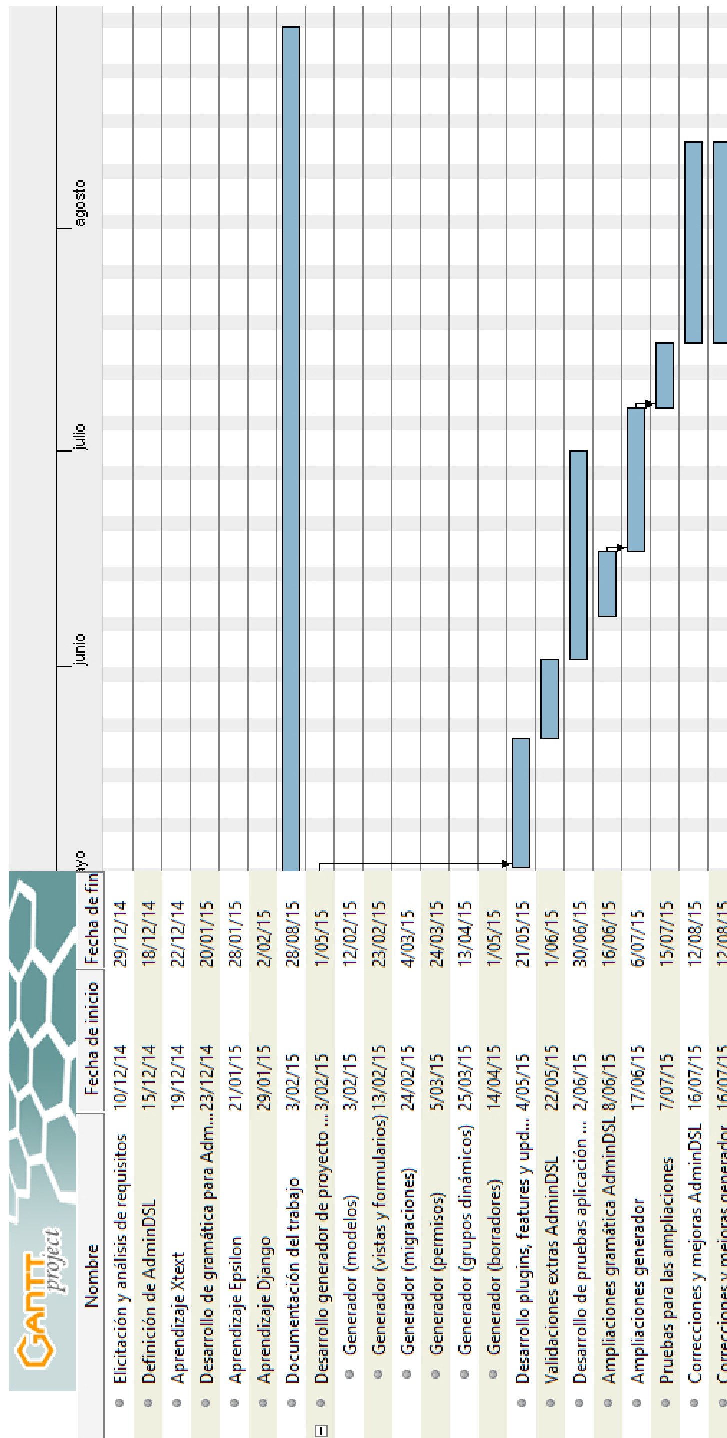


Figura 2.2: Diagrama de Gantt: parte 2.

CAPÍTULO 3

ANÁLISIS

3.1. Requisitos

Los principales requisitos a cumplir por este TFG son los siguientes:

- Definir y describir procesos administrativos: De una forma sencilla y fácil de comprender. Requisito que se cumple con el desarrollo del lenguaje específico del dominio y su editor asociado.

- Generación de aplicaciones web a partir de descripciones de procesos administrativos. Se cumple con el desarrollo del generador de código utilizando la tecnología Epsilon.

- Integración de las diferentes utilidades en un entorno de uso conocido. Requisito por el cual se desarrolla un plugin para Eclipse, entorno seleccionado para el despliegue de este TFG.

- El uso del DSL debe ser intuitivo y fácil de aprender en poco tiempo: El cumplimiento de este requisito es muy importante con el fin de mejorar la productividad.

- El DSL debe permitir definir los siguientes elementos:
    - Roles de usuario.
    - Procesos administrativos.
    - Secciones dentro de los documentos de los procesos administrativos.
    - Grupos dentro de las secciones. Estos pueden ser:
        - Estáticos y con un número fijo de instancias obligatorias.
        - Estáticos y con un número fijo de instancias no obligatorias.
        - Dinámicos y obligatorios (al menos uno añadido).
        - Dinámicos y no obligatorios (se permite no añadir ninguno).
    - Campos dentro de las secciones y grupos. Estos pueden ser:
        - De múltiples tipos, desde cadenas de texto a selectores múltiples.
        - Obligatorios o no obligatorios.
        - De valor único o que permite valores repetidos.
    - Entidades: Tanto a nivel de proceso como a nivel global.
    - Relaciones: De tantos niveles como se desee entre procesos, entidades o ambas.
    - Estados dentro de los procesos administrativos.
    - Permisos asociados a los roles de usuario para cada uno de los estados.
    - Transiciones de un estado a otro.
- Generación de código sencilla: Generar una aplicación web a través de la descripción dada por el DSL no debe suponer ningún tipo de esfuerzo extra.

- La aplicación web generada debe permitir comenzar, gestionar y controlar los procesos administrativos descritos ofreciendo una estructura básica reconocible por cualquier desarrollador que esté habituado a trabajar con este tipo de procesos.

- El código generado debe ser fácilmente comprensible por un desarrollador familiarizado con el marco de trabajo que se haya estipulado: Requisito importante para facilitar el mantenimiento de la aplicación.

3.2. Análisis de las herramientas existentes

En el contexto de los objetivos a cumplir por este trabajo, las principales herramientas, que abarcan un ámbito similar, son aquellas asociadas a BPM (*Business Process Model*).

Antes de empezar a describir las diferentes herramientas es necesario definir una serie de conceptos básicos que se verán a continuación.

Proceso de negocio

Los procesos son colecciones de actividades relacionadas y estructuradas que, como un todo, constituyen una actividad de una compañía u organización. Estos procesos de negocio pueden ser gestionados y controlados por sistemas conocidos como BPM o BPMS.

BPM

BPM [17] puede ser definida como una nueva categoría de software empresarial que permite a las empresas modelar, implementar y gestionar conjuntos de actividades interrelacionadas (procesos) de cualquier tipo, ya sea a través de los departamentos o involucrando a toda la empresa con extensiones para incluir la participación de clientes, proveedores y otros agentes como participantes en las tareas de los procesos.

3.2.1. AuraPortal Helium

AURA ofrece un único producto de software llamado AuraPortal Helium [18]. Es un software privativo y no gratuito. Posee una estructura modular que permite controlar desde un simple proyecto o la gestión de un departamento hasta la gestión completa integrada del conjunto de la empresa u organización, mediante un software de procesos BPM moderno, flexible y cuyo manejo está al alcance de cualquier persona sin necesidad de conocimientos técnicos.

3.2.2. Eunomia Process Builder

Eunomia Process Builder [19] es una herramienta privativa y no gratuita BPM eficiente y fácil de usar que permite diseñar procesos y generar sitios web para los mismos. Posee un editor de esquemas que permite diseñar diagramas de proceso utilizando diagramas de actividad de UML 2.4 o diagramas de procesos de BPMN 2.0. Además utiliza diagramas coreográficos o esquemas de trazas.

Tanto los metamodelos de UML como los de BPMN son extendidos con esta herramienta, que añade objetos tales como fases, sub-procesos, políticas, medida, entre otros, junto con sus atributos asociados y asociaciones.

3.2.3. Bonita BPM

Bonita BPM [20] es una plataforma de código abierto fundado en el desarrollo de aplicaciones BPM, establecida bajo la Licencia Pública General de GNU v2. Se trata de una plataforma para el desarrollo de procesos de negocio. Las principales características que ofrece son las siguientes:

- Modelar: Procesos de negocio a través de una suite de BPM.
- Construir: Aplicaciones que funcionan en cualquier dispositivo.

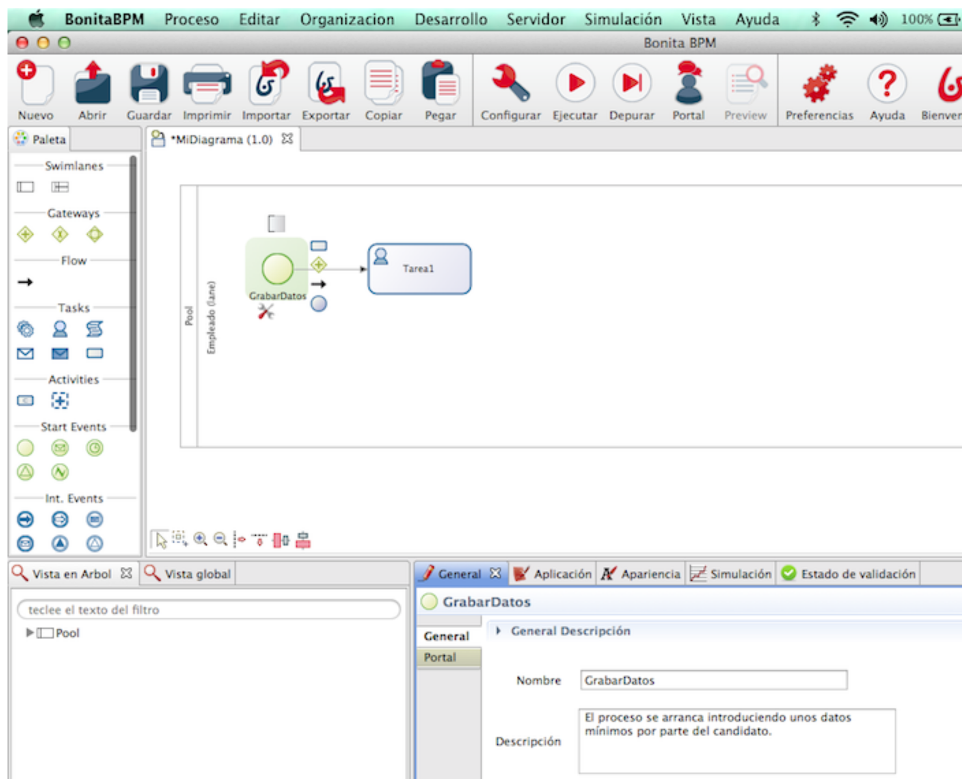


Figura 3.1: Bonita BPM: Añadiendo atributos en el proceso.

- Conectar: Datos de negocio y sistemas, y construir extensiones libremente.
- Actualizar: Aplicaciones en tiempo real, sin interrupción del servicio.
- Supervisar: Todo, incluso reparar, reproducir u omitir actividades fallidas.

A continuación se presenta un manual básico en el que se define un proceso utilizando esta herramienta.

Manual básico para la definición de un proceso con Bonita BPM [21]

Para comenzar se debe colocar el inicio del proceso, representado mediante un círculo. Un proceso tiene un pool o calle, que tiene un actor por

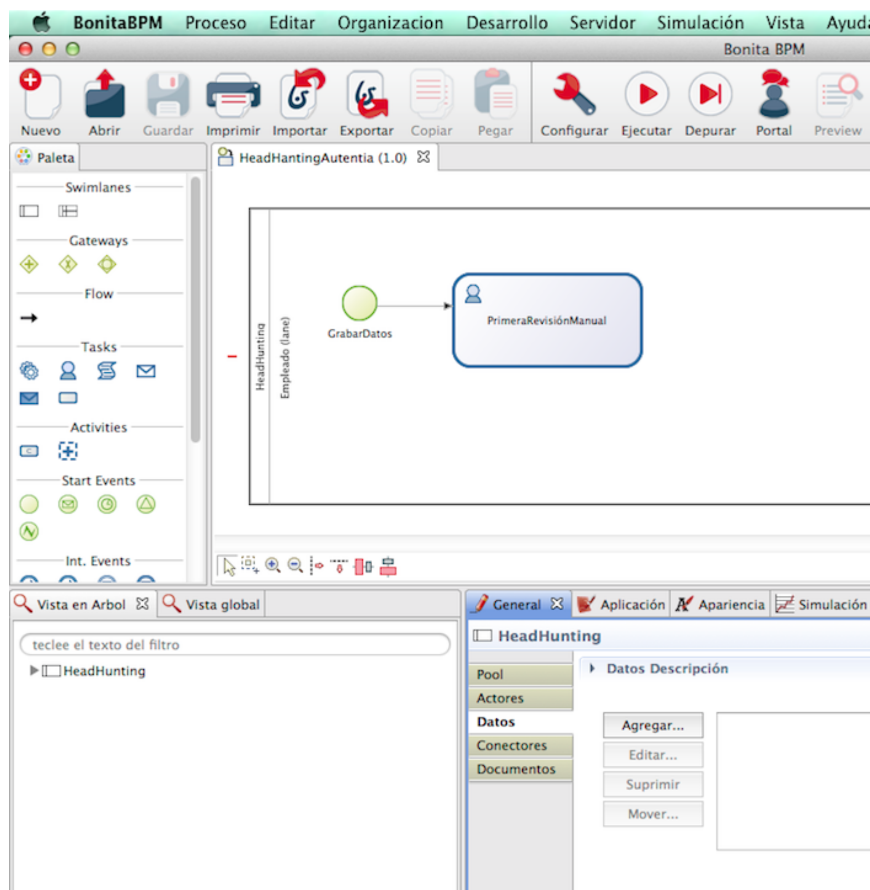


Figura 3.2: Bonita BPM: Añadiendo conjunto de datos.

defecto.

Marcando la calle principal se añaden atributos necesarios en el proceso (ver figura 3.1), los cuales estarán disponibles en cualquier tarea y serán persistentes automáticamente por el propio motor en las tablas internas.

Ahora, se añade todo el conjunto de datos que se tenga pensado utilizar en el ciclo de vida del proceso completo. En las mismas tareas se pueden declarar variables locales para ciertas operaciones o control de flujo. Véase figuras 3.2 y 3.3.

Al ejecutar el proceso se crea un formulario paginado en base a los campos globales al proceso creado tal y como se puede observar en la figura 3.4.

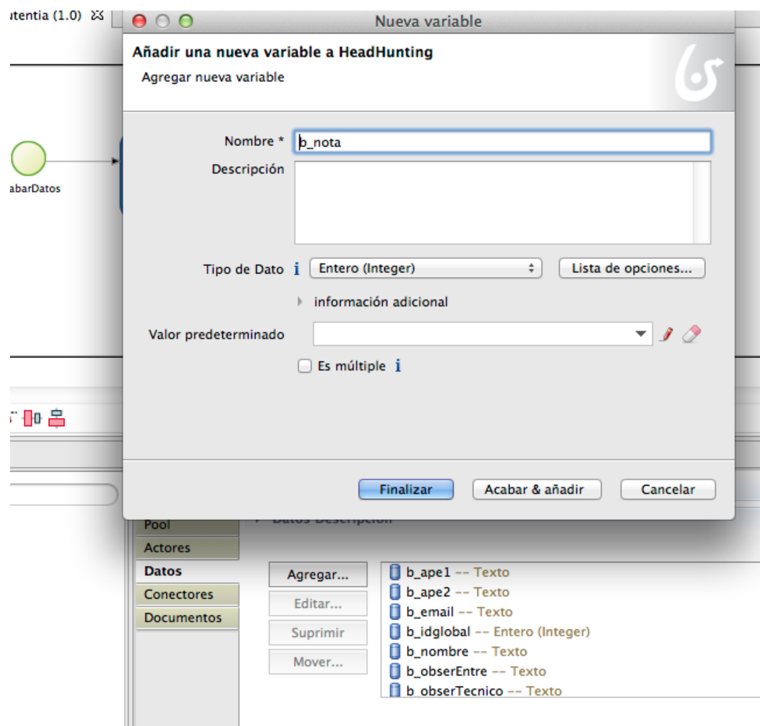


Figura 3.3: Bonita BPM: Declarando variables o campos globales.

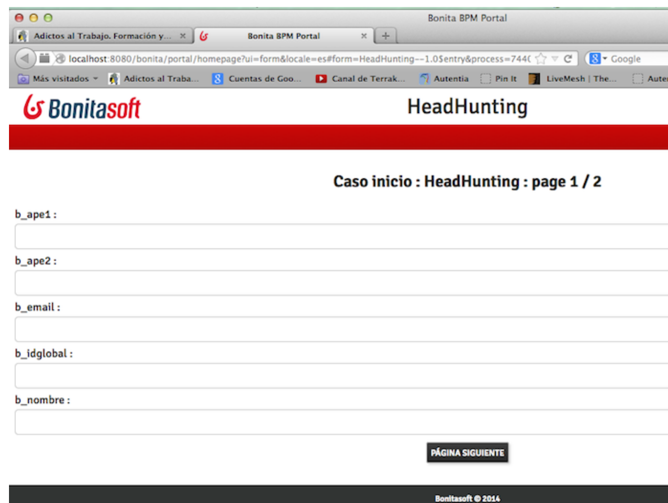


Figura 3.4: Bonita BPM: Primera ejecución del proceso con formulario asociado.

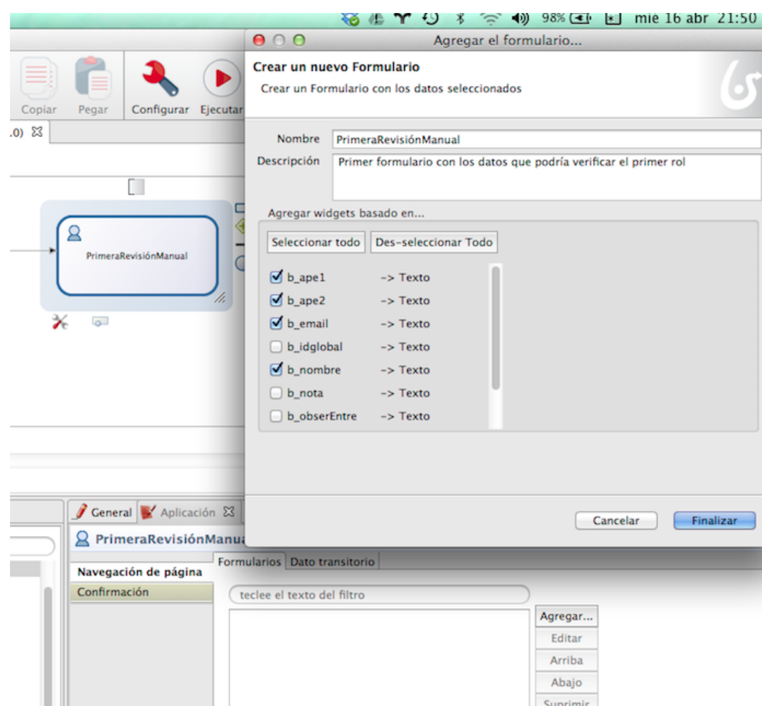


Figura 3.5: Bonita BPM: Editando formulario de una tarea.

Como la mayoría de las herramientas BPM, Bonita tiene como núcleo central una bandeja de tareas donde cada usuario dispone de un conjunto de labores potencialmente a iniciar. Cuando un usuario elige una tarea se la asigna (y ya no queda disponible para los demás). Al pinchar sobre ella dispone de un formulario donde completar la tarea.

El formulario puede ser por defecto, generado dinámicamente a partir del los campos declarados, o puede personalizarse. Para cambiar este formulario pinchamos en la tarea y, en la sección aplicación, elegimos los campos involucrados. Véase figura 3.5.

Además, se pueden reposicionar los campos del formulario y añadirles incluso una validación estándar como se muestra en la figura 3.6. También se pueden añadir nuevos campos (como el campo de tipo documento), imágenes y otros elementos que se pueden encontrar en el menú lateral o "pa-

leta”.

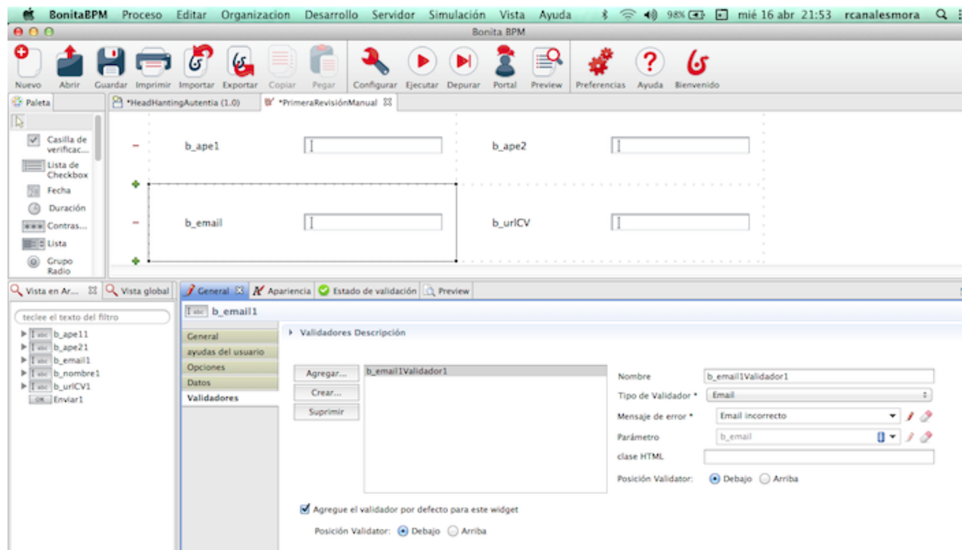
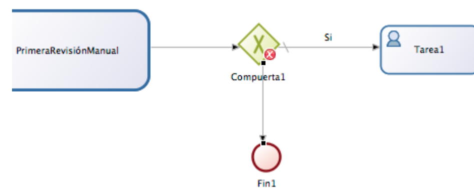


Figura 3.6: Bonita BPM: Posicionando campos del formulario y añadiendo validaciones.



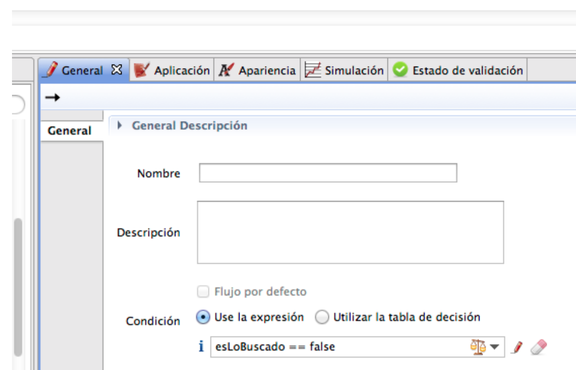


Figura 3.7: Bonita BPM: Añadiendo una bifurcación.

Normalmente en el ciclo de vida del proceso, en base a un cálculo o algún campo informado se hacen bifurcaciones o «gateways». Estas bifurcaciones pueden ser dadas por una condición AND (representada con un "+", OR (representada con un círculo) o XOR (representada con una "X"). Para añadirlas simplemente se arrastra su elemento representativo de la "paleta" y se conecta con los elementos que se crean convenientes (figura 3.7). En este ejemplo, la bifurcación hacia el final del proceso se realizará si el campo global "esLoBuscado" (previamente añadido) no se encuentra marcado, es decir, su valor es igual a "false".

Creando el resto de formularios, roles de participantes implicados, bifurcaciones y otros elementos que interesen, se tendría una aplicación básica orquestada dentro de un portal Web.

3.2.4. Conclusión del análisis de las herramientas

Tanto AuraPortal Helium como Eunomia Process Builder son herramientas privativas y no gratuitas, lo cual ya supone una desventaja en cuanto a las herramientas de código abierto como BonitaBPMN o la que se desarrolla con este TFG, AdminDSL.

Si bien estas tres herramientas BPM que se han descrito ofrecen una forma relativamente sencilla de modelar, construir y gestionar procesos de negocio. Sin embargo, esto se hace de forma muy generalizada y con un lenguaje bien definido y prácticamente cerrado, lo cual dificulta la adaptación y familiarización del personal que ha trabajado siempre en relación a procesos administrativos, que son un tipo específico de proceso de negocio, con herramientas que soportan procesos tan generales. Dicho esto, personalizar estas herramientas con el fin de mejorar la productividad puede ser más complicado que en sistemas basados en un dominio particular, puesto que en el segundo caso se puede utilizar un lenguaje específico de dominio más abierto a modificaciones.

Utilizar herramientas muy potentes para buscar soluciones a problemas en un dominio mucho más reducido puede ser también un problema por lo siguientes motivos:

- Elevada curva de aprendizaje: En muchas ocasiones, aprender a manejar una herramienta tan amplia para sólo tener que dar solución a un problema en un dominio muy reducido provoca un sobresfuerzo innecesario por parte del personal de desarrollo.
- Complicar las cosas más de lo normal: A veces ocurre que, al usar una herramienta que permite añadir múltiples características y funcionalidades extras, lo que en principio era una aplicación muy básica, se convierte en algo más complejo y difícil de mantener, violando uno de los principios de diseño del software, KISS («*Keep It Simple*»).

Dicho esto, encontrar una herramienta de fácil adaptación y personalización que se centre únicamente en procesos administrativos y que utilice un enfoque intuitivo sería lo ideal para el personal de desarrollo que debe desarrollar y mantener este tipo de procesos. Por eso, AdminDSL se presenta, en este contexto, como mejor opción frente a estas herramientas BPM.

3.3. Especificaciones de AdminDSL

En esta sección se describe el lenguaje AdminDSL, empezando por los requisitos que debe cumplir y describiendo su sintaxis abstracta y concreta.

3.3.1. Requisitos del lenguaje

A continuación, se especifican los requisitos que debe cumplir el lenguaje AdminDSL:

- Definir opciones generales. Destinadas a las plataformas que se quieran utilizar en el resultado de la generación final.

- Definir roles de usuario. Para poder agrupar los participantes de los procesos administrativos.

- Definir entidades. Que permitan abstraer conceptos que sean utilizables por los procesos administrativos.

- Definir procesos administrativos. Con todos los elementos necesarios.

    - Definir secciones. Que permita estructurar el documento formal.

        - Definir campos. A cumplimentar por los participantes.

        - Definir grupos. Que permita agrupar campos dentro de una sección.

    - Definir entidades. Que permitan abstraer conceptos utilizables sólo por el proceso administrativo donde se define.

    - Definir relaciones. Que permita asociar un proceso con otro/s proceso/s, una entidad con otra/s o un/os proceso/s y una/s entidad/es a nivel n-ario.

    - Definir estados. Por los que podrá transitar el proceso administrativo.

        - Definir permisos. Que defina si un cierto participante o grupo de participantes puede ver o editar un campo, grupo o sección del documento formal del proceso administrativo.

        - Definir transiciones. Con el fin de habilitar el cambio de estado en un proceso administrativo.

3.3.2. Sintaxis abstracta

En la sintaxis abstracta se definen los elementos del lenguaje y las relaciones entre ellos sin entrar en detalles de qué interpretación tendrán en la generación final obtenida a partir del mismo.

Diagramas de clase

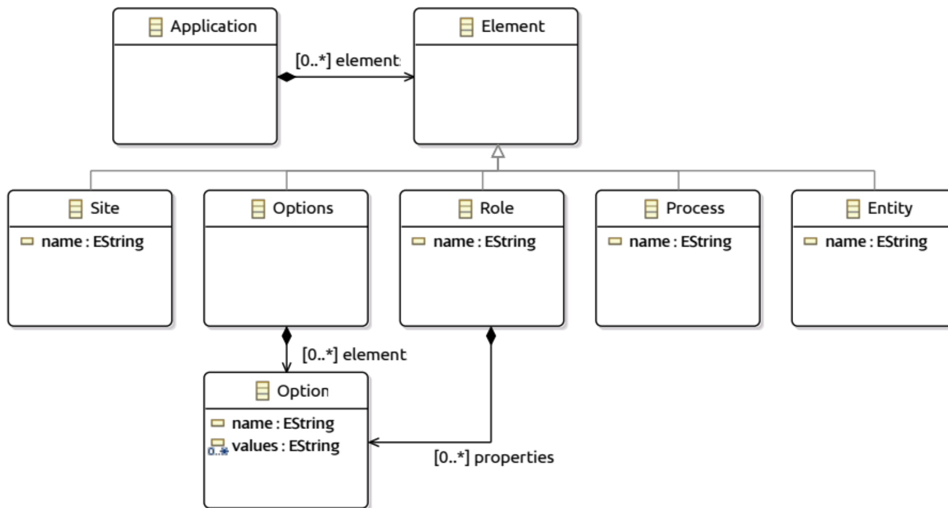


Figura 3.8: Diagrama de clases: Aplicación.

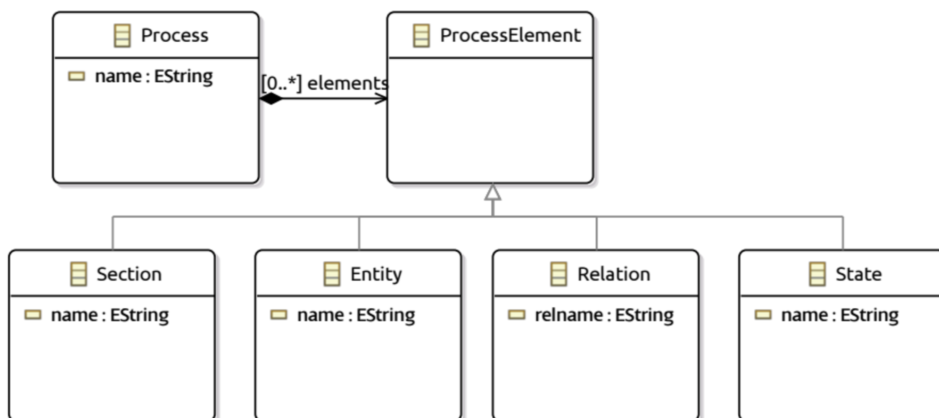


Figura 3.9: Diagrama de clases: Procesos.

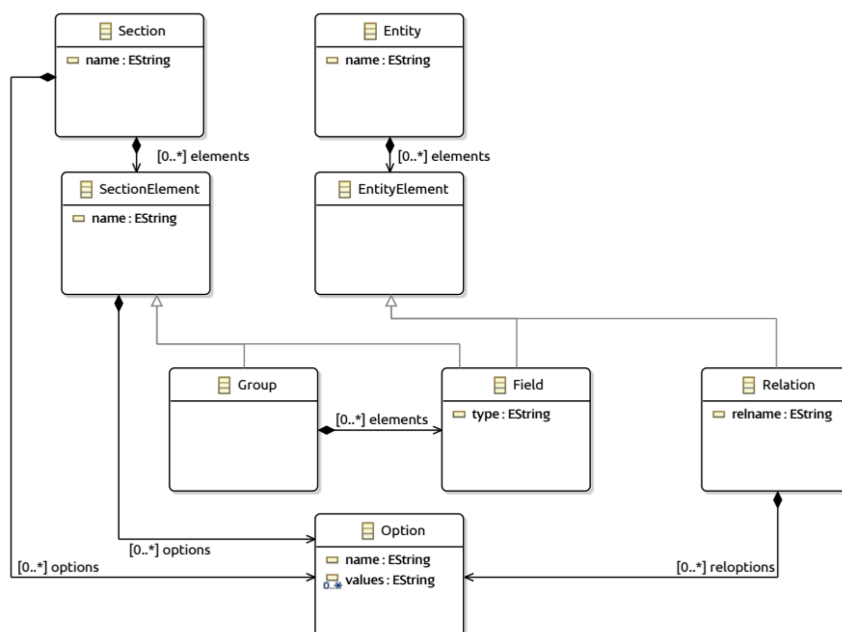


Figura 3.10: Diagrama de clases: secciones y entidades.

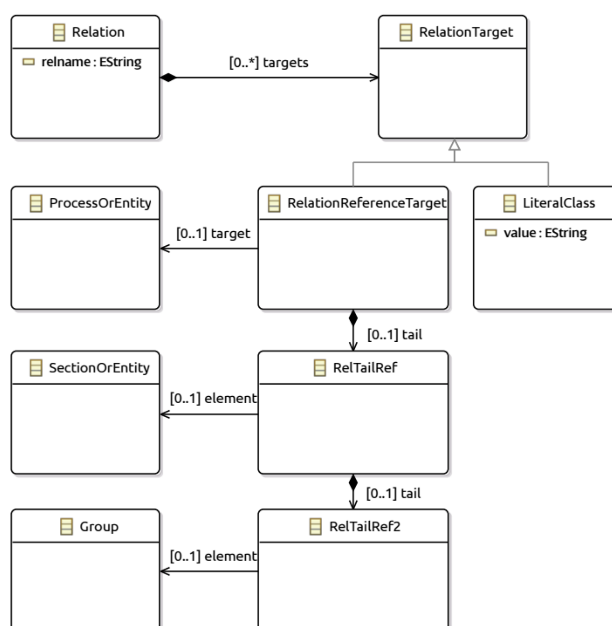


Figura 3.11: Diagrama de clases: Relaciones.

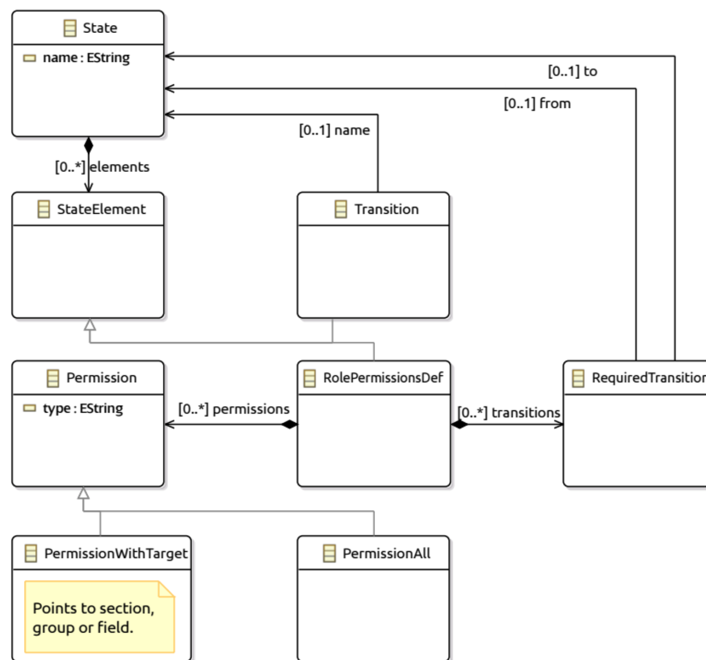


Figura 3.12: Diagrama de clases: Estados.

En la figura 3.8 se representa la sintaxis abstracta a nivel de `Application`. Este es el elemento raíz de la estructura del lenguaje, es singular y a partir del mismo se puede acceder a todos los elementos que se hayan especificado navegando a través de la jerarquía. En concreto, se pueden definir elementos `Site`, `Options`, `Role`, `Process` o `Entity`.

En la figura 3.9 se puede ver una representación donde participan los elementos relacionados con `Process` el cual es uno de los tipos de elementos de los cuales se compone un elemento `Application` tal y como ya se había mencionado previamente. Este diagrama implica que en el ámbito de un proceso se pueden definir elementos del tipo `Section`, `Entity`, `Relation` o `State`.

Tanto los elementos de tipo `Section` como los de `Entity` estarán formados por elementos de tipo `Field`. Además, en los de tipo `Section` se podrán definir elementos de tipo `Group`, que a su vez se componen de ele-

mentos `Field`, y en los de `Entity` elementos de tipo `Relation`. Véase figura 3.10.

Como se puede observar en la figura 3.11, un elemento `Relation` contiene *targets* u objetivos que pueden ser, o bien de tipo `LiteralClass` o bien referencias a otros elementos como pueden ser:

- Elementos de tipo `Process`.
- Elementos de tipo `Entity`.
- Elementos de tipo `Section` que se encuentra en un elemento `Process` particular.
- Elementos de tipo `Entity` que se encuentra en un elemento `Process` particular.
- Elementos de tipo `Group` que se encuentra en un elemento `Section` particular que a su vez forma parte de un elemento `Process`.

Por último, en el diagrama de la figura 3.12 se describe la estructura de los elementos relacionados con los de tipo `State`. Un elemento `State` se compone de elementos de tipo `StateRole` y de `Transition`. Los primeros, poseen una referencia a un elemento `Role` (aunque no aparezca en el diagrama por simplificación) y se componen de elementos `Permission`, además puede contener también elementos del tipo `RequiredTransition` que poseen dos referencias a elementos `State`. Los elementos `Transition`, por otro lado, sólo poseen una referencia a un elemento `State`. Esto nos permite conectar, en cierto modo, un elemento `State` con otros del mismo tipo a través de elementos `Transition`.

Las restricciones que no han podido ser cubiertas a través de esta representación se pueden ver en el apéndice B de este documento.

3.3.3. Sintaxis concreta

La sintaxis concreta define qué interpretación va a tener cada elemento del lenguaje, en este caso, en el dominio de procesos administrativos. Para llevar a cabo esta interpretación, se describe cada una de las sentencias del lenguaje dadas por cada uno de los elementos descritos en el apartado de sintaxis abstracta.

Sentencia `site`

Es una sentencia obligatoria que define el nombre de la aplicación donde estará contenida todos los elementos posteriormente definidos.

```
site nombre_aplicacion;
```

Sentencia `options`

Define opciones exclusivas para el marco de trabajo a generar, tales como la plantilla base que se va a utilizar o las app's que se van a importar en un proyecto Django (aquí también podrían definirse opciones propias de Symfony o cualquier otro entorno de trabajo web).

```
1 options {
2     django_base_template = "app/base.html";
3     django_extra_apps    = "nombre_app = SVN_URL";
4 }
```

Sentencia `role`

Define un rol de usuario que va a poder participar en los distintos procesos administrativos definidos.

```
1 role Piloto;
```

Sentencia `entity`

Define una entidad global. Una entidad es un elemento que posee características propias y capacidad para relacionarse con otros elementos.

```
entity Ciudad {
    tipo nombre;
}
```

Sentencia `relation`

Es una sentencia utilizable en el ámbito de las entidades y de los procesos, es decir, que debe ir colocada entre las llaves de una sentencia `entity` o `process`.

Define una relación o asociación entre el elemento donde se define y otra entidad o proceso. Su definición se divide en dos partes: la primera hace referencia a las entidades o procesos con las que se quiere relacionar. En esta parte se permite utilizar cadenas literales para permitir relaciones con modelos perteneciente a módulos externos. Además se pueden definir relaciones *n*-arias añadiendo más de una referencia. La segunda parte hace referencia al tipo de asociación, ya sea *many-to-many*, *one-to-one* o *many-to-one*. En caso de omisión de tipo, se interpreta como una asociación *many-to-one* si es una relación binaria. En caso de ser una relación en la que participan más de dos elementos se considerará *many-to-many*.

```
entity Avion {
    relation(Vuelo) vuelo_actual;
}
entity Aeropuerto {
    relation(Avion, type="ManyToMany") avion;
```

```
6  }
7  process Vuelo {
8      relation(Avion, Aeropuerto, type="ManyToMany")
           avion_aeropuerto;
9  }
```

Sentencia `process`

Define un proceso administrativo con todos los elementos que conlleva.

```
1  process Vuelo {
2      ...
3  }
```

Ámbito `process`: sentencia `relation`

Véase apartado **Sentencia `relation`**.

Ámbito `process`: sentencia `entity`

Define una entidad local dentro del proceso.

```
1  entity ModeloAvion {
2      tipo motor;
3  }
```

Ámbito `process`: sentencia `section`

Define una sección. Una sección es un conjunto de campos que forman un apartado en el formulario asociado al proceso. La sección puede contener opciones definidas.

```
section(label="Datos de los pasajeros") DatosPasajeros {
    tipo num_total_pasajeros;
}
```

Ámbito `section`: sentencia `group`

Define un grupo dentro de una sección. Un grupo es un conjunto de campos agrupados con la finalidad de separarlos del resto de campos de la sección donde se encuentra y permitir, entre otras opciones, mostrar un número fijo de instancias del mismo o hacerlo de forma dinámica a través del formulario. Al igual que la sección, los grupos pueden tener opciones definidas.

```
group(label="Documentos a entregar", count="dynamic", blank
   ="True") Docs {
    tipo fichero;
}
group infoPersonal {
    tipo nombre;
    ...
}
```

Ámbito `entity`, `section` o `group`: sentencia de definición de campos

Define un campo o atributo dentro de una entidad, sección o grupo. Este campo se declara por un tipo y un nombre.

```
tipo nombre_campo_o_atributo;
```

Para todos los tipos de campo se pueden definir opciones:

```
tipo(opcion1="valor1", opcion2="valor2") nombre_campo;
```

Opciones generales:

- `blank`: define si el campo es obligatorio o no. Valor “True” para campos no obligatorios, valor “False” u omisión de la opción para campos obligatorios.

  ```
  tipo(blank="True") nombre_campo;
  ```

- `unique`: define si el valor del campo debe ser único o no. Valor “True” para campos con valores únicos, valor “False” u omisión de la opción para campos con valores no únicos.

  ```
  tipo(unique="True") nombre_Campo;
  ```

- `label`: define el valor de la etiqueta en el formulario asociada al campo.

  ```
  tipo(label="Nombre del campo") nombre_campo;
  ```

Tipos de campos:

- `fullName`: para el nombre completo de una persona, de forma general, su nombre y apellido.

  ```
  fullName nombre_pasajero;
  ```

- `address`: para el domicilio de una persona.

  ```
  address direccion_pasajero;
  ```

- `identityDocument`: para el documento de identidad de una persona.

```
identityDocument id_pasajero;
```

- `birthdate`: para la fecha de nacimiento de una persona.

```
birthdate edad_pasajero;
```

- `phone`: para números de teléfono.

```
phone telefono;
```

- `email`: para correos electrónicos.

```
email correo;
```

- `currency`: para valores en unidades monetarias.

```
currency precio;
```

- `string`: campo cadena.

```
string descripcion_corta;
```

- `text`: para cadenas de una extensión mayor en un cuadro de texto.

```
text descripcion_larga;
```

- `date`: para fechas.

```
date fecha_salida;
```

- `integer`: para valores enteros.

```
integer numero_pasajeros;
```

- `decimal`: para valores decimales.

```
decimal altura;
```

- `float`: para valores flotantes.

```
float temperatura;
```

- `choice`: para definir selectores. Este campo posee las siguientes opciones propias:

    - `table`: para hacer referencia a una entidad a partir de la cuál se mostrarán las opciones del selector en función de las instancias existentes.

    ```
    choice(table="ModeloAvion") modelo_avion;
    ```

    - `process_table`: para hacer referencia a un proceso a partir del cuál se mostrarán las opciones del selector en función de las instancias existentes.

    ```
    choice(process_table="Vuelo") vuelo;
    ```

Las opciones “table” y “process_table” pueden ir acompañadas de la opción “multiple”, cuyo valor puede ser “True” o “False”, para definir si el selector es de tipo múltiple o no. Si se omite la opción, será un selector en el que sólo puede seleccionarse una opción.

```
1 choice(table="Aeropuerto", multiple="True") aeropuertos
    ;
```

```
2 choice(process_table="Vuelo", multiple="True") vuelos;
```

- `values`: para definir las distintas opciones del selector como cadenas literales.

  ```
  choice(values="Terminal A","Terminal B","Terminal C
      ") terminal;
  ```

Las opciones “table”, “process_table” y “values” son exclusivas, es decir, solamente se puede escoger una de ellas en la declaración de un mismo campo.

- `chain` y `chain_remote`: para asociar un campo de tipo “choice” con otro del mismo tipo creando una dependencia.

  Para que esta opción sea posible, la entidad o proceso de la opción “table” o “process_table” del campo “choice” padre tiene que contener otro campo “choice” o una relación “many-to-one” con la entidad o proceso de la opción “table” o “process_table” del campo “choice” hijo.

  La opción “chain” tiene que hacer referencia al nombre del campo en la sección o grupo que actúa como campo “choice” padre. La opción “chain_remote” tiene que hacer referencia al nombre de la relación o campo “choice” en la entidad o proceso definido en la opción “table” o “process_table” del campo “choice” padre que lo asocia con la entidad o proceso definido en la opción “table” o “process_table” del campo “choice” hijo.

  Si el valor de la opción “remote_chain” tiene que hacer referencia a un campo que se encuentra en una sección del proceso, se deberá definir como “nombre_proceso.nombre_sección.nombre_campo”.

Para comprender mejor la declaración de estos selectores dependiente se expone el siguiente ejemplo sencillo:

```
entity Ciudad {
    relation(Provincia) provincia_origen;
}
entity Provincia {
    ...
}
section section1 {
    choice(table="Provincia") provincia;
    choice(table="Ciudad", chain="provincia",
        chain_remote="provincia_origen") ciudad;
}
```

- `file`: para la subida de ficheros.

  ```
  file tarjeta_embarque;
  ```

- `bool`: para valores booleano.

  ```
  bool permitido;
  ```

- `url`: para URLs.

  ```
  url web;
  ```

Ámbito `process`: sentencia `state`

Define un estado dentro del proceso. Un estado es un elemento que permite asociar permisos de visibilidad y edición de campos, grupos o secciones entre un rol y el proceso, además de definir transiciones hacia otros estados.

Todos los procesos deben de tener dos estados definidos, uno denominado “initial”, que hace referencia a la fase de creación del proceso, y otro con la denominación pertinente que haga referencia a que el proceso ha sido creado. Además, lógicamente, debe existir una transición del estado “initial” al otro estado dada la decisión de un usuario, sino la creación del proceso sería imposible.

```
state initial {
    ...
}
state comenzado {
    ...
}
```

Ámbito `state`: sentencia `permissions`

Define permisos de visibilidad y/o edición para un rol respecto al proceso, una sección, un grupo o un campo. El nombre de la sentencia `permissions` debe coincidir con el nombre de un rol previamente definido para cumplir correctamente con la asociación.

```
role GestorAeropuerto;
role Pasajero;
process Vuelo {
    ...
    section DatosPasajeros {
        tipo campo_seccion;
        ...
        group Firmas {
            tipo campo_grupo;
            ...
        }
```

```
12      }
13      state initial {
14          permissions GestorAeropuerto {
15              editable all;
16          }
17          permissions Pasajero {
18              viewable DatosPasajeros.campo_seccion;
19              editable DatosPasajeros.Firmas;
20          }
21      }
22  }
```

La sentencia `permissions` puede ir acompañada de `from` o `from transition` haciendo referencia a una transición. La idea en esta declaración es que los permisos que se definan sólo se apliquen para los usuarios que han participado en una o varias transiciones dadas de una instancia del proceso, de esta forma se consiguen aislar los procesos de usuarios que pertenecen al mismo rol.

```
1   role Piloto;
2   process Vuelo {
3       ...
4       section InformacionSalida {
5           ...
6           group HorasSalidaLlegada {
7               ...
8           }
9       }
10      state initial {
11          permissions Piloto {
12              editable all;
13          }
14      }
```

```
15     state despegando {
16         permissions Piloto {
17             viewable all;
18         }
19         permissions Piloto from initial->despegando {
20             editable all;
21         }
22     }
23 }
```

Tipos de permisos:

- `viewable`: determina la visibilidad del proceso, una sección, un grupo o un campo. Cuando se define la visibilidad del proceso completo va seguido de `all`.

  ```
  viewable all;
  ```

  Si se trata de una sección, grupo o campo, éste debe ser referenciado del siguiente modo:

  Sección:

  ```
  viewable nombre_seccion;
  ```

  Campo de una sección:

  ```
  viewable nombre_seccion.nombre_campo;
  ```

  Grupo de una sección:

  ```
  viewable nombre_seccion.nombre_grupo;
  ```

  Campo de un grupo:

```
viewable nombre_seccion.nombre_grupo.nombre_campo;
```

- `editable`: determina los permisos de edición del proceso, una sección, un grupo o un campo. Se utiliza la misma gramática que para el permiso `viewable`.
- `deletable`: determina si las instancias de procesos ya creados pueden ser eliminadas por el rol en particular, por lo tanto, no puede ir asociado a ninguna sección, grupo o campo. Y no puede ir situado en el estado "initial" ya que es el estado de creación del proceso y va siempre seguido de `all`.

  ```
  deletable all;
  ```

Ámbito `state`: sentencia `transition`

Define el cambio de estado de un proceso. Se define sobre un estado en particular y hace referencia al estado al que se va a cambiar si se cumplen las condiciones definidas. Estas condiciones se definen a través de las opciones, que pueden ser:

- `decision_by`: define qué rol de usuario puede participar en la transición de un proceso dado para que se lleve a cabo.

  ```
  transition(decision_by="Piloto") despegando;
  ```

  El cambio de estado a `despegando` se producirá cuando un usuario, cuyo rol es `Piloto`, lo decida.

- `start_date`: define la fecha de comienzo a partir de la cual se puede llevar a cabo la transición.

  ```
  transition(start_date="2016/04/28-12:00:00")
      embarque;
  ```

  El cambio de estado a `embarque` se producirá de forma automática el día 28 de abril de 2016 a las 12:00:00.

- `end_date`: define la fecha límite a partir de la cual ya no se puede llevar a cabo la transición.

  ```
  transition(end_date="2016/04/28-13:00:00")
      cierreEmbarque;
  ```

  El cambio de estado a `cierreEmebarque` no se producirá de forma automática hasta el día 28 de abril de 2016 a las 13:00:00.

- `max`: define el número máximo de instancias del proceso que un usuario puede crear. Solamente puede ir en una transición del estado "initial" acompañada de la opción `decision_by`.

  ```
  process Vuelos {
      ...
      state initial {
          transition(decision_by="Piloto", max="1")
              comprar;
      }
  ```

```
}
```

Un usuario, cuyo rol es `Piloto`, no podrá iniciar más de un proceso `Vuelo`.

Estas opciones no son exclusivas, es decir, pueden ir juntas y el resultado de la condición de transición será el producto lógico de las mismas.

```
transition(decision_by="Azafata", start_date
    ="2016/04/28-14:00:00", end_date
    ="2016/04/28-15:00:00") servir_aperitivos;
```

El cambio de estado a `servir_aperitivos` se producirá cuando un usuario, cuyo rol es `Azafata`, lo decida entre las fechas del 28 de abril de 2016 a las 14:00:00 y el 28 de abril de 2016 a las 15:00:00.

3.4. Especificaciones del entorno desarrollado para Eclipse

En esta sección se describe el entorno que ha sido desarrollado para la herramienta Eclipse con el fin de acomodarla y dotarla de las funcionalidades requeridas para el uso del lenguaje AdminDSL y la generación de código a partir del mismo.

3.4.1. Modelo de casos de uso

Los casos de uso son una técnica de especificación de requisitos apropiada tanto en desarrollos estructurados como en desarrollos orientados a

objetos, siendo especialmente conveniente en este último caso, ya que sirve como referencia a lo largo de todo el ciclo de vida.

Se realiza el modelo de casos de uso identificando:

- Actores: Un actor representa un conjunto coherente de roles que interpretan los usuarios de los casos de uso al interaccionar con el sistema.

- Casos de uso: Un caso de uso describe un conjunto de secuencias de interacciones entre actores y el sistema. Con un caso de uso se describe un comportamiento deseado del sistema.

- Descripción de cada caso de uso: Dónde se indican el escenario principal y los posibles escenarios alternativos.

Diagrama de casos de uso

En la figura 3.13 se muestra el diagrama de casos de uso donde participa el actor principal (desarrollador) con cuatro casos de uso.

Los casos de uso de "Crear fichero APDSL", "Abrir fichero APDSL" y "Guardar fichero APDSL" podrían haberse definidos como heredados de casos de uso propios de Eclipse. Sin embargo, para aclarar mejor el funcionamiento del entorno de AdminDSL se han tomado como casos de uso independientes. Lógicamente, durante el desarrollo, estos casos de usos fueron tomados heredando de las funcionalidades de Eclipse. Por lo tanto, el actual diagrama está enfocado al lector de este documento que no tiene por qué tener conocimientos básicos sobre la herramienta Eclipse.

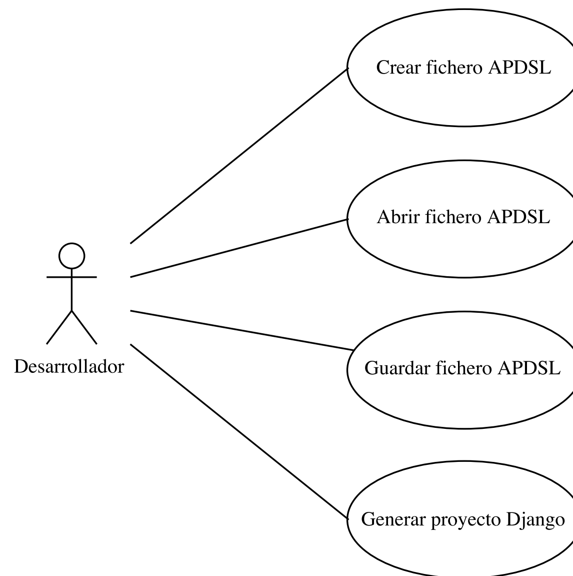


Figura 3.13: Diagrama de casos de uso: Entorno desarrollado para Eclipse.

Especificación de casos de uso

A continuación, se especifican cada uno de los casos de uso mostrados en el diagrama de la figura 3.13.

<u>**Crear fichero APDSL**</u>

- **Actores:**
    - Desarrollador. Entidad interesada en utilizar el lenguaje AdminDSL para definir procesos administrativos.
- **Descripción del escenario.** El desarrollador crea un fichero APDSL que utiliza el lenguaje AdminDSL a través del entorno Eclipse.
- **Precondición.** Se tienen los plugins de AdminDSL requeridos instalados en Eclipse.
- **Postcondición.** Se crea un fichero APDSL listo para ser editado.

- **Escenario principal:**

  1. El desarrollador crea un nuevo proyecto.
  2. El sistema muestra el proyecto creado en la perspectiva “Package Explorer”.
  3. El desarrollador selecciona el proyecto.
  4. El desarrollador utiliza el menú contextual superior de Eclipse y pulsa sobre “File” $\rightarrow$ “New” $\rightarrow$ “Other...”.
  5. El sistema muestra una interfaz de selección para el tipo de fichero que se quiere crear.
  6. El desarrollador selecciona “General” $\rightarrow$ “File” y pulsa sobre el botón “Siguiente”.
  7. El sistema muestra una interfaz donde se debe especificar el nombre del fichero.
  8. El desarrollador escribe el nombre del fichero con la terminación “.apdsl” y pulsa el botón de “Finalizar”.
  9. El sistema crea el fichero y abre el editor por defecto para su edición (editor de AdminDSL).

- **Escenarios secundarios:**

  1a. El desarrollador selecciona un proyecto ya creado. Se continúa a partir del paso 4.

  8a. El desarrollador escribe el nombre del fichero pero ya existe un fichero con ese nombre.

    1. El sistema muestra un aviso de que ya existe un fichero con ese nombre e inhabilita el botón “Finalizar”.
    2. El desarrollador vuelve a introducir un nombre de fichero con la terminación “.apdsl” hasta que sea único en su proyecto y pulsa el botón “Finalizar”.

3. Se continúa con el paso 9.

Guardar fichero APDSL

- **Actores:**
  - Desarrollador. Entidad interesada en utilizar el lenguaje AdminDSL para definir procesos administrativos.
- **Descripción del escenario.** El desarrollador guarda los cambios realizados sobre un fichero APDSL que utiliza el lenguaje AdminDSL a través del entorno Eclipse.
- **Precondición.** Se tiene el fichero a guardar abierto en un editor.
- **Postcondición.** Se guardan los cambios realizados en el fichero APDSL.
- **Escenario principal:**
  1. El desarrollador utiliza el menú contextual superior de Eclipse y pulsa sobre “File” $\rightarrow$ “Save”.
  2. El sistema guarda los cambios realizados sobre el fichero.
- **Escenarios secundarios:**

  1a. El desarrollador pulsa el atajo «Ctrl+S». Continúa Paso 2.

  1b. El desarrollador utiliza el menú contextual superior y pulsa sobre “File” $\rightarrow$ “Save As...”.

    1. El sistema muestra una interfaz para seleccionar el directorio donde se desea guardar y donde se puede especificar el nombre del fichero.
    2. El desarrollador se coloca sobre el directorio deseado, nombra el fichero y pulsa sobre el botón “Guardar”.

   3. El sistema guarda los cambios realizados sobre un fichero nuevo o sobrescribiendo el contenido de otro.

2a. El sistema muestra una interfaz para seleccionar el directorio porque se trata de un fichero recién creado.

   1. El desarrollador se coloca sobre el directorio deseado, nombra el fichero y pulsa sobre el botón “Guardar”.
   2. El sistema guarda los cambios realizados sobre un fichero nuevo o sobrescribiendo el contenido de otro.

Abrir fichero APDSL

- **Actores:**
    - Desarrollador. Entidad interesada en utilizar el lenguaje AdminDSL para definir procesos administrativos.
- **Descripción del escenario.** El desarrollador abre un fichero APDSL que utiliza el lenguaje AdminDSL a través del entorno Eclipse.
- **Precondiciones:**
    - Se tienen los plugins de AdminDSL requeridos instalados en Eclipse.
    - Existe un fichero APDSL previamente creado.
- **Postcondición.** Se abre el fichero APDSL con el editor de AdminDSL para su consulta o edición.
- **Escenario principal:**
    1. El desarrollador utiliza el menú contextual superior de Eclipse y pulsa sobre “File” → “Open file”.
    2. El sistema muestra una interfaz para seleccionar el fichero.

3. El desarrollador selecciona el fichero con extensión ".apdsl".
4. El sistema abre el fichero en el editor por defecto (AdminDSL).

- **Escenarios secundarios:**

  1a. El fichero existe ya en un proyecto abierto.

    1. El desarrollador hace doble click sobre el fichero ".apdsl" en la perspectiva "Package Explorer" de Eclipse.
    2. Continúa paso 4.

Generar aplicación web

- **Actores:**
  - Desarrollador. Entidad interesada en utilizar el lenguaje AdminDSL para definir procesos administrativos y generar una aplicación web.
- **Descripción del escenario.** El desarrollador genera a través del entorno Eclipse una aplicación web en el directorio que desee.
- **Precondiciones:**
  - Se tienen los plugins de AdminDSL requeridos instalados en Eclipse.
  - Existe un fichero APDSL previamente creado y sin fallos.
  - El fichero se encuentra en un proyecto existente y visible en la perspectiva "Package Explorer".
- **Postcondición.** Se crea una aplicación web para la gestión y el control de los procesos administrativos definidos utilizando el lenguaje AdminDSL.

- **Escenario principal:**

  1. El desarrollador hace click derecho sobre el fichero APDSL.
  2. El sistema muestra un menú contextual donde aparece la opción "Generate web application".
  3. El desarrollador pulsa sobre la opción "Generate web application".
  4. El sistema muestra una interfaz para seleccionar el directorio donde se desea crear la aplicación web.
  5. El desarrollador se coloca sobre el directorio deseado y pulsa el botón "Aceptar".
  6. El sistema genera la aplicación web en el directorio seleccionado y muestra una notificación de que ha sido una generación satisfactoria.

- **Escenarios secundarios:**

  4a. El sistema pregunta si se desea utilizar el mismo directorio que la vez anterior (de haberse generado ya esa aplicación previamente).

    1. Si el usuario pulsa el botón "Yes" se continúa con el paso 7.
    2. Si el usuario pulsa sobre el botón "No" se continúa con el paso 5.

  7a. (Excepción) El sistema muestra un aviso donde se especifica que la aplicación web no se ha podido generar y que se revise el "Error Log" de Eclipse.

3.4.2. Modelo de comportamiento del sistema

El modelo de comportamiento del sistema nos permite obtener la especificación del comportamiento, es decir, qué operaciones puede realizar

el actor participante con el sistema y qué acciones puede tomar el sistema con el fin de mantener una comunicación que permita al actor obtener los resultados deseados. Se utilizan los diagramas de secuencia para esta especificación.

Los diagramas de secuencia muestran la secuencia de eventos entre los actores y el sistema y nos permiten identificar las operaciones de este último.

En las figuras 3.14, 3.16, 3.15 y 3.17 se muestran, para cada caso de uso, los diagramas de secuencia asociados.

3.5. Especificaciones de la aplicación web generada

En esta sección se describe la aplicación generada a partir de los procesos administrativos definidos utilizando el lenguaje AdminDSL.

3.5.1. Modelo de casos de uso

A continuación se especifica el modelo de casos de uso de la aplicación web que se genera. Esta especificación ha sido realizada de la forma más generalizada posible. Dado que a través del lenguaje AdminDSL se pueden especificar detalles propios de cada proceso administrativo definido, los casos de uso van a estar enfocados a funcionalidades generales y comunes de las posibles aplicaciones web que se generen a partir de este lenguaje.

Para definir los actores se van a utilizar solamente dos, un superusuario que actúa como administrador de la aplicación a nivel interno y un usuario que representa a cualquiera que pertenezca a un cierto rol de los que se habrían definido con el lenguaje AdminDSL.

También se va a partir de la suposición de que solamente se ha definido un proceso administrativo aunque se permitan definir más en un mismo fichero APDSL con el fin de simplificar las descripciones.

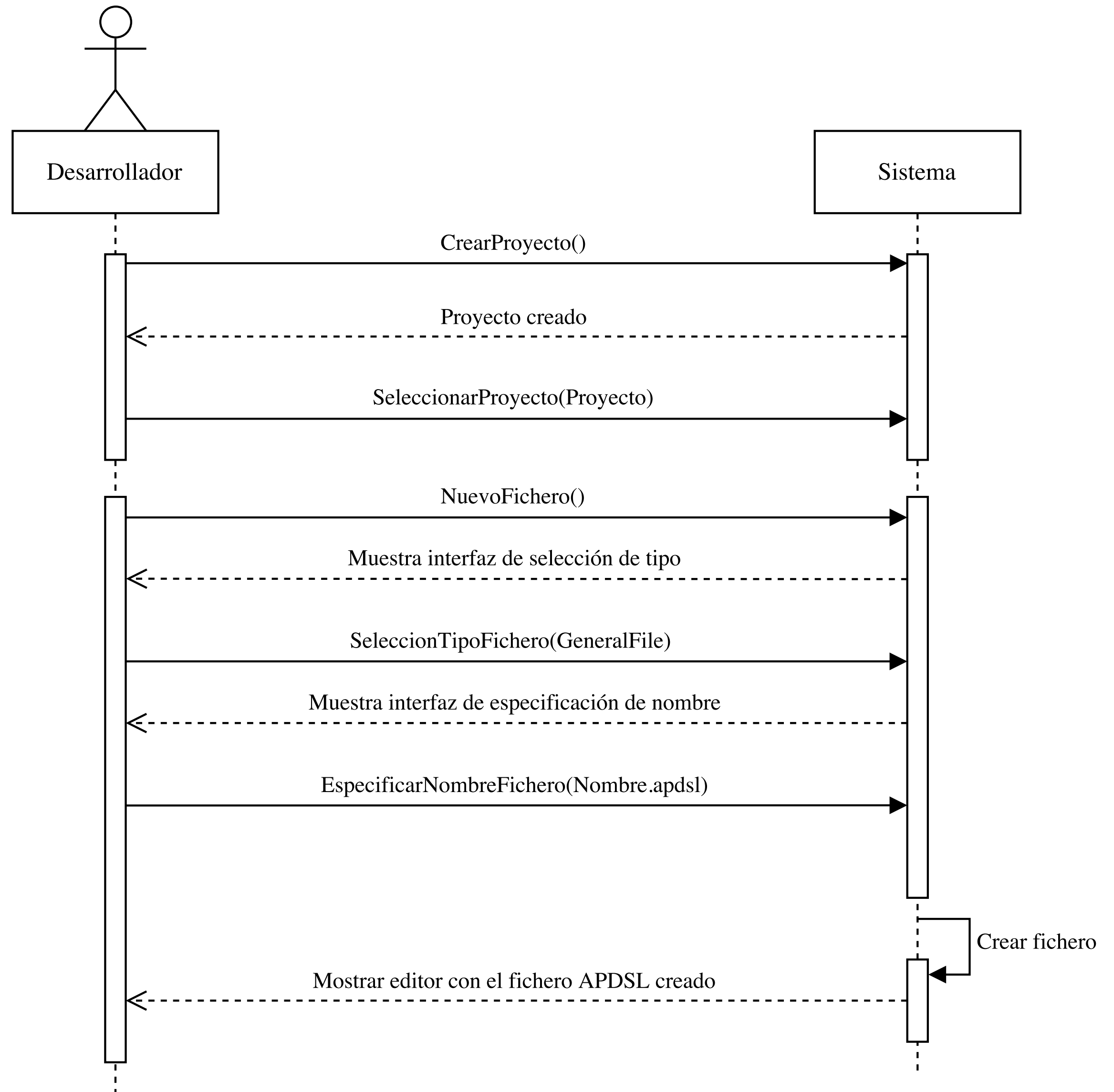


Figura 3.14: Diagrama de secuencia para el caso de uso: “Crear fichero APDSL”.

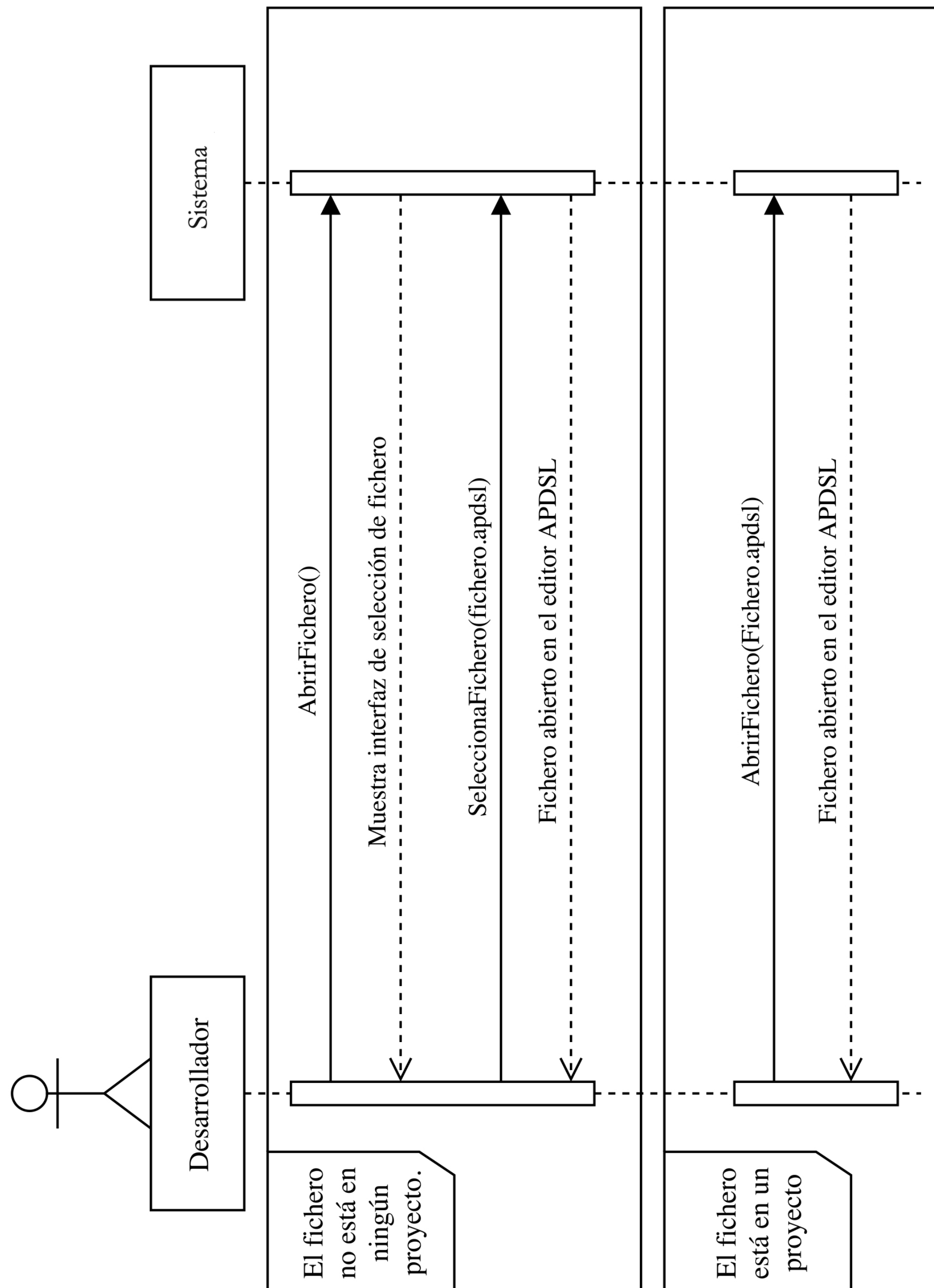


Figura 3.15: Diagrama de secuencia para el caso de uso: "Abrir fichero APDSL".

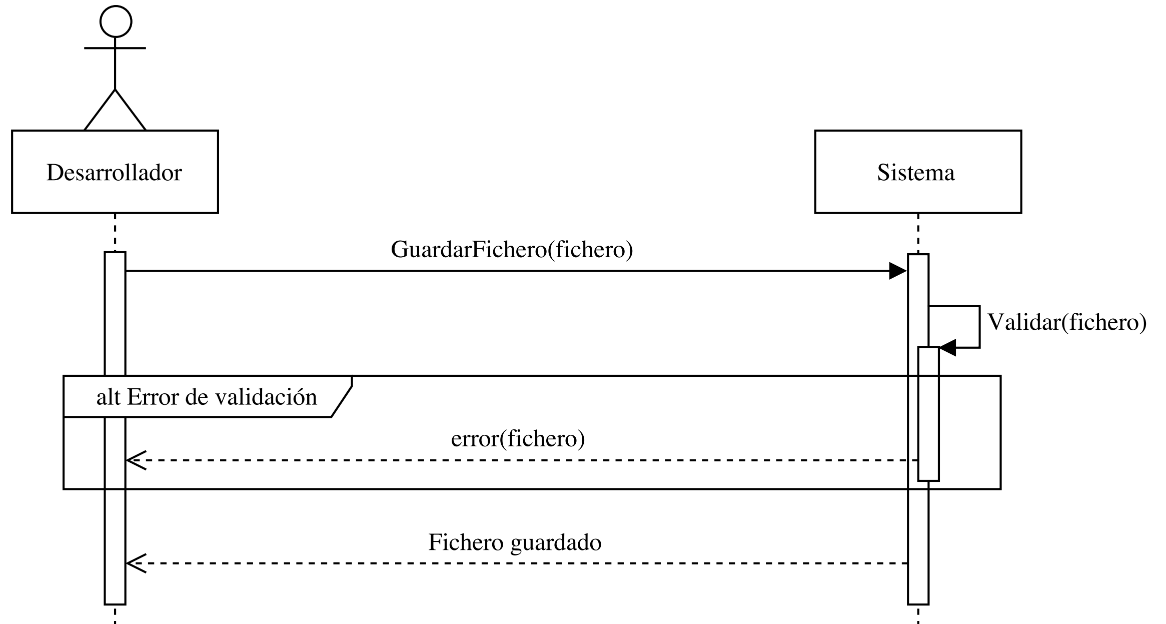


Figura 3.16: Diagrama de secuencia para el caso de uso: "Guardar fichero APDSL".

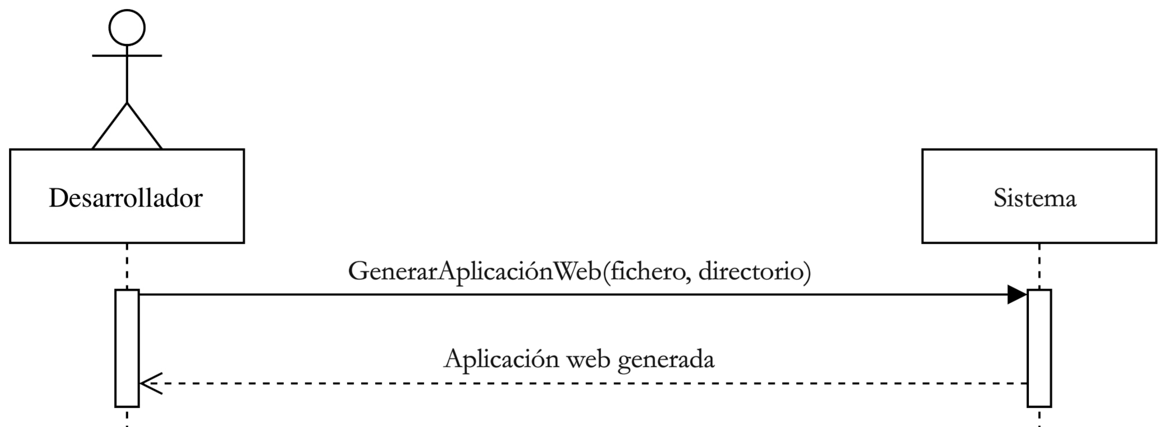


Figura 3.17: Diagrama de secuencia para el caso de uso: "Generar aplicación web".

Diagramas de casos de uso

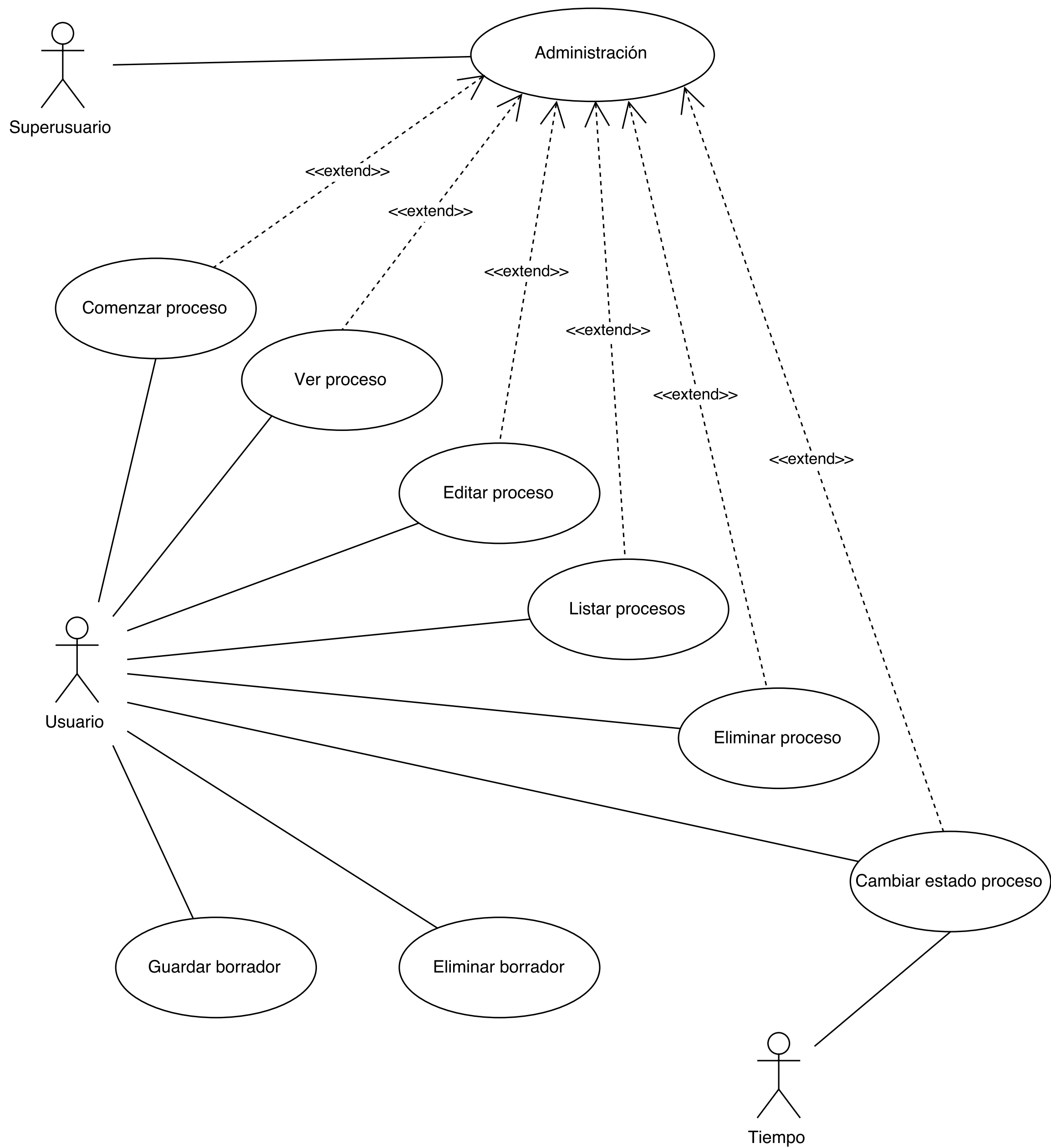


Figura 3.18: Diagrama de casos de uso.

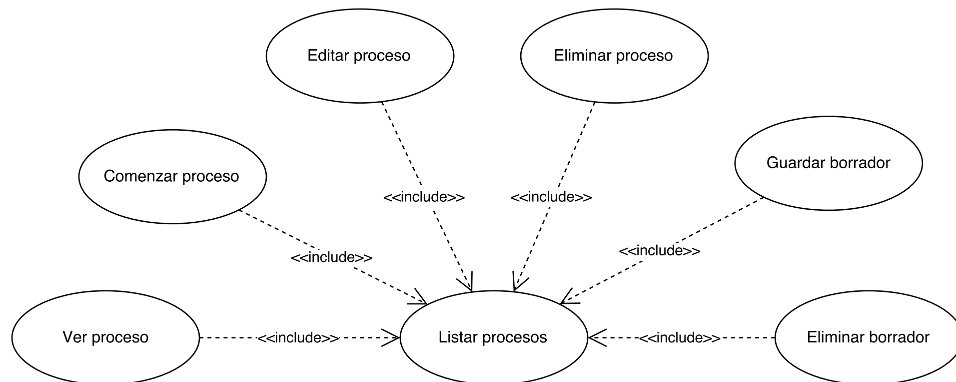


Figura 3.19: Diagrama de casos de uso.

Especificación de casos de uso

Comenzar proceso

- **Actores:**
    - Usuario. Entidad que hace referencia a un usuario perteneciente a un rol definido a través del lenguaje AdminDSL.
    - Superusuario. Entidad administradora que tiene acceso al sitio de administración de la aplicación web.
- **Descripción del escenario.** El usuario comienza un proceso partiendo de un estado inicial y dejándolo en otro estado siguiente, de forma que queda registrado en el sistema.
- **Precondiciones:**
    - El proceso que se quiere comenzar está definido en el sistema.
    - El usuario tiene los permisos necesarios para comenzar un proceso, es decir, puede editar al menos uno de sus campos para el estado “initial”.

- **Postcondición.** Se crea un nuevo proceso y se registra en el sistema.
- **Escenario principal:**
  1. «Include» Listar procesos.
  2. El usuario pulsa sobre el botón “Comenzar” del proceso que desea crear.
  3. El sistema muestra el formulario asociado al proceso.
  4. El usuario cumplimenta el formulario y pulsa el botón “Enviar”.
  5. El sistema valida que el contenido de los campos es correcto.
  6. El sistema registra el proceso con los datos cumplimentados en el formulario y cambia su estado actual (“initial”) por el siguiente estado correspondiente y envía una señal.
- **Escenarios secundarios:**

  6a. El sistema muestra errores en el formulario..
  1. El usuario modifica o cumplimenta los campos que contienen errores y pulsa el botón “Enviar”.
  2. Vuelta al paso 5.

Ver proceso

- **Actores:**
  - Usuario. Entidad que hace referencia a un usuario perteneciente a un rol definido a través del lenguaje AdminDSL.
  - Superusuario. Entidad administradora que tiene acceso al sitio de administración de la aplicación web.
- **Descripción del escenario.** El usuario consulta los campos del formulario del proceso asociado para ver sus contenidos.

- **Precondiciones:**
  - El proceso que se quiere ver existe en el sistema.
  - El usuario tiene los permisos necesarios para ver algún campo del proceso en cuestión.
- **Postcondición.** Se muestran los campos del formulario asociado al proceso administrativo y sus contenidos.
- **Escenario principal:**
  1. «Include» Listar procesos.
  2. El usuario pulsa sobre el botón “Editar” o “Consultar” del proceso en cuestión.
  3. El sistema muestra los campos del formulario y sus contenidos.
- **Escenarios secundarios.** No se contempla ningún escenario secundario.

Editar proceso

- **Actores:**
  - Usuario. Entidad que hace referencia a un usuario perteneciente a un rol definido a través del lenguaje AdminDSL.
  - Superusuario. Entidad administradora que tiene acceso al sitio de administración de la aplicación web.
- **Descripción del escenario.** Se muestran los campos del formulario asociado al proceso administrativo conveniente y se permite al usuario la edición de sus contenidos y el registro de los cambios en los mismos.
- **Precondiciones:**

    - El proceso que se quiere editar existe en el sistema.
    - El usuario tiene los permisos necesarios para editar algún campo del proceso en cuestión.

- **Postcondición.** Se registran los cambios en el proceso administrativo en cuestión.
- **Escenario principal:**
    1. «Include» Listar procesos.
    2. El usuario pulsa sobre el botón “Editar” del proceso que desea editar.
    3. El sistema muestra el formulario asociado al proceso.
    4. El usuario modifica o cumplimenta los campos del formulario y pulsa el botón “Enviar”.
    5. El sistema valida que el contenido de los campos es correcto.
    6. El sistema registra el proceso con los datos modificados o cumplimentados en el formulario y envía una señal.
- **Escenarios secundarios:**

    6a. El sistema muestra errores en el formulario..
      1. El usuario modifica o cumplimenta los campos que contienen errores y pulsa el botón “Enviar”.
      2. Vuelta al paso 5.

Listar procesos

- **Actores:**
    - Usuario. Entidad que hace referencia a un usuario perteneciente a un rol definido a través del lenguaje AdminDSL.

    - Superusuario. Entidad administradora que tiene acceso al sitio de administración de aplicación web.

- **Descripción del escenario.** El desarrollador accede a una vista donde se muestra una lista de procesos.

- **Precondición.** Existe al menos un proceso definido en el sistema.

- **Postcondición.** Se muestra un listado con los procesos administrativos definidos, se puedan comenzar o no, así como los procesos o borradores que ya han sido creados.

- **Escenario principal:**

    1. El usuario entra en la ruta principal de la aplicación.
    2. El sistema muestra el listado de procesos disponibles y no disponibles, así como los procesos o borradores que han sido creados según los permisos del usuario.

- **Escenarios secundarios:**

    1a. El usuario pulsa sobre el botón “TODOS LOS PROCESOS” del menú de navegación superior. Continúa el paso 2.

    1b. El usuario pulsa sobre el nombre del proceso deseado en el menú de navegación superior. Continúa el paso 2.

    2a. Si se ha optado por la ruta asociada a un nombre de proceso en el menú de navegación superior, el sistema muestra los procesos o borradores que han sido creados para ese tipo de proceso administrativo en cuestión.

Eliminar proceso

- **Actores:**

    - Usuario. Entidad que hace referencia a un usuario perteneciente a un rol definido a través del lenguaje AdminDSL.
    - Superusuario. Entidad administradora que tiene acceso al sitio de administración de aplicación web.

- **Descripción del escenario.** El usuario elimina un proceso de alguna de las listas mostradas.
- **Precondiciones:**
    - El proceso que se quiere eliminar existe en el sistema.
    - El usuario tiene los permisos necesarios para eliminar el proceso en cuestión.
- **Postcondición.** Se elimina un proceso administrativo, que ya ha sido comenzado, del sistema.
- **Escenario principal:**
    1. «Include» Listar procesos.
    2. El usuario pulsa sobre el botón "Eliminar" del proceso que desea eliminar.
    3. El sistema muestra un mensaje de confirmación.
    4. El usuario acepta el mensaje de confirmación.
    5. El sistema elimina la instancia del proceso administrativo del sistema y envía una señal.
- **Escenarios secundarios:**

    4a. El usuario cancela el mensaje de confirmación.

        1. El sistema no elimina la instancia del proceso administrativo.

Cambiar estado proceso

- **Actores:**
  - Usuario. Entidad que hace referencia a un usuario perteneciente a un rol definido a través del lenguaje AdminDSL.
  - Superusuario. Entidad administradora que tiene acceso al sitio de administración de la aplicación web.
- **Descripción del escenario.** Se produce un cambio de estado de un proceso administrativo activo, ya sea por decisión del usuario o de forma automática en un momento concreto.
- **Precondiciones:**
  - El proceso existe en el sistema.
  - En caso de ser un cambio por decisión, el usuario tiene los permisos necesarios para realizar dicho cambio sobre el proceso en cuestión.
  - El caso de ser un cambio automático, el proceso no tiene ningún campo obligatorio sin cumplimentar.
- **Postcondición.** El proceso cambia de estado.
- **Escenario principal:**
  1. «Include» Listar procesos.
  2. El usuario pulsa sobre el botón "Editar" del proceso que desea cambiar de estado.
  3. El sistema muestra el formulario asociado al proceso y dos botones de envío.
  4. El usuario pulsa sobre botón "Enviar y cambiar al estado de *estado*".
  5. El sistema valida que todos los campos estén correctamente y que el usuario tiene permitido dicho cambio de estado.

6. El sistema cambia de estado y envía una señal.

- **Escenarios secundarios:**

  1a. El sistema realiza el cambio automático del estado del proceso y envía una señal, si se cumple una cierta condición relacionada con la fecha de comienzo de la transición (de un estado a otro) o la fecha de fin de la transición o ambas.

  6a. El sistema muestra errores en el formulario.

    1. El usuario modifica o cumplimenta los campos que contienen errores y pulsa el botón “Enviar”.
    2. Vuelta al paso 5.

  6b. El sistema confirma que el usuario no puede cambiar de estado.

    1. Si no ha habido ningún error de validación de campos del formulario, el sistema registra los cambios que se hayan podido realizar el proceso sin cambiar de estado.

<u>**Administración**</u>

- **Actores:**
  - Superusuario. Entidad administradora que tiene acceso al sitio de administración de la aplicación web.
- **Descripción del escenario.** El usuario accede al panel de administración de la aplicación web.
- **Precondicióm.** El superusuario es administrador del sistema a nivel de aplicación.
- **Postcondición.** El superusuario tiene acceso a todas las funcionalidades asociadas a la consulta, creación, edición y eliminación de todos los modelos registrados en el sistema para su administración.

- **Escenario principal:**
  1. El superusuario accede al panel de administración haciendo click sobre el enlace facilitado por la interfaz web o mediante la ruta “admin”.
  2. El sistema muestra un menú donde se pueden realizar todas las funcionalidades necesarias para gestionar los modelos que han sido registrados en el sistema para su administración, estos son: usuarios, grupos de usuario, procesos, estados y transiciones.
  3. «Include» Listar procesos, «Include» Crear proceso, «Include» Editar proceso, «Include» Ver proceso, «Include» Eliminar proceso.
- **Escenarios secundarios.** No se contempla ningún escenario secundario.

<u>**Guardar borrador**</u>

- **Actores:**
  - Usuario. Entidad que hace referencia a un usuario perteneciente a un rol definido a través del lenguaje AdminDSL.
- **Descripción del escenario.** El usuario guarda un borrador de un proceso con el fin de continuar su creación o edición más adelante.
- **Precondiciones:**
  - El proceso del que se quiere guardar un borrador existe en el sistema o está siendo creado.
  - El usuario tiene los permisos necesarios para crear o editar algún campo del proceso en cuestión.

- **Postcondición.** Se registrar un borrador del proceso sin modificar la instancia del mismo qué solo puede ser gestionado por el usuario que lo ha guardado.
- **Escenario principal:**
  1. «Include» Listar procesos.
  2. El usuario pulsa sobre el botón “Editar” del proceso que desee.
  3. El sistema muestra el formulario asociado al proceso con un botón de guardado, además de los de envío.
  4. El usuario pulsa sobre el botón “Guardar”.
  5. El sistema registra el borrador y muestra además un botón de eliminación de borrador.
- **Escenarios secundarios.** No se contemplan escenarios secundarios.

<u>**Eliminar borrador**</u>

- **Actores:**
  - Usuario. Entidad que hace referencia a un usuario perteneciente a un rol definido a través del lenguaje AdminDSL.
- **Descripción del escenario.** El usuario elimina un borrador de un proceso con el fin de conservalo en su estado original (que podría no existir y por lo tanto no se crearía un proceso).
- **Precondiciones:**
  - El proceso del que se quiere eliminar un borrador existe en el sistema o está siendo creado.
  - Existe un borrador en curso para el proceso en cuestión.

  - El usuario tiene los permisos necesarios para crear o editar algún campo del proceso en cuestión.

- **Postcondición.** Se elimina un borrador del proceso de forma que el usuario vuelve a verlo en su estado original (si ya existe) o directamente no verlo (si se realizó el borrador en la fase de comienzo).

- **Escenario principal:**

  1. «Include» Listar procesos.
  2. El usuario pulsa sobre el botón "Eliminar" del borrador de proceso que desee.
  3. El sistema muestra una ventana de confirmación.
  4. El usuario pulsa sobre el botón de "Aceptar".
  5. El sistema elimina el borrador y en caso de que el borrador hiciera referencia a un proceso que aún no ha sido comenzado, también se eliminaría cualquier rastro del mismo.

- **Escenarios secundarios:**

  2a. El usuario pulsa sobre el botón "Editar" del borrador de proceso que desee.

    1. El sistema muestra el formulario asociado al borrador.
    2. El usuario pulsa sobre el botón "Eliminar borrador".
    3. El sistema muestra una ventana de confirmación.
    4. El usuario pulsa sobre el botón "Aceptar".
    5. El sistema elimina el borrador y muestra el formulario en su estado original o sale a la pantalla anterior (si el borrador era de un proceso aún no comenzado).

  4a. El usuario cancela el mensaje de confirmación.

    1. El sistema no elimina el borrador del proceso administrativo.

3.5.2. Modelo conceptual de datos

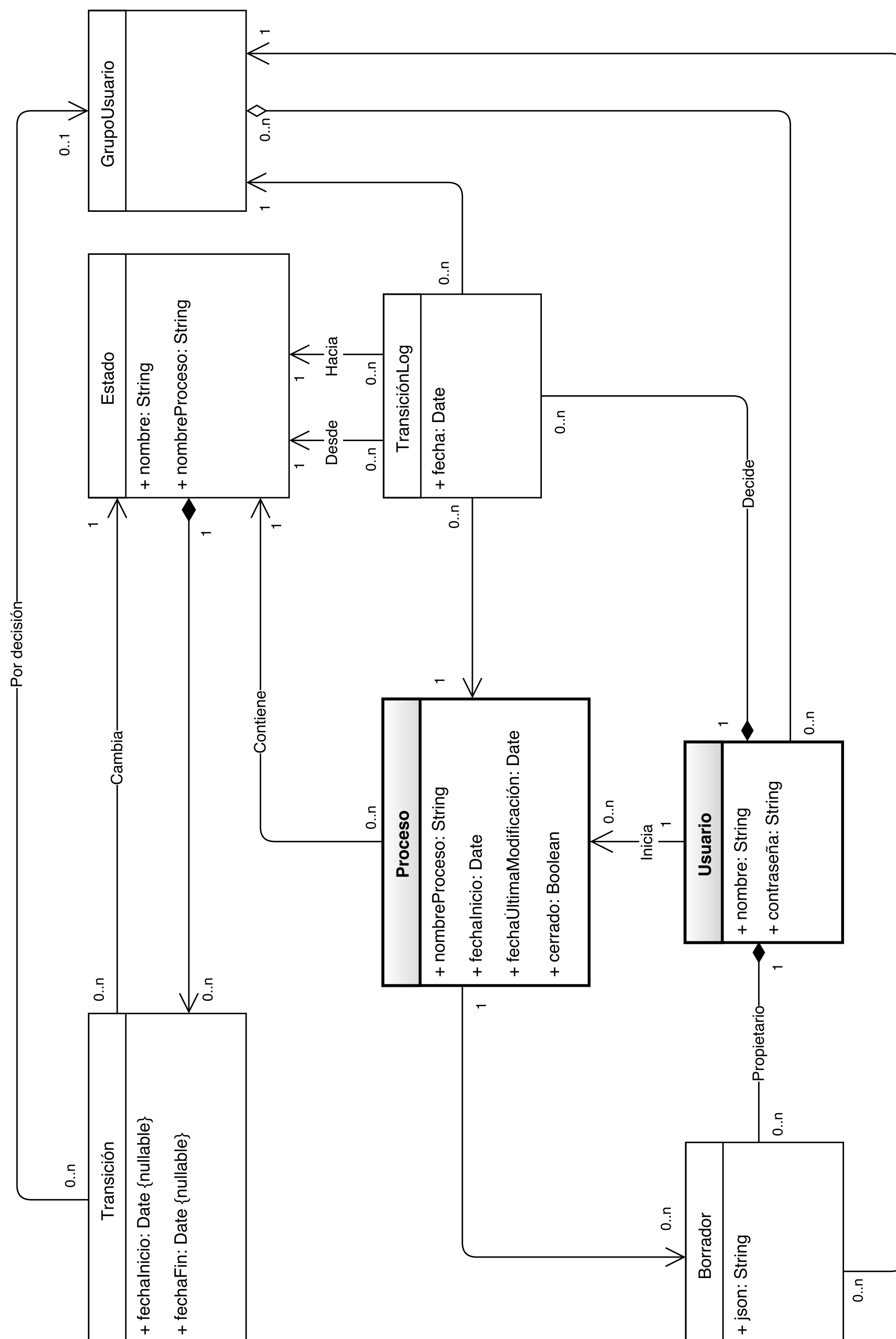


Figura 3.20: Diagrama conceptual de datos asociado al módulo base de la aplicación web generada.

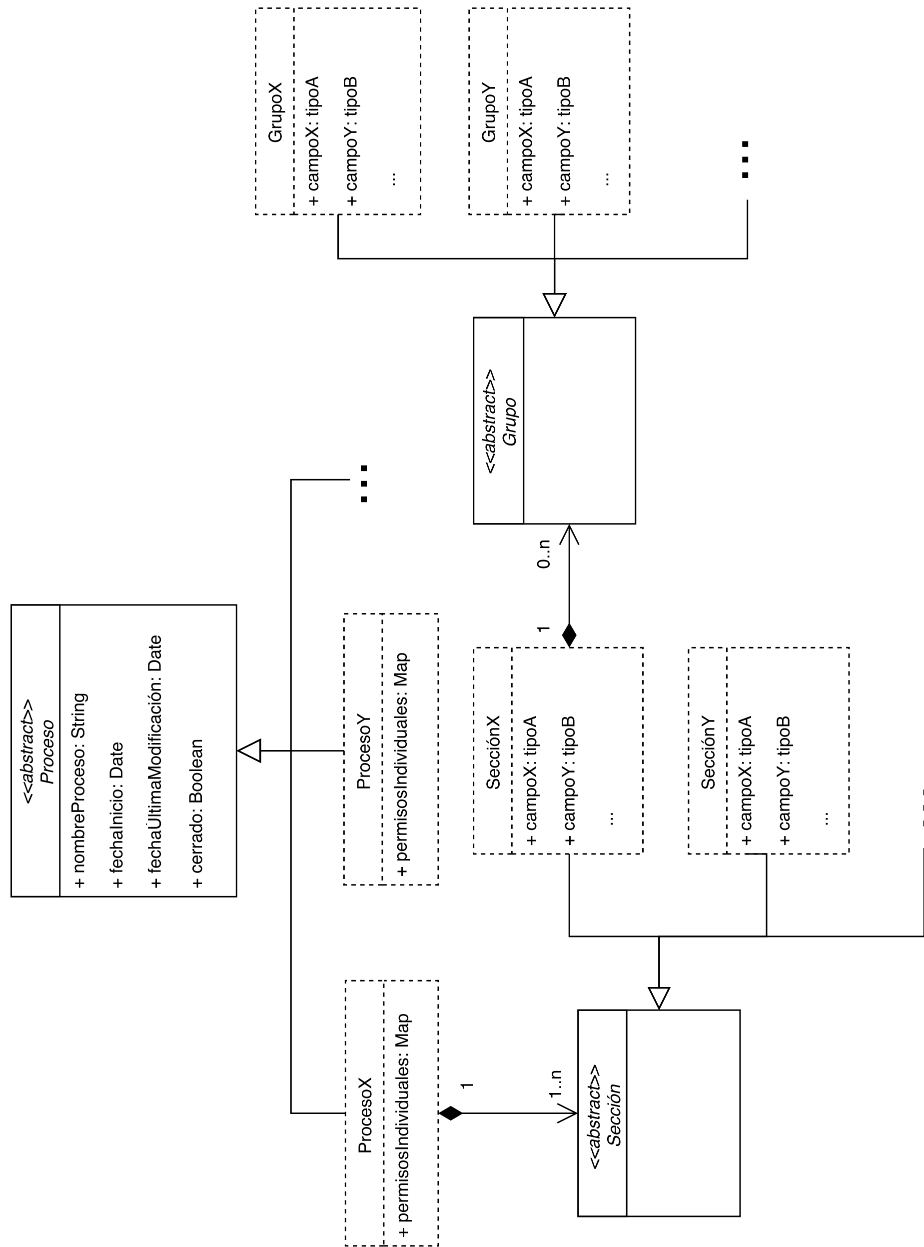


Figura 3.21: Diagrama conceptual de datos asociado a cada módulo de proceso administrativo de la aplicación web generada.

En la figura 3.20 se muestra el diagrama conceptual de datos del esqueleto central de la aplicación web a generar.

Por un lado, se encuentra la clase "Estado" que se compone de transiciones (aunque puede haber un estado sin transiciones), estas transiciones apuntan a otro estado y podrán venir asociadas por decisión de un grupo de usuario concreto o por tiempo a través de sus dos atributos.

Los procesos se encuentran en un estado y además se permite obtener un registro de los estados por los que ha navegado mediante la clase de asociación "TransitionLog". Cada proceso es iniciado por un usuario que pertenece a un cierto grupo de usuario y éste puede además crear borradores de los procesos con los que mantiene una relación de pertenencia.

En la segunda figura 3.21 se observa el modelo conceptual de datos asociado a los procesos administrativos vistos como documentos formales, que es lo que puede verse diferente en cuanto a estructura según cómo se hayan definido las secciones, grupos y campos de cada proceso administrativo.

Se encuentra una clase abstracta "Proceso" como clase padre de las clases asociadas a cada proceso administrativo que se haya podido definir a través del código de AdminDSL. De esta forma, por cada tipo de proceso administrativo que se haya definido en el DSL, se definirá una clase en la aplicación web generada. Esto resulta útil para facilitar la diferenciación de las mismas y poder abstraer la lógica general de un proceso a la lógica específica de un tipo concreto de proceso administrativo.

Con las secciones y los grupos ocurre básicamente lo mismo. De forma que la estructura quedaría de la siguiente forma:

Un proceso se compone de secciones (como mínimo una) las cuales agrupan un conjunto de campos como atributos en sí mismas. Estas secciones a su vez, pueden componerse de grupos que también agruparían un conjunto de campos.

3.5.3. Modelo de comportamiento del sistema

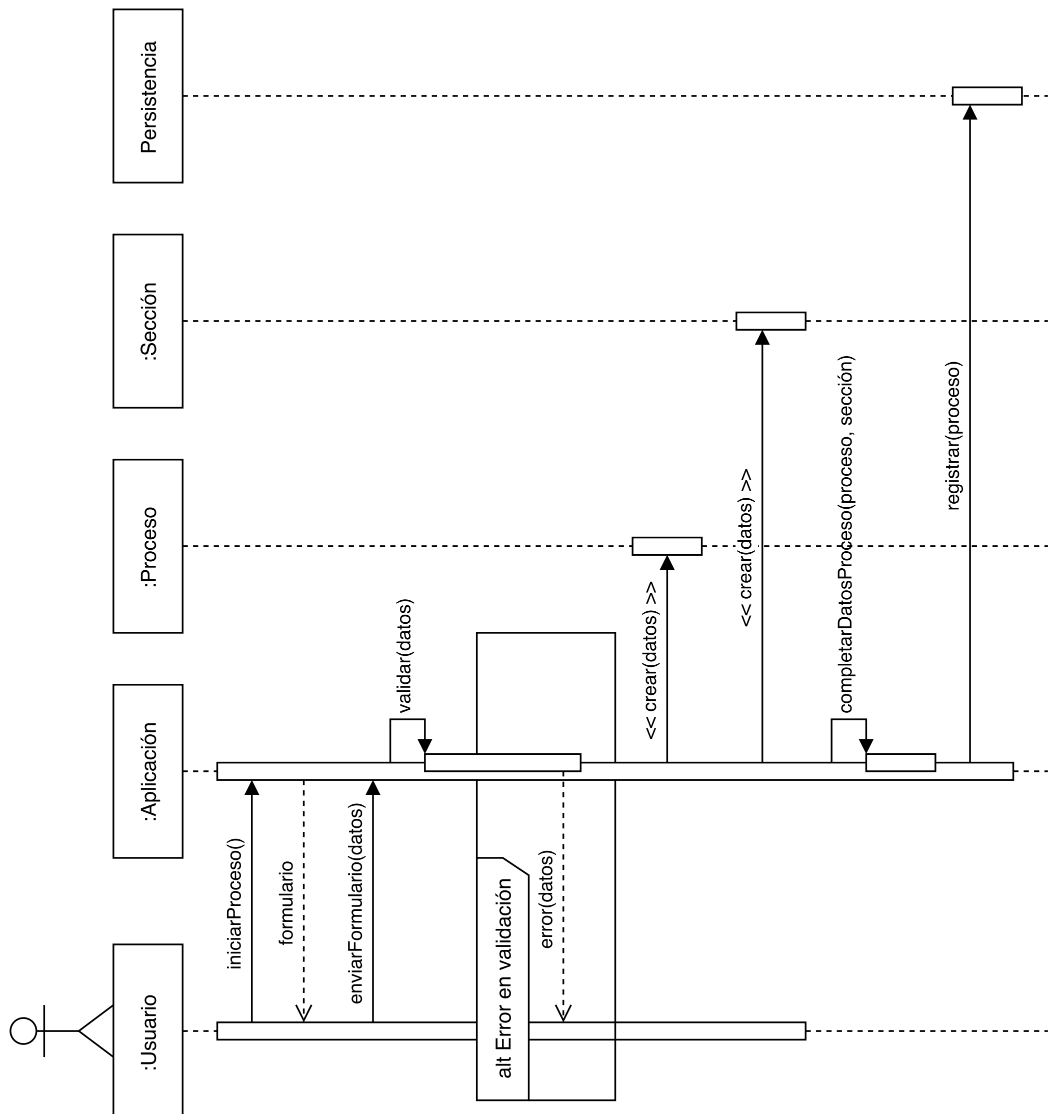


Figura 3.22: Diagrama de secuencia para el caso de uso: “Comenzar proceso”.

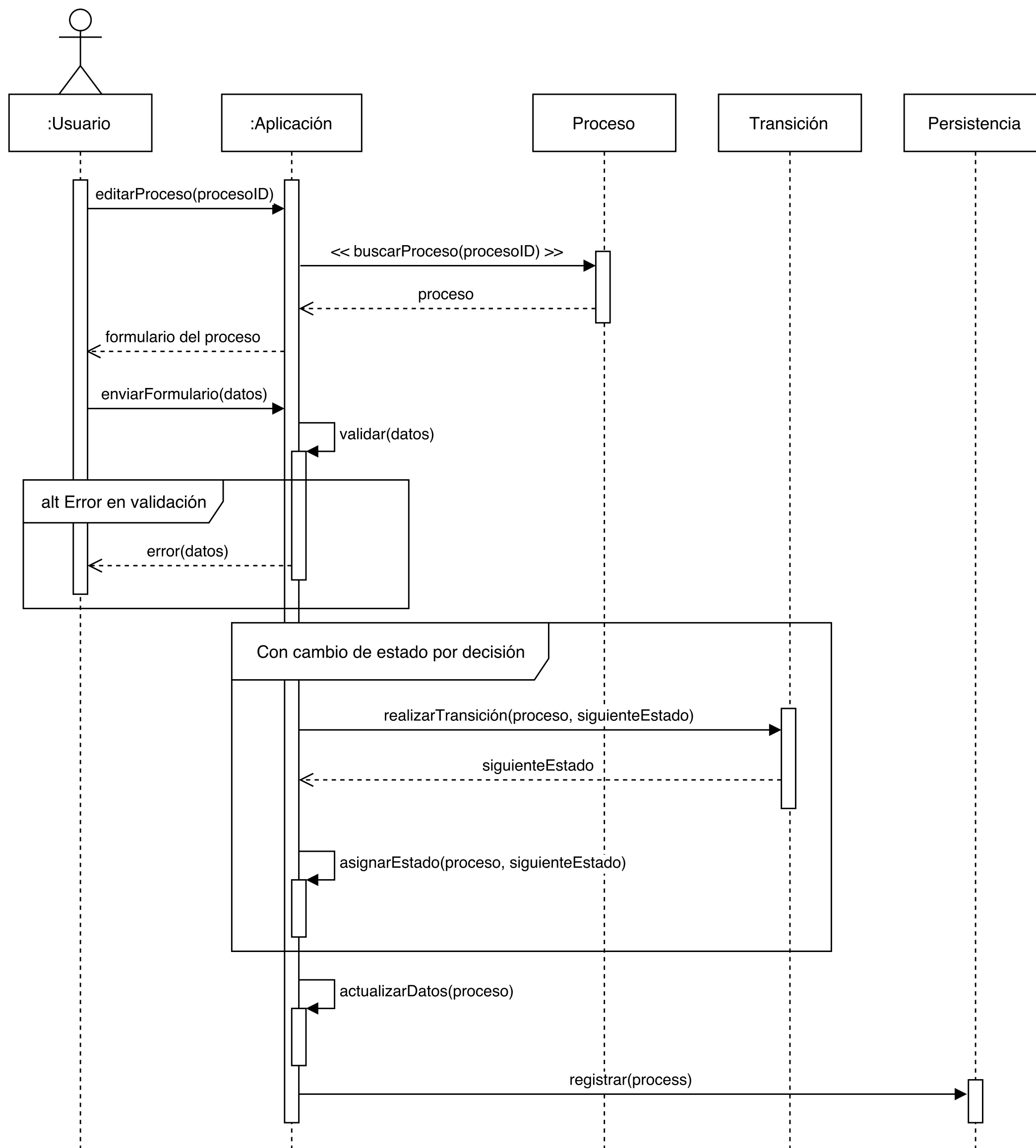


Figura 3.23: Diagrama de secuencia para el caso de uso: “Editar proceso”.

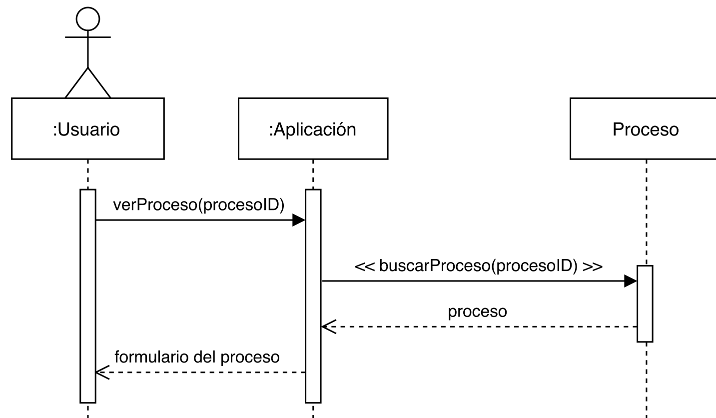


Figura 3.24: Diagrama de secuencia para el caso de uso: “Ver proceso”.

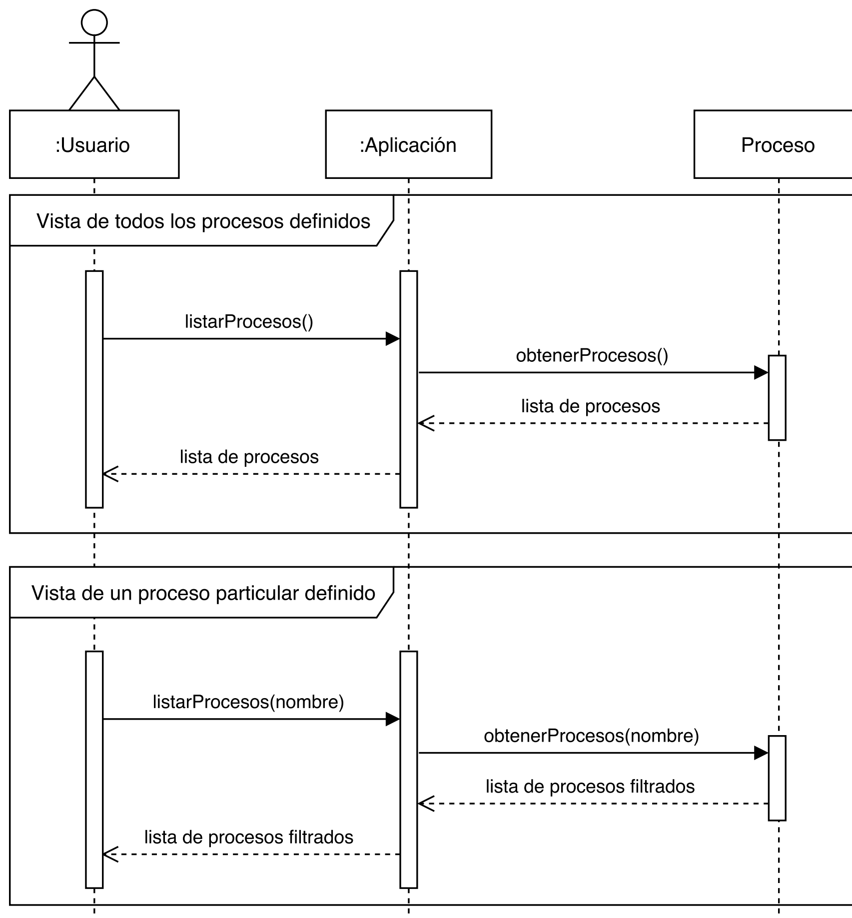


Figura 3.25: Diagrama de secuencia para el caso de uso: “Listar procesos”.

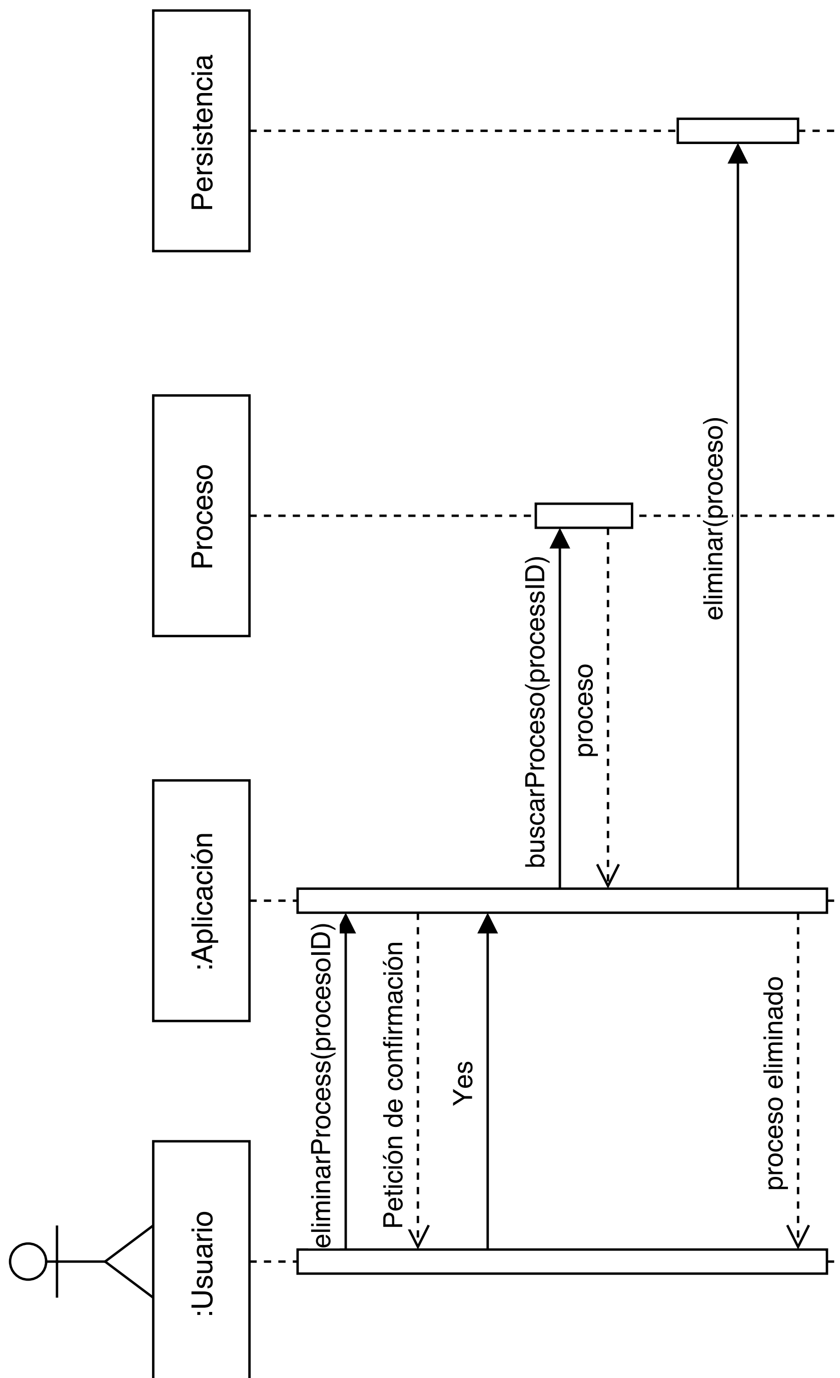


Figura 3.26: Diagrama de secuencia para el caso de uso: "Eliminar proceso".

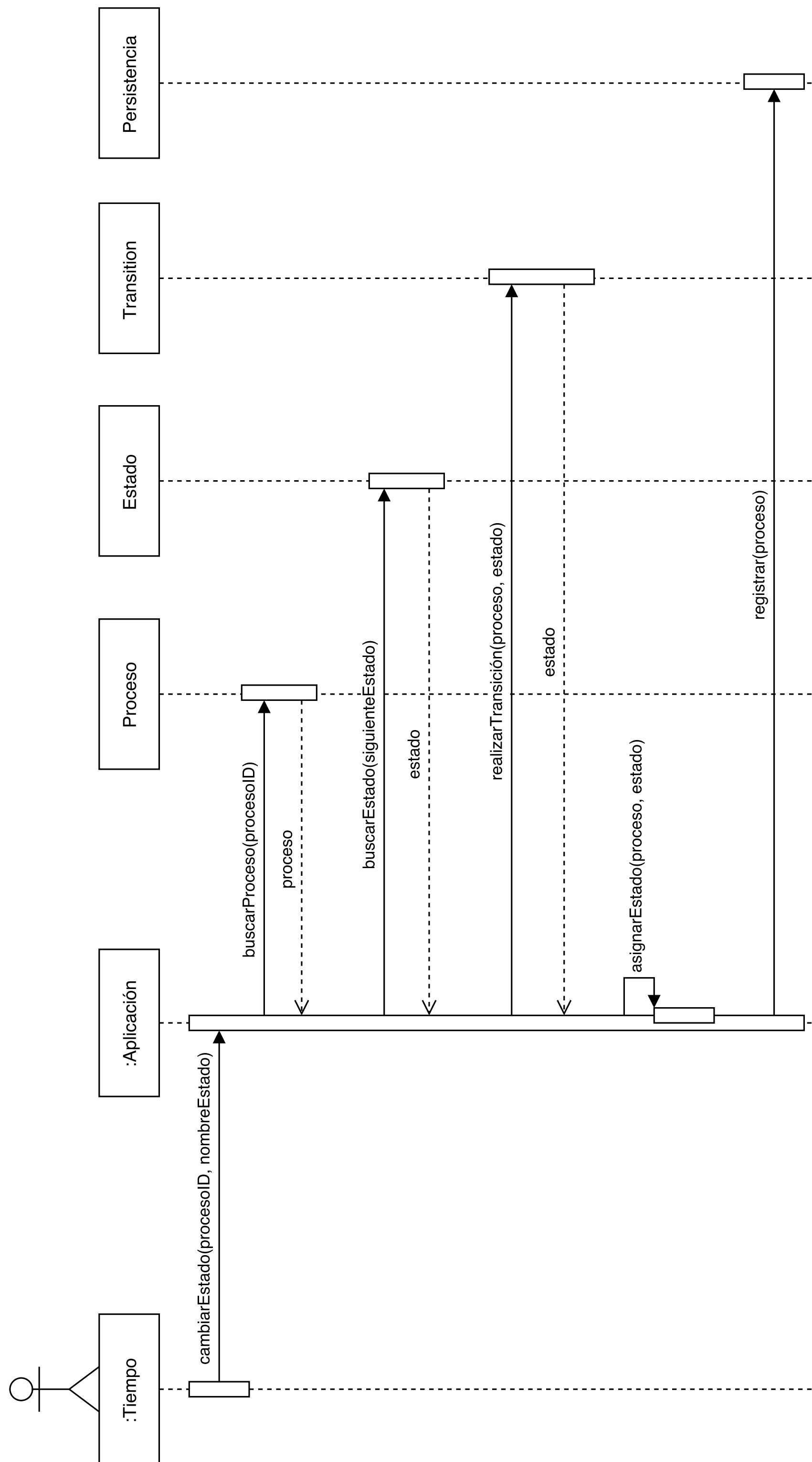


Figura 3.27: Diagrama de secuencia para el caso de uso: "Cambiar Estado proceso" con cambio automático. Véase figura 3.23 para cambio de estado por decisión.

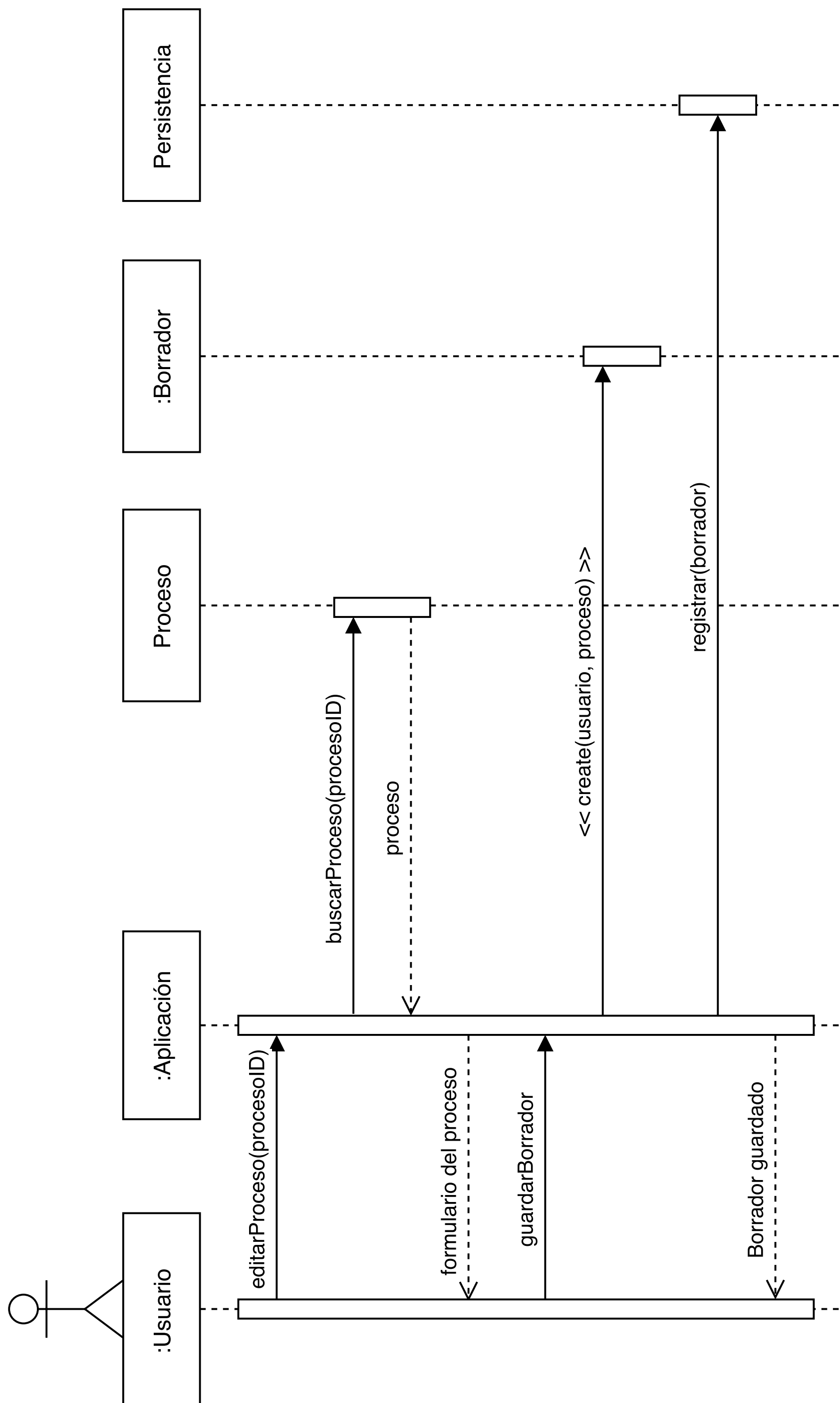


Figura 3.28: Diagrama de secuencia para el caso de uso: “Guardar borrador”.

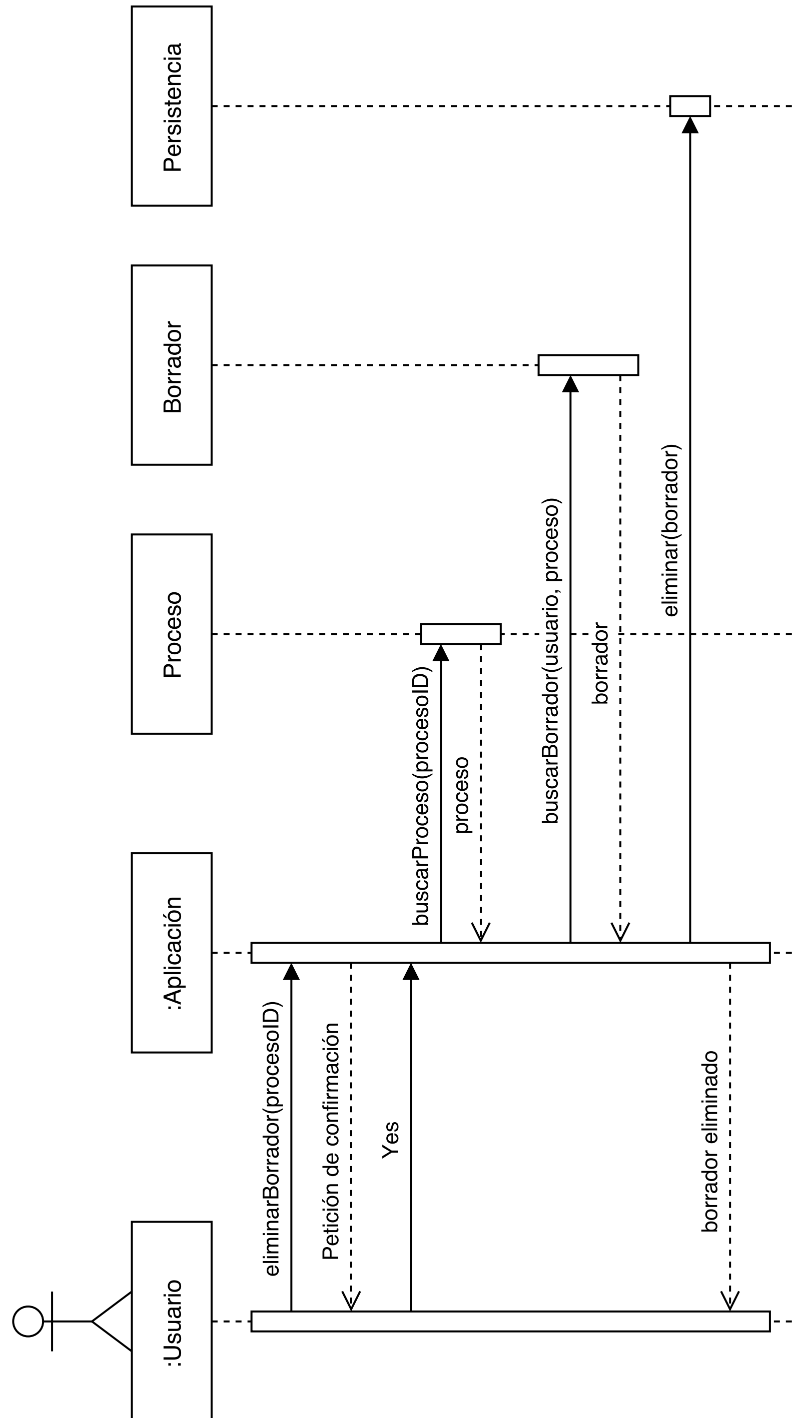


Figura 3.29: Diagrama de secuencia para el caso de uso: "Eliminar borrador".

CAPÍTULO 4
ADMINDSL

4.1. Diseño del editor basado en Xtext

Como ya se había explicado antes en el apartado de tecnologías del capítulo 1, Xtext se presenta como un entorno de ingeniería de lenguajes que permite tanto la generación del analizador como la de un AST así como la vinculación entre ambos.

A nivel de desarrollo, lo más importante es saber cómo trabajar con Xtext y para ello resulta crucial identificar los componentes del mismo y qué funcionalidades nos ofrecen. Para este trabajo se han utilizado los siguientes componentes (ver figura 4.1):

- *Proyecto xtext*. Donde se define el lenguaje y los aspectos internos del editor asociado. Este proyecto se compone de los siguientes elementos:
  - *GenerateDSL.mwe2*. Un generador de los componentes del lenguaje que se encarga principalmente de la generación del metamodelo y el entorno que le rodea.
  - *DSL.xtext*. Un fichero de definición del lenguaje. Donde se definen las reglas que conforman la gramática del lenguaje y a partir de

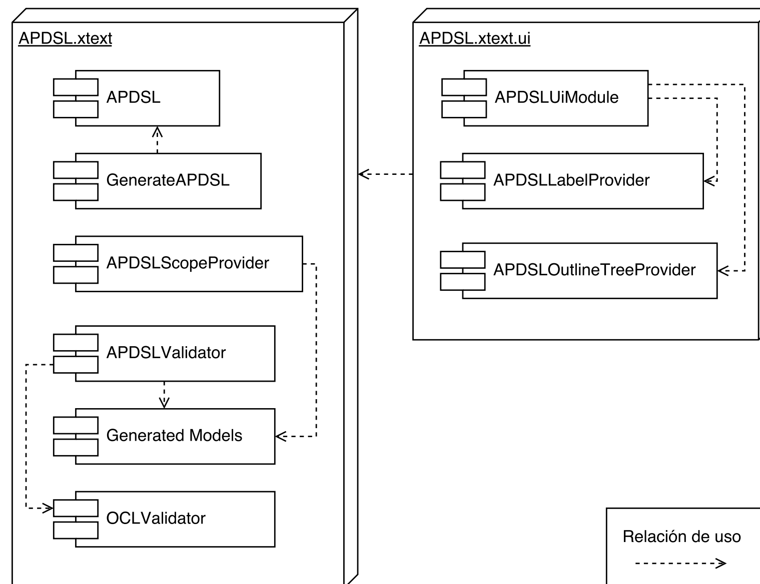


Figura 4.1: Diagrama de componentes del entorno Xtext para el desarrollo de AdminDSL

la cual se generará el metamodelo.

- *Validator.xtend*. Un módulo de validación donde se pueden añadir o modificar validaciones sobre el lenguaje.

- *ScopeProvider.xtend*. Un módulo para el soporte del alcance léxico con el fin de solventar problemas con ciertas referencias cruzadas.

- *Proyecto xtext.ui*. Donde se define la interfaz de usuario. Este plugin sólo ha sido utilizado para la modificación de la presentación de los elementos del lenguaje en la vista *outline*.

Algunos componentes no han sido mencionados puesto que no han sido utilizados durante el desarrollo de este trabajo.

4.2. Desarrollo del lenguaje AdminDSL

En este apartado se explica cómo se ha desarrollado el lenguaje AdminDSL donde ha sido utilizada la herramienta Eclipse junto con la tecnología Xtext.

Para facilitar ciertas explicaciones se exponen fragmentos de códigos que no tienen por qué ser iguales a los originales y en los que pueden haberse omitido ciertas líneas con el fin de facilitar su comprensión.

4.2.1. Definición del lenguaje

Para llevar a cabo la definición del lenguaje con Xtext se ha creado un proyecto Xtext denominado *es.uca.apdsl*. En el fichero "APDSL.xtext" del directorio *src* del proyecto *es.uca.apdsl.xtext*, que se crea automáticamente, es donde definimos la gramática de la siguiente forma vista a continuación.

Definición de reglas

Hay que partir desde la visión de una estructura jerárquica, un árbol de sintaxis abstracta, donde la raíz para este caso será denominada "Application" y contendrá todos los elementos (realmente visto como nodos) que se puedan definir con AdminDSL. Enfocado de otra manera (más orientada a objetos), un objecto de la clase "Application" tendrá un atributo "elements" donde se almacenarán entre cero y varios objetos de las clase "Element", que no es más que una superclase de las clases "Site", "Options", "Roles", "Entity" y "Process".

Se denominará regla a cada sentencia que define estas características en el lenguaje de Xtext. Por ejemplo, las reglas asociadas a "Application" y a "Element" serían las siguientes:

```
1 Application:
2     elements += Element*
```

```
;
Element:
    Site | Options | Role | Entity | Process
;
```

A partir de estas reglas, Xtext no sólo genera el analizador sino que además aporta un conjunto de clases del AST donde, por ejemplo, para la primera regla del ejemplo previo se encontraría la clase "Application" con un atributo "elements" como colección de objetos de la clase "Element".

Esto significa que en el lenguaje, a nivel global, podremos añadir sentencias del tipo "Site", "Options", "Roles", "Entity" y/o "Process". Dicho esto, se trata de definir una regla para cada nodo que defina su estructura y los subnodos (en caso de que tengan) que puedan venir asociados.

A modo de ejemplo, se define la regla "site" como sigue:

```
Site:
    'site' name=ID ';'
;
```

Como se puede ver en este ejemplo, las palabras reservadas o partes textuales de cada sentencia se definen entre comillas simples. Por otro lado, `name` se tratará como un atributo del elemento de tipo "Site" que contendrá el identificador que se le haya querido dar haciendo uso de este lenguaje. Esto quiere decir que se podrá definir sentencias del tipo "Site" del siguiente modo:

```
site aplicacion1;
```

Los atributos que se definan en la regla siempre pueden tener su propio ámbito donde se permitan añadir subnodos, como en el caso de "Application".

```
Process:
    'process' name=ID '{'
        (elements += ProcessElement)+
    '}'
;
ProcessElement:
    State | Entity | Section | Relation
;
```

En este caso se define la regla para la sentencia de tipo “Process”. En principio se debe escribir la palabra reservada “process” seguida de un identificador y una llave “{”. Esta apertura de llaves delimita el comienzo del ámbito del proceso terminado a través del cierre con “}”. Esto implica, a nivel del lenguaje, que todo lo definido entre las llaves de un proceso pertenecerá al mismo (almacenado en su atributo “elements” de forma directa o mediante una navegación entre los elementos relacionados).

Referencias cruzadas

En algunos casos interesa colocar una referencia a un nodo. Por ejemplo, si queremos definir un permiso para un rol de usuario debemos indicar que la sentencia del tipo “StateRole” hace referencia a un rol existente. Para ello, colocamos el nombre de la regla entre corchetes, como se podrá ver a continuación. A este tipo de referencias, se les denomina referencias cruzadas.

```
StateRole:
    'permissions' role=[Role] '{'
        (permissions += Permission)*
    '}'
;
```

Esto implica lo siguiente:

- Sentencia correcta:

```
role Jefe;
...
    permissions Jefe {
        ...
    }
```

- Sentencia incorrecta (no existe rol Trabajador):

```
role Jefe;
...
    permissions Trabajador {
        ...
    }
```

El problema de la ambigüedad

Mostrados estos ejemplos, lo más importante es mantener la definición del lenguaje libre de ambigüedades. Por ejemplo, dada la regla que se ha visto en el ejemplo de la sentencia "Site", no podríamos definir otra sentencia que tuviese la misma estructura coincidiendo en las palabras reservadas. Visto así, parece fácil y claro no caer en ambigüedades, pero cuando tenemos una serie de nodos y subnodos encadenados esto puede resultar más complicado de ver. Por ejemplo, las siguientes reglas poseen una ambigüedad:

```
Ejemplo1:
    'ejemplo' cadena=STRING ';'
;
Ejemplo2:
    'ejemplo' elements += EjemploElement* ';'
```

```
6  ;
7  EjemploElement:
8      ID | STRING | ANOTHERTYPE
9  ;
```

La ambigüedad reside en que se puede dar el siguiente caso.

Se tiene la siguiente sentencia:

```
1  ejemplo "Esto es un ejemplo";
```

El intérprete podría pensar que es una sentencia definida en la regla "Ejemplo1" donde el atributo "cadena" toma como valor "Esto es un ejemplo", pero también podría pensar que se trata de una sentencia "Ejemplo2" donde el atributo "elements" contiene un elemento del tipo "EjemploElement" que a su vez es un elemento "STRING" o sea "Esto es un ejemplo". Por lo tanto, este conjunto de reglas no son válidas en la definición de un lenguaje y Xtext informaría de que no es una estructura válida.

Solucionar problemas con referencias cruzadas

Si bien, cuando se trabaja con referencias cruzadas con más de un nivel de jerarquía o referencias a reglas que a su vez pueden especificarse en una u otra regla de forma exclusiva, se debe de "ayudar" en cierto modo al intérprete. Para ello se utiliza el módulo "ScopeProvider" que ofrece Xtext. Es un fichero en Java o Xtend que permite resolver problemas de alcance a nivel léxico que no se pueden solventar a nivel gramatical.

```
1  SectionElement:
2      Field | Group
3  ;
4  PermissionTarget:
5      section=[Section] (tail=TailRef)?
6  ;
```

```
7  TailRef:
8      '.' element=[SectionElement] (tail=TailRef)?
9  ;
10 ElementRef:
11     PermissionTarget | TailRef
12 ;
```

En este ejemplo anterior, se quiere poder definir sentencias que contengan referencias del tipo *sección* o *sección.campo* o *sección.grupo* o *sección.grupo.campo*.

En la regla "PermissionsTarget" se define que el primer elemento será una referencia cruzada a una sección que podrá venir o no acompañada de un nodo "TailRef". Con esto ya se permite la sentencia de tipo *sección* a solas. Ahora bien, en "TailRef" se define los elementos que van encadenados a la sección pero además, de forma recursiva, estos pueden ir encadenados a otro "TailRef". Estos "TailRef" contienen un elemento del tipo "SectionElement", que a su vez puede ser "Field" o "Group" y de manera opcional otro nodo "TailRef".

El problema surge en que el elemento de "TailRef" no se identifica como un campo o un grupo a través del intérprete que Xtext crea por defecto, de forma que no se puede realizar correctamente la comprobación de la referencia cruzada. Por ejemplo, la sentencia *sección.grupo.campo* sería correcta mientras que la sentencia *sección.campo.campo* no. De esta forma, comprobar la validez del tercer elemento no se lleva a cabo de manera ortodoxa sino que debe hacerse uso del módulo "ScopeProvider".

En este módulo se definen los siguientes métodos:

```
1  def IScope scope_TailRef_element(EObject context,
       EReference ref /* not used */) {
2      val container = context.eContainer as ElementRef
3      Scopes::scopeFor(container.refElements)
```

```
}
```

Este primero define el alcance (es decir los elementos que contiene) según sea un “PermissionTarget” o un “TailRef”, de ahí que se use la regla “ElementRef” que actúe como superclase de ambas.

Ahora toca definir qué elementos debe devolver la llamada a refElements según el tipo que sea:

```
def private dispatch refElements(PermissionTarget exp) {
    exp.section.elements
}
def private dispatch refElements(TailRef tailRef) {
    tailRef.element.elements
}
```

Como se puede observar arriba, si se trata de un “PermissionTarget” se devuelve los elementos que contenga el elemento almacenado en el atributo “section” que posee una referencia cruzada a un objeto de tipo “Section”. Y en caso de que sea un “TailRef” se obtendrán los elementos que contenga el elemento almacenado en el atributo “element”. Pero en este último caso, el atributo “element” contiene una referencia cruzada a una superclase “SectionElement”, luego también hay que aplicar una distinción, según se trate de un “Group” o un “Field”.

```
def dispatch elements(Section it) {
    elements.filter(typeof(SectionElement))
}
def dispatch Iterable<Field> elements(Group it) {
    elements
}
def dispatch Iterable<SectionElement> elements(Field it) {
    emptyList
```

```
9 }
```

Dado que “Field” no contiene ningún elemento se utiliza “emptyList” (lista vacía) de esta forma la sentencia *sección.campo1.campo2* nunca podrá ser válida porque *campo2* nunca aparecerá en una lista vacía.

4.2.2. Editor

El editor es desarrollado de forma totalmente automatizada por la tecnología Xtext. De forma que simplemente describiendo una gramática se crea un editor asociado a la misma, con los resaltados y las validaciones pertinentes. No obstante, la vista *outline* asociada al editor ha tenido que ser modificada a nivel de etiquetas, puesto que por defecto algunos de los elementos representados aparecían sin nombre o con un nombre poco representativo.

La vista *outline* es una vista asociada al editor que muestra todos los elementos que se han definido través del lenguaje mostrándolos de forma jerarquizada. Para ello, asocia una pestaña para cada elemento contenedor que permita desglosarlo en los elementos que contiene.

Para modificar las etiquetas de cada uno de los elementos que pueden ser mostrados en la vista *outline* se modifica el fichero *APDSLLabelProvider.xtend* situado en el proyecto *es.uca.apdsl.xtext.ui* que fue generado automáticamente en la creación inicial del proyecto Xtext.

Para editar la etiqueta de un elemento basta simplemente con añadir las siguientes líneas en el fichero antes mencionado:

```
1 class APDSLLabelProvider extends org.eclipse.xtext.ui.label
      .DefaultEObjectLabelProvider {
2     @Inject
3     new(org.eclipse.emf.edit.ui.provider.
          AdapterFactoryLabelProvider delegate) {
```

```
4           super(delegate);
5       }
6       def text(Site s) {
7           "Site " + s.name
8       }
9       def text(Role r) {
10          "Role " + r.name
11      }
12      ...
13  }
```

En este caso de ejemplo, se han definido dos etiquetas, una para los elementos de tipo “Site” y otra para los elementos de tipo “Role”. De esta forma, cuando se muestre un elemento de tipo “Site” con nombre “Web” en la vista *outline* aparecerá como “Site Web”.

4.2.3. Validador

Gran parte de las validaciones vienen dadas en la propia gramática y se acoplan de forma directa y automática al editor, tal y como se ha explicado en el apartado anterior. Por ejemplo, cuando se especifica *(elements += Element)+* al colocar el símbolo + se validará que debe haber uno o más elementos del tipo “Element” en el ámbito donde se haya situado.

Sin embargo, otras validaciones no pueden ser cubiertas a través de la gramática y deben ser definidas en el módulo “Validator” que aporta Xtext con el fin de que se integren de forma automática al mismo editor. Ocurre lo mismo que cuando definimos un diagrama de entidad relación o clases donde podemos definir ciertas restricciones pero otras deben ser definidas de forma separada. Generalmente para este tipo último de restricciones se utiliza el lenguaje de especificación OCL [22].

La clave es poder utilizar este lenguaje OCL para definir las validaciones

en lugar de utilizar Xtend con el fin de facilitar la comprensión y poder crear una relación directamente textual entre las especificaciones de las mismas y su desarrollo.

Para cumplir con esto solamente se necesita sobrescribir el método *register* en la clase APDSLValidator de forma que se cargue un fichero OCL que defina las validaciones, todo esto mediante el uso del paquete *Complete OCL* obtenido a través de la instalación del plugin *OCL Examples and Editors* que se encuentra en el sitio de instalación de software oficial de Eclipse.

```
1  override register(EValidatorRegistrar registrar) {
2      super.register(registrar);
3      val ePackage = APDSLPackage.eINSTANCE;
4      val oclURI = URI.createPlatformResourceURI(
5          "/es.uca.apdsl.xtext/ocl/APDSL.ocl",
6          true
7      );
8      registrar.register(
9          ePackage,
10         new CompleteOCLEObjectValidator(ePackage, oclURI)
11     );
12 }
```

En este fichero OCL primeramente se importa el metamodelo a partir del cual queremos declarar sus validaciones o restricciones, en este caso, el generado por Xtext para AdminDSL denominado "APDSL.ecore".

```
1  import ecore : '../model/generated/APDSL.ecore'
```

A continuación se van añadiendo las validaciones mediante invariantes con la nomenclatura propia de OCL y con algunas características añadidas, como la definición de mensajes personalizados, utilizando las clases del metamodelo importado.

Ejemplo de validación con *Complete OCL*:

```
1 context Application
2     inv NoMoreThanOneSite('There must be only one "Site"
         statement'):
3         Site.allInstances()->size() = 1
```

En el apéndice B se muestran todas las validaciones OCL que han sido añadidos en este fichero.

4.3. Diseño y desarrollo del entorno para Eclipse

En la figura 4.2 se muestra el diseño (o arquitectura) del entorno desarrollado para Eclipse. Principalmente, se compone de dos plugins: el que ofrece Xtext ligado al lenguaje y el que recoge Epsilon para la generación de código Django. Éste último viene acompañado de un menú contextual que permite al usuario realizar la generación de código a partir de un código AdminDSL pinchando en una opción particular.

A partir de Xtext y Epsilon se han obtenido los plugins y las fuentes necesarias. Sin embargo, es importante reunir estos elementos para poder elaborar extensiones que permitan integrar el trabajo realizado dentro de un entorno Eclipse y disponer de estas funcionalidades de forma directa y sencilla incluso para un usuario poco habitual de esta herramienta, de forma que se abstraiga totalmente de las tecnologías que se han utilizado para el desarrollo de este trabajo.

Xtext por sí solo ya genera las extensiones necesaria para hacer uso del editor con las validaciones y otras características que se hayan definido, todas reunidas en un plugin (el propio proyecto Xtext), por lo que ha permitido ahorrar tiempo y esfuerzo en este sentido. Sin embargo, para Epsilon, se ha tenido que preparar una extensión que permitiese generar el código mediante un simple botón u opción en un menú contextual.

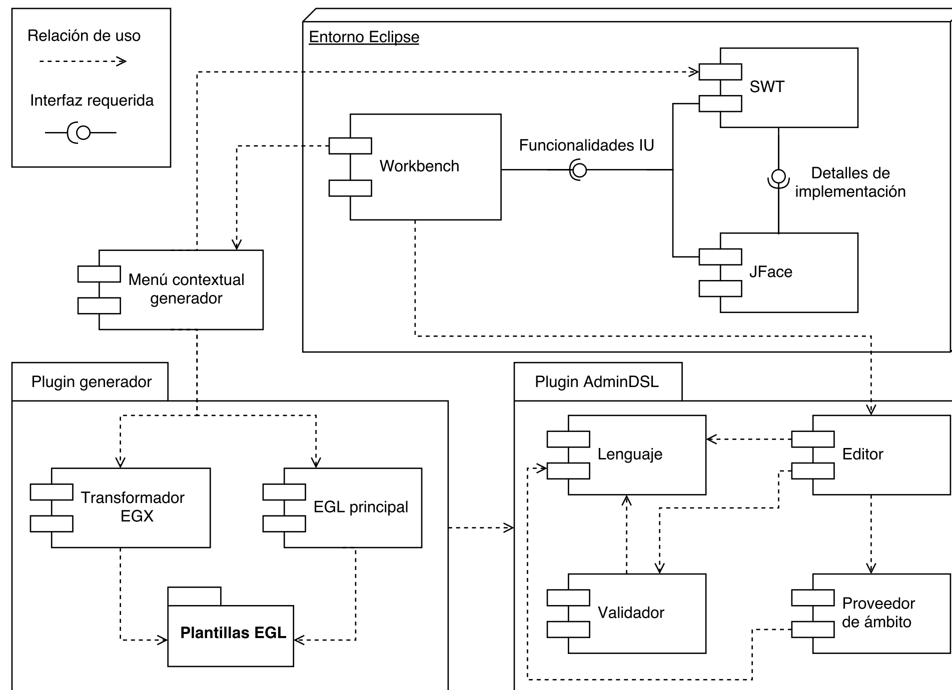


Figura 4.2: Diagrama de componentes del entorno desarrollado en Eclipse.

Por último, todo esto debía de empaquetarse de forma conjunta, con el fin de facilitar su instalación en cualquier entorno eclipse.

4.3.1. Extensión de menú contextual

Esta extensión hace referencia a la opción en el menú contextual de los ficheros que permite generar el código Django a través de un documento con extensión ".apdsl", que es la que se corresponde con el lenguaje AdminDSL. De forma que con un simple click derecho sobre este fichero en la vista del explorador de paquetes y seleccionando la opción "Generate Django Project" se genere de forma automática la aplicación web pertinente.

Para desarrollar esta extensión se ha creado un nuevo proyecto del tipo *Plug-in* que aporta Eclipse.

En esta extensión en desarrollo se definen las dependencias con Epsilon (que aporta el mecanismo de transformación e interpretación de las

plantillas), con la característica previamente desarrollada (que aporta las plantillas EGL y el transformador EGX) y el plugin de Xtext (que aporta el metamodelo del lenguaje). Posteriormente se define dónde y cómo debe aparecer la opción para generar el proyecto y cómo debe ejecutarse.

4.3.2. Paquete de instalación/actualización

Como ya se ha mencionado, la idea es tener todo reunido en un paquete de instalación de forma que un usuario que quiera utilizar el entorno que se ha desarrollado con este trabajo, sólo tenga que instalar un elemento donde se recojan todos los plugins o características necesarias. Esto se cubre con el desarrollo de un sitio de actualización o «update site», un paquete preparado para ser instalado (o para actualizar una versión previa instalada) desde el menú de instalación de nuevo software en Eclipse.

Primeramente, se ha reorganizado los plugins asociados al generador de Django y los plugins asociados al lenguaje AdminDSL en dos features respectivas de Eclipse que separa el editor del generador.

Una feature es una agrupación de plugins que aportan una determinada característica al entorno, y que es la unidad mínima de instalación para usuarios finales.

La idea es separar las implementaciones asociadas al generador de código para Django del resto de implementaciones. Esto facilita el uso de los plugins del lenguaje AdminDSL para elaborar otro generador o cualquier otro software diferente independiente del generador de Django sin tener fuentes inservibles de por medio. De modo que se tendría una feature donde se reúnen los plugins del editor de código AdminDSL, otra feature donde se encuentra todo el código fuente asociado al generador de código para Django en Epsilon y una feature que abarca la anterior junto con la extensión del menú contextual para la generación de código Django desde la interfaz de usuario.

Para terminar, el desarrollo del sitio de actualización se ha realizado señalando a través de un fichero XML las features a exportar, en este caso las dos principales que se han mencionado previamente, y configurándolo de forma que se generen los ficheros fuentes junto a las extensiones, para que las personas que se descarguen e instalen el sitio tengan acceso al código fuente del mismo. Se utiliza *Ant Build* ofrecido por Eclipse para ejecutar la elaboración del sitio de actualización a través de las opciones definidas en el fichero XML.

4.4. Herramientas utilizadas

En este apartado se describen las herramientas que se han utilizado para el desarrollo de este trabajo.

4.4.1. Eclipse con Xtext y Epsilon

Eclipse [8] se presenta como una comunidad para individuos y organizaciones que desean colaborar en el desarollo de software de código abierto. Sus proyectos están enfocados en la construcción de una plataforma de desarrollo abierta compuesta de entornos extensibles, herramientas y tareas para la construcción, el despliegue y el control de software durante todo el ciclo de vida.

Dicho esto, Eclipse (como programa) es una herramienta multiplataforma de fácil extensión compuesta de múltiples entornos y herramientas que nos permite el desarrollo del software con las tecnologías oportunas siempre y cuando existan extensiones que las añadan.

En este caso las tecnologías, ya mencionadas en más de una ocasión, que nos han servido para el desarrollo de este trabajo y que han sido instaladas sobre Eclipse han sido Xtext [7] y Epsilon [9]. Estas dos tecnologías se integran en Eclipse extendiendo sus funcionalidades con el fin de permitirnos

definir un DSL y un generador asociado de código respectivamente.

4.4.2. Control de versiones

Para el control de versiones se utiliza Git. Git es un sistema de control de versiones distribuido gratuito y de código abierto diseñado para manejar desde pequeños a grandes proyectos de forma rápida y eficiente.

La característica principal de Git, que lo aleja del resto de sistemas de control de versiones, es su modelo de ramas.

Git permite tener múltiples ramas locales que pueden ser totalmente independientes unas de otras. La creación, fusión y eliminación de aquellas lineas de desarrollo se realiza en pocos segundos.

Dado que casi todas las operaciones son ejecutadas de forma local, Git ofrece una gran ventaja en cuanto a velocidad frente a sistemas centralizados que tienen que comunicarse constantemente con un servidor.

Además, al ser distribuido te permite trabajar con un repositorio clonado en lugar de una copia de trabajo, lo cual implica que cada participante actúa como un servidor y por lo tanto no existe uno sólo central como en los sistemas de control de versiones centralizados.

CAPÍTULO 5

GENERADOR DE APLICACIONES DJANGO

5.1. Desarrollo del generador de código

Para la generación de código con Epsilon se crean dos ficheros principales para la transformación del código escrito en AdminDSL, uno para la salida en ficheros ejecutables, que requieren permisos de ejecución, y otros para el resto de ficheros a generar. Estos dos ficheros definen qué otros ficheros se van a generar a partir del código de AdminDSL haciendo referencia a una plantilla asociada y permitiendo enviar variables con ciertos valores útiles, además de dónde se van a crear y en qué orden (aunque esto último no es de utilidad para este tipo de transformaciones).

El generador de código parte de un metamodelo que se ha obtenido a través de Xtext donde viene definida toda la estructura del lenguaje. De forma que, a partir del mismo, se puede trabajar en una dinámica de orientación a objetos con los elementos que se han definido en el código escrito con AdminDSL utilizando el lenguaje EOL.

Véase la figura 5.1 para comprender mejor la estructura de los componentes del generador y su relación con el metamodelo provisto por Xtext.

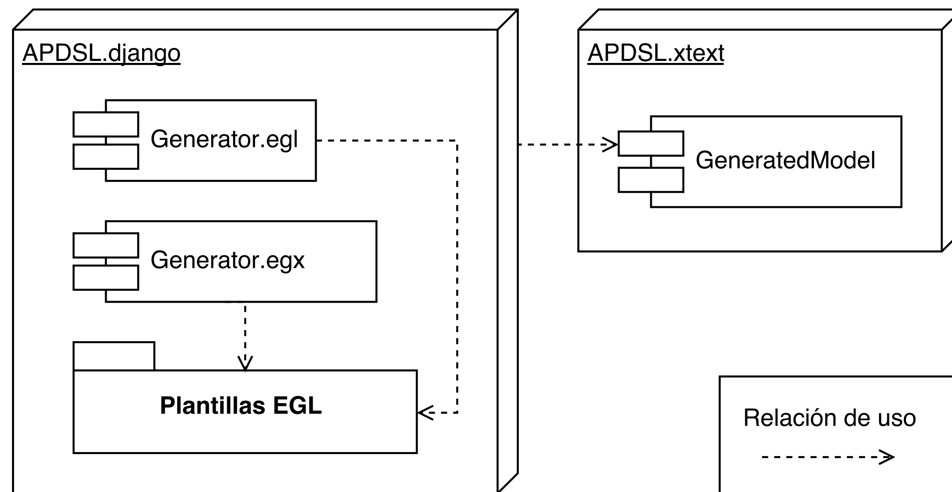


Figura 5.1: Diagrama de componentes del entorno Epsilon para el desarrollo del generador de código

De esta forma, con Epsilon, si queremos obtener todas las sentencias “Process” que hayan sido definidas en el código AdminDSL sólo se tiene que ejecutar la siguiente sentencia:

```
var procesos = Process.all
```

Teniendo en cuenta cómo se han definido los atributos en la gramática con Xtext, se puede navegar a través del árbol de sintaxis abstracta de una forma orientada a objetos.

```
1 var procesos = Process.all
2 for (proceso in procesos) {
3     var elements = proceso.elements;
4     ...
5 }
```

En este ejemplo estamos obteniendo todos los elementos de cada proceso y serviría para trabajar con ellos de algún modo.

Se pueden definir sentencias condicionales con *if, else*, *switch*, o de bucles

con *for* o *while* como en la mayoría de lenguajes de programación tradicionales.

La forma de desarrollar este generador se basa principalmente en que por cada fichero a generar existe una plantilla EGL asociada a partir de la cual se genera. Estas plantillas son "arrancadas" desde los ficheros principales. Dicho esto, la estructura de plantillas EGL queda prácticamente igual a la estructura de la aplicación final a generar.

5.1.1. Transformador EGX

Volviendo a los dos ficheros principales, tenemos un fichero con extensión EGX denominado "generator.egx" que sirve para generar ficheros ejecutables. Para ello se utiliza la siguiente nomenclatura:

```
1  rule manage
2      transform s : Application {
3          parameters {
4              var params : new Map;
5              params.put("projectName", Site.all.first.name);
6              return params;
7          }
8          template : "project/manage.py.egl"
9          target : Site.all.first.name + "/manage.py"
10         post { generated.executable = true; }
11     }
```

- En *parameters* definimos qué variables queremos enviarle a la plantilla EGL, en este caso, nos interesa el nombre de la aplicación dada mediante la sentencia de tipo "Site" que se sabe que debe haber sólo una.

- En *template* seleccionamos la ruta relativa del fichero de la plantilla

(incluyendo el nombre de la misma) a partir de la cuál se creará el fichero generado.

- En *target* seleccionamos la ruta relativa (incluyendo el nombre del fichero) de la salida del generador.

- Por último, con *post* especificamos que el fichero generado tenga permisos de ejecución.

5.1.2. Plantilla principal EGL

Para establecer la generación del resto de ficheros no ejecutables se utiliza el fichero "generator.egl".

Este fichero se puede dividir en dos partes: la primera que se encarga de generar los ficheros independientes de los procesos administrativos definidos utilizando AdminDSL y la segunda que se encarga de generar los ficheros según los procesos administrativos definidos.

La primera parte cuenta con la siguiente estructura (expresada en pseudocódigo para facilitar su comprensión):

```
1 Para Cada rutaPlantilla, rutaSalida En rutas Hacer
2     cargar(rutaPlantilla)
3     enviar(variabbles, rutaPlantilla)
4     generar(rutaSalida)
```

Para simplificar la lógica de este fichero generador, los nombres de las plantillas EGL son iguales que los nombres de los ficheros a generar añadiendo simplemente la extensión ".egl" al final.

Así el proceso de generación queda bastante sencillo:

- Se almacenan todas las rutas y nombres de ficheros que queremos generar en una colección.

- Se obtiene tanto la ruta de salida como la ruta de la plantilla correspondiente al fichero a generar por cada elemento contenido en esta colección.

- Se carga dicha plantilla a través de la ruta obtenida.

- Se envían las variables que sean necesarias a la plantilla.

- Se genera el fichero de salida con la ruta de salida.

Con la segunda parte se sigue la misma estructura pero teniendo en cuenta que las rutas de salidas serán dependientes del nombre que se le haya dado a cada proceso administrativo definido con AdminDSL.

Ahora sólo falta ver cómo funcionan las plantillas que son las que realmente realizan la traducción del código perteneciente a AdminDSL al código Python para Django.

5.1.3. Plantillas EGL

Es importante tener claro que en las plantillas hay dos formas diferentes de trabajar que pueden ser utilizadas de manera simultánea: una en la que se utiliza el lenguaje EOL y otra en la que se utiliza texto literal que irá directamente a la salida. Las partes que utilizan EOL se encuentran entre los delimitadores *[% ... %]*. Así, todo texto escrito en la plantilla fuera de estos delimitadores serán escritos en el fichero generado tal cual.

Dentro de estos delimitadores se tiene la posibilidad de utilizar las variables que hayamos enviado mediante *populate*. Además se puede añadir texto de salida a partir del valor de las variables con las que se trabajan en EOL colocando el símbolo '=' tras el primer delimitador, es decir, de la siguiente forma *[%= ... %]*, así como utilizar estructuras de control como se había visto en la plantilla principal *generator.egl*. Por último, se permiten utilizar métodos que generalmente son definidos en una plantilla aparte

no asociada a ninguna generación y que puede ser importada por cualquier otra.

Un ejemplo básico del uso de estas plantillas se puede ver en la generación de “models.py” para un proceso administrativo concreto. Por ejemplo, se muestra a continuación cómo se transformarían las entidades que se hayan definido dentro del ámbito de un proceso administrativo con AdminDSL en código Python a nivel de modelos para la aplicación Django.

```
1  [%
2  import '../util_field.egl';
3  %]
4  # -*- encoding: utf-8 -*-
5  from django.db import models
6  from base_admindsl.models import *
7  from django.utils.translation import ugettext_lazy as _
8  from smart_selects.db_fields import ChainedForeignKey
9  [%
10 var models := process.elements.select(e | e.type().name = "
       Entity");
11 for (_model in models) {%]
12 class [%=_model.name.firstToUpperCase()%](models.Model):
13     [%=printFields(_model)%]
14     [%=printRelations(_model, process.name)%]
15     //Mas codigo...
16     ...
```

Lo que se hace en este fragmento de plantilla es utilizar la variable *process* obtenida desde la plantilla principal “generator.egl” para obtener todos los elementos que son del tipo “Entity” (línea 10 del código superior). Una vez obtenido por cada uno de ellos, se crea la clase y se llaman a los métodos *printFields* y *printRelations* que imprimirán sobre el fichero los distintos campos y las relaciones definidas respectivamente.

Los métodos *printFields* y *printRelations* se encuentran en la plantilla `util_field.egl` y se definen del siguiente modo:

```
@template
operation printFields(section) {
    var fields := getFields(section);
    if (fields.size() > 0) {
        for (field in fields) {
            switch(field.type.toString()) {
                ...
                case "phone": %]
                [%=field.name%] = models.CharField(null=
                    True, max_length=50[%=printOptions(field
                    ,false)%])
                [% break;
                case "email": %]
                [%=field.name%] = models.EmailField(null=
                    True[%=printOptions(field,false)%])
                [% break;
                ...
            }
        }
    }
}
```

Este método, al llevar el decorador *@template* se tratará como parte de la plantilla que lo llama, es decir, es como una forma de trasladar el código de una plantilla a un método.

Sin embargo, el método *getFields* (véase debajo), que no lleva el decorador *@template*, actúa como un método común en los lenguajes de programación tradicionales devolviendo un resultado en este caso.

```
operation getFields(eObject) {
```

```
2      return eObject.elements.select(e | e.type().name = "
           Field");
3  }
```

Así pues, siguiendo esta nomenclatura y dinámica de desarrollo, se ha desarrollado cada una de las plantillas asociadas a cada fichero generado y se han colocado para su carga respectiva en uno de los dos ficheros principales con el fin de llevar a cabo la generación de una aplicación web Django completa.

5.2. Diseño de la aplicación web Django generada

El diseño de la aplicación web Django generada es similar a la arquitectura de cualquier proyecto Django. Se utiliza un entorno virtual que contiene todas las dependencias necesarias para el funcionamiento de la aplicación Django. En la figura 5.2 de la página 147 se puede ver un diagrama representativo de esta arquitectura.

Se trata de un modelo vista-controlador, donde por cada proceso administrativo se define una app.

Por cada app se recogen los siguientes componentes, los cuales pueden estar conectados tanto entre ellos como con otros componentes de otras apps:

- *Models*: Donde se definen los modelos de datos que contiene la aplicación.
- *Views*: Aquí se definen las vistas de la aplicación y es donde radica la mayor parte de la lógica de negocio de la aplicación.
- *Forms*: Se especifican los formularios que pueden ser cargados en las vistas.
- *Urls*: Para definir las rutas que tendrá la aplicación.

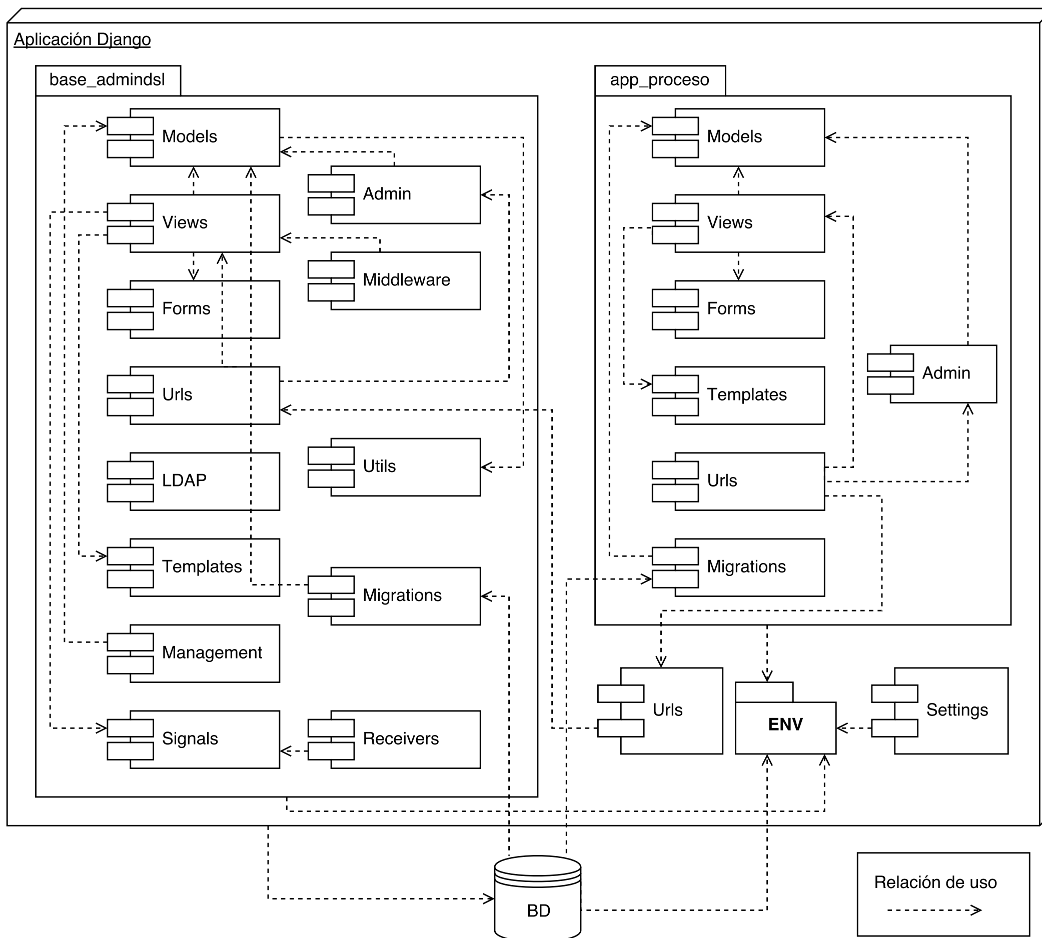


Figura 5.2: Diagrama de componentes de la aplicación Django generada.

- *Admin*: Donde se decide qué elementos (generalmente modelos) pueden ser gestionados en el panel de administración.
- *Templates*: Aquí se definen las plantillas para la visualización de las vistas de Django.
- *Migrations*: Para manejar la carga de datos en la base de datos. En las migraciones se definen las tablas, los elementos iniciales, entre otras cosas.

Además de las apps de cada proceso administrativo, existe una app básica y principal común para todos los proyectos Django generados a partir de AdminDSL denominada “base_admindsl”.

En esta app, además de los componentes anteriores, se añaden los siguientes:

- *Signals*: Para definir señales que serán enviadas con el fin de informar de que un evento ha ocurrido y hacer algo en consecuencia.
- *Receivers*: Para recoger señales que han sido enviada y realizar una acción determinada.
- *Middleware*: Aquí se definen lógicas de negocio comunes a diferentes vistas.
- *Utils*: Recoge métodos útiles para la implementación de la aplicación.
- *LDAP*: Aquí se puede configurar el sistema de autenticación LDAP.
- *Management*: Para definir órdenes a ejecutar a través de un terminal o consola de línea de comandos.

5.3. Generación de código: esqueleto base

En toda generación de código a partir del AdminDSL se crea un esqueleto base para Django. Este código base se encuentra localizado en la *app*

nombrada "base_admindsl" y en un directorio que recibe el mismo nombre que el escogido para la aplicación con la sentencia `site`. A continuación, se especifican y describen los elementos que se generan en este esqueleto.

5.3.1. Fichero de configuración "settings.py"

Este fichero se genera dentro de un subdirectorio cuyo nombre es el mismo que el del directorio raíz del proyecto.

La configuración por defecto contiene, entre otras opciones, lo siguiente:

- Base de datos SQLite3. Esta configuración en concreto viene situada en una región protegida, lo que quiere decir que cualquier modificación se mantendrá en regeneraciones de código posteriores.
- Identificación de usuarios por LDAP.
- Modo de depuración activado tanto a nivel general como a nivel de plantillas.

5.3.2. Fichero de configuración "urls.py"

Este fichero se encuentra en el mismo directorio que el fichero "settings.py" y define las rutas principales (primer nivel) disponibles en la aplicación django.

Este fichero contiene las siguientes rutas:

- Ruta raíz: dirige a la pantalla de autenticación de usuario.

```
url(r'^$', login_user)
```

- Admin site: dirige al panel de administración que ofrece Django por defecto.

```
url(r'^admin/', include(admin.site.urls))
```

- Login: dirige a la pantalla de autenticación de usuario.

```
url(r'^admin/', include(admin.site.urls))
```

- Logout: dirige a la pantalla de desconexión de usuario.

```
url(r'^logout/', logout, kwargs={'template_name': '
    base_admindsl/logout.html'}, name='logout')
```

- Processes: dirige al primer nivel de rutas definidas en “base_admindsl/-urls.py”

```
url(r'^processes/', include('base_admindsl.urls'))
```

- Ruta de proceso: se define una ruta por cada proceso definido en el DSL y dirige al primer nivel de rutas definidas en “app_proceso/urls.py”

```
url(r'^app_proceso/', include('app_proceso.urls'))
```

5.3.3. Modelos autogenerados y predefinidos

Estos modelos son creados independientemente del código que se haya definido en el lenguaje AdminDSL y se localizan en “base_admindsl/models.py”. Véase figura 5.3 para la representación en forma de diagrama de clases. Los tipos de atributos que se muestran son propios del marco de trabajo Django y no basados en la nomenclatura que aporta UML.

Modelo `Transition`

Define una transición entre un estado y otro. De esta forma, posee los siguientes atributos:

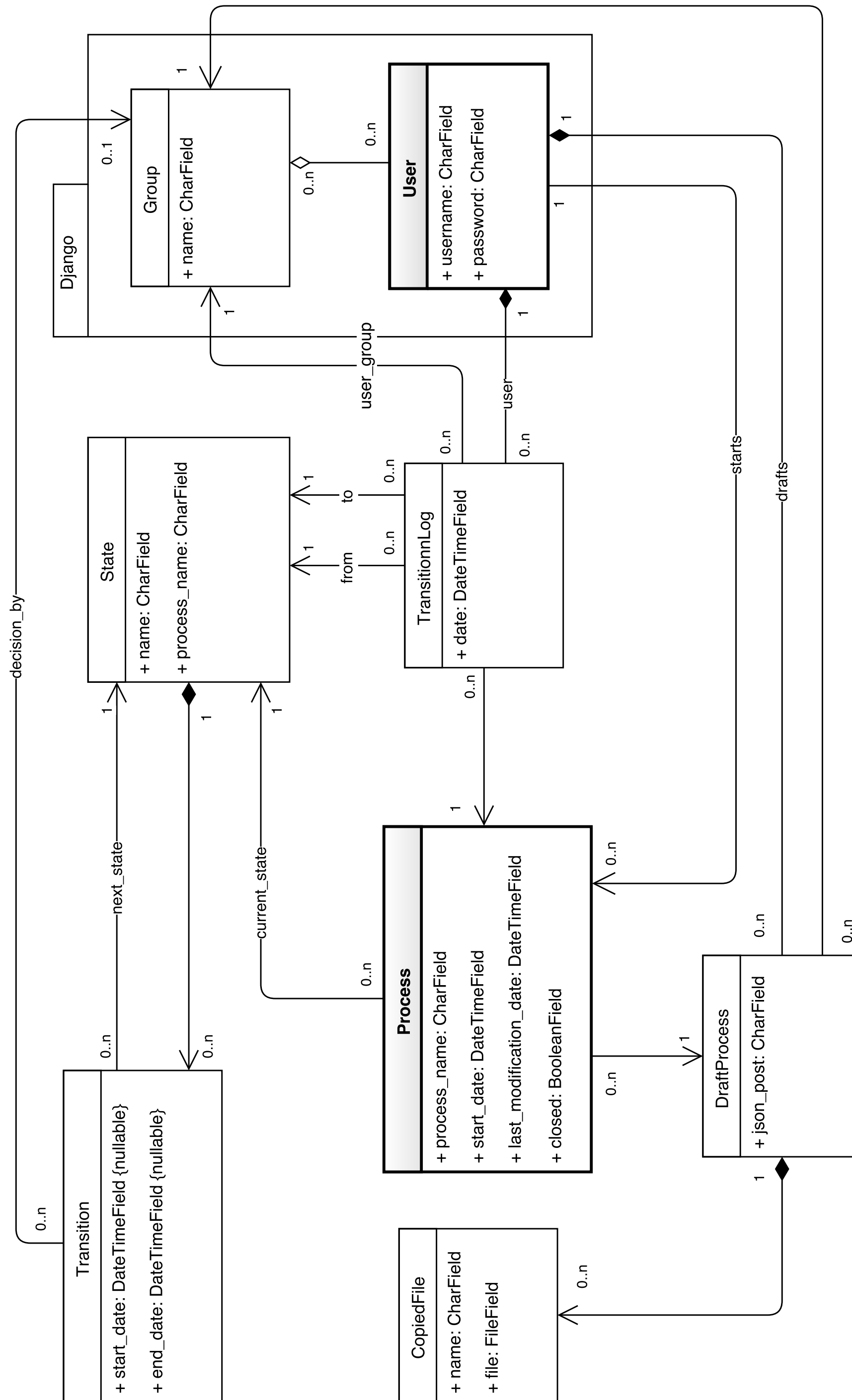


Figura 5.3: Diagrama de clases de los modelos autogenerados y predefinidos.

- `decision_by`: contiene el grupo de usuario de la clase `auth.Group` autorizado para realizar una transición. Este atributo puede ser nulo.
- `start_date`: contiene la fecha a partir de la cual se puede realizar la transición. Este atributo puede ser nulo.
- `end_date`: contiene la fecha a partir de la cual ya no se puede realizar la transición. Este atributo puede ser nulo.
- `state`: contiene el estado de la clase `State` desde el que parte la transición.
- `next_state`: contiene el estado de la clase `State` al que se llega una vez realizada la transición.

Nombraremos como transiciones automáticas a aquellas transiciones cuyo atributo `decision_by` es nulo, es decir, aquellas que sólo dependen del tiempo.

Esta clase define el siguiente método:

- `can_pass(self, user=None, group=None)`:
  - Si no se especifican los parámetros `user` ni `group`, comprueba si una transición puede realizarse de forma automática. Devuelve `True` o `False`.
  - Si se especifican los parámetros `user` y `group`, comprueba si ese usuario en ese grupo puede realizar la transición. Devuelve `True` o `False`.

Modelo `State`

Define un estado. Este estado en Django sólo posee dos atributos: `name`, que representa su propio nombre, y `process_name`, que representa el nombre del proceso donde puede participar.

En esta clase se definen los siguientes métodos:

- `is_initial(self)`: devuelve `True` si se trata del estado "initial" o `False` si ocurre lo contrario.

- `has_transition(self)`: devuelve `True` si existe alguna transición que parta de este estado o `False` si ocurre lo contrario.

- `make_a_transition(self, user=None, group=None, state=None)`: realiza una transición devolviendo el estado siguiente al que llega o `None` si no se puede realizar ninguna transición.

  - Si se especifican los parámetros `user` de la clase `auth.User`, `group` de la clase `auth.Group` y `state` como nombre del estado, devuelve el estado cuyo nombre coincide con el parámetro `state` si ese usuario en ese grupo puede realizar una transición del estado actual (desde el que se llama al método) al estado cuyo nombre coincide con el parámetro `state` o `None` si no es posible realizar la transición.
  - Si se especifican solamente los parámetros `user` de la clase `auth.User` y `group` de la clase `auth.User`, devuelve el primer estado al que se puede cambiar por decisión de ese usuario en ese grupo o `None` si no es posible realizar ninguna transición.
  - Si no se especifica ningún parámetro, devuelve el primer estado al que se puede cambiar de forma automática o `None` si no se puede realizar ninguna transición automática.

Modelo `TransitionLog`

Sirve para almacenar un historial de las diferentes transiciones que han sido realizadas por decisión de un usuario. Este historial es importante para gestionar los permisos de usuarios que han participado en ciertas transiciones.

Posee los siguientes atributos:

- `user`: el usuario que ha realizado la transición de la clase `auth.User`.
- `group`: el grupo al que pertenecía el usuario en esa transición de la clase `auth.Group`.
- `from_state`: el estado desde el que parte la transición de la clase `State`.
- `to_state`: el estado al que se llegó al realizar la transición de la clase `State`.
- `process`: el proceso en el que se realizó el cambio de estado de la clase `NombreProceso.NombreProcesoProcess`.

Modelo `Process`

Modelo abstracto que servirá como modelo base para los distintos procesos que se vayan a generar a partir del código del AdminDSL.

Posee 5 atributos:

- `init_user`: contiene el usuario creador del proceso.
- `current_state`: contiene el estado actual en el que se encuentra el proceso.
- `start_date`: contiene la fecha de creación del proceso.
- `last_modification_date`: contiene la fecha de última modificación del proceso.
- `closed`: señala si un proceso no puede cambiar de estado, es decir es un proceso en un estado cerrado.

Esta clase define los siguientes métodos:

- `add_to_transitionlog(self, user, group, from_state, to_state)`: crea una instancia nueva de la clase `TransitionLog` registrando que el usuario `user` en el grupo `group` a realizado una transición del estado `from_state` al estado `to_state`.

- `has_perm(self, user, group, state, permission_name)`: comprueba si el usuario `user` en el grupo `group` posee el permiso `permission_name` para el proceso en el estado `state`.

- `visible_by(self, user, group)`: comprueba si el proceso es visible por el usuario `user` en el grupo `group`, es decir, que puede ver o editar al menos alguna de sus secciones, grupos o campos.

- `editable_by(self, user, group)`: comprueba si el proceso es editable por el usuario `user` en el grupo `group`, es decir, que puede editar al menos alguna de sus secciones, grupos o campos.

- `deletable_by(self, user, group)`: comprueba si el proceso puede ser eliminado por el usuario `user` en el grupo `group`.

Los métodos no mencionados son métodos que sirven como apoyo para los métodos principales sí mencionados.

Modelo `MaxProcessInstance`

Permite definir el número máximo de instancias que un usuario perteneciente a un grupo puede crear sobre un mismo proceso.

Posee tres atributos:

- `group`: el grupo de usuario para el cual se limita el número de instancias.

- `process_name`: el nombre del proceso sobre el cual se limita el número de instancias.

- `max`: el valor del número máximo de instancias que se pueden crear.

Modelos **`BasePermSection`** y **`BasePermGroup`**

Es una clase abstracta que sirve como base para la clases que definen una sección o un grupo respectivamente dentro de un proceso. Esta clase permite gestionar los permisos de visibilidad o edición de los campos que contienen las secciones o grupos declarados utilizando los siguientes métodos:

- `is_editable(self, field, state, process)`: comprueba si el campo de nombre `field` es editable en el proceso `process` en el estado `state` por cualquier usuario independientemente del grupo al que pertenezcan. Útil para comprobar si un campo que posee campos obligatorios vacío es editable o no y evitar transiciones automáticas indeseables.

- `can_view(self, process, user, group, state, field)`: comprueba si el usuario `user` en el grupo `group` puede ver (de forma exclusiva, no editar) el campo de nombre `field` en el proceso `process` en el estado `state`.

- `can_edit(self, process, user, group, state, field)`: comprueba si el usuario `user` en el grupo `group` puede editar el campo de nombre `field` en el proceso `process` en el estado `state`.

Modelo **`DraftProcess`**

Modelo utilizado para almacenar datos no enviados ni registrados en procesos activos. Sirve como borrador de procesos para no tener que obligar al usuario a completar un formulario sin tener la opción de guardarlo y continuarlo en otro momento.

Contiene los siguientes atributos:

- `user`: el usuario al que pertenecen el borrador (el que ha guardado el formulario).
- `group`: el grupo al que pertenecen el usuario dueño del borrador.
- `process`: el proceso para el cual se está guardando la copia de los datos introducidos.
- `json_post`: almacena los datos del formulario serializados.

Modelo `CopiedFile`

Modelo que sirve para registrar los ficheros subidos por un usuario cuando se crea un borrador. Este modelo es necesario debido a que los ficheros en Django no son serializables.

Posee los siguientes atributos:

- `name`: nombre del fichero a guardar.
- `file`: fichero a guardar.
- `draft`: borrador asociado al fichero.

5.3.4. Clases para el manejo y control de formularios

Estas clases se encuentran en “base_admindsl/forms.py” y se describen a continuación. Véase figura 5.4 para la representación en forma de diagrama de clases.

Clase `UserFormSet`

Esta clase sustituye a la clase `BaseModelFormSet` que provee `django.forms` y que utiliza por defecto el método factoría `modelformset_factory` también dado por `django.forms`. El tipo `FormSet` en Django ofrece

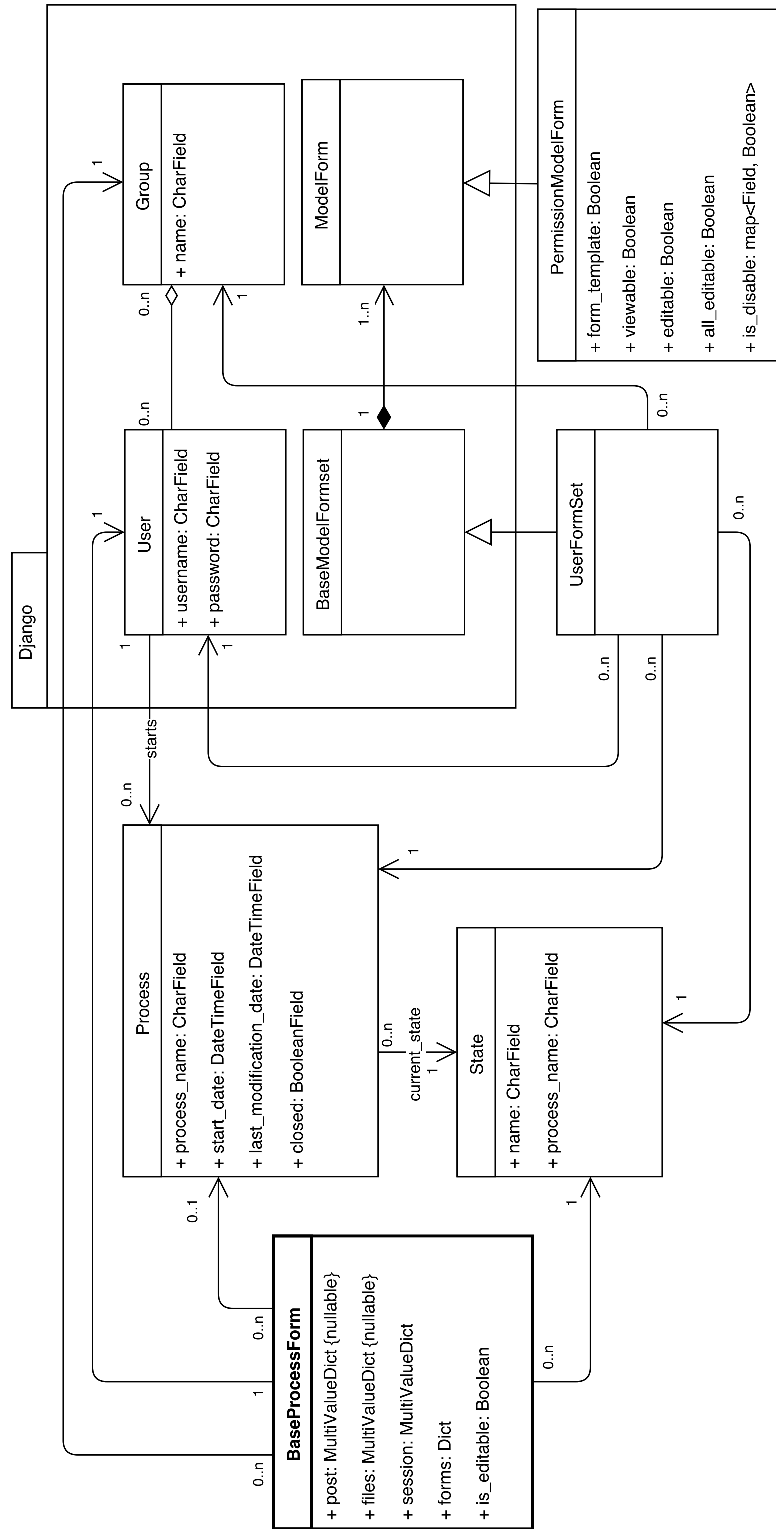


Figura 5.4: Diagrama de clases asociadas al manejo y control de formularios.

soporte para la creación de formularios con un número variable de elementos. Con `UserFormSet` se permiten declarar argumentos o parámetros de entrada adicionales en la creación de estos formularios. Estos argumentos son:

- `process`: el proceso para el cual se está creando el formulario de su grupo o sección.
- `user`: el usuario que va a operar con dicho formulario.
- `group`: el grupo al que pertenece el usuario que trabaja con el formulario.
- `state`: el estado en el que se trabaja con el formulario.

Estos argumentos adicionales son utilizados para la gestión de permisos de visibilidad o edición de los distintos campos del formulario.

Clase `PermissionModelForm`

Clase abstracta que hereda de la clase `ModelForm` que provee `django.forms` y que sirve como base para las clases que definen los distintos formularios.

Añade la lógica de gestión de permisos, deshabilitando los campos o mostrándolos de sólo lectura según corresponda, cuando se crea un formulario.

Clase `BaseProcessForm`

Clase que encapsula toda la lógica de gestión y control de formularios para las vistas en Django. Existen 4 situaciones diferentes en las que se tiene que crear una instancia de esta clase y de las cuales dependerá los parámetros de entrada en su construcción:

- **En la petición *GET* para crear un nuevo proceso**: *processForm = ProcessFormClass(user, group, initial_state, request.session)*. Se deben de pasar como parámetros de entrada el usuario de la clase *auth.User* que va a crear el proceso, el grupo de la clase *auth.Group* al que pertenece dicho usuario cuando lo está creando, el estado "initial" asociado al proceso que se va a crear y *request.session* que se obtiene directamente como parámetro de la vista actual.

  En este punto se presenta el formulario vacío del nuevo proceso a crear.

- **En la petición *POST* para crear un nuevo proceso**: *processForm = ProcessFormClass(request.user, group, initial_state, request.session, request.POST, request.FILES)*. Se deben de pasar los mismos parámetros que la petición *GET* añadiendo los parámetros *request.POST* y *request.FILES* que contienen toda la información enviada a través de la petición *POST*.

  En esta parte se procesarían los datos que ha introducido y enviado el usuario en el formulario de creación de un nuevo proceso.

- **En la petición *Get* para editar un proceso**: *processForm = ProcessFormClass(request.user, group, current_state, request.session, process= process, draft_process=draftProcess)*. Mismos parámetros de entrada que en la petición *GET* para crear un nuevo proceso pero añadiendo el proceso que se quiere editar y el borrador (*draft_process*) en caso de que lo tenga.

  Aquí se presenta el formulario de edición del proceso con los campos inicializados según el contenido previo del mismo.

- **En la petición *POST* para editar un proceso**: *processForm = ProcessFormClass(request.user, group, current_state, request.session, re-*

quest.POST, request.FILES, process, draftProcess). Mismos parámetros que en la petición *GET* pero añadiendo el request.POST y el request.FILES.

En este punto se procesarían los datos que han sido introducidos y enviados por el usuario en el formulario de edición.

Los métodos que posee esta clase son dependientes de la inicialización de los atributos durante la creación de la instancia y son los siguientes:

- *prepare_get_post(self)*: se encarga de crear los diferentes formularios por cada sección y grupo existente en el proceso que se quiere crear o editar. Además, asigna los valores de visibilidad para los formularios de grupos y secciones a nivel de formulario completo, es decir, un grupo o sección es visible si tiene al menos un campo visible, o en el caso de la sección, un campo o un grupo visible.

- *is_valid(self):* comprueba la validez de todos los formularios editables del proceso. Devuelve *True* si todos los valores de los campos editables de todos los formularios son válidos y *False* en caso de que uno de estos valores no sea válido.

- *save(self)*: guarda todos los datos de los diferentes formularios del proceso en la base de datos como instancias de los modelos correspondientes.

- *partial_save(self):* guarda en base de datos un borrador de los datos de los diferentes formularios del proceso.

- *prepare_context(self, context=)*: devuelve un *dict* con todo el contenido necesario para el manejo de los formularios u otra información adicional en las plantillas. Puede partir de un *dict* existente si se coloca como parámetro de entrada.

5.3.5. Métodos de vistas

Los métodos de vistas autogenerados se crean en el módulo de vistas situado en "base_admindsl/views.py". Estas vistas son generalizadas, es decir, válidas para cualquier proceso administrativo siguiendo la misma idea de las vistas genéricas de Django.

La idea se basa en definir a través del controlador de rutas qué elementos se van a presentar o manejar en estas vistas, dicho de otro modo, facilitar aquellos elementos que cambian respecto a una vista y otra que poseen el mismo conjunto de reglas de negocio. Con esto se evita en gran medida la duplicidad de código.

Por ejemplo, procesar un formulario para procesos administrativos no varía (en cuanto a reglas de negocio) entre un tipo de proceso y otro, exceptuando la clase correspondiente al proceso que debe usarse. Así, ofreciendo desde el controlador de rutas un nombre de proceso para la vista en cuestión (con el fin de que pueda obtener su clase) se puede desarrollar un código singular, generalizado y válido para todos los tipos de proceso, en lugar de tener una vista para cada uno de ellos donde sólo cambia el nombre de una clase y todo lo demás es exactamente igual.

A continuación se presentan las vistas que se ofrecen:

Vista *login_user(request)*

Método que inicia la sesión de un usuario. En caso de que el inicio de sesión sea satisfactorio, se le asigna al usuario (si no lo tenía ya asignado) uno de los grupos de usuario al que pertenece como activo.

Esta asignación se realiza a través de la variable *request.session* y bajo la clave *username* + *'activegroup'*. De forma que si queremos obtener el grupo activo actual del usuario con nombre de usuario *usuarioprueba* debemos obtenerlo de *request.session* de la siguiente forma:

request.session[‘usuariopruebagroupactive’]

Vista ***select_role_view(request)***

Método que cambia el grupo activo de un usuario almacenado en request.session[username + ‘groupactive’].

Vista ***available_process_list_view(request)***

Vista para mostrar cuáles son los procesos disponibles, es decir, los que pueden ser iniciados/creados por el usuario conectado. Posee transiciones válidas desde su estado inicial a otro estado.

Vista ***disabled_process_list_view(request)***

Vista para mostrar cuáles son los procesos no disponibles, es decir, los que no pueden ser iniciados/creados por el usuario conectado. No posee ninguna transición válida desde su estado inicial a otro estado.

Vista ***active_process_list_view(request)***

Vista para mostrar los procesos activos visibles por el usuario conectado, es decir, los que ya han sido iniciados y no se encuentran en un estado cerrado (sin transiciones).

Vista ***closed_process_list_view(request)***

Vista para mostrar los procesos cerrados visibles por el usuario conectado, es decir, los que ya han sido iniciados y se encuentran en un estado cerrado (sin transiciones).

Vista *process_list_view(request)*

Vista que reúne todas las vistas anteriores asociadas a procesos. Muestra cuáles son los procesos disponibles y los no disponibles para el usuario conectado y muestra todas las instancias de procesos visibles por el usuario que han sido iniciados independientemente del estado en el que se encuentren.

Vista *add_process(request, process_name)*

Vista de creación de proceso. Es una vista general que permite iniciar o crear un proceso. El parámetro *process_name* identifica de qué proceso se trata.

En esta vista, primero se comprueba si el proceso puede ser creado por el usuario actual y, posteriormente, se comprueba si el usuario ha superado el límite de instancias a crear para ese proceso, en caso de que exista dicho límite. Por último, se procesan los formularios asociados con el proceso utilizando la clase *BaseProcessForm* explicada en la sección 5.3.4.

Vista *edit_process(request, process_name, id)*

Vista de edición de proceso. Es una vista general que permite editar un proceso dado por el parámetro *id*. El parámetro *process_name* identifica de qué proceso se trata al igual que con la vista de creación de procesos. Funciona de forma parecida a la vista de creación de procesos, anteriormente explicada.

Primero se comprueba si el usuario actual puede editar o ver el proceso en cuestión, del mismo modo que lo hacía la vista de creación de procesos. Por último, se procesan los formularios asociados con el proceso utilizando la clase *BaseProcessForm* explicada en la sección 5.3.4.

Vista *delete_process(request, process_name, id)*

Vista para la eliminación de una instancia de proceso obtenida a partir de los parámetro *id* y *process_name*.

Vista *list(request, process_name)*

Vista para mostrar la lista de todos los procesos de nombre *process_name* visibles por el usuario conectado.

5.3.6. Señales y receptores

Las señales y receptores en Django son útiles para añadir reglas de negocio extras cuando ocurre un cierto suceso sin tener que retocar las líneas de código de las vistas donde se maneja dicho suceso. La idea principal consiste en enviar una señal cuando el suceso ocurra para que ésta sea capturada por un método receptor que tendrá las reglas de negocios oportunas a ejecutar.

Las señales se definen en el fichero “base_admindsl/signals.py” y los receptores en el fichero “base_admindsl/receivers.py”. A continuación se describen cada uno de estos elementos.

Señal *new_process_created*

```
1  new_process_created = Signal(providing_args=["user", "group
       ", "process" ])
```

Señal que debe enviarse cuando se crea un proceso (por defecto sólo se envía en la vista de creación de procesos). Cuando se envía esta señal se deben de pasar los siguientes parámetros: el usuario que crea el proceso, el grupo al que pertenece este usuario, y el proceso que ha creado.

Su receptor asociado por defecto es el método:

new_process_created_handler.

Señal *process_edited*

```
process_edited = Signal(providing_args=["user", "group", "
    process"])
```

Señal que debe enviarse cuando se edita un proceso (por defecto sólo se envía en la vista de edición de procesos). Cuando se envía esta señal se deben de pasar los siguientes parámetros: el usuario que edita el proceso, el grupo al que pertenece este usuario y el proceso que ha editado.

Su receptor asociado por defecto es el método:

process_edited_handler.

Señal *process_edited_with_transition*

```
process_edited_with_transition = Signal(providing_args=["
    user", "group", "process", "from_state", "to_state"])
```

Señal que debe enviarse cuando se edita un proceso con un cambio de estado (por defecto sólo se envía en la vista de edición de procesos). Cuando se envía esta señal se deben de pasar los siguientes parámetros: el usuario que edita el proceso, el grupo al que pertenece este usuario, el proceso que ha editado, el estado en el que se encontraba antes de la edición y el estado al que ha cambiado tras la edición.

Su receptor asociado por defecto es el método:

proces_edited_with_with_transition_handler.

Señal *process_automatic_transition*

```
process_automatic_transition = Signal(providing_args=["
    process", "from_state", "to_state"])
```

Señal que debe enviarse cuando se realiza una transición automática (por defecto sólo se envía en la orden de actualización de estados). Cuando se envía esta señal se deben de pasar los siguientes parámetros: el proceso que ha cambiado de estado, el estado en el que se encontraba previamente y el estado al que ha cambiado de forma automática.

Su receptor asociado por defecto es el método:

process_automatic_transition_handler.

Señal *process_deleted*

```
process_deleted = Signal(providing_args=["user", "group", "
    process_id", "process_name"])
```

Señal que debe enviarse cuando se elimina una instancia de un proceso (por defecto sólo se envía en la vista de eliminación de procesos). Cuando se envía esta señal se deben de pasar los siguientes parámetros: el usuario que elimina el proceso, el grupo al que pertenece este usuario, la id del proceso que se ha eliminado y el nombre de proceso de la instancia que se ha eliminado.

Su receptor asociado por defecto es el método:

process_deleted_handler.

5.3.7. Orden *update_states*

Se define en el fichero “base_admindsl/management/commands/update_state.py”. Se puede llamar desde la consola de comandos de la siguiente forma:

```
python manage.py update_states
```

Esta orden se encarga de actualizar los estados de todas las instancias activas de procesos. Si un proceso contiene campos obligatorios editables para cierto estado y se encuentra en dicho estado, no podrá realizar ninguna transición automática, por lo tanto la orden *update_state* no tendrá efecto sobre el mismo.

El objetivo es que esta orden venga integrada en algún planificador a nivel de sistema, como por ejemplo, un trabajo cron o el planificador de Windows, con el fin de que se realice una actualización periódica de los estados en los que se encuentra cada uno de los procesos administrativos de la aplicación.

5.4. Generación de código: AdminDSL a Django

Sentencia *site*

Define el nombre del directorio raíz del proyecto Django generado.

Sentencia *options*

Define opciones propias de Django, tales como, la plantilla base que se va a utilizar o las app's que se van a importar en un proyecto Django.

```
1 options {
2     django_base_template = "app/base.html";
3     django_extra_apps    = "app_name = SVN_URL";
4 }
```

Sentencia *role*

Define un grupo de usuario en "migrations/groups.py" del app "base_admin-dsl".

- En AdminDSL:

```
role Bosses;
```

- En Django:

  “base_admindsl/migrations/groups.py”:

```
def register_groups(apps, schema):
    Group = apps.get_model('auth', 'Group')
    boss, created = Group.objects.get_or_create(
        name='Boss')
    if created:
         logger.info('Bosses Group created')
class Migration(migrations.Migration):
    operations = [migrations.RunPython(
        register_groups),]
```

Sentencia *entity*

Si se coloca en el ámbito global, define un modelo en el fichero “models.py” de la app “base_admindsl”. Si se coloca en el ámbito de un proceso, define un modelo en el fichero “models.py” de la app correspondiente al proceso.

- En AdminDSL:

```
entity Name {
    type attribute;
}
```

- En Django:

  “base_admindsl/models.py”:

```
1  class Name(models.Model):
2      attribute = models.TypeField(null=True)
```

Sentencia *relation*

Si es una relación binaria, añade un campo del tipo *ForeignKey*, *ManyToManyField* o *OneToOneField* (dependiendo del valor de la opción “type”) en el modelo que define el elemento donde se ha colocado. Si es una relación *n*-aria, se crea una nueva clase con el mismo nombre que la relación seguido de “relation” y se añade un campo por cada entidad asociada del tipo *ForeignKey*.

- En AdminDSL:

```
1  entity Entity1 {
2      relation(Entity2) entity2;
3  }
4  entity Entity2 {
5      relation(Process1, type="ManyToMany") process1;
6  }
7  process Process1 {
8      ...
9  }
10 process Process2 {
11     relation(Process1, "auth.User", type="ManyToMany")
           process2_process1_user;
12 }
```

- En Django:

  “base_admindsl/models.py”:

```
1 class Entity1(models.Model):
2     entity2 = models.ForeignKey(base_admindsl.
          Entity2, null=True)
3
4 class Entity2(models.Model):
5     process1 = models.ManyToManyField(Process1.
          Process1Process)
```

“Process2/models.py”:

```
1 class Process2Process(Process):
2     ...
3
4 class Process2_process1_user(models.Model):
5     process2 = models.ForeignKey("Process2.
          Process2Process", related_name="
          process2_process1_user", null=True)
6     process1 = models.ForeignKey("Process1.
          Process1Process", related_name="
          process2_process1_user", null=True)
7     user = models.ForeignKey("auth.User",
          related_name="process2_process1_user", null=
          True)
8     class Meta:
9         unique_together = ('process2', 'process1',
              'user', )
```

Sentencia *process*

Define una nueva app con los ficheros “models.py”, “forms.py”, “urls.py”, “views.py”, “templates/add_processname.html”, “templates/edit_processname.html”, “migrations/permissions.py” y “migrations/states.py”, cuyos con-

tenidos serán dependientes de los elementos definidos dentro de su ámbito. Se define, a su vez, un modelo y un formulario en “models.py” y “forms.py” representativo del mismo proceso.

En la figura 5.5 de la página 173 se muestra una diagrama de clases representativo de la jerarquía que se genera cuando se definen procesos, secciones dentro del mismo y grupos dentro de estas últimas.

```
1 process ProcessName {
2     ...
3 }
```

Estas sentencias generarán código en los distintos ficheros contenidos en la app del proceso.

Sentencia *relation*

Véase sentencia *relation* de la página 170.

Sentencia *entity*

Define un modelo en “models.py” dentro de la app del proceso.

- En AdminDSL:

```
1 process ProcessName {
2     entity Name {
3         type(unicode="True") attribute;
4     }
5 }
```

- En Django:

  “ProcessName/models.py”:

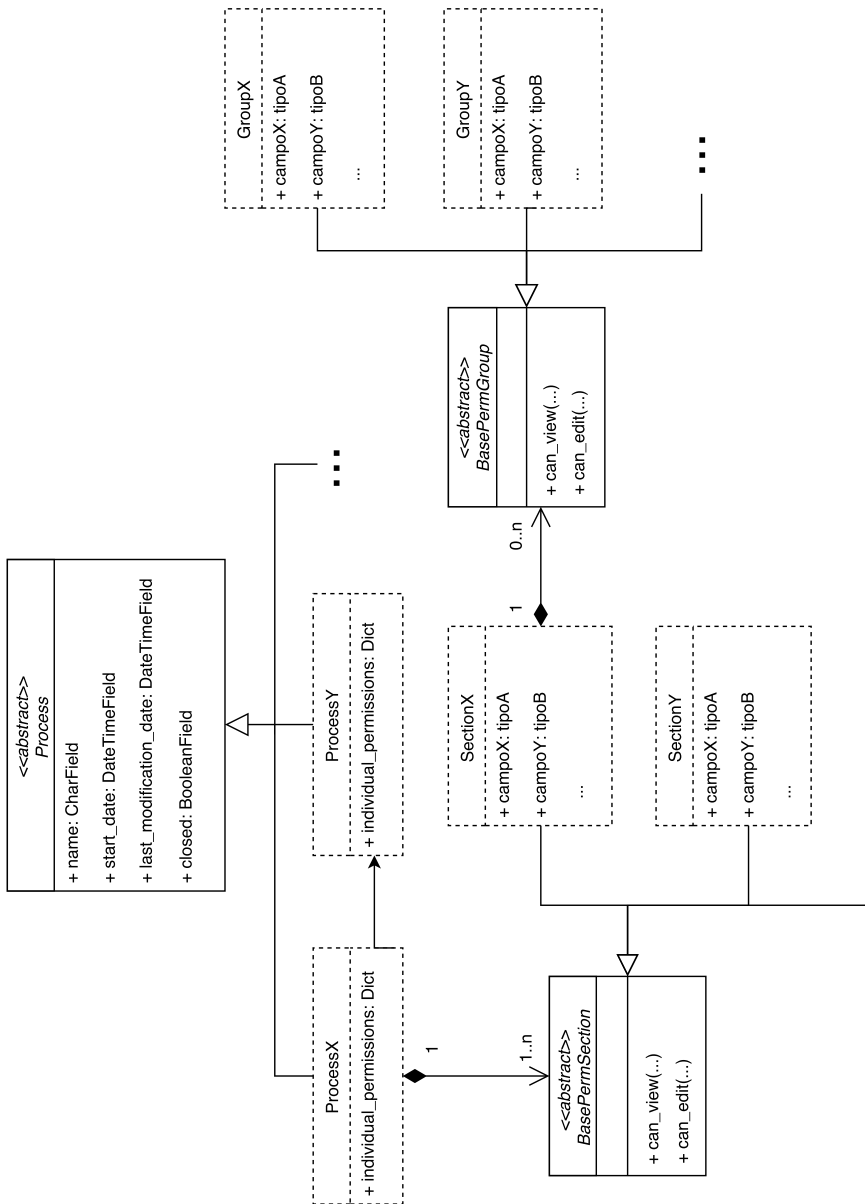


Figura 5.5: Diagrama de clases de los modelos generados en cada app asociada a un proceso administrativo.

```
1  class Name(models.Model):
2      attribute = models.TypeField(null=True)
3      def __unicode__(self):
4      # protected region Name_unicode on begin
5          return unicode(self.attribute)
6      # protected region Name_unicode end
```

Sentencia *section*

Crea un modelo asociado al proceso en el que se encuentra y un formulario asociado a dicho modelo representativo de la sección.

- En AdminDSL:

```
1  process Process1 {
2      section Sect1 {
3          type fieldname;
4      }
5  }
```

- En Django:

  "ProcessName/models.py":

```
1  class Sect1Process1Section(BasePermSection):
2      process = models.ForeignKey(Process1Process)
3      fieldname = models.TypeField(null=True)
```

  "ProcessName/forms.py":

```
1  class Sect1Process1SectionForm(PermissionModelForm)
       :
2      class Meta:
```

```
3           model = Sect1Process1Section
4           fields = ['fieldname', ]
5       def clean_fieldname(self):
6           if self.is_disabled['fieldname']:
7               return self.instance.fieldname
8           else:
9               return self.cleaned_data.get('fieldname
                    ')
```

Sentencia *group*

Crea un modelo asociado asociado a la sección en la que se encuentra y un formulario asociado a dicho modelo representativo del grupo.

- En AdminDSL:

```
1 process P1 {
2     section S1 {
3         group G1 {
4             type fieldname;
5         }
6     }
7 }
```

- En Django:

  "ProcessName/models.py":

```
1 class G1S1P1Group(BasePermGroup):
2     section = models.ForeignKey(S1P1Section)
3     fieldname = models.TypeField(null=True)
```

  "ProcessName/forms.py":

```
1 class G1S1P1GroupForm(PermissionModelForm):
2     class Meta:
3         model = S1P1Section
4         fields = ['fieldname', ]
5     def clean_fieldname(self):
6         if self.is_disabled['fieldname']:
7             return self.instance.fieldname
8         else:
9             return self.cleaned_data.get('fieldname
                 ')
```

Sentencia de definición de campos

Añade un campo/atributo dentro de un modelo.

Las opciones generales que se definen para los campos en el lenguaje AdminDSL tienen la siguiente correspondencia con las opciones de los campos en Django:

- *blank:*
  - En AdminDSL:

    ```
    type(blank="True") name;
    ```

  - En Django:

    ```
    name = models.TypeField(null=True, blank=True)
    ```

- *unique:*
  - En AdminDSL:

```
type(unique="True") name;
```

- En Django:

```
name = models.TypeField(null=True, unique=True)
```

- *label:*
  - En AdminDSL:

```
type(label="Nombre") name;
```

  - En Django:

```
name = models.TypeField(null=True, verbose_name="
    Nombre")
```

Los tipos de campo en el lenguaje AdminDSL tienen la siguiente correspondencia con los tipos de Django:

- *fullName:*
  - En AdminDSL:

```
fullName user_name;
```

  - En Django:

```
1 user_name_firstname = models.CharField(null=True,
    max_length=100)
2 user_name_lastname = models.CharField(null=True,
    max_length=100)
```

- *address:*
  - En AdminDSL:

```
address user_address;
```

  - En Django:

```
1 user_address_country = models.CharField(null=True,
      max_length=100)
2 user_address_town = models.CharField(null=True,
      max_length=100)
3 user_address_postcode = models.CharField(null=True,
       max_length=100)
4 user_address_streetname = models.CharField(null=
      True, max_length=100)
5 user_address_housenumber = models.CharField(null=
      True, max_length=100)
```

- *identityDocument:*
  - En AdminDSL:

```
identityDocument user_id;
```

  - En Django:

```
1 USER_ID_TYPE_CHOICES = (
2     ('NIF', 'NIF:'),
3     ('NIE', 'NIE:'),
4     ('CIF', 'CIF:'),
5     ('PASSPORT', 'PASSPORT:')
6 )
```

```
7  user_id_type = models.CharField(null=True,
      max_length=8, choices=USER_ID_TYPE_CHOICES,
      default='NIF', verbose_name="Type:")
8  user_id_digits = models.CharField(null=True,
      max_length=50, verbose_name="Digits:")
```

- *birthdate:*
  - En AdminDSL:

    ```
    birthdate user_age
    ```

  - En Django:

    "app/models.py/class":

    ```
    user_age = models.DateField(null=True)
    ```

    "app/forms.py/classform":

    ```
    user_date = forms.DateField(widget=
       SelectDateWidgetWithReadonly(years=range(
       datetime.date.today().year, datetime.date.
       today().year-200, -1)))
    ```

- *phone:*
  - En AdminDSL:

    ```
    phone phone_number;
    ```

  - En Django:

    ```
    phone_number = models.CharField(null=True,
       max_length=50)
    ```

- *email:*
  - En AdminDSL:

    ```
    email main_email;
    ```

  - En Django:

    ```
    main_email = models.EmailField(null=True)
    ```

- *currency:*
  - En AdminDSL:

    ```
    currency cash;
    ```

  - En Django:

    ```
    cash = CurrencyField(null=True, max_digits=50,
        decimal_places=2)
    ```

- *string:*
  - En AdminDSL:

    ```
    string short_description;
    ```

  - En Django:

    ```
    short_description = models.CharField(null=True,
        max_length=200)
    ```

- *text:*

- En AdminDSL:

```
text long_description;
```

- En Django:

```
long_description = models.TextField(null=True)
```

- *date:*

- En AdminDSL:

```
date meeting_date
```

- En Django:

"app/models.py/class":

```
meeting_date = models.DateField(null=True)
```

"app/forms.py/classform":

```
meeting_date = forms.DateField(widget=
    SelectDateWidgetWithReadonly(years=range(
    datetime.date.today().year+10, datetime.date
    .today().year-100, -1)))
```

- *integer:*

- En AdminDSL:

```
integer member_count;
```

- En Django:

```
member_count = models.IntegerField(null=True)
```

- *decimal:*
    - En AdminDSL:

      ```
      decimal grade;
      ```

    - En Django:

      ```
      grade = models.DecimalField(null=True, max_digits
          =50, decimal_places=4)
      ```

- *float:*
    - En AdminDSL:

      ```
      float temperature;
      ```

    - En Django:

      ```
      temperature = models.FloatField(null=True)
      ```

- *choice:*
    - *table:*
        - En AdminDSL:

          ```
          choice(table="EntityName") entityname;
          choice(table="EntityName", multiple="True")
              entityname;
          ```

- En Django:

```
1 entityname = models.ForeignKey("app.EntityName",
      null=True)
2 entityname = models.ManyToMany("app.EntityName")
```

- *process_table:*
  - En AdminDSL:

```
1 choice(process_table="ProcessName") processname;
2 choice(process_table="ProcessName", multiple="
     True") processesnames;
```

  - En Django:

```
1 processname = models.ForeignKey("ProcessName.
     ProcessNameProcess", null=True,
     limit_choices_to = ~models.Q(
     current_state__name = "initial"))
2 processesnames = models.ManyToManyField("
     ProcessName.ProcessNameProcess",
     limit_choices_to = ~models.Q(
     current_state__name = "initial"))
```

- *values:*
  - En AdminDSL:

```
choice(values="A","B","C") available_options
    ;
```

  - En Django:

```
1 AVAILABLE_OPTIONS_CHOICES = (
2     (None, _("Choose an option")),
```

```
3       ('A', 'A'),
4       ('B', 'B'),
5       ('C', 'C'),
6   )
7   available_options = models.CharField(null=True,
        choices=AVAILABLE_OPTIONS_CHOICES, default=
        None, max_length=250)
```

- *chain* y *chain_remote:*
    - En AdminDSL:

```
1   process Process1 {
2       section Section1 {
3           choice(process_table="Process2")
               process2;
4           ...
5       }
6       ...
7   }
8   process Process2 {
9       ...
10  }
11  process Process3 {
12      section Section1 {
13          choice(process_table="Process1")
               process1;
14          choice(process_table="Process2", chain="
               process1", remote_chain="Process1.
               Section1.process2")
15          ...
16      }
17      ...
18  }
```

◦ En Django:

"Process1/models.py":

```
1  class Process1Process(Process):
2      ...
3
4  class Section1Process1Section(
       BasePermSection):
5      process = models.ForeignKey(
           Process1Process, null=True)
6      process2 = models.ForeignKey("Process2.
           Process2Process", null=True)
7      ...
```

"Process2/models.py":

```
1  class Process2Process(Process):
2      ...
```

"Process3/models.py":

```
1  class Process3Process(Process):
2      ...
3
4  class Section1Process3Process(
       BasePermSection):
5      process = models.ForeignKey(
           Process3Process,null=True)
6      process1 = models.ForeignKey("Process1.
           Process1Process", null="True")
7      process2 = ChainedForeignKey(
8          "Process2.Process2Process",
```

```
9           chained_field="process1",
10          chained_model_field="
               process1section1section__process2
               ",
11          show_all=False,
12          null=True,
13      )
14      ...
```

- *file:*
  - En AdminDSL:

    ```
    file document_uploaded;
    ```

  - En Django:

    ```
    document_uploaded = models.FileField(null=True)
    ```

- *bool:*
  - En AdminDSL:

    ```
    bool closed;
    ```

  - En Django:

    ```
    closed = models.BooleanField(default=False)
    ```

- *url:*
  - En AdminDSL:

```
url web;
```

- En Django:

```
web = models.URLField(null=True)
```

Sentencia *state*

Esta sentencia crea una instancia de la clase “State” en las migraciones del *app* asociado al proceso que la contiene.

- En AdminDSL:

```
1 state initial {
2     ...
3 }
```

- En Django:

  “app/migrations/states.py”:

```
1 def register_states(apps, schema):
2     initial, created = State.objects.get_or_create(
         name='initial', process_name='app')
3     if created:
4         logger.info('initial state of the process
             app created')
5 class Migration(migrations.Migration):
6     operations = [migrations.RunPython(
         register_states),]
```

Sentencia *permissions* en el ámbito de un estado

Crea permisos en las migraciones de la *app* asociada al proceso que contiene el estado donde se han declarado. En Django, el nombre (o *codename*) del permiso creado es de la forma:

proceso_estado_editable/viewable_all,

proceso_estado_sección_editable/viewable_all,

proceso_estado_sección_campo_editable/viewable,

proceso_estado_sección_grupo_editable/viewable_all,

proceso_estado_sección_grupo_campo_editable/viewable.

- En AdminDSL:

```
1  role bosses;
2  role workers;
3  process process1 {
4      ...
5      section section1 {
6          type sectionfield;
7          ...
8          group group1 {
9              type groupfield;
10             ...
11         }
12     }
13     state initial {
14         permissions bosses {
15             editable all;
16         }
17         permissions workers {
18             viewable section1;
```

```
19              editable section1.group1;
20          }
21      }
22  }
```

- En Django:

  “process1/migrations/permissions.py”:

```
1  def register_custom_permissions(apps, permissions):
2      ContentType = apps.get_model("contenttypes", "
           ContentType")
3      Permission  = apps.get_model("auth", "
           Permission")
4      # create a content type for the app
5      ct, created = ContentType.objects.get_or_create
           (app_label='process1', model='
           process1process',)
6      if created:
7          print "  Adding content type '%s'" % ct
8      # create permissions
9      for codename, name in permissions:
10         p, created = Permission.objects.
               get_or_create(codename=codename,
               content_type=ct, defaults={'name': name
               })
11         if created:
12             print "Adding custom permission '%s'" %
                   codename
13 def apply_custom_permissions(apps, schema):
14     permissions = (
15     ('process1_initia_editable_all', 'All fields of
           the process1 in the state initial are
```

```
       editable'),
16     ('process1_initial_section1_viewable_all', 'All
        fields in the section section1 of the
       process process1 in the state initial are
       viewable'),
17     ('process1_initial_section1_group1_editable_all
       ', 'All fields in the group group1 of the
       section section1 of the process process1 in
       the state initial are editable'),
18     )
19     register_custom_permissions(apps, permissions)
20     Group = apps.get_model('auth', 'Group')
21     Permission = apps.get_model('auth', 'Permission
       ')
22     #Permissions for the state initial:
23     grupo = Group.objects.get(name='Bosses')
24     permission = Permission.objects.get(codename='
       process1_initia_editable_all')
25     grupo.permissions.add(permission)
26     grupo = Group.objects.get(name='workers')
27     permission = Permission.objects.get(codename='
       process1_initial_section1_viewable_all')
28     grupo.permissions.add(permission)
29     grupo = Group.objects.get(name='workers')
30     permission = Permission.objects.get(codename='
       process1_initial_section1_group1_editable_all
       ')
31     grupo.permissions.add(permission)
32 class Migration(migrations.Migration):
33     dependencies = [
34         ('contenttypes', '0001_initial'),
35         ('base_admindsl', 'groups'),
36     ]
```

```
37        operations = [migrations.RunPython(
              apply_custom_permissions),]
```

Sentencia *transition* en el ámbito de un estado

Crea una instancia de la clase “Transition” y la asocia con la instancia de la clase “State” asociada al estado donde se ha declarado la transición en el código del AdminDSL.

- *decision_by*: crea la instancia de la clase “Transition” asignándole el grupo de usuario asociado al rol definido en el código del AdminDSL al atributo “decision_by”.
  - En AdminDSL:

```
1  state initial {
2      ...
3      transition(decision_by="bosses") state1;
4  }
```

  - En Django:

    “app/migrations/states.py”:

```
1  def register_states(apps, schema):
2      initial, created = State.objects.
           get_or_create(name='initial',
           process_name='app')
3      ...
4      group = Group.objects.get(name='Bosses')
5      next_state = State.objects.get(name='state1
           ',process_name='app')
6      state1InitialTransition, created =
           Transition.objects.get_or_create(
```

```
            decision_by=group, state=initial,
            next_state=next_state)
7       if created:
8           logger.info('initial to state1
                transition of the process app
                created')
9   class Migration(migrations.Migration):
10      operations = [migrations.RunPython(
            register_states),]
```

- *start_date*: crea la instancia de la clase "Transition" asignándole el valor de la fecha definida en el código del AdminDSL al atributo "start_date".

    - En AdminDSL:

```
1   state initial {
2       ...
3       transition(start_date="2016/04/28-14:00:00")
            state1;
4   }
```

    - En Django:

        "app/migrations/states.py":

```
1   def register_states(apps, schema):
2       initial, created = State.objects.
            get_or_create(name='initial',
            process_name='app')
3       ...
4       start_date = datetime.strptime
            ("2016/04/28-14:00:00", "%Y/%m/%d-%H:%M
            :%S")
```

```
5       next_state = State.objects.get(name='state1
            ',process_name='app')
6       state1InitialTransition, created =
            Transition.objects.get_or_create(
            start_date=start_date, state=initial,
            next_state=next_state)
7       if created:
8           logger.info('initial to state1
                transition of the process app
                created')
9   class Migration(migrations.Migration):
10      operations = [migrations.RunPython(
            register_states),]
```

- *end_date*: crea la instancia de la clase “Transition” asignándole el valor de la fecha definida en el código del AdminDSL al atributo “end_date”.

    - En AdminDSL:

```
1   state initial {
2       ...
3       transition(end_date="2017/02/20-12:00:00")
            state1;
4   }
```

    - En Django:

      “app/migrations/states.py”:

```
1   def register_states(apps, schema):
2       initial, created = State.objects.
            get_or_create(name='initial',
            process_name='app')
3       ...
```

```
4      end_date = datetime.strptime
          ("2017/02/20-12:00:00", "%Y/%m/%d-%H:%M
          :%S")
5      next_state = State.objects.get(name='state1
          ',process_name='app')
6      state1InitialTransition, created =
          Transition.objects.get_or_create(
          end_date=end_date, state=initial,
          next_state=next_state)
7      if created:
8          logger.info('initial to state1
              transition of the process app
              created')
9  class Migration(migrations.Migration):
10     operations = [migrations.RunPython(
          register_states),]
```

- *max*: crea una instancia de la clase "MaxProcessInstances", asociando el nombre del proceso con el grupo de usuario definido por el rol que participa en la transición en el código del AdminDSL y con un número entero que representa el número máximo de instancias de dicho proceso que se pueden crear.

  - En AdminDSL:

```
1  process p1 {
2      ...
3      state initial {
4          transition(decision_by="bosses", max="2")
              state1;
5      }
6  }
```

- En Django:

  "app/migrations/states.py":

```
1 def register_states(apps, schema):
2     #State and transitions creation
3     ...
4     group = Group.objects.get(name='Bosses')
5     ...
6     if group:
7         bossesmax, created = MaxProcessInstances
              .objects.get_or_create(group=group,
              process_name='p1',max=2)
8 class Migration(migrations.Migration):
9     operations = [migrations.RunPython(
          register_states),]
```

5.5. Herramientas utilizadas

En este apartado se describen las herramientas que se han utilizado para el desarrollo de este trabajo.

5.5.1. Eclipse con Xtext y Epsilon

Véase apartado 4.4.1.

5.5.2. PyCharm

PyCharm es un entorno de desarrollo integrado o IDE para Python que posee soporte para diferentes entornos de trabajo tales como Django. Facilita el desarrollo, la depuración y la puesta en marcha de las aplicaciones web desarrolladas gracias al soporte de autocorrecciones, resaltado de errores,

refactorización automatizada y navegación entre las distintas implementaciones propias de Python, lo cual facilita la detección de errores de manera sustancial.

Esta herramienta se ha utilizado principalmente para poner en práctica la aplicación web Django generada a través de AdminDSL.

5.5.3. Control de versiones

Véase apartado 4.4.2.

CAPÍTULO 6 PRUEBAS

6.1. Plan de pruebas

6.1.1. Alcance

Las pruebas se han realizado en torno a los dos elementos principales de este trabajo: el entorno para el lenguaje AdminDSL y las aplicaciones web generadas a partir del mismo.

6.1.2. Tiempo y lugar

Con respecto al entorno de AdminDSL las pruebas se han realizado por cada adición de funcionalidad o aspecto nuevo tanto a nivel de lenguaje como a nivel del editor.

Para las pruebas de la aplicación generada se han realizado a medida que se añadía un nuevo subsistema o funcionalidad, dado que el grado de modificación de aspectos más minuciosos habría traído demasiado volumen de casos de pruebas que posiblemente fuesen innecesarios por motivos de cambios en el código o eliminación del mismo.

Los cambios y mejoras en las etapas iniciales de desarrollo tanto en una

parte como en la otra eran muy frecuentes por lo que se fueron realizando pruebas manuales y de forma menos repetible, esperando a tener estructuras o aspectos fijos antes de comenzar a realizar las pruebas automatizadas.

6.1.3. Naturaleza

Todas las pruebas han sido enfocadas en torno a la especificación del sistema, es decir, a nivel del DSL se han realizado pruebas comprobando que se cumplen todos los requisitos del mismo. En cuanto a las aplicaciones web generadas se han probado cada uno de los subsistemas que lo componen de forma que se correspondan tanto en funcionamiento como en las salidas obtenidas con los requisitos especificados.

6.2. Pruebas de AdminDSL

La primera idea que se presenta es que la tecnología Xtext utilizada para el desarrollo del lenguaje y editor viene provista de validaciones que disminuye en gran medida la aparición de errores en el código y/o ambigüedades en las reglas definidas que puedan afectar al funcionamiento del editor. De esta forma, gracias a este entorno se puede asegurar una calidad alta en el lenguaje desarrollado por lo que se han dado menos prioridad a las pruebas de esta parte.

Sin embargo, esto no es todo lo que envuelve a AdminDSL. Este lenguaje viene acompañado de validaciones extras y de métodos adicionales para el alcance de las referencias cruzadas que ya se han explicado en el capítulo anterior. Y estos métodos y validaciones sí pueden introducir incongruencias en el editor del lenguaje por lo que han tenido que ser probadas de forma más exhaustiva.

6.2.1. Pruebas de validaciones

Respecto a las validaciones, se debe comprobar que los errores se muestran de manera acorde a su especificación y no hay mejor forma de realizar esta comprobación que tratando con el propio editor. Generalmente, hay cinco tipos de validaciones y por lo tanto cuatro tipos de comprobaciones:

- **Validaciones del número de elementos**: en ciertos ámbitos se requiere un cierto número de elementos, por ejemplo, una estructura "process" debe contener al menos una estructura "section".

  Para comprobar estas validaciones se estudian los valores límites de la restricción y se definen los elementos según los mismos, probando los valores más cercanos por cada sentido de cada uno de estos valores límites, incluyendo los mismos y un valor alejado de estos.

  Por ejemplo, para la restricción de "al menos una sección dentro de cada proceso" se probaría con cero secciones, con una sección, con dos y con más de dos (en este caso -*n* secciones no tendría sentido), resultando un aviso de error sólo en el primer caso.

- **Validaciones de elementos duplicados**: para probar este tipo de validaciones simplemente tenemos que declarar dos elementos iguales dentro del mismo ámbito, dos elementos diferentes dentro del mismo ámbito y dos elementos iguales pero en ámbitos diferentes. Por último comprobar que el primer caso resulta en un aviso de error en el editor y que el resto no.

- **Validación de nombres ilegales**: para probar esta validación se debe de utilizar diferentes nombres contra las expresiones regulares o palabras reservadas con el fin de comprobar que se muestran correctamente los avisos de errores oportunos.

- **Validación de referencias**: algunas referencias cruzadas no vienen

definidas a nivel de la gramática del lenguaje y deben de comprobarse a través de las validaciones extras.

Simplemente se coloca una referencia a un elemento inexistente en el mismo ámbito para comprobar que se muestra un error, y que no se muestran errores colocando uno que ya viene definido en el ámbito correspondiente.

- **Validación de elementos dependientes**: existen ciertas opciones que sólo pueden colocarse dentro de un cierto ámbito o elemento, o incluso que deben ir acompañadas de otras.

  Para realizar la comprobación, se coloca una opción fuera del elemento o el ámbito correspondiente para comprobar que se muestra un aviso de error, así como colocarlo sin el acompañamiento del elemento requerido, según sea el tipo de dependencia que se tenga.

Parte del problema no se basa únicamente en las comprobaciones de que las validaciones funcionan correctamente sino en el estudio de qué validaciones son necesarias y no se encuentran definidas para el editor de AdminDSL. Esto es un trabajo que se realiza durante la fases de análisis y especificación del lenguaje, pero muchas de estas validaciones no se toman en cuenta hasta que el lenguaje en sí no es utilizado.

En el siguiente capítulo de validación se expone el uso del lenguaje AdminDSL por parte del personal del Área de Informática de la UCA. Gracias a la colaboración de este personal se han podido encontrar ausencias o mejoras en las validaciones definidas para AdminDSL.

6.2.2. Pruebas de navegabilidad y alcance

Xtext ofrece un proyecto preparado para el desarrollo de pruebas para AdminDSL donde se puede comprobar, entre otros detalles, qué los elementos que conforman el lenguaje se asocian correctamente, y que el alcance de

las referencias entre los mismo es el adecuado. Sin embargo, en este trabajo no se ha utilizado dicho proyecto ni módulo de pruebas puesto que se ha encontrado redundante teniendo en cuenta la segunda parte de este trabajo.

Dado que se ha desarrollado el generador de código para aplicaciones web Django y que éste cubre en su totalidad la disposición de todos los elementos que se definen en el código de AdminDSL, se ha tratado el propio generador como batería de pruebas.

Para ello, se ha elaborado una aplicación que utiliza todos los aspectos del lenguaje, creando múltiples procesos administrativos, cada uno con un grupo de funcionalidades distintas y agrupadas. A partir de este código se ha generado una aplicación web Django comprobando que el código se ha generado correctamente a partir del mismo. Principalmente se ha enfocado esta comprobación en que los elementos obtenidos a partir del código en AdminDSL, ya sea de forma directa o a través de la navegabilidad de los mismos, vienen dispuestos correctamente. Este código se encuentra completo en el fichero "*testexample.apdsl*" del proyecto "*es.uca.apdsl.testexample*" dentro del directorio "*examples*".

6.3. Pruebas de la aplicación Django con Selenium

Al igual que con AdminDSL, es fundamental probar que la aplicación web generada en Django funciona correctamente ofreciendo cada unas de las funcionalidades especificadas. Para ello, el marco de trabajo Django ofrece un módulo de pruebas donde se permite integrar la tecnología Selenium.

Selenium [23] permite definir casos de pruebas donde se puede tomar el control de un navegador, en este caso Firefox, y navegar sobre la aplicación web generada en ejecución realizando comprobaciones oportunas y permitiendo al módulo de pruebas de Django tratar los elementos de los que se componen para realizar validaciones.

Lo que se ha planteado es obtener una aplicación web Django generada a partir del fichero “*testexample.apdsl*” mencionado en el apartado anterior, el cual posee todos los aspectos del lenguaje y que, por lo tanto, genera una aplicación web que abarca todas las funcionalidades posibles respecto a cualquier otra. A partir de esta aplicación se trata un conjunto de casos de prueba por cada subsistema o funcionalidad principal que posea, de forma que con la ejecución de cada caso de prueba se cubra todo el código que pueda participar en dicho subsistema o funcionalidad. Todo esto enfocado a que cada caso de prueba funcione tanto a nivel unitario como a nivel de integración.

Cada caso de prueba contiene una serie de asertos o validaciones de forma que se prueba a nivel unitario ciertas partes de código que participan en una funcionalidad concreta y, a su vez, el conjunto de estas validaciones comprueban a nivel de integración que la funcionalidad o subsistema funciona correctamente.

Lógicamente, aunque se trate de cubrir todo el código, el número de combinaciones posibles de entradas para cada funcionalidad es demasiado grande. Por lo tanto, probar de forma completa cada una de las funcionalidades es algo inviable en este nivel. Por ello, el enfoque de las pruebas está basado en cada una de las funcionalidades principales de la aplicación donde se ha tratado de probar tanto aquellos aspectos más generales y básicos como los más difíciles de rastrear a nivel de código.

Para cubrir el mayor número de posibilidades y detectar errores lo más viable es colocar este trabajo en un entorno de producción (como ha sido el caso) y ponerlo en manos de distinto personal de confianza que consigan encontrar errores o vulnerabilidades que no eran posibles de detectar mediante los casos de pruebas definidos.

A continuación se exponen los subsistemas que se han probado así como los casos de pruebas que se han definido para cada uno de estos. Saber

que para las pruebas se han creado tres roles de usuario: “Low”, “Medium”, “Super”, cada uno con permisos diferentes según sea el proceso.

6.3.1. Subsistemas y casos de prueba

Subsistema de autentificación

- **Inicio de sesión** con datos correctos. Se inicia sesión con un usuario existente en el sistema y se comprueba que la sesión es iniciada con éxito sin errores.
- **Inicio de sesión con datos incorrectos.** Se inicia sesión con una contraseña equivocada y se comprueba que se muestra un error de que los datos son incorrectos.
- **Cierre de sesión**. Se inicia y se cierra la sesión del usuario. Se comprueba que no hay ningún usuario conectado en el sistema.

Subsistema de gestión de procesos administrativos

Los casos de pruebas que se han definido son los que se exponen a continuación:

- **Creación de procesos.**
  1. Se inicia sesión con un usuario “Low”.
  2. Se comienza un proceso “Process2”.
  3. Se rellenan ciertos campos del formulario.
  4. Se envía el formulario.
  5. Se comprueba que el proceso ha sido creado.
  6. Se consulta el proceso y se comprueba la integridad de los datos.
- **Edición de procesos.**

1. Se inicia sesión con un usuario “Low”.
2. Se comienza un proceso “Process2”.
3. Se rellenan ciertos campos del formulario.
4. Se envía el formulario.
5. Se comprueba que el proceso ha sido creado.
6. Se edita el proceso cambiando el valor de uno de sus campos.
7. Se envía de nuevo el formulario.
8. Se consulta el proceso y se comprueba que los datos han sido correctamente actualizados.

- **Edición de procesos compartido por distintos roles.**

1. Se inicia sesión con un usuario “Low”.
2. Se comienza un proceso “Process3” y se rellenan los campos visibles.
3. Se cierra sesión con el usuario “Low”.
4. Se inicia sesión con un usuario “Super”.
5. Se edita el proceso previamente comenzado y se rellenan los campos que antes no eran visibles dejando los cumplimentados por el usuario “Low” tal y como están.
6. Se cierra sesión con el usuario “Super”.
7. Se inicia sesión con un usuario “Middle”.
8. Se consulta el proceso y se comprueba la integridad de los datos que han introducido los dos usuarios previos.
9. Se cierra sesión con el usuario “Middle”.
10. Se inicia sesión con el usuario “Super”.
11. Se edita el proceso, en concreto los valores de los campos que había introducido el usuario “Low” inicialmente.

12. Se cierra sesión con el usuario “Super”.
13. Se repiten pasos 7-9.

- **Eliminación de procesos.**

1. Se inicia sesión con un usuario “Low”.
2. Se comienza un proceso “Process3”.
3. Se pulsa el botón de “Eliminar” del proceso creado que ahora aparece en la lista de procesos.
4. Se acepta la alerta mostrada.
5. Se comprueba que el proceso ya no aparece en la lista de procesos y que, por lo tanto, ha sido eliminado.

Subsistema de control de transiciones de procesos

- **Creación de proceso que supone cambio de estado por decisión.**

1. Se inicia sesión con un usuario “Low”.
2. Se comprueba que el proceso “Process1” aparece en la lista de procesos disponibles y que, por lo tanto, se puede comenzar. Esto implica que la transición del estado “initial” a otro estado es posible por un usuario “Low”.
3. Se comprueba que dicho proceso no aparece en la lista de procesos no disponibles.
4. Se cierra sesión con el usuario “Low”-
5. Se inicia sesión con un usuario “Middle”.
6. Se comprueba que el proceso “Process1” no aparece en la lista de procesos disponible dado que la fecha de inicio de la transición de “initial” a otro estado aún no ha pasado.
7. Se cierra sesión con el usuario “Middle”.

8. Se inicia sesión con un usuario “Super”.
9. Se comprueba que el proceso “Process1” no aparece en la lista de procesos disponible dado que la fecha de fin de la transición de “initial” a otro estado ya ha pasado.
10. Se cierra sesión con el usuario “Super”.
11. Se repiten el caso de prueba pero esta vez con el proceso “Process2”. Se realizan las mismas comprobaciones. La diferencia es que para el proceso “Process2” sólo existe una condición de fecha para cada transición ya sea de inicio o de fin mientras que para el proceso “Process1” existen una fecha de inicio y otra de fin por cada transición donde se incumple una de ellas para los casos de los roles “Middle” y “Super”.

- **Cambio de estado por decisión.**

1. Se inicia sesión con un usuario “Low”.
2. Se comienza un proceso “Process1”.
3. Se comprueba que el proceso se encuentra en el estado “state1”.
4. Se consulta el proceso y se pulsa sobre el envio del formulario con cambio del estado “state1” a “state2”.
5. Se comprueba que el proceso ha cambiado de estado.

- **Cambio de estado automático.**

1. Se inicia sesión con un usuario “Low”.
2. Se comienza un proceso “Process7”.
3. Se comprueba que el proceso se encuentra en el estado “state1”.
4. Se ejecuta la orden “update_states”.
5. Se comprueba que el proceso ha cambiado de “state1” a “state2”.
6. Se ejecuta de nuevo la orden “update_states”.

7. Se comprueba que el proceso no ha cambiado del estado “state2” dado que contiene campos obligatorios sin rellenar (los cuales no eran visibles en el estado “state1”).
8. Se edita el proceso y se rellenan todos los campos obligatorios.
9. Se ejecuta la orden “update_states”.
10. Se comprueba que el proceso ha cambiado de “state2” a “state3”.

Subsistema de control de permisos

- **Comprobación de visibilidad de campos.**
  1. Se inicia sesión con un usuario “Low”.
  2. Se comienza un proceso “Process1”.
  3. Se consulta el proceso y se comprueba que todos los campos del formulario se pueden ver pero no editar.
  4. Se cierra sesión con el usuario “Low”.
  5. Se inicia sesión con un usuario “Middle”.
  6. Se consulta el proceso y se comprueba que sólo posee visibilidad de ciertos campos según se ha definido en sus permisos a través del código de AdminDSL y que no puede editar nada.
  7. Se cierra sesión con el usuario “Middle”.
  8. Se inicia sesión con un usuario “Super”.
  9. Se comprueba que el proceso creado no aparece en la lista de procesos puesto que este rol no posee ningún permiso de visibilidad para ninguno de sus campos.
  10. Se comprueba que el acceso a través de la ruta conociendo su identificador tampoco es posible.
- **Comprobación de permisos de edición de campos.** Mismo caso de prueba que el anterior cambiando el proceso “Process1” por “Process2”

y comprobando que los campos dados se pueden editar en lugar de solamente verlos.

Subsistema de validaciones de campos

- **Validaciones de campos obligatorios.**

  1. Se inicia sesión con un usuario “Low”.
  2. Se comienza un proceso “Process4” donde todos sus campos son obligatorios y de diferente tipo.
  3. Se trata de enviar el formulario dejando un sólo campo sin rellenar.
  4. Se comprueba que el campo en cuestión muestra un error de que debe ser rellenado y que el formulario no se envía.
  5. Se repiten los pasos 4-5 de forma que el único campo sin rellenar sea diferente hasta probar con todos los campos del formulario.
  6. Por último, se rellenan todos los campos del formulario y se envía.
  7. Se comprueba que el proceso se ha creado con éxito con los datos introducidos.

Subsistema de gestión de grupos dinámicos en formularios

- **Creación sin adición de grupos.**

  1. Se inicia sesión con un usuario “Low”.
  2. Se comienza un proceso “Process5” el cual posee dos grupos dinámicos, uno que requiere al menos una instancia obligatoria con ciertos campos obligatorios y otro que no requiere ninguna instancia pero con ciertos campos obligatorios en caso de que se añada alguna.

3. Se trata de enviar el formulario sin rellenar los campos obligatorios del primer grupo.
4. Se comprueba que se muestra un error de formulario porque existen campos obligatorios sin rellenar.
5. Se rellenan los campos obligatorios del grupo uno y se envía el formulario.
6. Se comprueba que el proceso se ha creado con éxito y la integridad de sus datos.

- **Creación con adición y eliminación.**

  1. Se inicia sesión con un usuario “Low”.
  2. Se comienza un proceso “Process5”.
  3. Se añade una instancia nueva del primer grupo.
  4. Se elimina la instancia creada.
  5. Se comprueba que no existe botón de eliminar cuando sólo queda una instancia (puesto que se requiere al menos una).
  6. Se añade una instancia nueva del segundo grupo.
  7. Se elimina la instancia creada.
  8. Se trata de enviar el formulario sin rellenar los campos obligatorios del primer grupo.
  9. Se comprueba que se muestra un error de que existen campos obligatorios no rellenados.
  10. Se rellenan los campos obligatorios.
  11. Se comprueba que el proceso se ha creado correctamente, la integridad de sus datos así como que no aparezcan grupos que habían sido eliminados.

- **Creación con una adición.**

1. Se inicia sesión con un usuario “Low”.
2. Se comienza un proceso “Process5”.
3. Se rellenan los campos obligatorios del primer grupo.
4. Se añade una instancia nueva del primer grupo.
5. Se trata de enviar el formulario.
6. Se comprueba que se muestra un error de que existen campos obligatorios no rellenados.
7. Se rellenan los campos obligatorios de la nueva instancia del primer grupo.
8. Se envía el formulario.
9. Se comprueba que el proceso se ha creado correctamente y la integridad de sus datos.

- **Creación con dos adiciones.**

1. Se inicia sesión con un usuario “Low”.
2. Se comienza un proceso “Process5”.
3. Se rellenan los campos obligatorios del primer grupo.
4. Se añade una instancia nueva del primer grupo y dos nuevas del segundo.
5. Se rellenan los campos obligatorios de la nueva instancia del primer grupo y de la primera instancia del segundo grupo.
6. Se trata de enviar el formulario.
7. Se comprueba que se muestra un error de que existen campos obligatorios no rellenados en la última nueva instancia del segundo grupo.
8. Se rellenan los campos obligatorios de la segunda instancia del segundo grupo.

9. Se envía el formulario.
10. Se comprueba que el proceso se ha creado correctamente y la integridad de sus datos.

- **Edición con una eliminación.**

1. Se inicia sesión con un usuario “Low”.
2. Se comienza un proceso “Process5”.
3. Se rellenan los campos obligatorios del primer grupo.
4. Se añade una instancia nueva del primer grupo.
5. Se rellenan los campos obligatorios de la nueva instancia del primer grupo.
6. Se envía el formulario.
7. Se edita el proceso, se elimina la primera instancia del primer grupo y se envía el formulario.
8. Se consulta el proceso y se comprueba que la primera instancias del primer grupo ya no aparece y que la integridad de los datos de la segunda se mantiene correcta.

Subsistema de gestión de borradores de procesos

- **Creación de borradores.**

1. Se inicia sesión con un usuario “Low”.
2. Se comienza un proceso “Process2”.
3. Se rellenan ciertos campos del proceso y se guarda el formulario en lugar de enviarlo.
4. Se comprueba que el proceso aparece en la lista de borradores.
5. Se comprueba la integridad de los datos del proceso.

- **Edición de borradores con eliminación.**

  1. Se inicia sesión con un usuario “Low”.
  2. Se comienza un proceso “Process2”.
  3. Se guarda el formulario en lugar de enviarlo.
  4. Se comprueba que el proceso aparece en la lista de borradores.
  5. Se edita el proceso y se comprueba la integridad de los datos del borrador y se envía.
  6. Se comprueba que el proceso ahora aparece en la lista de procesos activos y que no aparece ningún borrador.
  7. Se edita el proceso cambiando el valor de alguno de sus campos y se guarda como borrador.
  8. Se elimina el borrador y se comprueba que el proceso vuelve a su estado anterior.

- **Eliminación de borradores.**

  1. Se inicia sesión con un usuario “Low”.
  2. Se comienza un proceso “Process3”.
  3. Se rellenan ciertos campos del proceso y se guarda el formulario en lugar de enviarlo.
  4. Se comprueba que el proceso aparece en la lista de borradores.
  5. Se elimina el borrador.
  6. Se comprueba que tanto el borrador como el proceso ya no existen en ninguna lista, puesto que no se envió ningún dato.

- **Creación y eliminación de borradores con ficheros, selectores múltiples y grupos dinámicos.**

  1. Se inicia sesión con un usuario “Low”.

2. Se comienza un proceso “Process8”.
3. Se añade una instancia nueva, además de la ya existente, del grupo dinámico.
4. Se rellenan los campos obligatorios de ambas instancias donde se incluye seleccionar un fichero y seleccionar opciones de un selector múltiple.
5. Se envía el formulario.
6. Se edita el proceso y se elimina la primera instancia del grupo y se modifican los datos de la segunda.
7. Se guarda como borrador.
8. Se comprueba que el proceso aparece ahora como borrador en la lista de procesos.
9. Se edita el borrador y se comprueba la integridad de los datos.
10. Se elimina el borrador y se comprueba que la instancia del grupo eliminada vuelve a aparecer y que los datos de la segunda se han restablecido.

6.3.2. Cobertura de las pruebas

Utilizando la herramienta Coverage integrada en la aplicación Django generada, que permite medir la cobertura de código para programas en Python, se han obtenido estadísticas para saber qué líneas de código de toda la aplicación han participado en las pruebas.

Omitiendo el código situado en el entorno virtual y centrando la estadística en el código generado a partir del código en AdminDSL se ha alcanzado un nivel de cobertura del 82 % con un total de 4500 líneas de código.

Si bien una pérdida de 6 % se debe al código asociado a la orden “update_state”, puesto que el 85 % del código se asocia con procesos que no han sido sujetos a las pruebas para evitar redundancia, ya que la estructura

del código es exactamente la misma. Otro 2 % se debe a que el componente de vistas de cada proceso no se utiliza (se utiliza el generalizado en “base_admindsl”), pero es código útil para que los desarrolladores puedan hacer uso del mismo y personalizar vistas para procesos particulares. Si omitimos estos dos últimos factores la cobertura alcanzaría un nivel del 90 % del código teniéndose un total de 4042 líneas de código.

6.3.3. Medidas de rendimiento

Para medir el rendimiento de la aplicación web se ha utilizado la herramienta Debug Toolbar [24] integrada en la misma. Esta herramienta permite obtener estadísticas varias respecto al rendimiento, consultas en la base de datos, movimiento de datos a través de la aplicación, etc.

A través de la información obtenida con esta herramienta se ha tratado de reducir el número de consultas a la base de datos con el fin de mejorar el rendimiento. Además se ha facilitado la detección de errores y la realización de trazas para comprobar el funcionamiento correcto de la aplicación.

Cabe mencionar que dada la cantidad de posibilidades entre los permisos de un proceso, sección, grupo o campo y los roles participantes o los propios usuarios, las vistas genéricas autogeneradas para la visualización de las listas de procesos pueden resultar lentas. Sin embargo, estas vistas no están enfocadas a ser utilizadas en producción sino que están pensadas para servir de guía al desarrollador que pueda ver el funcionamiento del flujo de reglas de negocio de forma más clara y sencilla.

CAPÍTULO 7

VALIDACIÓN

7.1. Artículos

A lo largo del desarrollo de este trabajo se han publicado dos artículos en relación al mismo. En estos se han presentado tanto el DSL como el generador de código asociado. Ambos artículos han sido aceptados y presentados, lo que ha resultado en una experiencia positiva tanto para los autores como para el mismo trabajo, habiéndose encontrado puntos que mejorar y nuevas posibilidades de cara al futuro.

7.1.1. Fifth International Symposium on Business Modeling and Software Design (BMSD 2015)

BMSD [25] es un foro internacional que une a investigadores y profesionales interesados en el modelado de negocios y su relación con el diseño software. Las áreas particulares de interés son: modelos de negocios y los requisitos; modelos de negocios y los servicios; modelos de negocios y el software; arquitectura de sistemas de información.

El artículo presentado para el BMSD 2015 fue aceptado como artículo

corto y presentado en Milán. La aceptación fue grata y dos de los asistentes se ofrecieron como beta testers.

Uno de los puntos de interés trató sobre llevar el enfoque del DSL a un ámbito separado de la Universidad de Cádiz y poder generalizar los procesos que se manejan a cualquier otro tipo. Esta sugerencia ha despertado la posibilidad de llevar este trabajo a un ámbito más extenso con más posibilidades y por supuesto con mayor número de personas de interés, lo cual resulta muy positivo ya que se mantiene la vida del proyecto como un proyecto activo en desarrollo y no sólo en mantenimiento.

7.1.2. Jornadas de Ingeniería del Software y Bases de Datos XX (JISBD 2015)

Las JISBD [26] son un foro de referencia en la investigación de la Ingeniería del Software y las Bases de Datos en el ámbito iberoamericano. A lo largo de los años, el evento ha servido para que los investigadores de España, Portugal y Latinoamérica presentasen sus trabajos y establecieran una comunidad muy sólida alrededor de ambas disciplinas. En 2015, las Jornadas celebran su XX edición. Es, por ello, una ocasión para hacer balance del camino recorrido, por un lado, y de consolidar el papel dinamizador de la comunidad a la que aloja, por otro.

En este caso, el artículo presentado en el año 2015 ha sido aceptado como artículo completo. Las valoraciones aportadas por los revisores han sido generalmente positivas puntualizándose ciertos apartados que han permitido una mejora de este trabajo en ciertos aspectos que se mencionan a continuación:

- Mejoras en la gramática de AdminDSL: evitando incongruencias en los nombres de las reglas o aportando una estructura más clara y fácil de entender.

- Integración de validaciones en OCL al validador del editor asociado a AdminDSL: se propuso especificar validaciones en OCL con el fin de conocer qué restricciones poseía AdminDSL y el editor asociado. La solución que se ha llevado a cabo ha sido aprovechar esta especificación para implementar las validaciones pertinentes, de forma que la propia especificación en OCL ha servido de forma directa y textual para el desarrollo de las mismas.

- Mayor fuerza en la justificación del uso de este trabajo frente a herramientas BPMN.

7.2. Aplicaciones

En el Área de Informática de la UCA se ha puesto en marcha este trabajo con el desarrollo de ciertas aplicaciones que se presentan a continuación. Estas aplicaciones han permitido detectar errores que durante la fase de pruebas no habían salido a luz. Además se han estudiado nuevas necesidades que implican ampliaciones tanto en AdminDSL como en el generador de código.

Cada una de las aplicaciones han sido especificadas mediante reuniones donde se ha utilizado AdminDSL como lenguaje principal para la especificación de las mismas así como diagramas de clases UML. En estas reuniones y en el seguimiento de las distintas aplicaciones han participado Inmaculada Medina Bulo, Juan José Caballero Muñoz, Jesús Franco Oliva, Gerardo Aburruzaga García, José Manuel Frías Carnero y José Luis Ortega Seijo.

7.2.1. Planes de mejora de titulaciones

El desarrollador de la FUECA José Luis Ortega Seijo ha sido el encargado de llevar a cabo este proyecto haciendo uso de AdminDSL y el generador de aplicaciones web Django asociado. Ha contado con el apoyo del personal

del Área de Informática en las fases de elicitación, análisis y especificación. Y con mi apoyo personal en el desarrollo de la aplicación con el fin de poner a prueba la fase de adaptación con el DSL y el código generado a partir del mismo.

La aplicación consiste en ofrecer una plataforma de gestión de los planes de mejora para las titulaciones que se ofertan en la Universidad de Cádiz, así como elementos asociados a los mismos (responsables, títulos, centros, etc.). Además, estos planes de mejora se componen de acciones también gestionadas por la aplicación a partir de las cuales se pueden realizar seguimientos.

Durante la fase de especificación de esta aplicación se realizó una ampliación de AdminDSL con el fin de que permitiese definir relaciones entre procesos con entidades y que éstas pudieran tanto binarias como ternarias. Se implementó de forma que se podían formar relaciones n-arias y se cumplió con esta demanda en menos de una semana.

Durante la realización de esta aplicación también se detectó una carencia en AdminDSL y es la ausencia de elementos para definir un panel de administración (diferente al que aporta Django) para poder crear, modificar y consultar entidades, ya que esto sólo puede hacerse a nivel de procesos administrativos. Esto implica cambios más severos en el generador y por lo tanto aún no se ha desarrollado, aunque ya se encuentra diseñada la estructura que será generada para cumplir con este objetivo.

Sin embargo, la idea fue tratar a las entidades como procesos administrativos que vienen dotados de la gestión oportuna y simplemente eliminar de estos la lógica no útil para este caso. Estas acciones tomadas no consumieron un gran esfuerzo y permitieron la continuación y finalización de la aplicación de forma satisfactoria a falta de la aprobación final de los clientes. Si bien, este tipo de aplicaciones han abierto las puertas a nuevas mejoras o cambios futuros que impliquen, entre otras cosas, la disminución de mode-

los generados para el tratamiento de los procesos administrativos.

7.2.2. Seguimiento del Plan Estratégico de la Universidad de Cádiz

Esta aplicación es llevada a cabo por Jesús Franco Oliva, quien ha aportado muchas sugerencias con el fin de mejorar tanto el lenguaje AdminDSL como el código generado, tales como la adición de nuevas secciones protegidas, mejoras que permitan regenerar código evitando complicaciones o mejora en la visualización de los listados de proceso.

Su adaptación con el entorno Eclipse para el uso de AdminDSL ha sido algo más lenta que para la aplicación anterior. Sin embargo, esto ha permitido una mejor puesta a prueba del lenguaje dado que se han podido detectar más mejoras y se han podido realizar correcciones por el hecho de haberse hecho uso del mismo con más detenimiento.

La aplicación se basa en ofrecer una gestión sobre la información que se presenta en el Plan Estratégico de la Universidad de Cádiz (PEUCA) con el fin de poder obtenerla y generarla según se crea conveniente.

7.2.3. Encuestas sobre puestos de trabajo

Esta aplicación ha sido desarrollada por mí mismo. Se puede decir que era la más acorde a la especificación de AdminDSL y por lo tanto ha sido la que menos código extra ha necesitado una vez generada la aplicación web Django en sus versiones iniciales.

En muy poco tiempo se han conseguido resultados muy cercanos a los del primer prototipo, teniéndose una buena aceptación por parte de los clientes.

Juan José Caballero Muñoz e Inmaculada Medina Bulo han sido los encargados de hacer un seguimiento del progreso del desarrollo de esta aplicación aportando sugerencias y mejoras que han sido muy útiles para la

satisfacción de los solicitantes.

En un principio, la aplicación consistía en la realización de encuestas sobre puestos de trabajo de la Universidad de Cádiz. Estas encuestas eran realizadas por empleados, supervisadas y completadas por los jefes, y finalmente aprobadas por un administrador. Con esto se cubría con los requisitos del primer prototipo.

Para el segundo prototipo, las encuestas solamente tenían que ser revisadas por los jefes y estos tenían la opción de generar una ficha (otro formulario) de forma automática a partir de los datos de las encuestas revisadas. Posteriormente, el analista era el encargado de generar a partir de estas fichas anteriores una nueva ficha que finalmente es llevada a la ficha final por un supervisor, encargado de revisarla y dejarla lista para la firma de un gerente.

Dado que por cada ficha se requería un formulario muy similar al de las encuestas, se decidió clonar la *app* asociada a estas para cada una de las fichas. En este caso, no se utilizó la regeneración del código a partir de las descripción definida haciendo uso de AdminDSL puesto que la mayor parte de las modificaciones se realizaron sobre las plantillas HTMLs que no se encuentran acotadas en regiones protegidas. La *app* principal *base_admindsl* fue reutilizada para cada uno de los nuevos procesos, pero tuvo que ser retocada de forma mínima, ya que se tuvo que añadir autocompletado de campos a partir de información externa.

A partir del desarrollo de esta aplicación se ha podido poner a prueba a fondo tanto el DSL como el código generado a partir del mismo. Quedó demostrado que para aplicaciones acordes a la definición del propio lenguaje se pueden conseguir resultados finales de forma muy rápida. Además, se han permitido modificaciones posteriores sobre el código generado y se han añadido nuevos procesos ajustándose perfectamente al comportamiento base inicial sin ninguna complicación extra.

El desarrollo de esta aplicación también ha contribuido en el estudio de posibles mejoras y cambios (principalmente en el código generado) como trabajo futuro.

7.2.4. Registro y seguimiento de presupuestos

La última aplicación que hace uso del trabajo actualmente es la encargada de realizar el registro y seguimiento de presupuestos. Está aplicación se encuentra aún en fases iniciales de desarrollo. Se ha especificado mediante AdminDSL y generado la aplicación web Django base a partir de la cual se realizará el desarrollo por parte de José Luis Ortega Seijo.

En esta aplicación se ha desarrollado la estructura para la gestión de entidades de forma manual con el fin de que sirva de ejemplo base para la futura ampliación del generador Django asociado a AdminDSL y conseguir que se integre en el mismo.

CAPÍTULO 8

CONCLUSIONES Y TRABAJO FUTURO

8.1. Resultados obtenidos

La idea de este TFG inicialmente era conseguir facilitar el desarrollo y mantenimiento de aplicaciones web para procesos administrativos que se desarrollan en la Universidad de Cádiz mediante el uso de AdminDSL y el generador de aplicaciones Django asociado.

Este objetivo se ha cumplido dentro del Área de Informática aportando a las distintas aplicaciones desarrolladas una misma estructura o esqueleto básico. Además se ha facilitado la especificación de requisitos a través del uso de AdminDSL, haciéndolos más visibles y más cercano a la aplicación final, y un ahorro del esfuerzo dado que se evita implementar las aplicaciones desde cero.

8.2. Impresiones personales

Bajo un punto de vista personal, el desarrollo de este trabajo ha aportado una gran fuente de nuevos conocimientos y habilidades. Dado que se ha utilizado una variedad considerable de conceptos, tecnologías, herramientas y

lenguajes de programación diferentes; la adquisición de nuevos conocimientos y de buenas prácticas de desarrollo ha sido indiscutible. Sin olvidar que se han mejorado las habilidades de comunicación con el personal de interés y con los clientes de las aplicaciones que se han desarrollado utilizando este TFG, siendo partícipe de proyectos en el Área de Informática gracias al uso del mismo.

Añadir que en este TFG se ha tenido que afrontar problemas que difícilmente han sido resueltos previamente dada la composición de tecnologías utilizadas y sobre todo dados los aspectos que ofrece el propio AdminDSL, especificados de manera personal bajo el estudio de las necesidades, cuyo resultado final se refleja en una aplicación web utilizando la tecnología Django, la cual no viene preparada por defecto para abarcar muchos de estos aspectos.

Además de lo mencionado anteriormente, la satisfacción personal de haber podido ser de ayuda para la experiencia de trabajo del personal del Área de Informática y querer llevar esa mejora a una comunidad más extensa, ha sido la principal fuente de energía para llevar a cabo este TFG con ambición y coraje.

8.3. Trabajo futuro

Se espera que este trabajo sea utilizado en más áreas dentro de la UCA y fuera de esta institución, donde ya han habido partes interesadas gracias a los artículos publicados para las JISBD 2015 y el BMSD 2015.

Tras la puesta en marcha de este TFG en el Área de Informática también se originó la idea de llevar el uso del lenguaje AdminDSL a un nivel más alto en cuanto a especificación. En la mayoría de aplicaciones web para procesos administrativos, los desarrolladores no sólo se conforman con la especificación de procesos administrativos como tales sino que además ne-

cesitan de otros elementos participantes como pueden ser la especificación de estados globales o paneles de administración («backends») para entidades. Estas ideas ya han sido registradas para futuras mejoras del lenguaje y el generador e incluso se ha comenzado su desarrollo.

Además de añadir estas nuevas funcionalidades, también se han ido planteando mejoras a nivel de calidad de código generado. Entre las ideas que se han planteado se encuentran las siguientes:

- Disminución del número de modelos para la definición de procesos administrativos: actualmente se cuenta con un modelo para el proceso, tantos modelos como secciones conforman el documento formal de dicho proceso y tantos modelos como grupos de campos existan, teniéndose las relaciones pertinentes entre ambos. La idea es reducir la cantidad de modelos, por ejemplo, definiendo las secciones en el propio modelo del proceso denominando sus campos con el nombre de la sección a la que pertenece además de su propia denominación mediante un separador.

- Hacer uso de las vistas basadas en clases en Django. Estas vistas permiten, en pocas líneas de código, desarrollar todo lo necesario para la gestión de entidades en una aplicación Django (creación, modificación, eliminación y listado). Ofrece una serie de especificaciones que permiten ofrecer un nivel de granuralidad adecuado para el mantenimiento y la simplificación del código.

Lógicamente estos cambios afectarían en gran medida a muchas de las funcionalidades que ofrece el código generado a partir de AdminDSL, tales como la gestión de permisos en múltiples niveles (proceso, secciones, grupos o campos) o el uso de borradores que permitan guardar a nivel individual el contenido de un formulario pendiente de envío, entre otras. Por ello, se han dejado estos cambios apartados del proyecto actual, dado el grado de

modificación que conlleva.

Dicho esto, cabe destacar que este trabajo seguirá siendo un proyecto activo y abierto a sugerencias donde se llevará a cabo un continuo mantenimiento y estudio para la adición de nuevas mejoras y facilidades con el fin de mejorar la experiencia de los usuarios que lo utilicen. Todo esto gracias, en parte, al hecho de que se trate con un DSL propio, lo cual facilita la adaptación a nuevos cambios manteniendo de forma correcta lo que ya funcionaba previamente.

APÉNDICE A
GRAMÁTICA COMPLETA DE ADMINDSL

En este apartado se expone la gramática completa del lenguaje AdminDSL según se ha definido utilizando la tecnología Xtext.

```
1  Application:
2      elements += Element*
3  ;
4  Element:
5      Site | Options | Role | Process | Entity
6  ;
7  Site:
8      'site' name=ID ';'
9  ;
10 Options:
11     'options' '{'
12         (elements += Option ';')+
13     '}'
14 ;
15 Role:
16      ('role' name=ID ';') |
17      ('role' name=ID '{'
```

```
18          (properties += Option ';')*
19       '}')
20   ;
21   Process:
22       'process' name=ID '{'
23           (elements += ProcessElement)+
24       '}'
25   ;
26   ProcessElement:
27       State | Section | Entity | Relation
28   ;
29   Entity:
30       'entity' name=ID '{'
31           elements += EntityElement*
32       '}'
33   ;
34   EntityElement:
35       Field | Relation
36   ;
37   Section:
38       'section' ('(' options += Option (',' options += Option
             )* ')')? name=ID '{'
39           (elements += SectionElement)+
40       '}'
41   ;
42   SectionElement:
43       Field | Group
44   ;
45   Field:
46       type=FieldType ('(' options += Option (',' options +=
             Option)* ')')? name=ID ';'
47   ;
48   Group:
```

```
49      'group' ('(' options += Option (',' options += Option)*
            ')')? name=ID '{'
50          elements += Field*
51      '}'
52  ;
53  Relation:
54      'relation' '(' targets += RelationTarget (',' targets
           += RelationTarget)* (',' reloptions += Option (','
           reloptions += Option)*)? ')' relname=ID ';'
55  ;
56  RelationTarget:
57      RelationReferenceTarget | LiteralClass
58  ;
59  LiteralClass:
60      className=STRING
61  ;
62  RelationReferenceTarget:
63      target=[ProcessOrEntity] (tail=RelTailRef)?
64  ;
65  ProcessOrEntity:
66      Process | Entity
67  ;
68  RelTailRef:
69      ('.' element=[SectionOrEntity] (tail=RelTailRef2)?)
70  ;
71  SectionOrEntity:
72      Section | Entity
73  ;
74  RelTailRef2:
75      '.' element=[Group]
76  ;
77  RelElementRef:
78      RelationTarget | RelTailRef | RelTailRef2
```

```
79  ;
80  Option:
81      name=ID '=' values += STRING (',' values += STRING)*
82  ;
83  State:
84      'state' name=ID '{'
85          (elements += StateElement)*
86      '}'
87  ;
88  StateElement:
89      Transition | StateRole
90  ;
91  Transition:
92      'transition' '(' options += Option (',' options +=
           Option)* ')' state=[State] ';'
93  ;
94  StateRole:
95      'permissions' role=[Role] ('from' 'transition'?
           requiredTransitions += RequiredTransition (','
           requiredTransitions += RequiredTransition)* )? '{'
96          (permissions += Permission)*
97      '}'
98  ;
99  RequiredTransition:
100     from=[State] '->' to=[State]
101 ;
102 Permission:
103     PermissionAll | PermissionWithTarget
104 ;
105 PermissionAll:
106     type=PermType ALL ';'
107 ;
108 PermissionWithTarget:
```

```
109     type=PermType targets += PermissionTarget (',' targets
            += PermissionTarget)* ';'
110 ;
111 PermissionTarget:
112     section=[Section] (tail=TailRef)?
113 ;
114 TailRef:
115     '.' element=[SectionElement] (tail=TailRef)?
116 ;
117 ElementRef:
118     PermissionTarget | TailRef
119 ;
120 terminal ALL:
121     'all'
122 ;
123 enum FieldType:
124     FULLNAME = 'fullName' |
125     ADDRESS = 'address' |
126     IDENTITYDOCUMENT = 'identityDocument' |
127     EMAIL = 'email' |
128     CURRENCY = 'currency' |
129     BIRTHDATE = 'birthdate' |
130     DATE = 'date' |
131     PHONE = 'phone' |
132     STRING_T = 'string' |
133     INTEGER_T = 'integer' |
134     DECIMAL_T = 'decimal' |
135     FLOAT = 'float' |
136     CHOICE = 'choice' |
137     FILE = 'file' |
138     BOOL = 'boolean' |
139     TEXT = 'text' |
140     URL = 'url'
```

```
141 ;
142 enum PermType:
143     EDITABLE = 'editable' |
144     VIEWABLE = 'viewable' |
145     DELETABLE = 'deletable'
146 ;
```

APÉNDICE B

RESTRICCIONES DE ADMINDSL CON OCL

En este apartado se expone las restricciones del lenguaje AdminDSL que no han podido ser cubiertas a través de la gramática utilizando el lenguaje de especificación OCL para su representación.

B.1. Contexto de *Application*

- Debe haber al menos una sentencia *Site* definida pero no más de una.

```
1  context Application
2      inv NoMoreThanOneSite('There must be only one "Site
           " statement'):
3          Site.allInstances()->size() = 1
```

B.2. Contexto de *Role*

- No pueden existir dos elementos *Role* con el mismo nombre.

```
1  context Role
2      inv DuplicatedName('Duplicated "Role" name'):
```

```
3         Role.allInstances()->forAll(r : Role | r <>
             self implies r.name <> self.name)
```

B.3. Contexto de *Process*

- No pueden existir dos elementos *Process* con el mismo nombre.

```
1 context Process
2     inv DuplicatedName('Duplicated "Process" name'):
3         Process.allInstances()->forAll(p : Process | p
             <> self implies p.name <> self.name)
```

- El nombre del proceso tiene que empezar por una letra y sólo puede contener letras y números.

```
1 context Process
2     inv InvalidName('Invalid "Process" name'):
3         self.name.matches('[a-zA-Z]+[0-9]*[a-zA-Z]*')
```

- Un elemento *Process* debe contener al menos un elemento *Section*.

```
1 context Process
2     inv NoSectionFound('No section found. There must be
          at least one section defined'):
3         self.elements->select(e : Section | e.
             oclIsTypeOf(Section))->size() > 0
```

- Un elemento *Process* debe contener al menos un elemento *State* de nombre *Initial*.

```
1 context Process
2     inv NoInitialFound('State "initial" not found'):
```

```
3         self.elements->select(e : State | e.oclIsTypeOf
              (State) and e.name = 'initial')->size() = 1
```

B.4. Contexto de *Section*

- No pueden existir dos elementos *Section* con el mismo nombre dentro de un mismo elemento *Process*.

```
1  context Section
2      inv DuplicatedName('Duplicated "Section" name: "' +
           self.name + '"'):
3          Section.allInstances()->forAll(s : Section | s
               <> self implies (s.name <> self.name or s.
               oclContainer() <> self.oclContainer()))
```

- Un elemento *Section* debe contener al menos un elemento ya sea de tipo *Field* o *Group*.

```
1  context Section
2      inv EmptySection('A Section cannot be empty. Define
           at least one element: Field or Group'):
3          self.elements->size() > 0
```

B.5. Contexto de *Group*

- No pueden existir dos elementos *Group* con el mismo nombre dentro de un mismo elemento *Section*.

```
1  context Group
2      inv DuplicatedName('Duplicated "Group" name: "' +
          self.name + '"'):
```

```
3       Group.allInstances()->forAll(g : Group | g <>
          self implies (g.name <> self.name or g.
          oclContainer() <> self.oclContainer()))
```

- Un elemento *Group* debe contener al menos un elemento de tipo *Field*.

```
1   context Group
2     inv EmptySection('A Group cannot be empty. Define
        at least one Field'):
3       self.elements->size() > 0
```

B.6. Contexto de *Field*

- No pueden existir dos elementos *Field* con el mismo nombre dentro de un mismo elemento contenedor.

```
1   context Field
2     inv DuplicatedName('Duplicated "Field" name: "' +
        self.name + '"'):
3       Field.allInstances()->forAll(f : Field | f <>
          self implies (f.name <> self.name or f.
          oclContainer() <> self.oclContainer()))
```

- Los elementos *Field* de tipo “choice” tienen que tener una opción “values”, “table” o “process_table” de forma obligatoria y exclusiva.

```
1   context Field
2     inv InvalidChoice('Invalid "choice" field. It must
        have a "values", "table" or "process_table"
        option defined exclusively'):
3       self.type = FieldType::CHOICE implies (
```

```
4             self.options->select(o : Option | o.name =
                 'values' or o.name = 'table' or o.name='
                 process_table')->size() = 1
5         )
```

- La opción “chain” en un elemento *Field* de tipo “choice” sólo es válida junto a la opción “table” o “process_table”.

```
1 context Field
2     inv InvalidChainChoice('Invalid "chain" option in a
         "choice" field. There is no "table" or "
        process_table" option defined and it is required
        '):
3         (self.type = FieldType::CHOICE and self.options
            ->exists(o : Option | o.name = 'chain'))
4         implies (
5             self.options->exists(o: Option | o.name = '
                 table' or o.name = 'process_table')
6         )
```

- La opción “remote_chain” en un elemento *Field* de tipo “choice” sólo es válida junto a la opción “chain”.

```
1 context Field
2     inv InvalidRemoteChain('Invalid "remote_chain"
        option in a "choice" field without "chain"
        option'):
3         (self.type = FieldType::CHOICE
4         and self.options->exists(o : Option | o.name =
            'remote_chain'))
5         implies (
6             self.options->exists(o: Option | o.name = '
                 chain')
```

```
7         )
```

- La opción “multiple” en un elemento *Field* de tipo “choice” no es válida junto a la opción “values”.

```
1 context Field
2     inv InvalidMultiple('Invalid "multiple" option in a
          "choice" field with "values" option defined'):
3         (self.type = FieldType::CHOICE
4         and self.options->exists(o : Option | o.name =
              'multiple'))
5         implies (
6             self.options->exists(o: Option | o.name = '
                  table' or o.name = 'process_table')
7         )
```

B.7. Contexto de *State*

- No pueden existir dos elementos *State* con el mismo nombre dentro de un mismo elemento *Process*.

```
1 context State
2     inv DuplicatedName('Duplicated "State" name: "' +
        self.name + '"'):
3         State.allInstances()->forAll(s : State | s <>
              self implies (s.name <> self.name or s.
              oclContainer() <> self.oclContainer()))
```

- El elemento *State* de nombre “initial” debe contener al menos una transición con la opción “decision_by”.

```
1 context State
```

```
2     inv InitialWithoutTransitionByDecision('"initial"
         state must have at least one transition with a "
         decision_by" option'):
3         self.name = 'initial'
4         implies self.elements->exists(
5             e : Transition | e.oclIsTypeOf(Transition)
                 and
6             e.options->exists(o : Option | o.name = '
                 decision_by')
7         )
```

B.8. Contexto de *Transition*

- Elementos *Transition* en el *State* "initial" deben tener al menos la opción "decision_by".

```
1 context Transition
2     inv AuthomaticTransitionInInitial('Transitions in "
         initial" state require "decision_by" option'):
3         self.State.name = 'initial'
4         implies self.options->exists(o : Option | o.
             name = 'decision_by')
```

- No pueden existir dos elementos *Transition* iguales contenidos en un mismo elemento *State*, es decir, con las mismas opciones y valores.

```
1 context Transition
2     inv DuplicatedTransition('Duplicated transitions'):
3         Transition.allInstances()->forAll(
4             t : Transition | self <> t implies (self.
                 state <> t.state or not(self.options->
                 forAll(
```

```
5                   o1 : Option | t.options->exists(
6                       o2: Option | o1.name = o2.name and
                            o1.values = o2.values)
7                   )
8                   and t.options->forAll(
9                       o1 : Option | self.options->exists(
                            o2: Option | o1.name = o2.name
                            and o1.values = o2.values))
10                  )
11                  or self.State <> t.State
12              )
13          )
```

B.9. Contexto de *StateRole*

- No se pueden definir elementos *StateRole* que contengan elementos del tipo *RequiredTransitions*.

```
1   context StateRole
2       inv InvalidStateRoleInitial('Permissions with
            required transitions cannot be defined in the
            initial state'):
3            self.State.name = 'initial' implies
4            self.requiredTransitions->size() = 0
```

B.10. Contexto de *PermissionWithTarget*

- No puede existir un elemento de tipo *PermissionWithTarget* que sea del tipo “deletable”.

```
1   context PermissionWithTarget
```

```
2       inv InvalidPermissionDeletable('Invalid permission
           type "deletable". It must be followed by "all"
           instead process elements"'):
3           self.type <> PermType::DELETABLE
```

B.11. Contexto de *Permission*

- No puede existir un elemento de tipo *Permission* del tipo “deletable” definido dentro del elemento *State* “initial”.

```
1   context Permission
2       inv InvalidPermissionDeletableInitial('Invalid
           permission type "deletable". It cannot be in the
            "initial" state'):
3           self.type = PermType::DELETABLE implies self.
               StateRole.State.name <> 'initial'
```

B.12. Contexto de *Option*

- No puede existir un elemento de tipo *Option* con el mismo nombre dentro de un mismo elemento contenedor.

```
1   context Option
2       inv DuplicatedName('Duplicated "Option" name: "' +
           self.name + '"'):
3           Option.allInstances()->forAll(o : Option| o <>
               self implies (o.name <> self.name or o.
               oclContainer() <> self.oclContainer()))
```

- No debería de haber más de un valor para cualquier elemento *Option* que no sea “values”.

```
context Option
    inv MultipleValues('This option have multiple
        values but it will only be taken the first one')
        :
        self.name <> 'values' implies self.values->size
            () = 1
```

- La opción “type” sólo puede tener los siguientes valores: “OneToOne”, “ManyToOne” o “ManyToMany”.

```
context Option
    inv InvalidRelationTypeValue('Invalid type option
        with value: "' + self.values->first() + '". The
        available values are "OneToOne", "ManyToOne" or
        "ManyToMany"'):
        (self.Relation <> null
        and self.name = 'type')
        implies (
            self.values->first() = 'OneToOne'
            or self.values->first() = 'ManyToOne'
            or self.values->first() = 'ManyToMany'
        )
```

- La opción “count” sólo puede tener un valor numérico entre comillas dobles o el valor “dynamic”.

```
context Option
    inv InvalidCountValue('Invalid "count" option with
        value: "' + self.values->first() + '". The
        available values are "dynamic" or a quoted
        integer (example: "1", "2")'):
        (aPDSL::Group.allInstances()->exists(g : Group
```

```
            | g.options->includes(self))
4       and self.name = 'count')
5       implies (
6           self.values->first() = 'dynamic'
7           or self.values->first().matches('[0-9]+')
8       )
```

- La opción “table” sólo puede hacer referencia a un elemento *Entity* que se encuentre en el mismo ámbito que el elemento contenedor, ya sea, dentro del mismo elemento *Process* o a nivel global.

```
1   context Option
2       inv InvalidChoiceEntityValue('Invalid "table"
           option with value: "' + self.values->first() + '
           ". It must reference an existing entity in the
           same scope'):
3           self.name = 'table' implies (
4           (self.oclContainer().oclAsType(aPDSL::Field).
               Section <> null and
5           self.oclContainer().oclAsType(aPDSL::Field).
               Section.Process.elements->exists(
6           e : Entity | e.oclIsTypeOf(Entity) and e.name =
                self.values->first()
7           ))
8           or
9           (self.oclContainer().oclAsType(aPDSL::Field).
               Group <> null and
10          self.oclContainer().oclAsType(aPDSL::Field).
               Group.Section.Process.elements->exists(
11          e : Entity | e.oclIsTypeOf(Entity) and e.name =
                self.values->first()
12          ))
13          or Application.allInstances()->forAll(a :
```

```
            Application | a.elements->exists(e : Entity
            | e.oclIsTypeOf(Entity) and e.name = self.
            values->first()))
14        )
```

- La opción “process_table” debe hacer referencia a un elemento *Process* existente.

```
1 context Option
2     inv InvalidChoiceProcessValue('Invalid "
        process_table" option with value: "' + self.
        values->first() + '". It must reference an
        existing process'):
3         self.name = 'process_table' implies (
4             Process.allInstances()->exists(p : Process
                | p.name = self.values->first())
5         )
```

- La opción “decision_by” debe hacer referencia a un elemento *Role* existente.

```
1 context Option
2     inv InvalidDecisionBy('Invalid "decision_by" option
         with value: "' + self.values->first() + '". It
        must reference an existing role'):
3         self.name = 'decision_by' implies (
4             Role.allInstances()->exists(r : Role | r.
                name = self.values->first())
5         )
```

- Las opciones “blank”, “unicode_name” y “unique” sólo pueden tener el valor “True” o “False”.

```
1 context Option
2     inv InvalidBooleanValue('Invalid boolean type
        option with value: "' + self.values->first() + '
        ". Available values are only "True" or "False"')
        :
3        self.name = 'blank' or self.name = '
            unicode_name' or self.name = 'unique'
            implies (
4             self.values->first() = 'True' or self.
                values->first() = 'False'
5        )
```

- La opción "max" sólo puede ser definida en elementos *Transition* que se encuentren en el elemento *State* "initial".

```
1 context Option
2     inv MaxOnlyForInitial('Option "max" can be only
        defined in state "initial"'):
3        (self.name = 'max' and self.Transition <> null)
             implies self.Transition.State.name = '
            initial'
```

APÉNDICE C

MANUAL DE INSTALACIÓN E IMPLANTACIÓN

En este apéndice se explican cómo instalar y preparar el entorno y las herramientas con el fin de poder hacer uso del lenguaje AdminDSL y poder generar una aplicación web Django a partir de esta, así como los requisitos necesarios para poderse llevar a cabo esta instalación. Además se especifican los pasos previos requeridos para la puesta en marcha de dicha aplicación generada.

C.1. Requisitos de entorno

Requisitos para el desarrollador

- Distribución GNU/Linux basada en Debian: Debian, Ubuntu, Guadalinex, etc.
- Entorno Eclipse Luna para la instalación mostrada en el apartado C.2.2.
- ANTLR [11], Xtext, Epsilon y OCL Examples and Editors en Eclipse. Véase sección C.2.2 de este apéndice.

Requisitos del servidor para el alojamiento de las aplicaciones web generadas

- Se recomienda un sistema operativo Linux/Unix. Pero como toda aplicación web Django, también puede ser ejecutada bajo sistemas operativos Windows o Mac OS X.
- Python.
- Dependiendo de la base de datos que se quiera utilizar:
  - Paquete *psycopg2* para PostgreSQL.
  - Controlador de base de datos como *mysqlclient* para MySQL.
  - Paquete *cx_Oracle* para el uso de Oracle.
- Pip.
- Virtualenv.
- LDAP. En concreto el paquete *libldap2-dev* para sistemas Linux/Unix.

Se ofrece una guía de instalación donde se localizan todas estas dependencias en la página web oficial de Django [3]:

`https://docs.djangoproject.com/en/1.8/intro/install/`

Enlace visitado por última vez el 30/08/2015.

C.2. Entorno para AdminDSL y el generador de aplicaciones Django

La instalación de este entorno se puede realizar de dos formas diferentes: instalando una herramienta Eclipse que ya contiene los plugins requeridos o instalando los plugins ofrecidos por un sitio de actualización en un entorno Eclipse que previamente había sido instalado.

C.2.1. Herramienta Eclipse Luna

Ofrecer una herramienta Eclipse Luna preparada, con las dependencias requeridas instaladas, para hacer uso del lenguaje y el generador es una vía útil para aquellas personas que no se encuentran familiarizadas con esta herramienta y entorno.

La herramienta se ofrece como un directorio comprimido en un fichero ZIP que será aportado a través de cualquier plataforma de descarga que haya disponible o a través de un repositorio Git ya sea en el *redmine* del departamento de ingeniería informática o de forma pública en el sitio web de GitHub [27].

Para la instalación de esta herramienta simplemente debe de extraerse el contenido del fichero ZIP en cualquier directorio deseado donde se alojará la herramienta. Hecho esto se puede comenzar a usar arrancando el ejecutable denominado “eclipse” que se encuentra en ese directorio.

C.2.2. Plugins a través de Eclipse

Para aquellos desarrolladores que desean integrar el lenguaje y el generador a su herramienta Eclipse (recordar que se requiere la versión Luna) ya instalada, se aporta un sitio de actualización («update site» en Eclipse) a partir del cual se puede instalar los plugins y dependencias requeridas para el funcionamiento correcto de este trabajo.

En primer lugar se requiere la instalación de ciertas dependencias para el uso del plugin desarrollado por este trabajo:

ANTLR

Para la instalación de ANTLR sobre el entorno Eclipse simplemente abrir la herramienta Eclipse y seguir los pasos que se muestran a continuación:

1. Acceder mediante el menú contextual superior a “Help” → “Eclipse

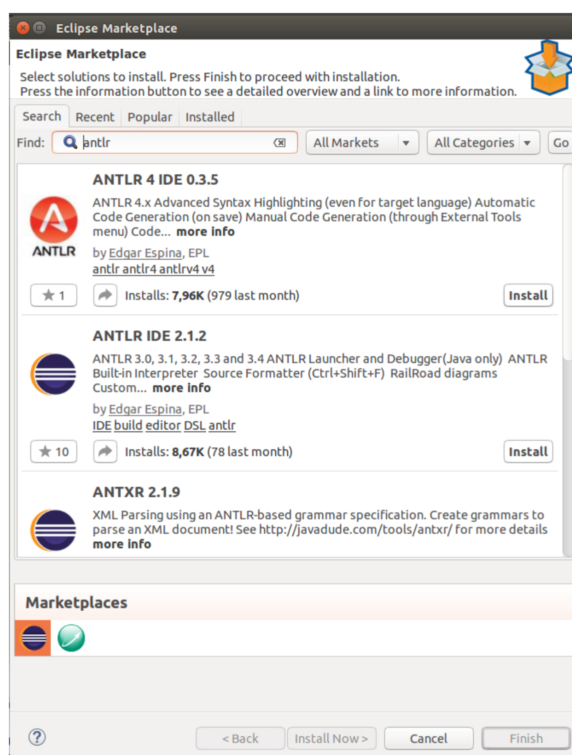


Figura C.1: Marketplace en Eclipse para la instalación de ANTLR IDE 4.

Marketplace”.

2. Escribir en la barra de búsqueda “ANTLR” sin las comillas.

3. Seleccionar “ANTLR 4 IDE” y pulsar sobre el botón “Install”. Véase figura C.1.

4. Confirmar la instalación pulsando sobre “Confirm >” y luego pulsar “Finish”.

5. Si se muestra un aviso de instalación pulsar en “Ok” para continuar.

6. Listo. ANTLR comenzará a instalarse y solicitará el reinicio de la aplicación Eclipse al terminar.

Xtext

Xtext se puede instalar en Eclipse desde su sitio de actualización oficial. Para ello, seguir los siguientes pasos:

1. Acceder mediante el menú contextual superior a "Help" $\rightarrow$ "Install New Software...".
2. A continuación pegar sobre la barra de "Work with:" la siguiente dirección:

   `http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases/`

   y pulsar la tecla "Enter".
3. Seleccionar Xtext SDK de la categoría Xtext mostrada en la lista de paquetes software y pulsar sobre el botón "Next" hasta pulsar el botón "Finish".
4. Si se muestra un aviso de instalación pulsar en "Ok" para continuar.
5. Listo. Xtext comenzará a instalarse y solicitará el reinicio de la aplicación Eclipse al terminar.

Epsilon

Epsilon, al igual que Xtext, se puede instalar en Eclipse desde su sitio de actualización oficial. Para ello, seguir los siguientes pasos:

1. Acceder mediante el menú contextual superior a "Help" $\rightarrow$ "Install New Software...".
2. A continuación pegar sobre la barra de "Work with:" la siguiente dirección:

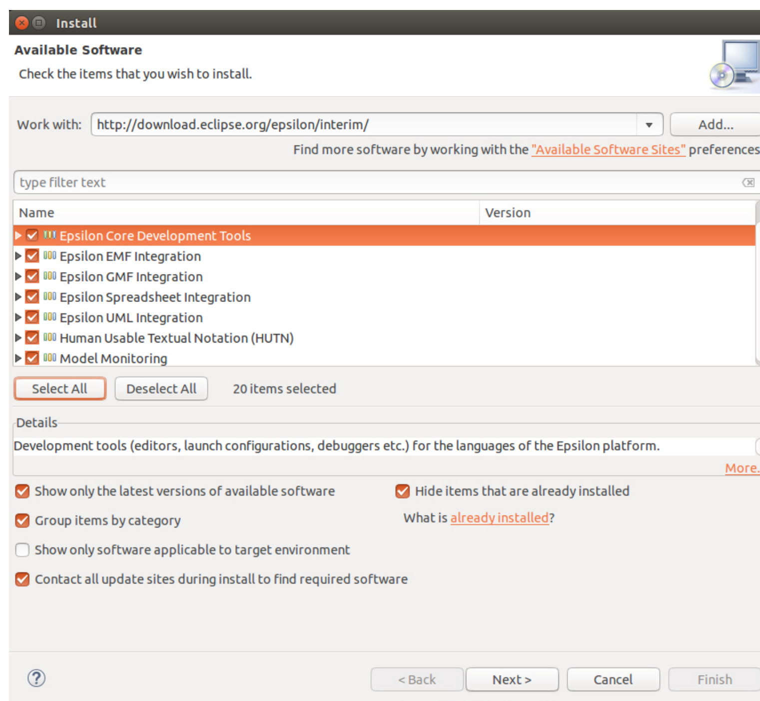


Figura C.2: Ventana de instalación en Eclipse para Epsilon.

`http://download.eclipse.org/epsilon/interim/`

y pulsar la tecla "Enter".

3. Seleccionar todas las categorías mostradas en la lista de paquetes software y pulsar sobre el botón "Next" hasta pulsar el botón "Finish". Véase figura C.2

4. Si se muestra un aviso de instalación pulsar en "Ok" para continuar.

5. Listo. Epsilon comenzará a instalarse y solicitará el reinicio de la aplicación Eclipse al terminar.

OCL Examples and Editors

OCL Examples and Editors se instala de la misma forma que Xtext o Epsilon, sin embargo se usa la dirección del sitio de actualización propio de Eclipse que varía según la versión de la herramienta que se está utilizando. Para llevar a cabo la instalación sea cual sea la versión, seguir los siguientes pasos:

1. Acceder mediante el menú contextual superior a “Help” → “Install New Software...”.

2. A continuación seleccionar sobre la barra de “Work with:” la opción –All Available Sites– y pulsar la tecla “Enter”. Es posible que esta operación tarde unos segundos.

3. Seleccionar “OCL Examples and Editors” de la categoría “Modelling” y pulsar sobre el botón “Next” hasta que aparezca el botón “Finish” y pulsarlo. Véase figura C.3.

4. Listo. OCL Examples and Editors comenzará a instalarse y solicitará el reinicio de la aplicación Eclipse al terminar.

AdminDSL y generador

El sitio de actualización para la instalación del entorno para AdminDSL y el generador se ofrece como un directorio comprimido en un fichero ZIP, aportado de la misma forma que la herramienta eclipse del apartado anterior. Se deberá extraer el contenido y seguir los pasos que aparecen a continuación:

1. Acceder a través del menú contextual al menú de instalación de nuevo software accediendo a “Help” → “Install new software...”.

2. Añadir un nuevo sitio de actualización pulsando sobre el botón “Add..”.

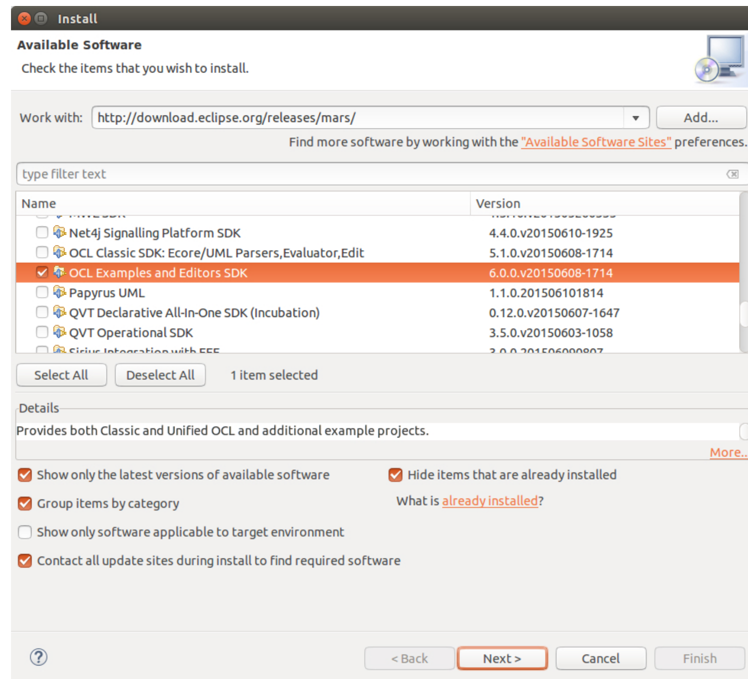


Figura C.3: Ventana de instalación para OCL Examples and Editors.

3. Seleccionar ahora “Local...” y seleccionar el directorio extraído previamente del sitio de actualización.

4. Marcar la categoría AdminDSL la cual puede desplegarse en los diferentes plugins asociados al editor del lenguaje y al generador de código para Django. Véase figura C.4

5. Seguir las instrucciones y continuar con las instalación pulsando en los botones “Next” y aceptando el acuerdo de licencia cuando se solicite.

6. Una vez se comience a instalar el software solicitará una nueva confirmación dado que se tratará de un software no registrado por la fundación Eclipse y se mostrará una advertencia al respecto.

7. Por último, se solicitará el reinicio de la herramienta, lo cual debe aprobarse, para poder comenzar a hacer uso del editor para AdminDSL

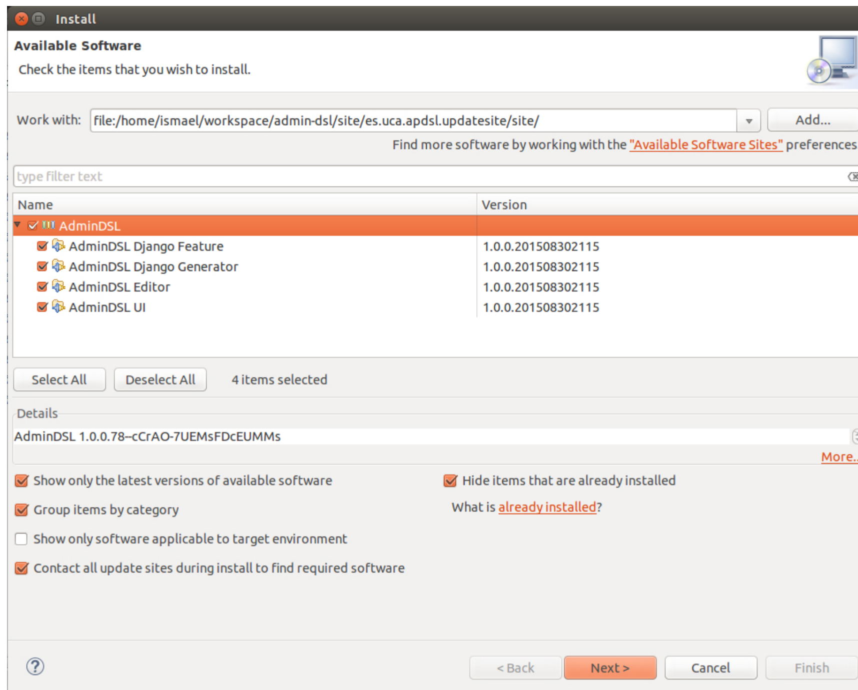


Figura C.4: Ventana de instalación para AdminDSL.

y el generador de aplicaciones Django.

C.3. Aplicación web generada

Una vez haya sido generada la aplicación web Django, utilizando la extensión de este trabajo, se debe instalar ciertas dependencias y ejecutar una serie de órdenes necesarias para la puesta en marcha de la aplicación siguiendo los pasos a continuación:

- Se abre una terminal situada en el directorio que hayamos seleccionado (y nombrado a través de la sentencia *site*) para la generación de la aplicación.

- Se ejecuta el fichero “install_dependencies.sh” como sigue:

```
./install_dependencies.sh
```

- Se solicitará la contraseña de superusuario que deberá ser aportada para la correcta instalación de las dependencias.
- Se solicitará un nombre y una contraseña para el acceso a la administración de la aplicación Django.
- Se instalarán todas las dependencias mostrando salidas que indican el progreso de la misma y los errores (en caso de que los hubiese).

Una vez instaladas las dependencias se puede comenzar a la ejecución de la aplicación Django desde la terminal mediante las siguientes órdenes:

```
source ENV/bin/activate
```

Para la activación del entorno virtual donde se han instalado todas las dependencias.

```
python manage.py runserver
```

Para la ejecución de la aplicación web.

APÉNDICE D

MANUAL DE USUARIO

D.1. Cómo empezar: abriendo el editor

Para comenzar a usar el editor de AdminDSL se deben realizar los siguientes pasos:

1. Abrir la herramienta Eclipse.
2. Usar el menú contextual superior para crear un nuevo proyecto:
   a) Pulsar sobre la opción “File” → “New” → “Project...”.
   b) En el menú de selección que se muestra se selecciona el tipo “General” → “Project” y se pulsa “Next”. Véase figura D.1.
   c) Se escribe ahora el nombre deseado para proyecto y se pulsa sobre “Finish”. Véase figura D.2.
   d) En el explorador de paquetes (*Package Explorer*) aparecerá el proyecto creado. Si no se muestra el explorador de paquetes acceder a “Window” → “Show View” → “Package Explorer” en el menú contextual superior.

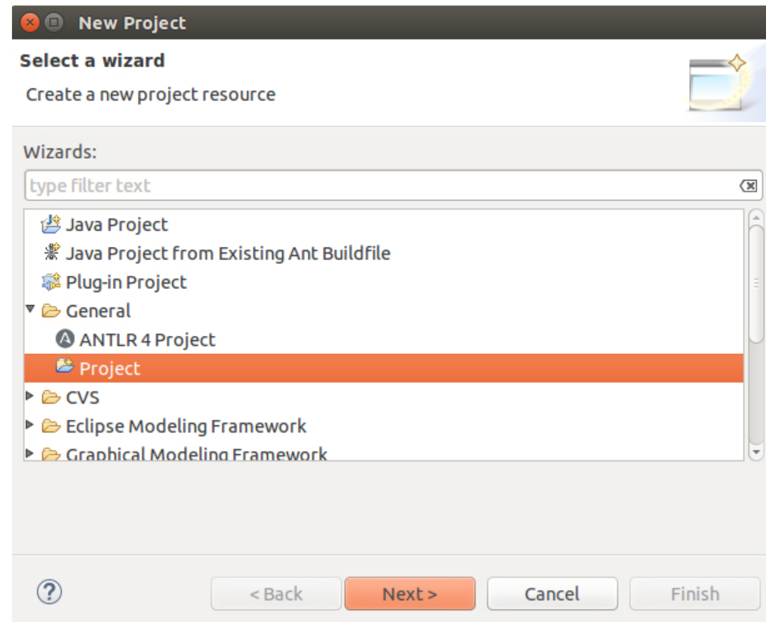


Figura D.1: Ventana de selección de tipo de proyecto.

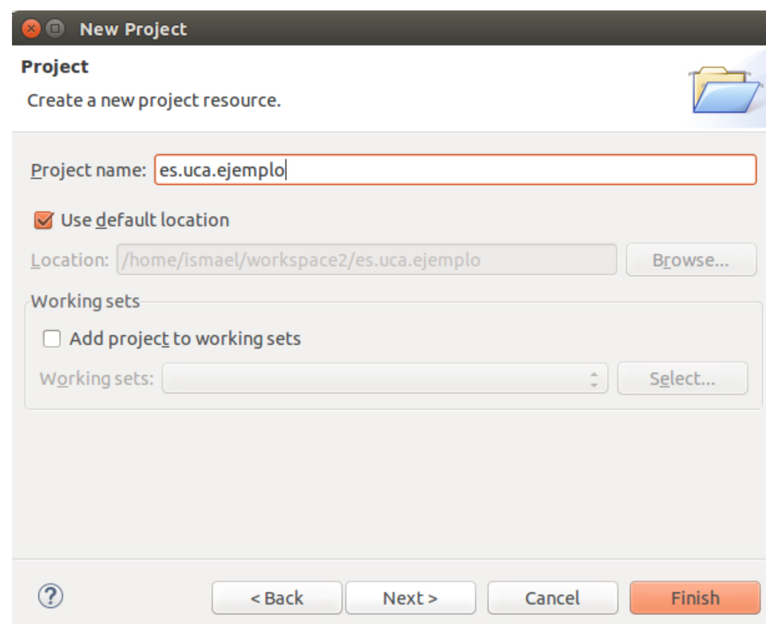


Figura D.2: Ventana para especificar el nombre del proyecto.

3. Creado el proyecto, añadimos un nuevo fichero donde escribir el código en AdminDSL. Para ello:

    a) Acceder al menú contextual del proyecto creado haciendo click derecho sobre el mismo en el explorador de paquetes.

    b) Seleccionar "New" → "Other...".

    c) En el menú de selección que se muestra se selecciona el tipo "General" → "File" y se pulsa "Next".

    d) Se escribe ahora el nombre deseado para el fichero añadiendo la terminación ".apdsl" (esto último es importante para poder utilizar el editor AdminDSL) y pulsar sobre "Finish". Véase figura D.3 de la página 260.

    e) Saltará una ventana de confirmación preguntando si se desea añadir *Xtext Nature* al proyecto donde hemos creado dicho fichero. Pulsamos "Yes" (en otro caso no se podrá utilizar el editor AdminDSL).

    f) El fichero creado aparecerá ahora dentro del proyecto que se había creado y además se abrirá automáticamente el editor (el fichero se podrá abrir posteriormente haciendo doble click sobre el mismo).

D.2. Trabajando con AdminDSL

Abierto el fichero con el editor de AdminDSL se puede comenzar a definir procesos administrativos haciendo uso del DSL. A continuación se explica cómo realizar el código para poder generar una aplicación web básica a partir del mismo.

Es importante que el lector conozca previamente y de forma general el lenguaje AdminDSL sirviéndose del apartado 3.3.3, página 73 del capítulo 3

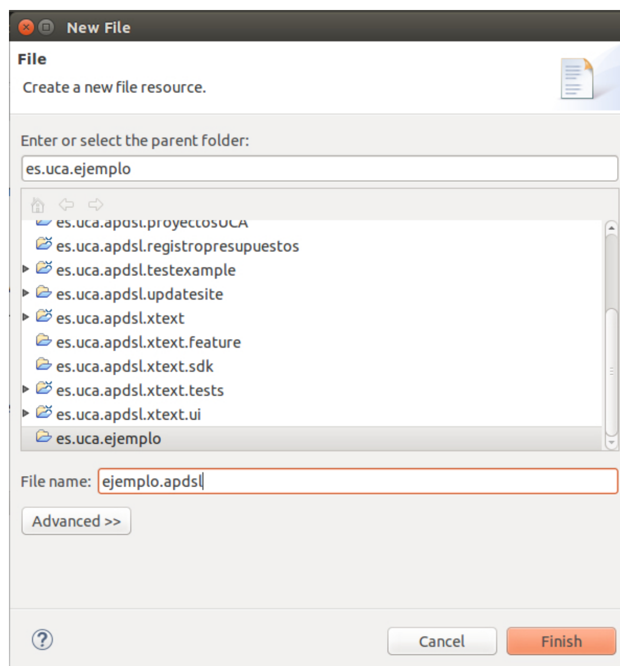


Figura D.3: Ventana para especificar el nombre del fichero y crearlo.

de Análisis. En ese apartado se presentan todos los elementos de AdminDSL que pueden ser utilizados mientras que en el ejemplo que se presentará a continuación sólo se utilizarán los más básicos con el fin de facilitar la comprensión del mismo.

La aplicación que se va a definir en este apartado trata sobre un examen dividido en dos partes, cada una a realizar por estudiantes y revisadas por profesores. El examen comienza en una fecha determinada pero sólo se le mostrará al estudiante inicialmente la primera parte. Una vez pasado el tiempo para la realización de la primera parte, se mostrará la segunda parte y no se podrá editar esta primera. Terminada la segunda parte (ya sea por decisión del estudiante o porque se ha pasado la fecha límite de realización), el examen pasa a revisión, de la cual sólo puede encargarse un usuario profesor. Al profesor se le muestra un campo donde añadir la nota y se le permitirá cerrar el proceso permitiendo a los estudiantes ver qué nota

han sacado.

D.2.1. Definir nombre de la aplicación

Para definir el nombre de la aplicación se debe escribir la siguiente sentencia:

```
site Escuela;
```

En este caso, se ha nombrado la aplicación como “Escuela”.

D.2.2. Definir roles participantes en la aplicación

Se definen los roles de profesor y estudiante de la siguiente forma:

```
1 role Profesor;
2 role Estudiante;
```

D.2.3. Definir un proceso administrativo

En este caso el proceso administrativo va a hacer referencia al examen, el cual se dividía en dos partes, para ello definimos lo siguiente:

Secciones y grupos

```
1 ...
2 process Examen {
3     entity Respuesta3 {
4         string(unicode_name="True") respuesta;
5     }
6     section(label="Datos personales") personal {
7         fullName alumnoNombre;
```

```
8           identityDocument alumnoId;
9           email mail;
10      }
11      section(label="Cuestionario ejemplo") cuestionario {
12          group(label="Primera parte") parte1 {
13              string(label="¿Es esto un ejemplo?", blank="
                    True") c1;
14              choice(label="Selecciona la opción correcta",
                    values="A1","A2","A3", blank="True") c2;
15              choice(label="Selecciona la respuesta correcta
                    ", label="Respuesta3", blank="True") c3;
16          }
17          group(label="Segunda parte") parte2 {
18              currency(label="¿Cuántos euros cuesta?", blank
                    ="True") c4;
19              integer(label="¿2+6?", blank="True") c5;
20          }
21      }
22      section{label="Evaluación del cuestionario") evaluacion
             {
23          float nota;
24      }
25  }
```

Como se puede ver en el código superior, se ha definido un proceso administrativo denominado “Examen” con dos secciones: “personal” y “cuestionario”. La primera se ha colocado para que el estudiante escriba sus datos personales y la segunda es el cuestionario del examen que debe realizar. Como se había mencionado anteriormente, el cuestionario viene dividido en dos partes por lo que se han definido dos grupos: “parte1” y “parte2”. También se ha definido una sección para la evaluación del cuestionario con un campo flotante donde colocar la nota que ha obtenido el estudiante.

Dada la opción «blank="True"» se han definido todos los campos del cuestionario como opcionales, permitiendo entregar un examen en blanco. Además se han colocado etiquetas personalizadas para escribir las cuestiones como tales.

Por último se ha definido una entidad para la respuesta de la cuestión "c3", de forma que se tendrá que seleccionar una de las instancias existente de la entidad "Respuesta3". Esto permitiría al administrador de la aplicación poder añadir nuevas respuestas.

Permisos, estados y transiciones

En primer lugar, queremos establecer una fecha de comienzo, es decir, una fecha a partir de la cual los estudiantes podrán comenzar el examen. Para ello definimos el estado "initial", el cual hace referencia a la creación del proceso, donde se colocan permisos para que el estudiante pueda ver solamente la primera parte del cuestionario junto a la sección de datos personales.

```
1  ...
2  process Examen {
3      //Secciones y grupos
4      ...
5      state initial {
6          permissions Estudiante {
7              editable personal, cuestionario.parte1;
8          }
9          transition(max="1", decision_by="Estudiante",
               start_date="2015/09/23-16:00:00", end_date
               ="2015/09/23-20:00:00") parte1;
10     }
11     state parte1 {
12         permissions Profesor {
```

```
13              viewable all;
14          }
15          permissions Estudiante from initial->parte1 {
16              editable personal, cuestionario.parte1;
17          }
18      }
19  }
```

Como se puede observar en el código, se ha añadido un estado nuevo "parte1" así como una transición en el estado "initial" hacia este estado. Esto significa que el usuario podrá comenzar un examen entre las fechas señaladas en las opciones "start_date" y "end_date" y que, una vez comenzado, pasará al estado "parte1" donde se mantendrían los mismos permisos mencionados anteriormente para el estudiante que ha participado en dicha transición (es decir, sólo el que lo ha comenzado para evitar que un estudiante pueda ver o editar el examen de otro estudiante) y nuevos permisos para los profesores que podrán ver el examen completamente sin poder editar nada.

A continuación se debe establecer la fecha límite para la primera parte del examen y que comience la segunda parte de forma automática. Para ello se define un nuevo estado donde el estudiante podrá ver y editar la segunda parte con la transición automática oportuna en el estado "parte1".

```
1  ...
2      ...
3      state parte1 {
4          permissions Profesor {
5              viewable all;
6          }
7          permissions Estudiante from initial->parte1 {
8              editable personal, cuestionario.parte1;
9          }
```

```
10          transition(start_date="2015/09/23-18:00:00") parte2
                ;
11      }
12      state parte2 {
13          permissions Profesor {
14              viewable all;
15          }
16          permissions Estudiante from initial->parte1 {
17              viewable cuestionario.parte1;
18              editable personal, cuestionario.parte2;
19          }
20      }
21      ...
22  ...
```

Ya sólo faltaría definir la finalización del examen y evaluación del mismo.

```
1   ...
2       ...
3       state parte2 {
4           permissions Profesor {
5               viewable all;
6           }
7           permissions Estudiante from initial->parte1 {
8               viewable cuestionario.parte1;
9               editable personal, cuestionario.parte2;
10          }
11          transition(decision_by="Estudiante", end_date
                ="2015/09/23/20:00:00) evaluacion;
12          transition(start_date="2015/09/23/20:00:00")
                evaluacion;
13      }
14      state evaluacion {
```

```
15          permissions Profesor {
16              viewable all;
17              editable evaluacion;
18          }
19          permissions Estudiante from initial->parte1 {
20              viewable personal, cuestionario;
21          }
22          transition(decision_by="Profesor") finalizado;
23      }
24      state finalizado {
25          permissions Profesor {
26              viewable all;
27          }
28          permissions Estudiante {
29              viewable evaluacion;
30          }
31          permissions Estudiante from initial->parte1 {
32              viewable all;
33          }
34      }
35      ...
36  ...
```

Para el estado “parte2” se han definido dos transiciones una que permita al estudiante terminar el cuestionario y pasar el examen a evaluación, y otra que se realiza automáticamente cuando se termina el límite de tiempo, es decir, cuando se supera la fecha dada.

Además de los estados mencionados, se ha añadido el estado de “finalizado” para que los estudiantes puedan ver las notas de los exámenes realizados o revisar su propio examen.

Terminada la definición de estados se tendría el código AdminDSL listo para poder generar a partir del mismo una aplicación web, lo cual se verá

en el siguiente apartado del manual.

```
1  site Escuela:
2
3  role Profesor;
4  role Estudiante;
5
6  process Examen {
7      entity Respuesta3 {
8          string(unicode_name="True") respuesta;
9      }
10     section(label="Datos personales") personal {
11         fullName alumnoNombre;
12         identityDocument alumnoId;
13         email mail;
14     }
15     section(label="Cuestionario ejemplo") cuestionario {
16         group(label="Primera parte") parte1 {
17             string(label="¿Es esto un ejemplo?", blank="
                   True") c1;
18             choice(label="Selecciona la opción correcta",
                   values="A1","A2","A3", blank="True") c2;
19             choice(label="Selecciona la respuesta correcta
                   ", label="Respuesta3", blank="True") c3;
20         }
21         group(label="Segunda parte") parte2 {
22             currency(label="¿Cuántos euros cuesta?", blank
                   ="True") c4;
23             integer(label="¿2+6?", blank="True") c5;
24         }
25     }
26     section{label="Evaluación del cuestionario") evaluacion
            {
```

```
27          float nota;
28      }
29      state initial {
30          permissions Estudiante {
31              editable personal, cuestionario.parte1;
32          }
33          transition(max="1", decision_by="Estudiante",
                start_date="2015/09/23-16:00:00", end_date
                ="2015/09/23-20:00:00") parte1;
34      }
35      state parte1 {
36          permissions Profesor {
37              viewable all;
38          }
39          permissions Estudiante from initial->parte1 {
40              editable personal, cuestionario.parte1;
41          }
42          transition(start_date="2015/09/23-18:00:00") parte2
                ;
43      }
44      state parte2 {
45          permissions Profesor {
46              viewable all;
47          }
48          permissions Estudiante from initial->parte1 {
49              viewable cuestionario.parte1;
50              editable personal, cuestionario.parte2;
51          }
52          transition(decision_by="Estudiante", end_date
                ="2015/09/23/20:00:00) evaluacion;
53          transition(start_date="2015/09/23/20:00:00")
                evaluacion;
54      }
```

```
55      state evaluacion {
56          permissions Profesor {
57              viewable all;
58              editable evaluacion;
59          }
60          permissions Estudiante from initial->parte1 {
61              viewable personal, cuestionario;
62          }
63          transition(decision_by="Profesor") finalizado;
64      }
65      state finalizado {
66          permissions Profesor {
67              viewable all;
68          }
69          permissions Estudiante {
70              viewable evaluacion;
71          }
72          permissions Estudiante from initial->parte1 {
73              viewable all;
74          }
75      }
76  }
```

D.3. Generando la aplicación web Django

Preparado el código AdminDSL y guardado en el fichero que habíamos creado en el apartado D.1 se puede generar una aplicación web Django siguiendo los pasos que se muestran a continuación:

1. Se accede al menú contextual del fichero ".apdsl" donde se encuentra el código hacinedo click derecho sobre el mismo. Es importante asegurarse de que el código que se muestra en el editor se ha guardado

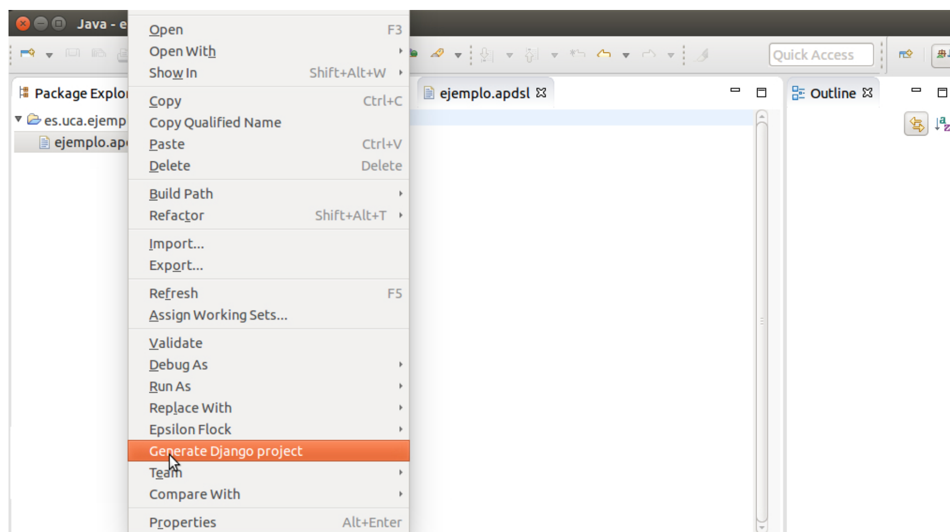


Figura D.4: Ventana para especificar el nombre del proyecto.

antes de la generación.

2. A continuación se selecciona la opción “Generate Django project”. Véase figura D.4.

3. Se mostrará ventana de selección de directorio si es la primera vez que se desea generar el proyecto. En caso de no ser la primera vez, se preguntará si se desea volver a utilizar el último directorio seleccionado previamente.

4. Se selecciona el directorio donde queremos que se almacene la aplicación web y se pulsa “Aceptar”.

5. Pasados unos segundos, se mostrará una ventana de que la aplicación web ha sido generada con éxito y la aplicación se encontrará en el directorio escogido con el nombre que se haya definido en la sentencia “site”.

Para la puesta en marcha de la aplicación web Django visitar el apartado C.3 en la página 255 de este documento.

APÉNDICE E
MANUAL DEL DESARROLLADOR

E.1. Crear una feature en Eclipse

Para ello, simplemente se crea un nuevo proyecto del tipo *Feature* usando el menú contextual de Eclipse y se añade, si es necesario, un directorio que contenga código fuente que se requiera. A continuación, se selecciona en la pestaña de “Plug-ins”, situada en el fichero “feature.xml”, los plugins que se desean integrar en dicha feature (ejemplo en la figura E.1). Por último, se marca en la pestaña “Build” el directorio dentro de la feature donde se encuentra el código fuente (en caso de que hubiese) para que sea empaquetado y pueda ser utilizado por otros plugins o features que posean a esta como dependencia.

E.2. Crear extensión de menú contextual en Eclipse

Para definir una nueva opción en el menú contextual que ofrece el explorador de paquetes de Eclipse se realizan los siguientes pasos:

1. Crear un nuevo plugin de Eclipse: para ello se crea un nuevo proyecto

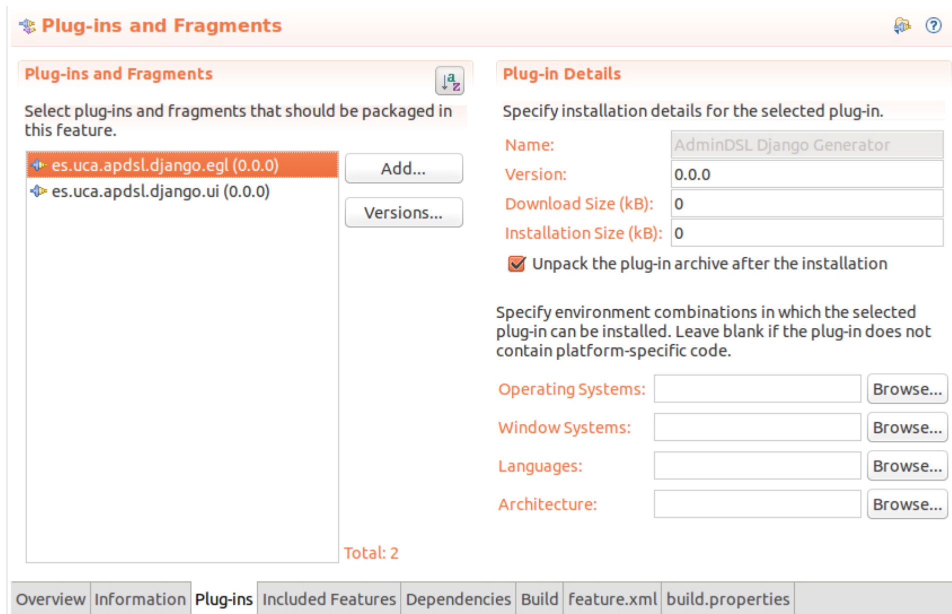


Figura E.1: Añadiendo plugins a la feature

del tipo *Plug-in Project* usando el menú contextual.

2. Se añade en la pestaña de "Extensions" del fichero "plugin.xml" una nueva extensión, seleccionando el tipo *org.eclipse.ui.menus*.

3. Dentro de este menú se añade un *menuContribution* con URI *popup:org.eclipse.jdt.ui.PackageExplorer*. Esto indica que estamos seleccionando el menú contextual que se muestra al hacer click derecho sobre un fichero en la vista del explorador de paquetes.

4. Ahora se añade dentro de este *menuContribution* una orden o *command* con id "*es.uca.apdsl.plugin.generator*" y con *label* "Generate Django project".

5. Para finalizar con esta extensión, como queremos que esta opción sólo aparezca para ficheros con extensión ".apdsl", se añaden una serie de condiciones que deshabilitan esta opción para casos que no cumplan con esta premisa.

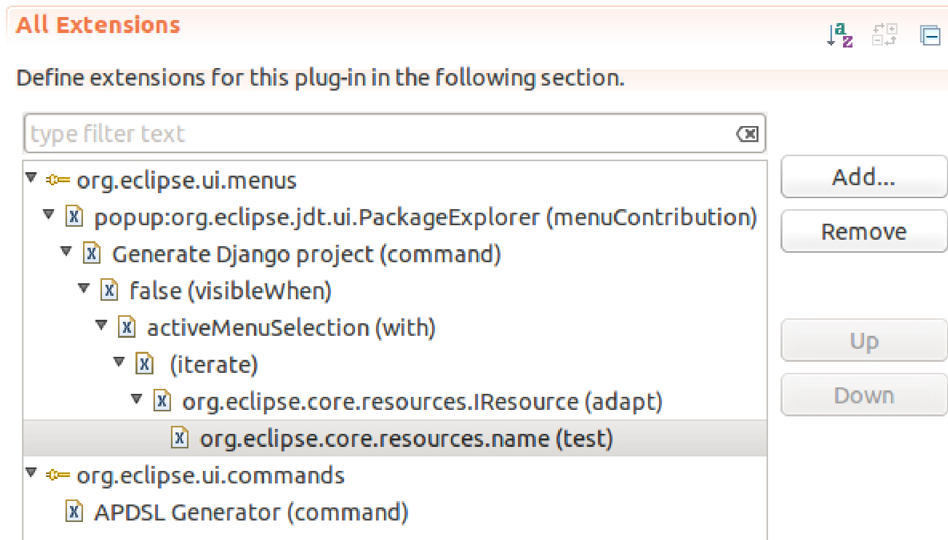


Figura E.2: Estructura de la extensión del menú contextual para Eclipse

6. Una vez definida la extensión, falta definir qué va a hacer la orden que habíamos señalado en el paso 3. Para ello, se añade una nueva extensión del tipo *org.eclipse.ui.commands*.

7. Sobre esta última extensión, se añade una orden o *command* y se coloca como id la misma que la que se colocó para la orden del paso 3 con el fin de que exista esa correspondencia.

8. Por último, se coloca como controlador por defecto el fichero con ruta *es.uca.apdsl.plugin.handler.GeneratorHandler* que será creado posteriormente y que contendrá el código a ejecutar cuando se seleccione dicha opción.

En la figura E.2 se muestra cómo queda la estructura de la extensión del menú contextual junto con su orden.

Controlador *GeneratorHandler*

Para el desarrollo de este controlador se utiliza el lenguaje Java y se define una clase (*GeneratorHandler*) que hereda de *AbstractHandler* que ya

provee Eclipse y simplemente se sobrescribe el método *execute* que esta última clase ofrece.

Dicho esto, el código se divide en dos partes o métodos principales:

- El primero de ellos (*createOutput*) se ocupa de la interfaz que permite seleccionar un directorio de salida (donde se quiere situar la aplicación web generada) o mantener la selección previa.

- El segundo (*runEGL*) se encarga de la transformación del código AdminDSL en la aplicación web Django utilizando la tecnología Epsilon y el metamodelo obtenido a través de Xtext.

E.3. Pruebas con Django + Selenium

En este apartado se explicará cómo se pueden desarrollar nuevas pruebas con Django y Selenium así como que procedimiento debe llevarse a cabo para la ejecución de las mismas.

E.3.1. Desarrollo de pruebas

Preparación del entorno

Para desarrollar una prueba primero debemos ir al directorio “tests” situado en la app que queremos probar. Para las aplicaciones generadas con este trabajo lo ideal sería colocarlo en el directorio “base_admindsl/tests” ya que este contiene la estructura básica y central así como asociaciones con el resto de elementos de las otras app’s. Dentro de este directorio debe haber un fichero vacío denominado “__init__.py” para que sea detectado como un paquete en Django.

En segundo lugar, se deberían importar el fichero “utils.py” situado en el directorio “examples/es.uca.apdsl.testexample/tests” del repositorio de este TFG.

Este fichero contiene una especialización de la clase StaticLiveServer-TestCase propia (clase principal donde se definen y ejecutan las pruebas) para permitir que la base de dato se restablezca con los datos iniciales por cada prueba que se realiza evitando excepciones o errores por problemas en la integridad de la misma.

Además en "utils.py" se pueden encontrar diversos métodos que resultan útiles a nivel general, aunque algunos de ellos están adaptados a la aplicación generada a partir de "testexample.apdsl".

Ahora se crea un nuevo fichero denominado "test_nombredelaprueba.py" donde "nombredelaprueba" sería un nombre a selección del desarrollador que identifique el conjunto de casos de pruebas.

Importaciones básicas

Se deben importar las siguientes clases (lógicamente muchas importaciones serán dependientes del caso de prueba a desarrollar por lo que se han colocado las que suelen ser más generales):

- WebDriver, que establece la conexión entre las pruebas y el navegador Firefox.

```
from selenium.webdriver.firefox.webdriver import
    WebDriver
```

- Algunas excepciones según vayan siendo necesarias, por ejemplo:

```
from selenium.common.exceptions import
    NoSuchElementException
from selenium.common.exceptions import
    NoAlertPresentException
```

- Clase By para escoger el tipo de filtro en la selección de elementos web.

```
from selenium.webdriver.common.by import By
```

- Fichero de configuración de la aplicación para obtener la configuración establecida: conexión a BD, etc.

```
from Test import settings
```

- Select, si se desea trabajar con selectores en formularios de la aplicación web.

```
1 from selenium.webdriver.support.select import Select
```

- Métodos y la especialización de StaticLiveServerTestCase desde "utils.py".

```
from utils import *
```

- Sistema operativo para poder seleccionar directorios en los campos de tipo file.

```
import os
```

Definición de casos de prueba

Realizadas las importaciones oportunas, se comienza con la definición de casos de prueba. Para ello, se crea una clase representativa del conjunto de casos de prueba que herede de la clase CustomStaticLiveServerTestCase importada desde "utils.py".

Para facilitar la explicación, se definirán casos de pruebas asociados a un ejemplo concreto, en este caso, de autenticación de usuarios.

Se definirá un método dentro de la clase definida por cada caso de pruebas que se quiera desarrollar del siguiente modo:

```
1 class TestsAuthentication(CustomStaticLiveServerTestCase):
2     def test_start_session(self):
3         ...
4     def test_start_session_with_false_credentials(self):
5         ...
6     def test_close_session(self):
7         ...
```

Ahora se deben definir los pasos y las validaciones oportunas por cada caso de prueba. Para ello, primero se debe obtener el controlador del navegador con el fin de poder movernos a través de la aplicación desde el código.

```
driver = instance.driver
```

De esta forma, la variable "driver" nos permitirá tener acceso al navegador Firefox y permitirá la ejecución de órdenes a través del código que permitan navegar por la aplicación web y validar elementos que se muestren en la misma.

Para acceder a una cierta URL a través del navegador se utiliza la siguiente línea de código:

```
driver.get('%s%s' % (instance.live_server_url, '/
    ruta_deseada'))
```

Acceso a elementos del WebDriver

Para buscar un elemento se pueden utilizar diferentes métodos que vienen dados por Selenium, como por ejemplo:

```
1 driver.find_element_by_name("username").clear()
2 driver.find_element_by_name("username").send_keys(name)
3 driver.find_element_by_id("password").clear()
4 driver.find_element_by_id("password").send_keys(name)
5 driver.find_element_by_css_selector("button[type=\"submit
    \"]").click()
```

Como se muestra en el ejemplo, también se pueden encadenar otros métodos del WebDriver sobre los elementos encontrados que permitan hacer un "click" de ratón, limpiar o rellenar el contenido de un cierto input, etc.

Para más información sobre los métodos y funcionalidades que ofrece Selenium para Python se ofrece una API del WebDriver en el sitio web de Selenium-Python [28].

Validaciones con aserciones

Una aserción es una condición que verifica que el estado de la aplicación es acorde a lo requerido o esperado (por ejemplo, que algo esperado se encuentra realmente presente).

Generalmente, un caso de prueba contiene un conjunto de aserciones aunque en ciertos casos con una sola validación es suficiente para probar que un elemento o subsistema funciona correctamente para un cierto caso de prueba.

Django permite definir aserciones de distinto tipo donde principalmente se encuentran:

- assertEqual: recibe dos elementos y verifica que sean iguales.
- assertTrue: verifica que una condición sea verdadera.
- assertFalse: verifica que una condición sea falsa.

Para hacer uso de estas aserciones en el código de un caso de prueba se debe de realizar lo siguiente:

```
self.assertFalse(is_element_present(driver, By.CSS_SELECTOR
    , "li.error"), u"FAILURE: Credentials don't match.")
```

Donde el primer parámetro es la condición que debe verificarse comprobándose que sea falsa y el segundo parámetro es el mensaje que debe aparecer en caso de que la verificación falle.

Para el caso de "assertEqual", se deberán introducir tres parámetros donde los dos primeros serán comparados y el tercero será el mensaje de fallo de la verificación.

E.3.2. Ejecución de pruebas

Las pruebas en Django pueden ejecutarse desde diferentes niveles:

- **Ejecución total:** ejecuta todas las pruebas definidas en la aplicación web. Para ello, se ejecuta la siguiente instrucción en consola:

  ```
  python manage.py test
  ```

- **Ejecución parcial:** ejecuta un subconjunto de pruebas o casos de pruebas de la aplicación web. En la ejecución parcial se pueden distinguir los siguientes casos:

  - Ejecución de pruebas de una cierta app: ejecuta todas las pruebas definidas en el directorio "tests" de cierta app.

    ```
    python manage.py tests app.tests
    ```

  - Ejecución de pruebas asociadas a una cierta app: a diferencia de la anterior, esta ejecuta todas las pruebas que posean alguna referencia a la app seleccionada.

```
python manage.py tests app
```

- Ejecución de una prueba concreta: suponiendo que la clase que hemos definido se denomina "TestsAuthentication" sería de la siguiente forma.

  ```
  python manage.py tests app.tests.
     TestsAuthentication
  ```

  O seleccionando el fichero donde se encuentra las pruebas para evitar ambigüedades.

  ```
  python manage.py tests app.tests.
     test_authentication.TestsAuthentication
  ```

- Ejecución de un cierto caso de prueba: en este caso, solo se ejecuta uno de los métodos definidos en la clase de pruebas que se haya seleccionado.

  ```
  python manage.py tests app.tests.
     TestsAuthentication.test_start_session
  ```

  O seleccionando el fichero donde se encuentra las pruebas para evitar ambigüedades.

  ```
  python manage.py tests app.tests.
     test_authentication.TestsAuthentication.
     test_start_session
  ```

BIBLIOGRAFÍA

[1] Sitio web del Área de Informática de la UCA. `http://informatica.uca.es/`, 2015. Última visita: 15.07.2015.

[2] Sitio web oficial de Symfony framework. `https://symfony.com/`, 2015. Última visita: 15.07.2015.

[3] Sitio web oficial de Django framework. `https://www.djangoproject.com/`, 2015. Última visita: 15.07.2015.

[4] M. Fowler. *Domain Specific Languages*, capítulo 1–2, páginas 3 y 27–39. Addison-Wesley Professional, 2010.

[5] M. Mernik, J. Heering y A. M. Sloane. *When and how to develop domain-specific languages*, páginas 316–344. ACM Computing Surveys, 2005.

[6] D. C. Schmidt. *Model-Driven Engineering*, páginas 25–31. IEEE Computer Society, 2006.

[7] Sitio web del proyecto Xtext en Eclipse. `https://eclipse.org/Xtext/`, 2015. Última visita: 15.07.2015.

[8] Sitio web oficial de Eclipse. https://eclipse.org/, 2015. Última visita: 15.07.2015.

[9] Sitio web del proyecto Epsilon en Eclipse. http://www.eclipse.org/epsilon/, 2015. Última visita: 15.07.2015.

[10] R. B. Hernández. *Introducción a Flex y Bison*, páginas 3–14. http://webdiis.unizar.es/asignaturas/LGA/material_2004_2005/Intro_Flex_Bison.pdf, 2004-05. Capítulos 1–2. Última visita: 15.07.2015.

[11] Sitio web oficial de ANTLR. http://www.antlr.org/, 2014. Última visita: 15.07.2015.

[12] A. Holovaty y J. Kaplan-Moss. The Django Book. http://www.djangobook.com/en/2.0/, 2007. Última visita: 25.07.2015.

[13] K. Beck y C. Andres. *Extreme Programming Explained: Embrace Change, 2nd Edition*. Addison-Wesley Professional, 2004. Capítulos 1, 3–5.

[14] D. Kolovos, L. Rose, A. García-Domínguez y R. Paige. *THE epsilon BOOK*, páginas 13–14, 23–56, 101–116. The Eclipse Foundation, 2015.

[15] LaTeX Project Team. LaTeX - a document preparation system. http://www.latex-project.org/, 2010.

[16] L. Lamport. *LaTeX - A Document Preparation System: User's Guide and Reference Manual, Second Edition*. Pearson / Prentice Hall, 1994.

[17] Aura Portal. What is bpm? http://www.auraportal.com/en/what-is-bpm--business-process-management, 2015. Última visita: 26.07.2015.

[18] Sitio web oficial de Aura Portal. http://www.auraportal.com/, 2015. Última visita: 26.07.2015.

[19] Sitio web oficial de Eunomia Process Builder. http://www.eunomia-process.com/, 2015. Última visita: 26.07.2015.

[20] Sitio web oficial de Bonita BPM. http://www.bonitasoft.com/, 2015. Última visita: 26.07.2015.

[21] R. Canales-Mora. Primeros pasos con Bonita BPM Community 6.2.6. http://www.adictosaltrabajo.com/tutoriales/bonita-bpm-primeros-pasos/, 2014. Última visita: 02.08.2015.

[22] Object Management Group. Especificación oficial de Object Constraint Language 2.4. http://www.omg.org/spec/OCL/2.4/, 2014. Última visita: 19.08.2015.

[23] Sitio web oficial de SeleniumHQ. http://www.seleniumhq.org/, 2015. Última visita: 15.08.2015.

[24] Repositorio github de Debug Toolbar. https://github.com/django-debug-toolbar/django-debug-toolbar, 2015. Última visita: 15.08.2015.

[25] IICREST. Business Modeling and Software Design. http://www.is-bmsd.org/, 2015. Última visita: 08.09.2015.

[26] U. de Cantabria. Jornadas de Ingeniería del Software y Bases de Datos. http://www.istr.unican.es/sistedes2015/jisbd.html, 2015. Última visita: 08.09.2015.

[27] Sitio web oficial de GitHub. https://github.com/, 2015. Última visita: 21.08.2015.

[28] API del WebDriver de Selenium-Python. http://selenium-python.readthedocs.org/en/latest/api.html, 2011-14. Última visita: 07.09.2015.