ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA INFORMÁTICA

APERTURA DE DATOS EN PROYECTOS DJANGO

José Manuel Llerena Carmona

diciembre de 2013

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA EN INFORMÁTICA

APERTURA DE DATOS EN PROYECTOS DJANGO

- Departamento: Ingeniería Informática
- Directores del proyecto: Iván Ruíz Rube y Juan Manuel Dodero Beardo
- Autor del proyecto: José Manuel Llerena Carmona

Cádiz, diciembre de 2013

Fdo: Iván Ruíz Rube Fdo: Juan M. Dodero Beardo Fdo: José M. Llerena Carmona

Agradecimientos

A mis padres, por haberme dado la oportunidad de estudiar, ya que sin ellos nunca lo podría haber conseguido.

A mi novia Isabel, por haber estado siempre a mi lado apoyándome.

A mis tutores Don Iván Ruíz Rube y Don Juan Manuel Dodero Beardo, a los que les agradezco sinceramente toda la ayuda y consejos que me han prestado.

Resumen

Hoy en día es mucha la información que circula a través de internet, y aunque la gran mayoría sea comprensible por el ser humano, las máquinas no disponen de la capacidad de razonamiento necesario para comprender en muchos de los casos de qué trata dicha información o con qué está relacionada. De esta forma, mediante el marcado de nuestros datos haciendo uso de vocabularios específicos, podemos dotar a nuestro contenido web de un mayor valor semántico (web semántica), de tal forma que estemos indicando explícitamente qué es la información que estamos publicando.

En el presente proyecto titulado Apertura de datos en aplicaciones Django, se ha desarrollado una herramienta para el framework de desarrollo web Django, el cual se encuentra desarrollado bajo el lenguaje de programación Python. Esta herramienta o plugin, se encargará de facilitar al usuario la publicación controlada de los datos de sus proyectos Django en internet, haciendo uso de los vocabularios u ontologías disponibles en internet para tal finalidad.

En la web están disponibles multitud de ontologías disponibles para los usuarios que deseen hacer uso de ellas. Cada una de ellas relacionadas con una temática diferente o con muchas temáticas a la vez, como por ejemplo puede ser, la música, la ingeniería, el cine, electrónica, etc...

A día de hoy, multitud de organizaciones, tanto públicas como privadas, usuarios y software están adaptando sus productos a la web semántica para la apertura de sus datos. Un ejemplo bastante claro es el Open Government (o gobierno abierto), donde se defiende que los temas del Gobierno y Administración Pública se abran a los ciudadanos, de tal forma que se transmita transparencia en sus gestiones. También en el ámbito del comercio, aplicaciones bastante populares de comercio electrónico, están adaptando o han adaptado sus aplicaciones para la publicación de sus datos, confiriendo de esta forma una mayor proyección de los productos.

A raíz de todo esto comentado, surge la idea de realizar una herramienta para un framework de desarrollo web cada vez más usado, que permita adaptar los proyectos existentes a la publicación de datos, sin necesidad de redesarrollar los proyectos por completo.

Por lo tanto, la herramienta que se ha desarrollado en el presente proyecto, permitirá a partir de sus datos realizar un etiquetado de los mismos para luego publicar estos en internet. Para desarrollar esto, la presente herramienta ofrece al usuario las siguientes funcionalidades:

- Captación de los datos de los que está compuesto nuestro proyecto Django.
- Carga de espacios de nombres y ontologías definidas en la web que sirvan para el propósito de los datos que se desean publicar.
- Posibilidad de elegir qué datos van a ser publicados y cuales no lo serán.
- Componente para la interrelación de los espacios de nombres y los datos de nuestro proyecto.
- Componente para la publicación de nuestros datos en diferentes formatos existentes.

Palabras clave: OpenData, LinkedData, RDF, RDFa, Microdata, ontología, namespace, RDF/Ntriples, RDF/Turtle, XML, Django, Python, SPARQL, D2Rq.

Índice general

Índice de figuras

Índice de tablas

Parte I

Prolegómeno

Capítulo 1

Introducción

A lo largo del siguiente documento, se describirán las motivaciones que han llevado a la realización del proyecto, así como los objetivos, estructura, requisitos y demás características técnicas, implementación y pruebas del mismo, además de los manuales necesarios para su utilización.

Como punto inicial, en el presente capitulo se describen la motivaciones para la realización del proyecto, los objetivos y cómo se encuentra estructurada la documentación.

1.1. Motivación

Actualmente existe una gran cantidad de datos distribuidos por toda la red, pero éstos no se encuentran publicados ni organizados de forma que otros usuarios u organizaciones puedan hacer uso de ellos, o mejor dicho, de forma que otras aplicaciones externas puedan entenderlos. Especificando un formato común para publicar dicha información, se podrían obtener grandes cantidades de información en la web, a disposición de cualquier usuario u organización que la desease.

Hoy en día muchos Gobiernos (OpenGovernment) están abriendo parte de sus datos en la Web, beneficiando así a multitud de usuarios, los cuales pueden captar estos datos y generar estadísticas, informes, estudios, generando transparencia en la gestión del gobierno. O como en el caso del comercio electrónico, si un comerciante abre sus datos en la Web, los buscadores pondrán enlazar mucho mejor sus productos, y ofrecer mejores resultados de búsqueda para su Web de comercio electrónico. Como conclusión, podemos decir que la apertura de datos, es beneficiosa en multitud de casos totalmente distintos, tanto como el marketing, investigación, gobierno, etc...

A raíz de todo esto, surgen multitud de vocabularios (ontologías) donde se definen una serie de entidades y propiedades de las mismas, con las cuales podemos relacionar nuestros datos. También existen multitud de estándares los cuales nos permiten realizar la publicación de los datos en un formato conocido. Por lo que, cuando vayamos a abrir nuestros datos podremos

darle una estructura conocida por todos, de forma que aquella persona que vaya a hacer uso de ellos, pueda conocer a qué hacen referencia los mismos. De esta forma, conseguimos darle un significado semántico a los datos que estamos mostrando en nuestra web. Además, podremos definir relaciones entre los distintos datos, tanto los pertenecientes a nuestros proyectos como de otras fuentes externas, lo cual se conoce como LinkedData.

1.2. Descripción del sistema actual

En la actualidad, para el framework de desarrollo de aplicaciones web Django, no existen aplicaciones (o éstas ofrecen pocas funcionalidades o se encuentran obsoletas) que permitan realizar la apertura de datos de proyectos Django a través de un menú de configuración intuitivo, haciendo uso de cualquiera de las diferentes ontologías disponibles en internet para el marcado de los datos, como pueden ser por ejemplo:

- **Schema [GMY12]:** proporciona un vocabulario para marcar los contenidos web, el cual es reconocido por los principales buscadores, como son: Google, Bing, Yahoo o Yandex.
- **FOAF [FOA10]:** acrónimo de Friend Of A Friend, define una ontología para la publicación de relaciones entre personas y datos en internet.
- **Good relations [Hep12]:** similar a los anteriores, define un vocabulario específico para el comercio electrónico. Dicho vocabulario, ya ha sido incluido en la especificación de Schema.

Por otro lado, actualmente si que existen herramientas de este tipo realizadas para otros frameworks de desarrollo de aplicaciones Web, como puede ser el caso de una gema realizada para el framework Ruby on Rails, bajo el nombre de EasyData[1], desarrollada también en la Universidad de Cádiz, que ha obtenido buenos resultados. Al igual que también conocidas aplicaciones de comercio electrónico realizadas en PHP, como puede ser Magento o Prestashop, también están siguiendo esta tendencia y se están adaptando al espacio de la Web semántica, vista las ventajas que ofrece. Por lo tanto, a raíz de la utilidad del mismo y que prueba de ello cada vez más tecnologías se están adaptando, sería interesante disponer de una herramienta de este tipo, en un framework como Django, que cada vez está más presente en el total de aplicaciones existentes en la Web.

1.3. Objetivos y alcance del proyecto

El objetivo principal del presente proyecto es el de proporcionar al desarrollador de aplicaciones Django, una herramienta (conocida a partir de ahora como EasyData/Django) la cual le permita realizar la apertura **controlada** de los datos de su proyecto Django. Esto es el producto final que se obtendrá al finalizar dicho proyecto, pero este proyecto se puede desgranar en objetivos más específicos o subobjetivos, donde la unión de estos formen la totalidad del proyecto. Estos objetivos más específicos, se pueden dividir en tres, los cuales son:

[1]El proyecto se encuentra alojado en el repositorio `https://github.com/jnillo/Linked_data`

OBJ-001	**Resolver modelos de la aplicación**
Autor	José Manuel Llerena Carmona
Descripción	Un primer objetivo será realizar una parte software, que sea capaz mediante introspección, de obtener los distintos modelos, junto con los fields y relaciones y los tipos de estos dos últimos que componen el proyecto Django donde se encuentra instalado nuestro proyecto. Esta estructura se almacenará, para posteriormente poder realizar la configuración de cómo se va a llevar a cabo la publicación de los datos.
Subobjetivos	▪ Confeccionar una estructura de base de datos, que permita almacenar la estructura de modelos que forman el proyecto Django donde se encuentra nuestra aplicación. ▪ Desarrollar una porción de software, la cual sea capaz de obtener la estructura de los modelos del proyecto Django donde nos encontramos, y almacenar esta estructura en la base de datos confeccionada anteriormente.

Tabla 1.1: Objetivo - 001 - Resolver modelos de la aplicación

OBJ-002	**Interfaz de configuración**
Autor	José Manuel Llerena Carmona
Descripción	Un segundo objetivo consistirá en proporcionar al usuario una aplicación Web de configuración, donde podrá detallar cómo se va a realizar la apertura de datos de su proyecto. En este apartado, el usuario deberá de indicar con qué entidades de los vocabularios disponibles en la aplicación, se relacionan los modelos de su proyecto, y con qué propiedades de dichas entidades, se relacionan los atributos y relaciones de cada uno de los modelos. Además se podrá especificar para cada uno de los modelos, cuales serán visibles y cuales no (no todos los datos son susceptibles de ser publicados, ya que todo proyecto puede contener datos sensibles o que se encuentren protegidos por leyes de protección de datos).
Continúa en la siguiente página	

Continuación de la tabla	
OBJ-002	**Interfaz de configuración**
Subobjetivos	▪ Proporcionar un apartado dentro del proyecto, donde el usuario sea capaz de cargar los diferentes vocabularios u ontologías que va a utilizar para la publicación de los datos. ▪ Proporcionar un apartado donde se listen los distintos modelos que componen al proyecto Django, de tal forma que para cada uno de los modelos, pudiésemos especificar con qué entidades de las propuestas por las distintas organizaciones se corresponden. ▪ Un apartado más dónde para cada modelo, se pudiera realizar la correspondencia (mapeo) de cada uno de los fields del modelo con las propiedades de la entidad asociada al modelo o cualquier otra disponible en el namespace. ▪ Un último apartado, donde para cada modelo y field o relación de nuestros modelos, podamos indicar si estos son visibles o no.

Tabla 1.2: Objetivo - 002 - Interfaz de configuración

OBJ-003	**Publicación de datos**
Autor	José Manuel Llerena Carmona
Descripción	Un tercer y último objetivo sería el de proporcionar al usuario, una serie de herramientas, las cuales le permitan realizar la publicación de los datos de su proyecto. Estas le permitirían al usuario utilizar los distintos tipos de notaciones más comúnmente usados, como pueden ser RDF/XML, RDF/Turtle, RDF/Ntriples, Microdata o RDFa.

Tabla 1.3: Objetivo - 003 - Publicación de datos

Por lo tanto, como conclusión a todo lo comentado, el alcance del proyecto será el de proporcionar al usuario, una aplicación Django, que le permita realizar la apertura de los datos de forma controlada de sus proyectos Django en la Web. Para ello, la aplicación le proporcionará al usuario, la posibilidad de usar la mayor parte de los formatos permitidos en la Web semántica para la publicación de datos en internet. Así como también le permitirá la carga de las diferentes ontologías existentes en la web, de tal forma que la aplicación se adaptará a cualquier nuevo espacio de nombres que pueda surgir en un futuro.

1.4. Definiciones

Framework
en el ámbito del desarrollo software, se entiende por framework a una plataforma software para desarrollar aplicaciones, la cual sirve como base para el desarrollo.

Modelo
es uno de los componentes del patron Modelo-Vista-Controlador. El modelo se encarga de representar y gestionar toda la información con la que el sistema trabaja.

Vista
es otro de los componentes del patron Modelo-Vista-Controlador. En el caso de Django, la vista hace referencia al controlador, aunque exista otro componente con el nombre de vista. Las vistas en Django se encarga de responder a eventos y tratar la información realizando peticiones a los modelos. Las vistas pueden utilizar las plantillas, para mostrar los datos generados en ellas.

Plantilla
es otro de los componentes del patron Modelo-Vista-Controlador. En el caso de Django, la plantilla hace referencia a la vista del patrón MVC. La plantilla se encarga de presentar los datos en un formato adecuado para la interacción.

Ontología o Namespace
dentro de la aplicación que se va a desarrollar, se conoce por ontología o namespace, a cada uno de los vocabularios definidos que están disponibles para que los usuarios realicen el marcado del contenido de sus datos.

Python
[Fou13b] es el lenguaje de programación que se va a utilizar para el desarrollo del proyecto. Este lenguaje de programación es multiparadigma, ya que permite la utilización de diferentes paradigmas de programación (orientado a objetos, funcional o imperativa). Este lenguaje de programación también se caracteriza por ser multiplataforma y de código abierto. Para más información se puede consultar la web oficial de Python.

Django
se trata de un framework de desarrollo web de código abierto, el cual se utilizará para la realización del proyecto. Este framework está escrito en el lenguaje de programación Python y está basado en el modelo MVC (Modelo Vista Controlador).

RDF
[RDF04a] es el acrónimo de Resource Description Framework. Está diseñado por el World Wide Web Consistorium como un modelo de datos para metadatos. Se usa para la descripción conceptual y modelado de la información para recursos web.

RDFa
es el acrónimo de Resource Description Framework in Attributes. Se trata de una serie de extensiones para lenguajes HTML, XHTML y ciertos documentos basados en XML recomendadas por W3C, de forma que se permita dotar de contenido semático a los documentos web.

Microdata
: es una especificación HTML de WHATWG (Web Hypertext Application Technology Working Group), para enlazar metadatos al contenido existente en páginas web.

Web semántica
: es un conjunto de actividades promovidas por el W3C con la finalidad de crear tecnologías que permitan la publicación de datos legibles por aplicaciones informáticas.

1.5. Organización del documento

Este documento tiene como finalidad, describir todo el proceso de desarrollo de la aplicación Django para la apertura de datos de proyectos y su posterior publicación, empezando por la descripción de los objetivos iniciales y requisitos esenciales que deberá reunir dicho proyecto. Se mostrará una planificación del desarrollo del proyecto, incluyendo costes estimados y diagramas de Gantt que ilustren el avance del proyecto en el tiempo.

En este documento también se plasmará todo el ciclo de vida de desarrollo, desde la fase de análisis y diseño de los requisitos del proyecto, definiendo los distintos casos de usos, diagramas con la organización de los datos, actores implicados o diagramas de secuencia e interacción que pudieran resultar del mismo. Siguiendo por la implementación del mismo y las pruebas realizadas, las cuales verifiquen la calidad y robustez del mismo. Además, en este documento también se comentarán las posibles mejoras o ampliaciones futuras que pudieran considerarse.

Por último, proporcionará al usuario tanto un manual de uso, como un manual de instalación y explotación del mismo, de tal forma que cualquier usuario sin conocimientos previos de la aplicación, pueda hacer uso de ella y sacarle el máximo partido posible.

Capítulo 2

Planificación

En esta sección se describen todos los aspectos relativos a la planificación del proyecto: metodología, organización, costes, planificación y gestión de riesgos.

2.1. Metodología de desarrollo

Como metodología de desarrollo para la elaboración del proyecto, nos hemos basado en el modelo de ciclo de vida de desarrollo en espiral. Este modelo de desarrollo se centra en dividir el proyecto en segmentos más pequeños y proporcionar más facilidad de cambio en el proceso de desarrollo. En este modelo, se combinan las principales características del desarrollo iterativo, con las del desarrollo en cascada, de forma que en cada uno de las iteraciones en la espiral, recoge todas fases del modelo en cascada, de tal forma, que al final de cada iteración, obtenemos una porción de software, y en las sucesivas iteraciones se obtienen versiones incrementales de la misma pieza software. Tal y como explicamos en el capítulo anterior, nuestro proyecto lo hemos dividido en varias piezas software "independientes", de tal forma haciendo uso de este modelo, nos permitirá realizar el desarrollo por partes, obteniendo en cada una de las iteraciones de la espiral, una de las piezas software definidas anteriormente.

Para el modelo de ciclo de vida de desarrollo en espiral, se definen cuatro fases, las cuales son las que se repiten en cada una de las iteraciones de la espiral. Estas fases son las siguientes:

- **Definición de objetivos:** Exceptuando la planificación inicial, que es llevada a cabo al comienzo del proyecto, en esta fase en cada una de las iteraciones de la espiral se especifican y analizan los requisitos que deberá cumplir el componente software, al final de la iteración para cumplir cada uno de los objetivos en los que hemos dividido el proyecto.

- **Análisis de riesgos:** En esta fase se evalúan los posibles riesgos y eventos no deseados que puedan surgir durante la ejecución de la iteración, para que en la fase de desarrollo y pruebas, se puedan evaluar los mismos y buscar las alternativas posibles para solventarlos.

- **Desarrollo y pruebas:** En este apartado se realizan las tareas propias que su nombre indica, se lleva a cabo el diseño e implementación de los requisitos especificados, y se prueban los mismos. Además, en este apartado, como se acaba de comentar, se evalúan las posibles alternativas para solventar los posibles riesgos existentes. Al final de esta fase, se obtiene la nueva versión incremental del software.
- **Planificación:** En esta fase se revisan el producto obtenido, y se decide si se continúa con la siguiente iteración.

2.2. Planificación del proyecto

A continuación se va a realizar una estimación temporal del tiempo que se empleará en la elaboración del proyecto. Para ello, vamos a dividir el proyecto en una serie de fases, las cuales según la complejidad y extensión de las mismas, se le asignará un tiempo estimado para su finalización.

2.2.1. Especificación de las Fases

Las fases en las que vamos a dividir nuestro proyecto para realizar la estimación temporal, son las siguientes:

- **Estudio inicial:** este apartado hace referencia al estudio de la propuesta de proyecto, redacción del proyecto y motivaciones del mismo. Comprende desde las primeras reuniones con los tutores del proyecto donde se propone el mismo, hasta el momento de la asignación de este.
- **Preparación del entorno:** esta parte se refiere a todas las tareas realizadas para la preparación del entorno de trabajo para elaborar el proyecto. Desde la solicitud de apertura de un espacio de trabajo y un repositorio SVN para alojar el proyecto, hasta la selección e instalación de las herramientas necesarias para realizar el mismo, como pueden ser el IDE, el sistema de control de versiones, servidores o configuración de la máquina.
- **Fase de Análisis**[1]**:** el apartado de fase de análisis, se corresponde totalmente con la fase de análisis del ciclo de vida del software. Aquí se especifican los requisitos funcionales y no funcionales que debe de poseer la aplicación, los distintos actores que pueden intervenir en él, se detallan los casos de uso y se hace una especificación de los datos que manejará nuestra aplicación.
- **Fase de Diseño**[1]**:** dentro del ciclo de vida del software, esta fase es la que aparece justo a continuación de la fase de análisis. Es esta fase se baja un nivel de abstracción más,

[1]Estos apartados se repetirán varias veces a lo largo del desarrollo del proyecto, ya que al hacer uso de una metodología de desarrollo en espiral, en cada una de las iteraciones de la metodología, tendremos que volver a revisar cada uno de los requisitos, para determinar si se cumplen o si estos se adaptan a los requisitos verdaderos de la aplicación, y continuar con la implementación y/o corrección de los requisitos existentes y realizar las pruebas de los mismos.

respecto de la fase anterior. Aquí se detallará a nivel de operaciones, como se llevarán a cabo las distintas funciones que debe cumplir nuestro software. Además, se proporcionará un diagrama de persistencia de los datos que maneja nuestra aplicación en una base de datos.

- **Implementación[1]:** esta fase hace referencia a el apartado más técnico del desarrollo del producto. En esta fase se pasa de la especificación en papel, a la codificación del mismo en el lenguaje de programación seleccionado. Esta fase es de las más extensas y a su vez de las más importantes, ya que es donde se obtendrá el producto software final.
- **Pruebas[1]:** esta fase, no menos importante que la de implementación, se llevará a cabo a la vez que la implementación. La fase de pruebas, será la responsable del control de calidad, de tal forma que en ella se descubran los posibles errores tanto de codificación como de funcionamiento de la aplicación. En esta fase se asegurará que el software contenga unos mínimos de calidad, seguridad y fiabilidad.
- **Documentación:** Este apartado abarca todo el ciclo de vida del desarrollo del proyecto, y en él se detallarán todas y cada una de estas fases. Esta fase abarcará desde el momento inicial del proyecto, hasta el final del mismo. Este apartado además de los detalles técnicos de la aplicación, incluirá los manuales de usuario, las instrucciones de instalación y despliegue, las estimaciones de tiempo y conclusiones.

2.2.2. Diagrama de Gantt Inicial

En el diagrama de Gantt de la Figura 2.1, se pueden apreciar cada una de las fases comentadas anteriormente, expresadas en función del tiempo que se estima que se empleará en cada una de ellas:

2.2.3. Diagrama de Gantt con los tiempos reales

A continuación también se muestra el diagrama de gantt (Figuras 2.2 y 2.3), el cual se ha obtenido una vez finalizado el proyecto, con los tiempos reales utilizados, de forma que se pueda realizar una comparación, con los tiempos que se estimaron en un principio.

Tal y como se puede apreciar en las figuras, los tiempos estimados de duración del proyecto se han ajustado bastante bien a los tiempos que finalmente se han necesitado para la ejecución del proyecto. Inicialmente se estimó una duración total de 12 meses, desde octubre de 2012 hasta octubre de 2013, habiéndose excedido el proyecto únicamente en un mes (finalizado en noviembre de 2013).

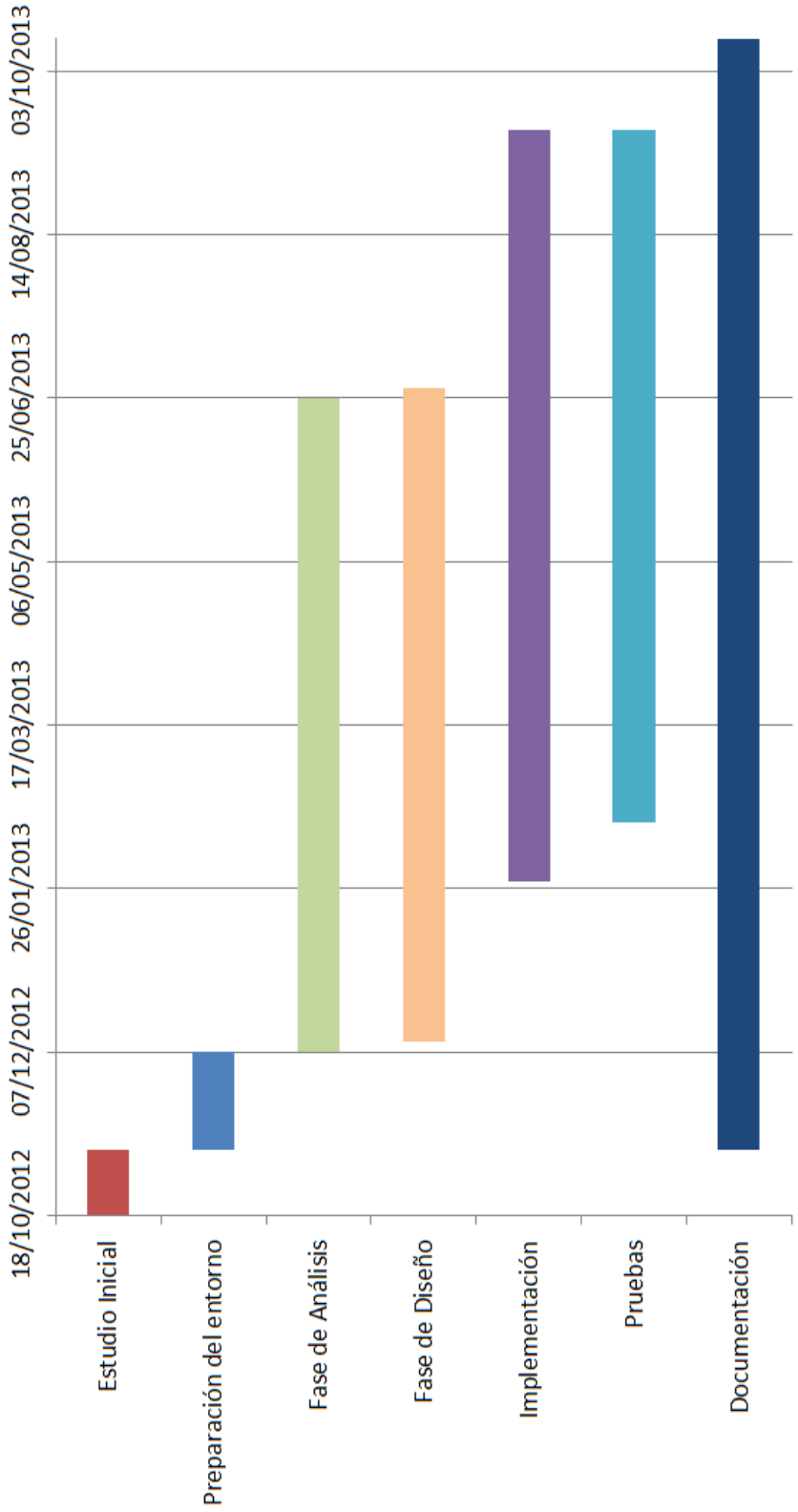


Figura 2.1: Diagrama de Gantt con estimación temporal

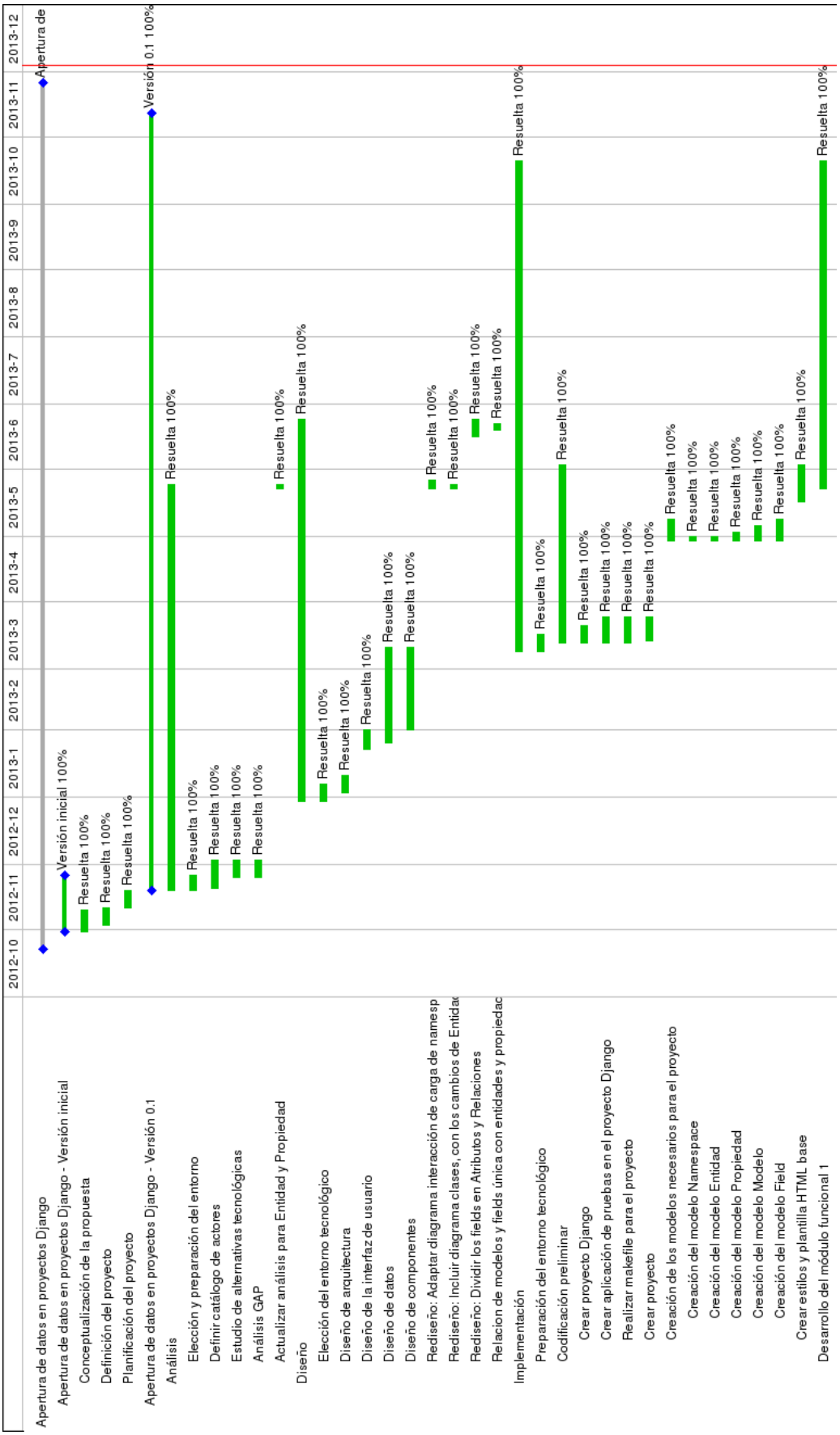


Figura 2.2: Diagrama de Gantt con los tiempos finales (Parte 1)

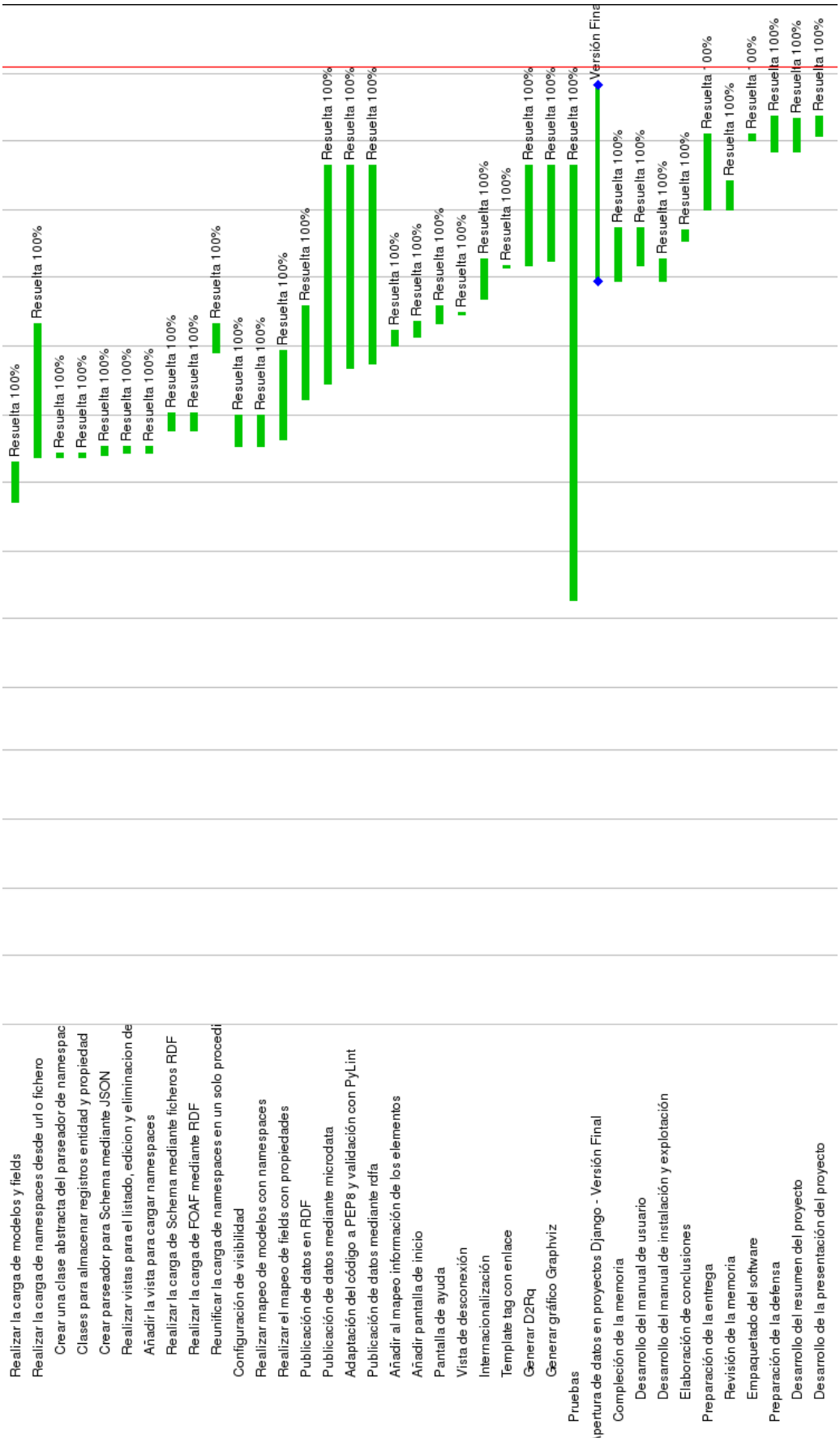


Figura 2.3: Diagrama de Gantt con los tiempos finales (Parte 2)

2.3. Organización

2.4. Roles existentes

El conjunto de personas que componen a la organización involucrada en el desarrollo del proyecto, son las siguientes:

- **Desarrollador:** El desarrollador del proyecto, es el alumno que está llevando a cabo dicho proyecto informático. Desempeña los papeles de analista informático, de desarrollador informático e ingeniero de test. Sus funciones son las de analizar y diseñar el software, así como de desarrollar el mismo cumpliendo las especificaciones, realizar los tests de calidad oportunos y documentar todo el proceso de desarrollo.
- **Tutores:** Los tutores del proyectos, tienen como cometido guiar al alumno en el desarrollo del mismo. Se puede decir que realizan un papel similar al cliente que contrata la realización del software, se encargan de proponer el desarrollo del proyecto y de proponer los objetivos que debe cumplir el mismo. Además, también ayudan al alumno en el proceso de desarrollo, aportando ideas y posibles soluciones.

Durante el desarrollo del proyecto, el alumno como se acaba de indicar, será el indicado de realizar todo el proceso del ciclo de vida del software. Puntualmente, mantendrá reuniones con los tutores del proyecto, de forma que estos puedan tener una visión del avance del proyecto a lo largo del tiempo. Los tutores del proyecto, en cada una de estas reuniones, podrán aportar posibles ideas, o nuevos objetivos que deba cumplir el software, guiando al alumno en la consecución del mismo.

2.5. Recursos inventariables

En todo proyecto informático, además del factor humano, entran en escena otra serie de factores, como son los tecnológicos, los cuales servirán al alumno en la consecución de sus objetivos. En la Tabla 2.1, se muestra un inventario de las herramientas, tanto hardware como software, utilizadas para la elaboración del proyecto:

Recurso inventariable	**Descripción**	
Hardware	Procesador	AMD Athlon II x3 460
	Memoria RAM	8GB DDR3
	Tarjeta Gráfica	NVIDIA GeForce GTX 550 Ti
	Disco Duro	Seagate 1TB SATA3 7200rpm
	Grabadora DVD	LG GH24NS90 DVD 24X
continua en la siguiente página		

continuación de la página anterior		
Elemento inventariable	**Descripción**	
	Monitor	Samsung SyncMaster 2243LNX
Sistema operativo	Linux Mint 14 Nadia	
Editor de textos	Gedit 3.4.1	
Python	Versión 2.7	
Django	Versión 1.4.0	
iPython	Interprete interactivo de Python	
Dia	Software para la generación de gráficos.	
Gimp	Software para la edición de imágenes.	
ConceptDraw Office	Software para la generación de diagramas.	
Apache	Servidor web.	

Tabla 2.1: Tabla con inventario de materiales

Para los recursos de tipo software, se ha intentado usar en todo momento software de tipo libre, de tal forma que se reduzcan los costes al máximo.

2.6. Costes

Para llevar a cabo una estimación de los costes de producción, a parte de tener una estimación del tiempo y recursos necesarios para llevar a cabo el proyecto, también es necesario conocer el precio unitario de cada uno de estos recursos. Si bien, los recursos materiales es de un coste conocido, ya que conocemos de cuales de ellos necesitamos disponer, y el coste de los mismos. Respecto a materiales como puede ser el ordenador personal, mobiliario, y demás elementos de los que debe de poseer cualquier empresa, no deben de tenerse en cuenta su coste total, ya que estos no son específicos para un puesto en concreto, sino que se mantendrán en la empresa y su precio se amortizará a lo largo del tiempo. Normalmente el tiempo de amortización de los mismos, se estima en tres años, por lo que al haber estimado una duración de un año para el proyecto, contemplaremos solo un 33 % del precio de los mismos. De esta forma, si suponemos que conociendo que el coste del equipo informático es de 500€, nos quedará un coste total de 165€.

Para estimar el costo de los recursos humanos que intervendrán en el desarrollo del proryecto software, deberemos de tener en cuenta tanto el número de personas implicadas, como la duración del proyecto, como el sueldo que habría que abonarle a cada uno de los participantes en el proyecto. Este proyecto, al tratarse de un proyecto de pequeña envergadura, será realizado únicamente por una persona, que trabajará durante 12 meses, que son los que se han estimado para la duración del proyecto.

En la siguiente Tabla (2.2) se reflejan los distintos recursos necesarios para elaborar el proyecto, con los costos y unidades necesarias asociados de los mismos:

Recurso	Descripción	Coste
Ingeniero Informático	Es el encargado de realizar tanto la documentación del proyecto, como la implementación del mismo.	1 persona/mes
Ordenador Personal	Es el equipo que utilizará el ingeniero informático en su puesto de trabajo.	165€
Local y mobiliario	Son los gastos del local y derivados del mismo (luz, agua, mesa, etc...) donde se va a realizar el proyecto.	30€/mes
Material de oficina	Hace referencia a papel, bolígrafos, etc...	50€

Tabla 2.2: Tabla de costos de los recursos

Una vez conocemos todos los recursos implicados en la elaboración del proyecto, es necesario tener una estimación del sueldo promedio de un programador informático en España. Infojobs posee una herramienta [Inf13], la cual nos permite consultar el sueldo promedio según el puesto de trabajo que desempeñe el trabajador. Tal y como se puede observar, el salario medio de un ingeniero informático en España, ronda la cuantía de los 17567 € anuales. Aunque, para la realización de este proyecto, no se va a tener una dedicación total, sino que se tratará de una dedicación parcial (aproximadamente una cuarta parte de lo que dedicaría un trabajador normal), por lo que del total de horas mensuales que un trabajador normal dispone (alrededor de unas 152 horas), estimamos que se emplearán aproximadamente unas 38 horas mensuales, de tal forma que calculando la proporción, nos queda un coste total de 366 € por mes, para la dedicación que va a tener el ingeniero al proyecto. Finalmente podemos decir que, si el proyecto tiene un tiempo estimado de 12 meses, y el trabajador percibirá una cantidad de 366 € cada uno de los meses, el coste total del trabajador será de 4.392 €.

De esta forma, una vez que conocemos tanto la cantidad como el costo de todos los recursos implicados en la elaboración del proyecto (tanto humanos como materiales) y conocemos también la estimación temporal que hicimos anteriormente, podemos estimar un costo total para el proyecto, el cual se refleja en la Tabla 2.3:

Unidades	Recurso	Coste unitario	Coste total
12	Ingeniero informático	366€	4.392€
1	Ordenador personal	165€	165€
12	Local	30€	360€
1	Material de oficina	50€	50€
		Total	**4.967€**

Tabla 2.3: Tabla de coste total

Los costes finales se han calculado en función de 12 meses, ya que en la planificación que se ha hecho anteriormente y que podemos apreciar en el diagrama de Gantt (Figura 2.1), el coste temporal total comprendía desde Octubre de 2012 hasta Septiembre de 2013, lo que hace un

total de 12 meses.

2.7. Gestión de riesgos

Dentro de todo proyecto, pueden darse a lugar una serie de riesgos, los cuales complicarían o retrasaría los plazos fijados en la planificación del mismo. Dentro de estos riesgos, pueden entrar en juego distintos factores, ya sean humanos o de cualquier otro tipo de índole. En este apartado, vamos a hacer una descripción de los posibles riesgos que podrían tener lugar, los cuales comprometiesen el cumplimiento de la planificación. Para la descripción de estos posibles riesgos, nos vamos a basar en los descritos en el MAGERIT V3 [MAG12] para determinar los riesgos de tipo genérico que se puedan dar, y además citaremos otra serie de riesgos específicos del proyecto, los cuales también retrasarían los plazos estimados en la planificación. Todos estos riesgos deben de estar descritos dentro de la política de protección de datos de cualquier empresa.

- **Riesgos de tipo genéricos.**
  - **Errores del administrador (E.2):** equivocaciones de personas con responsabilidades de instalación y operación.
  - **Destrucción de información (E.18):** perdida de la información almacenada del proyecto.
  - **Pérdida de equipos (E.25):** la pérdida de equipos provoca directamente la carencia de un medio para prestar los servicios, es decir una indisponibilidad.
  - **Indisponibilidad del personal (E.28):** ausencia accidental del puesto de trabajo: enfermedad, alteraciones del orden público, etc...
  - **Daños por agua (I.2):** escapes, fugas, inundaciones: posibilidad de que el agua acabe con los recursos del sistema.
  - **Contaminación mecánica (I.3):** vibraciones, polvo, suciedad, ...
  - **Avería de origen físico o lógico (I.5):** fallos en los equipos y/o fallos en los programas. Puede ser debida a un defecto de origen o sobrevenida durante el funcionamiento del sistema.
  - **Corte del suministro eléctrico (I.6):** cese de la alimentación de potencia.
- **Riesgos de tipo específicos del proyecto.**
  - **Cambios en la especificación de los vocabularios:** si se introdujesen cambios a la hora de la especificación de los vocabularios, haciendo uso de una nomenclatura diferente, habría que modificar el parseador de ficheros XML que se encargue de dicha función, añadiendo la nueva nomenclatura.
  - **Cambios en la estructura de la base de datos:** si por la aparición de nuevas necesidades o por deficiencias en la estuctura actual se debiesen hacer cambios en la base de datos, se debería de volver nuevamente al apartado de diseño, para definir la nueva estructura de la misma, determinar los sistemas donde afectarán dichos cambios y si fuese necesario, definir herramientas de migración de los datos a las nuevas estructuras, para evitar la pérdida de datos.

- **Modificación de funcionalidades existentes:** si alguna de las funcionalidades especificadas en el proyecto se modificasen, se debería de volver al apartado de análisis y diseño, para modificar los diagramas de caso de uso e interacción de dicha funcionalidad, para posteriormente pasar a la modificación de la funcionalidad.

Para la subsanación de los riesgos que provocan una pérdida de datos, o un deterioro de los mismos, como son los riesgos marcados con los códigos E.2, E.18, E.25, I.2, I.3, I.5 e I.6, se hará uso de un sistema de control de versiones (en nuestro caso se hace uso del sistema subversion), el cual permita en todo momento mantener una copia de cada una de las versiones que existen del proyecto, garantizando la disponibilidad e integridad de los datos. Asímismo, este sistema de control de versiones, se encontrará en otra máquina ajena a la máquina donde se está desarrollando el proyecto, y en un edificio distinto. De esta forma, evitamos posibles pérdidas de información por causas naturales, subidas de tensión, o deterioro de componentes físicos de la máquina donde se está realizando el proyecto.

En el ámbito de los riesgos provocados por causas humanas, debidas a la indisponibilidad del persona, al tratarse de un equipo de desarrollo bastante reducido, compuesto únicamente por una persona, las precauciones que se han tomando, han sido las de contemplar las posibles bajas por enfermedad o indisponibilidad por parte del personal, dentro de los plazos estipulados de realización del proyecto, intentando ajustar los plazos siempre todo lo posible, pero dando siempre cierto margen en los tiempos (teniendo siempre en cuenta que los costes y tiempos se encuentren siempre dentro de límites razonables), de tal forma que un posible contratiempo de este tipo, no altere de sobremanera los tiempos de entrega, prologándolos demasiado en el tiempo.

Parte II

Desarrollo

Capítulo 3

Análisis de Requisitos

En esta sección se presenta el catálogo de requisitos del sistema de información. Para ello se detallarán los actores del sistema, los requisitos funcionales, los requisitos de información, los requisitos no funcionales y las reglas de negocio. Luego se describen las diferentes alternativas tecnológicas y el análisis de la brecha entre los requisitos planteados y la solución base seleccionada.

3.1. Catálogo de actores

Dentro del funcionamiento de la aplicación, intervienen una serie de actores, los cuales son los encargados de activar cada una de las operaciones de las que dispone la aplicación. Estos actores, los cuales intervienen dentro de los casos de uso de la aplicación, son los que se describen a continuación:

3.1.1. Administrador del sistema

ACT-001	**Administrador del sistema**
Descripción	El actor administrador representa a toda aquella persona con acceso a la aplicación, que cuente con permisos de superusuario. Este actor es el encargado de la configuración y administración de la aplicación.

Tabla 3.1: Actor - 001 - Administrador del sistema

3.1.2. Usuario externo

ACT-002	**Usuario externo**
Descripción	El actor Usuario externo (persona o sistema) representa a toda aquella persona externa al proyecto donde va a ser usada la aplicación, de forma que vaya a hacer uso de los recursos públicos de ella, cuando acceda a la misma.

Tabla 3.2: Actor - 002 - Usuario externo

3.2. Requisitos funcionales

A continuación se realiza una descripción de los distintos requisitos funcionales que deberá de cumplir la aplicación. Se van a describir cada uno de estos requisitos funcionales como casos de uso, donde mostraremos los actores implicados en el desarrollo de los mismos, así como las precondiciones, postcondiciones y los posibles escenarios alternativos que se puedan dar.

Primeramente vamos a mostrar los diagramas de Casos de Uso, para posteriormente pasar a la descripción de cada uno de estos. Los casos de usos los dividiremos en dos grupos, aquellos que pertenecen al subsistema del administrador (las operaciones que el administrador y solo el administrador debe poder hacer) y el subsistema perteneciente al usuario externo (estas operaciones podrán ser realizadas por cualquier usuario).

3.2.1. Diagramas de Casos de Uso

Casos de Uso del Usuario externo

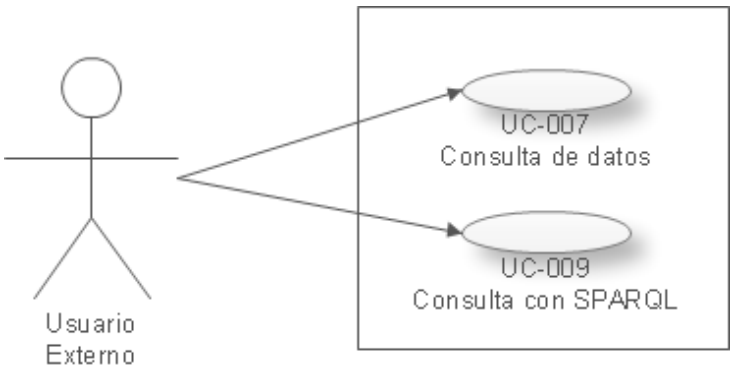


Figura 3.1: Diagrama de CU del Usuario externo

Casos de Uso del Administrador

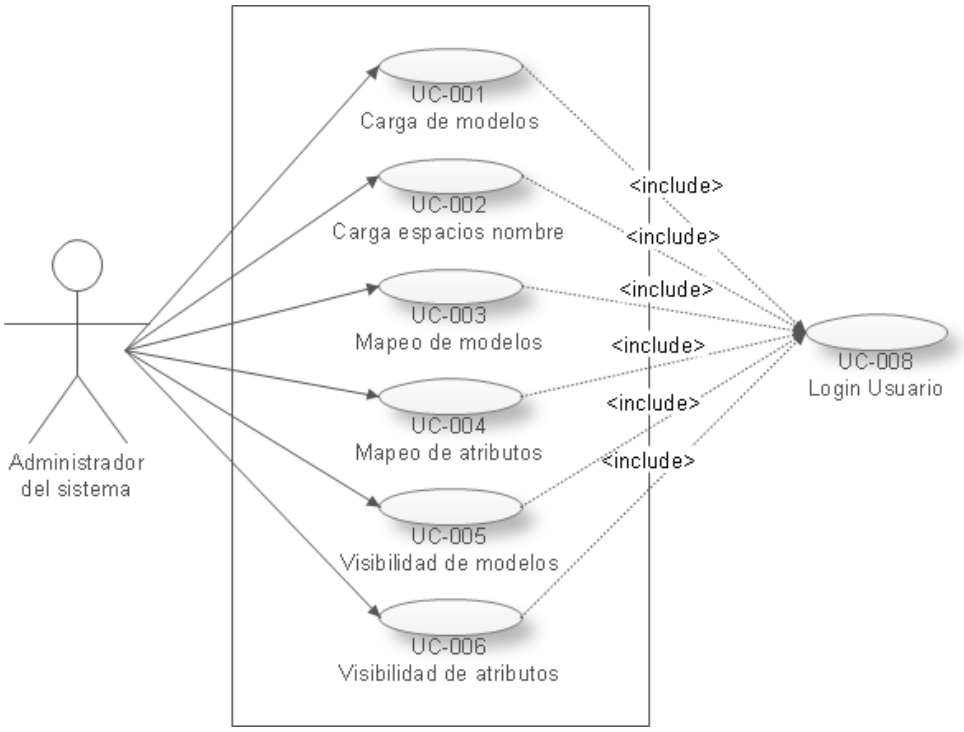


Figura 3.2: Diagrama de CU del Administrador

3.2.2. Descripción de los Casos de Uso

En este apartado, realizamos una descripción a fondo de cada uno de los Casos de Uso mostrados anteriormente. Para cada uno de ellos, indicamos los actores que intervienen en el, la descripción del mismo, posibles flujos alternativos que puedan darse, y así como la descripción de precondiciones y postcondiciones que deberán de cumplirse.

Carga de modelos

UC-001	**Carga de modelos**
Descripción	Carga de modelos y fields del proyecto Django donde se va a usar la aplicación.
Referencia	Tabla A.1

Tabla 3.3: Descripción Caso de Uso - 001 - Carga de modelos

Carga de espacios de nombres

UC-002	**Carga de espacios de nombres**
Descripción	Se debe de realizar una serie de procedimientos los cuales permitan a la aplicación cargar la estructura de los diferentes namespaces existentes, además de permitir actualizar y eliminar estos.
Referencia	Tabla A.2

Tabla 3.4: Descripción Caso de Uso - 002 - Carga de espacios de nombres

Mapeo de modelos

UC-003	**Mapeo de modelos**
Descripción	Este caso de uso describe el procedimiento mediante el cual el administrador del sistema, describe en la aplicación la relación de cada uno de los modelos del proyecto Django, con las entidades de los distintos namespaces que existen en la aplicación.
Referencia	Tabla A.3

Tabla 3.5: Descripción Caso de Uso - 003 - Mapeo de modelos

Mapeo de los atributos

UC-004	**Mapeo de los atributos**
Descripción	Este caso de uso describe el procedimiento mediante el cual el administrador del sistema, describe en la aplicación la relación de cada uno de los atributos de los modelos del proyecto Django, con las propiedades de las entidades de los distintos namespaces que existen en la aplicación.
Referencia	Tabla A.4

Tabla 3.6: Descripción Caso de Uso - 004 - Mapeo de los atributos

Establecer visibilidad de modelos

UC-005	**Establecer visibilidad de modelos**
Descripción	Este caso de uso describe el procedimiento que debe de seguir el administrador del sistema, para especificar aquellos modelos del proyecto Django de los que podrán publicarse los datos.
Referencia	Tabla A.5

Tabla 3.7: Descripción Caso de Uso - 005 - Establecer visibilidad de modelos

Establecer visibilidad de los atributos

UC-006	**Establecer visibilidad de los atributos**
Descripción	Este caso de uso describe el procedimiento que debe de seguir el administrador del sistema, para especificar aquellos atributos de los modelos del proyecto Django de los que podrán publicarse los datos.
Referencia	Tabla A.6

Tabla 3.8: Descripción Caso de Uso - 006 - Establecer visibilidad atributos

Publicación de los datos

UC-007	**Consulta de datos**
Descripción	Muestra los datos por pantalla usando los estándares RDF/XML, RDF/Ntriples, RDF/Turtle, RDFa o Microdata, existentes para la publicación de datos en internet.
Referencia	Tabla A.7

Tabla 3.9: Descripción Caso de Uso - 007 - Consulta de datos

Inicio de sesión

UC-008	**Inicio de sesión**
Descripción	Realiza el login del usuario dentro de la aplicación web.
Referencia	Tabla A.8

Tabla 3.10: Descripción Caso de Uso - 008 - Inicio de sesión

Consulta con SPARQL

UC-009	**Consulta con SPARQL**
Descripción	Debe de permitirse al usuario realizar consultas sobre los datos utilizando el lenguaje SPARQL.
Referencia	Tabla A.9

Tabla 3.11: Descripción Caso de Uso - 009 - Consulta con SPARQL

3.2.3. Diagramas de Secuencia

Una vez hemos descrito cada uno de los casos de uso que deberá incluir la aplicación, el siguiente paso es plasmar gráficamente el comportamiento de estos. De esta forma, se muestran a continuación, los diagramas de secuencia de cada uno de los casos de uso de la aplicación.

UC-001

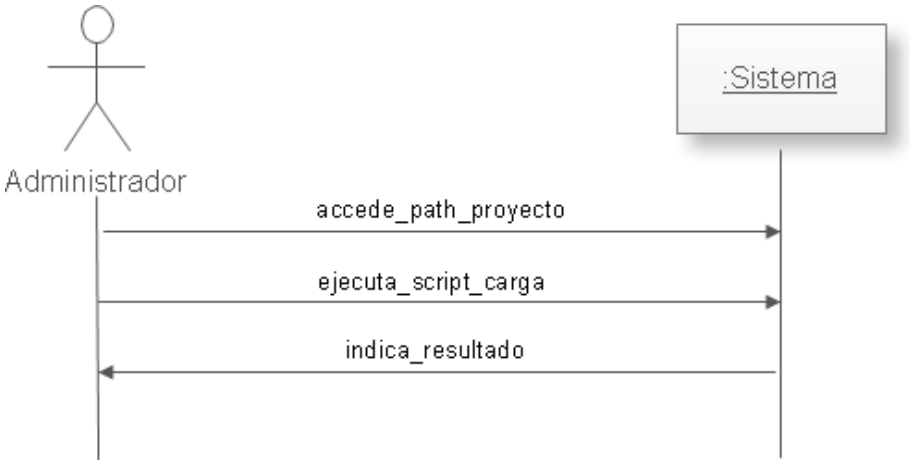


Figura 3.3: Diagrama de Secuencia CU-001

UC-002

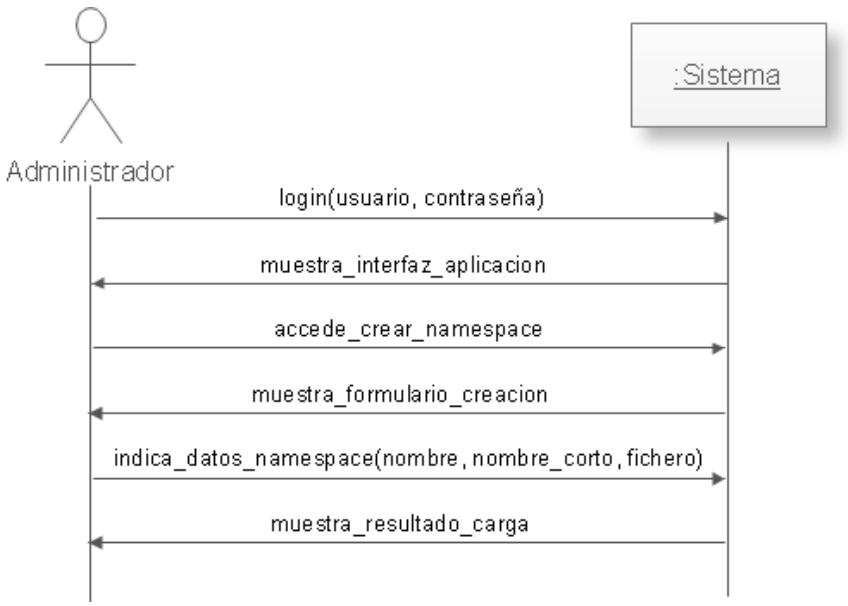


Figura 3.4: Diagrama de Secuencia CU-002

UC-003

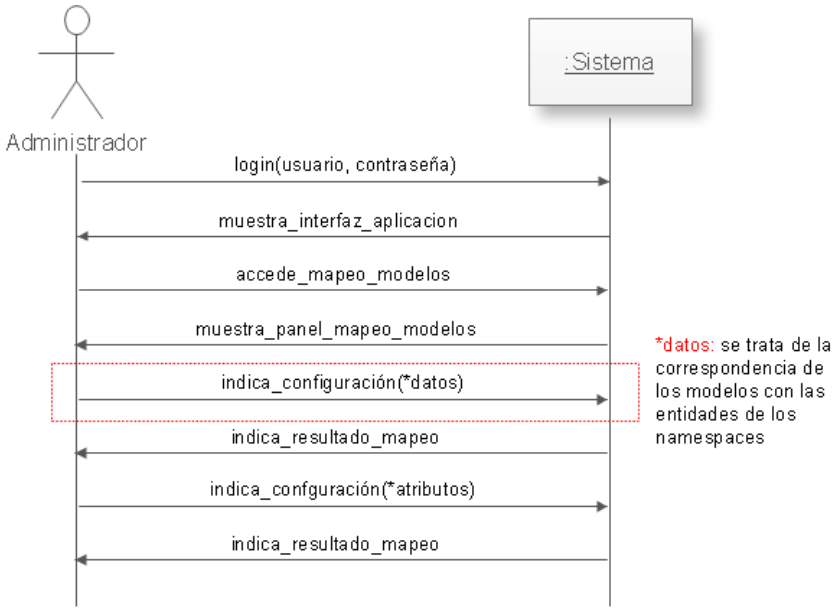


Figura 3.5: Diagrama de Secuencia CU-003

UC-004

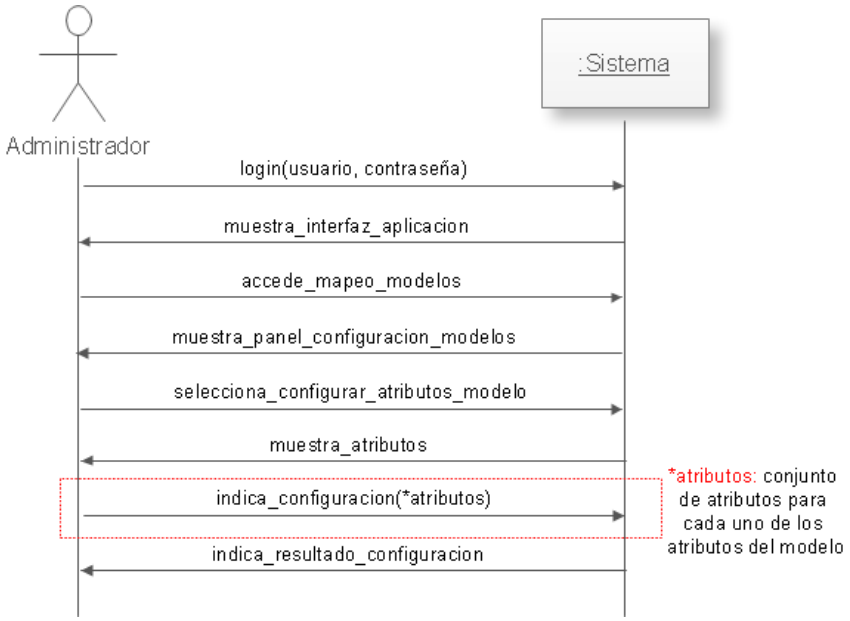


Figura 3.6: Diagrama de Secuencia CU-004

UC-005

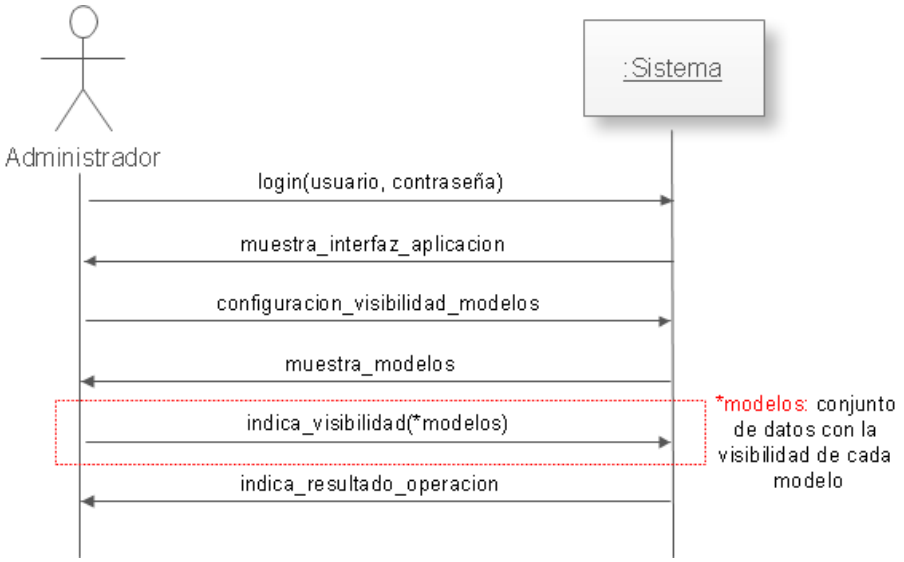


Figura 3.7: Diagrama de Secuencia CU-005

UC-006

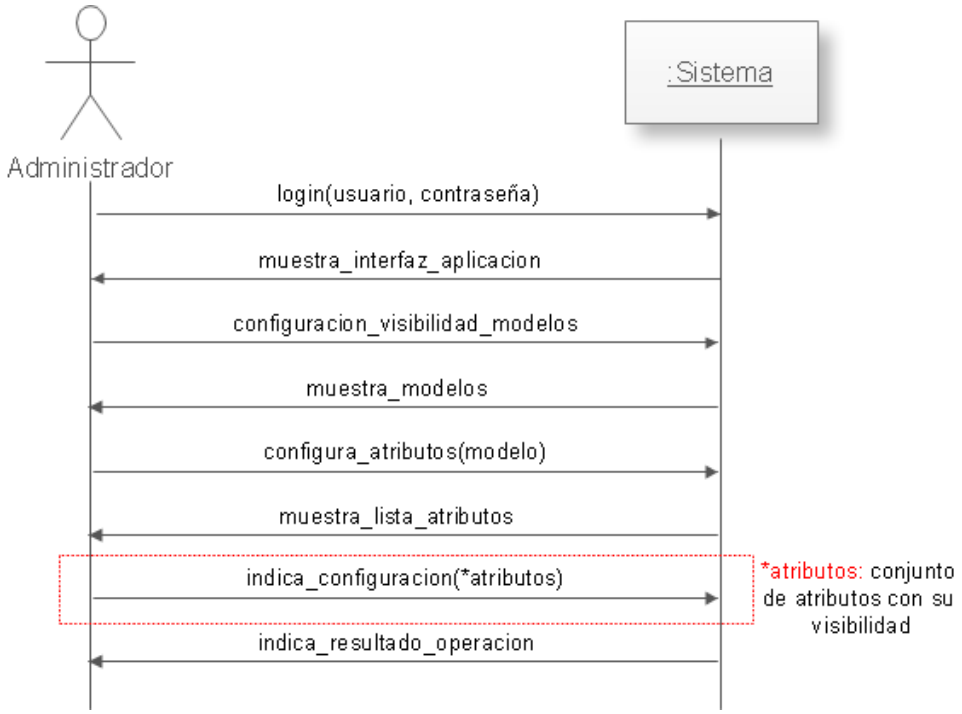


Figura 3.8: Diagrama de Secuencia CU-006

UC-007

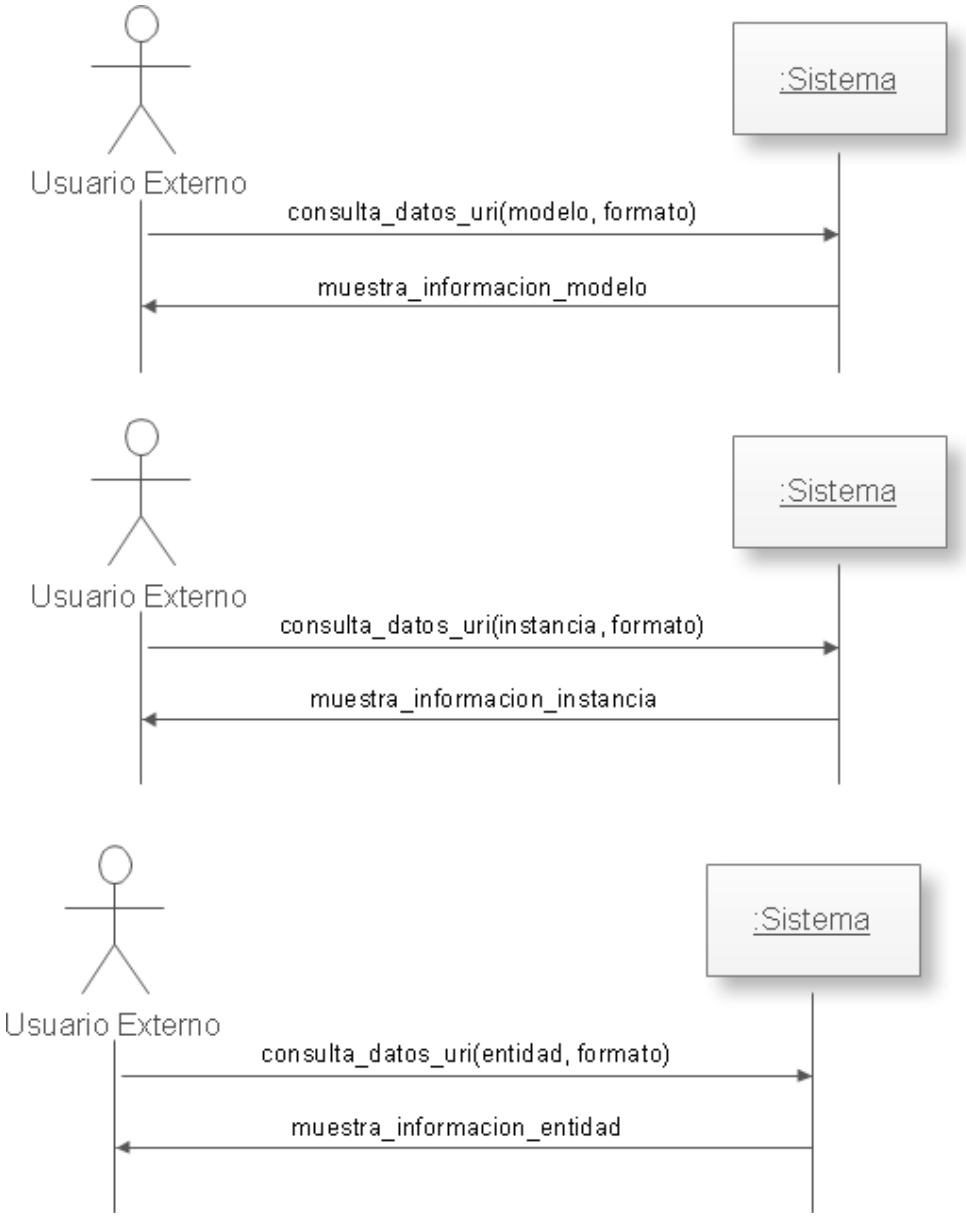


Figura 3.9: Diagrama de Secuencia CU-007

UC-008

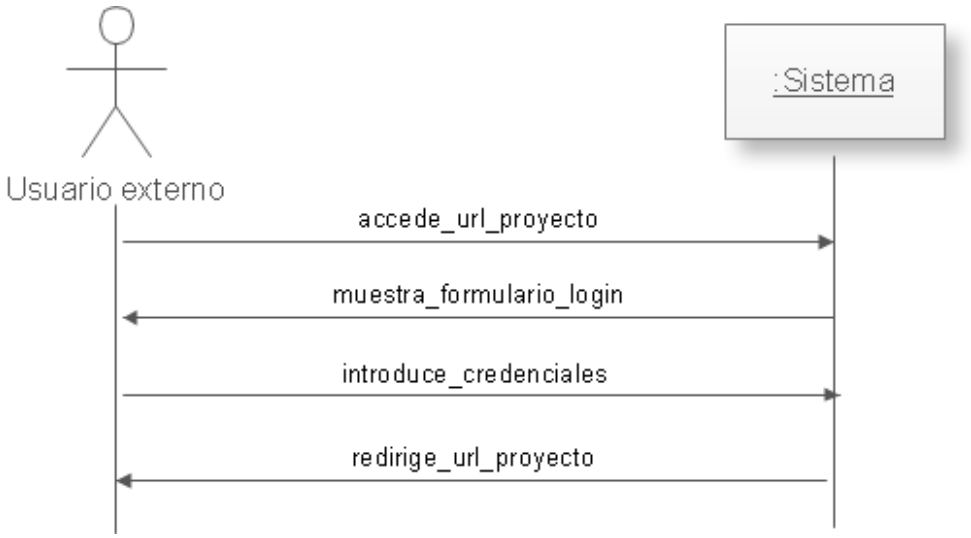


Figura 3.10: Diagrama de Secuencia CU-008

UC-009

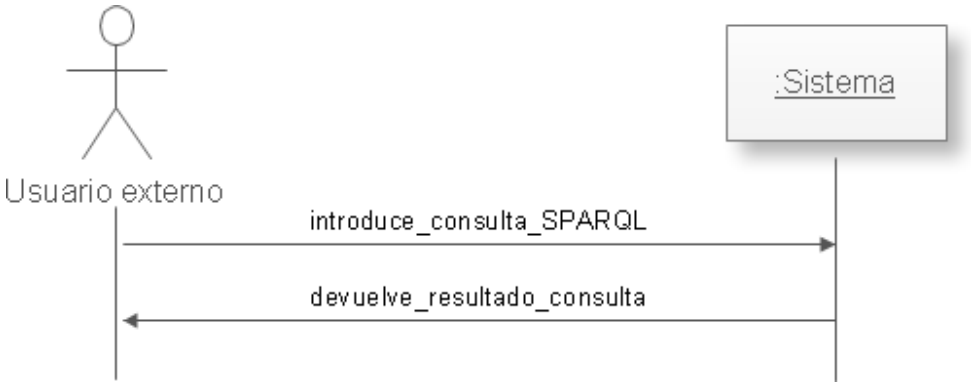


Figura 3.11: Diagrama de Secuencia CU-009

3.3. Requisitos de información

En esta sección se describen los requisitos de gestión de información (datos) que el sistema debe gestionar. Para ello, utilizando el formato de especificación propuesto por UML, detallamos a continuación, cada uno de los requisitos de información de la aplicación:

3.3.1. Descripción de los requisitos de información

IRQ-001	**Información de los namespaces**
Descripción	Se debe almacenar la información acerca de los distintos namespaces que almacena la aplicación, de tal forma que se tenga la información acerca del namespace, de las entidades que componen a este, y de las propiedades que componen a la entidad.
Referencia	Tabla A.10

Tabla 3.12: Descripción IRQ - 001 - Información de los namespaces

IRQ-002	**Información de los modelos**
Descripción	Se debe almacenar la información acerca de los distintos modelos que componen al proyecto Django, de tal forma que se tenga la información acerca del modelo, y de los fields que componen al modelo.
Referencia	Tabla A.11

Tabla 3.13: Descripción IRQ - 002 - Información de los modelos

3.3.2. Diagrama conceptual de datos

Una vez que hemos descrito cada uno de los datos que necesitamos que nuestra aplicación Django almacene, mostramos un diagrama conceptual de datos UML, donde se puede apreciar como se relacionan cada uno de estos datos entre ellos. Además, a parte de las distintas clases que almacena la aplicación, podremos identificar, los atributos, relaciones y restricciones adicionales.

Si observamos el diagrama de clases, está divido en dos partes, numeradas cada una de ellas. La parte número 1, se corresponde con el requisito de información *IRQ-001* (Tabla A.10), y el apartado número 2, se corresponde con los requisitos de información descritos en *IRQ-002* (Tabla A.11).

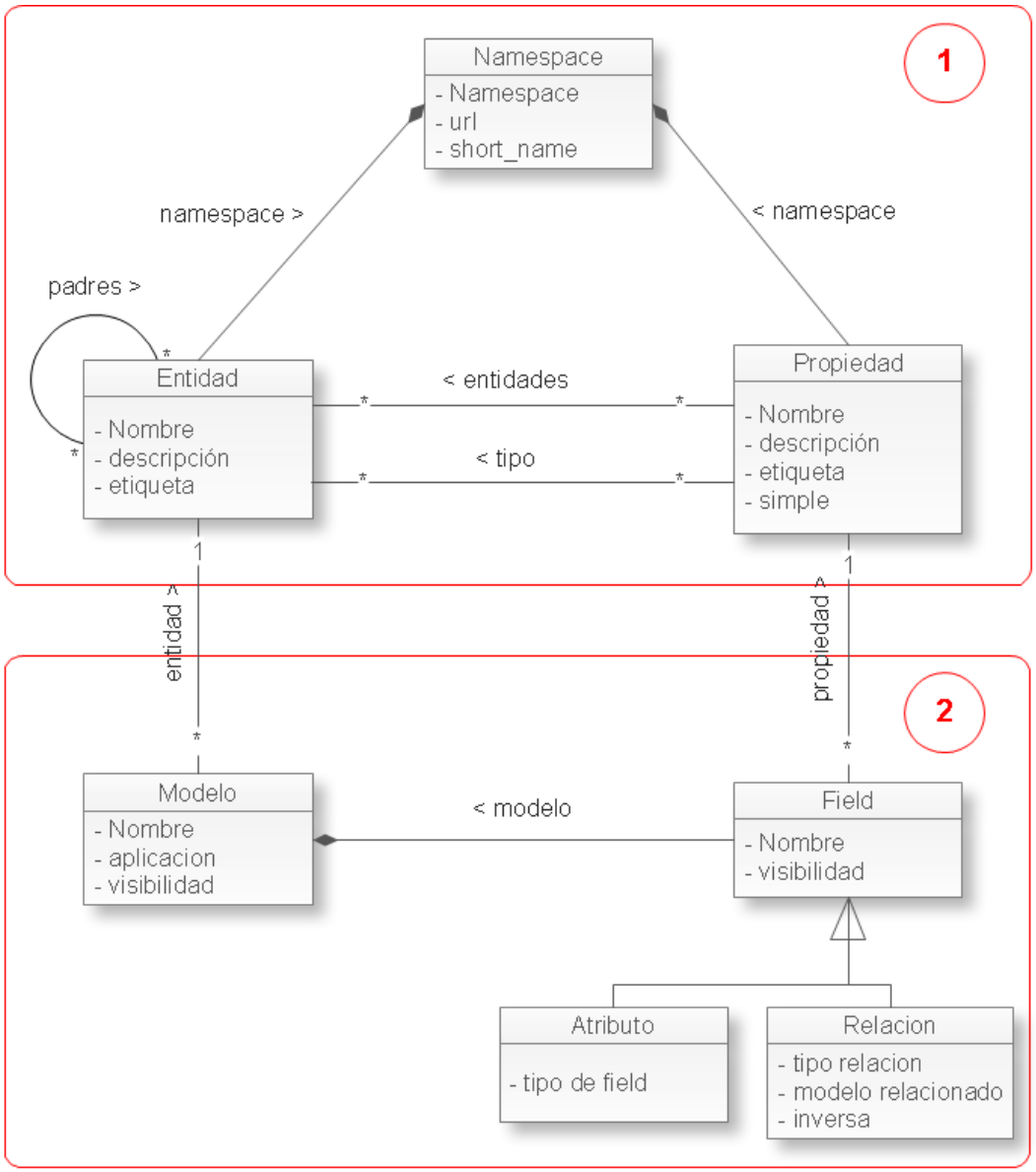


Figura 3.12: Diagrama conceptual de datos UML

3.4. Requisitos no funcionales

A continuación se detalla una descripción de otra serie requisitos, relacionados con la calidad del software, conocidos como requisitos no funcionales, que el sistema deberá satisfacer. Estos requisitos son los siguientes:

- **Seguridad:** dicha aplicación no será accesible a usuarios no autorizados y solo se publicarán

aquellos datos expresamente indicados.

- **Estándares de las organizaciones:** deberá de adaptarse a cada uno de los estándares impuestos por las distintas organizaciones desarrolladoras de las especificaciones para la apertura de datos.
- **Portabilidad:** la aplicación debe de estar diseñada, haciendo uso siempre de tecnologías y herramientas independientes de la plataforma, de forma que pueda usarse bajo la gran mayoría de sistemas operativos actuales.
- **Mantenibilidad:** Esta aplicación se encuentra desarrollada como un módulo a parte, de forma que no sea dependiente de un proyecto en cuestión. Además, la aplicación deberá de estar perfectamente documentada, y se usará la guía de buenas maneras para la escritura de código Python PEP8 ([RW12]), de tal forma que el código sea lo suficientemente claro.
- **Extensibilidad:** La aplicación debe de ser fácilmente extensible, pudiéndose adaptar fácilmente a nuevos estándares para la apertura de datos.
- **Interfaz:** La aplicación debe de poseer una interfaz de configuración sencilla e intuitiva, de tal forma que un usuario no experimentado, pueda aprender a manejarla sin dificultades.
- **Entorno tecnológico:** La aplicación forzosamente debe de estar elaborada bajo el lenguaje de programación Python y el framework Django, ya que es el principal objetivo de este proyecto, proporcionar una herramienta para la apertura de datos en aplicaciones Django. El resto de tecnologías a usar, es indiferente, siempre y cuando respeten los requisitos que se acaban de imponer.

3.5. Reglas de negocio

A lo largo del desarrollo del sistema, además de los distintos requisitos que ha de cumplir la aplicación, en lo referente a funcionalidades, información o propiedades que deben de cumplir el sistema, hay que tener en cuenta las denominadas reglas de negocio, es decir, el conjunto de restricciones, normas o políticas de la organización que deben ser respetadas por el sistema. A continuación se detallan, una lista de las distintas reglas de negocio que debe de cumplir la aplicación:

1. El sistema no debe de permitir que un usuario que no posea los permisos de administrador, pueda consultar datos, los cuales están declarados como privados.
2. El sistema no permitirá acceder al apartado de configuración de la publicación de datos a usuarios que no posean credenciales de administrador.

3.6. Estudio de alternativas tecnológicas

La finalidad de este apartado, sería de comentar las diferentes alternativas tecnológicas que existen, las cuales nos permitirían obtener el producto que queremos desarrollar, comentando

posibles pros y contras de cada una de estas, de forma que nos ayude a decidirnos por cual de ellas sería la mejor. Aunque en nuestro caso, no será necesario definir estas, ya que el principal objetivo de este proyecto informático, es el de dotar de una cierta cualidad (apertura de datos en internet) a una tecnología en concreto (Django), por lo que no tendría sentido en un principio estudiar otras alternativas tecnológicas.

3.7. Análisis GAP

Este proyecto no se encuentra basado en ningún software base, ya que vamos a realizar una aplicación software completamente desde cero. Si cabe comentar, que el proyecto se va a realizar en Python, para el framework Django, por lo que si considerásemos el framework Django como un software base a partir del cual vamos a desarrollar nuestro proyecto, este nos proporciona una serie de herramientas las cuales nos simplificarán el desarrollo del mismo. Entre todas estas herramientas que comentamos, cabría destacar las siguientes:

- Herramientas para el acceso a los distintos modelos de las aplicaciones. Esto nos facilitará enormemente la introspección que deberemos de realizar, para captar los modelos de los que se encuentran compuestos los proyectos Django.
- Herramientas para la gestión de las sesiones y del log de la aplicación.
- Herramientas para la generación de formularios, incluyendo comprobaciones de seguridad ante distintos tipos de ataques.
- Manejador propio para las bases de datos, con un lenguaje propio de consultas, de tal forma que podemos realizar la aplicación de forma que esta sea completamente independiente del SGBD que use el proyecto donde vaya a ser implantada la aplicación.
- Una comunidad, la cual proporciona un soporte continuo del framework Django.

El proyecto aunque no toma como base para su desarrollo la plataforma D2Rq, genera ficheros de configuración para ésta, por lo que para el desarrollo de este módulo se ha debido de realizar un estudio del funcionamiento de esta plataforma, y adaptar la generación del fichero de configuración a ésta. La plataforma D2Rq, es de código abierto, y proporciona un sistema para el acceso a los datos de bases de datos relacionales, como si se tratasen de grafos RDF virtuales de solo lectura. De esta forma, ofrece a los usuario acceso mediante RDF al contenido relacional de las bases de datos, sin la necesidad de tener que replicar estos sobre un almacenamiento en formato RDF.

Capítulo 4

Diseño del Sistema

En este capítulo se recoge la arquitectura general del sistema de información, el diseño de la interfaz de usuario, el diseño físico de datos, el diseño de componentes software y la parametrización del software base.

4.1. Diseño de la arquitectura

En esta sección se define la arquitectura general del sistema de información, especificando las distintas particiones físicas del mismo, la descomposición lógica en subsistemas de diseño y la ubicación de cada subsistema en cada partición, así como la especificación detallada de la infraestructura tecnológica necesaria para dar soporte al sistema de información.

4.1.1. Arquitectura física

En este apartado, describimos los principales componentes hardware que forman la arquitectura física de nuestro sistema, recogiendo por un lado los componentes de servidor y los componentes de sistemas externos con los que colabora nuestro sistema y por otro, los componentes hardware de cliente.

Este proyecto no precisa de una configuración física específica para su funcionamiento, ya que se trata de un plugin para proyectos Django. Bastará únicamente un servidor que cumpla con los requisitos mínimos necesarios para alojar proyectos Django (para más información, se puede consultar la documentación oficial en la web del proyecto Django [Fou13a]). Es más, esta configuración dependerá en toda medida del proyecto Django donde se hará uso de este plugin, ya que este será el que marque principalmente los requisitos mínimos de los sistemas físicos, en función de su carga de trabajo, disponibilidad, capacidad de respuesta u otros requisitos impuestos para dicho software.

4.1.2. Arquitectura lógica

La arquitectura lógica del sistema está formada por los elementos software (servicios, aplicaciones, librerías, frameworks, etc.) que componen el software base, más el software desarrollado para cumplir los requisitos de la aplicación. También, se recogen los componentes de sistemas externos con los que interactúa nuestro sistema, así como los componentes software del lado cliente.

De esta forma, en este apartado de arquitectura lógica, vamos a describir todos los elementos software que componen el software base de la aplicación, además del propio software a desarrollar. De esta forma, podemos citar los siguientes componentes software:

- Primeramente, como acabamos de comentar, nos encontramos el elemento software que compone a nuestro proyecto, que no es más que un plugin para añadir a proyectos existentes, el cual permite al usuario la publicación de datos en internet siguiendo diferentes estándares conocidos.
- Este plugin está desarrollado, para un tipo de aplicaciones en concreto, aquellas que están desarrolladas haciendo uso del framework Django, por lo que el plugin está desarrollando de igual forma, haciendo uso de Django. Por lo que será necesario que la máquina posea una copia instalada de dicho framework.
- El framework Django, es un framework cuya finalidad es el desarrollo rápido de aplicaciones web, haciendo uso del patrón MVC. Este framework está desarrollado haciendo uso de la tecnología Python, por lo que es requisito indispensable que toda máquina en la que vaya a usarse dicho proyecto, tenga soporte para Python.
- A su vez, dicho proyecto estará alojado en algún servidor, ya sea Apache (nuestro caso,) o cualquiera de los múltiples servidores disponibles a día de hoy. Un requisito de cualquiera de estos servidores, será el que tenga soporte para python (en el caso de Apache podemos usar mod_python o wsgi).
- Por último, el sistema operativo donde se vaya a instalar todos los componentes anteriores, debe de tener soporte tanto para el servidor web donde vamos a incluir nuestro proyecto, como para la tecnología Python, ya que el resto de componentes comentados, funcionan sobre estas dos piezas. Por suerte, Python es un lenguaje de programación multiplataforma, el cual tiene soporte para los principales sistemas operativos existentes en la actualidad. Además, existen multitud de servidores web, como el comentado anteriormente (Apache), que poseen soporte también en los principales sistemas operativos existentes.
- En otro lugar, encontraremos aquellas tecnologías del lado del cliente, que también forma parte del proyecto o que intervienen en él. La tecnología que vamos a utilizar en el proyecto del lado del cliente es JavaScript. Se conoce por del lado del cliente, porque esto se ejecuta en la máquina del cliente, y no en el servidor. Además, también utilizaremos tecnologías como CSS para los estilos del proyecto.

A continuación, se muestra un gráfico (Figura 4.1) donde se puede apreciar mejor, las distintas capas que acabamos de describir anteriormente, y cómo interactuarían estas entre ellas,

desde el nivel más bajo (el sistema operativo) hasta la capa capa más alta, compuesta por el proyecto Django y el plugin que estamos desarrollando en este proyecto.

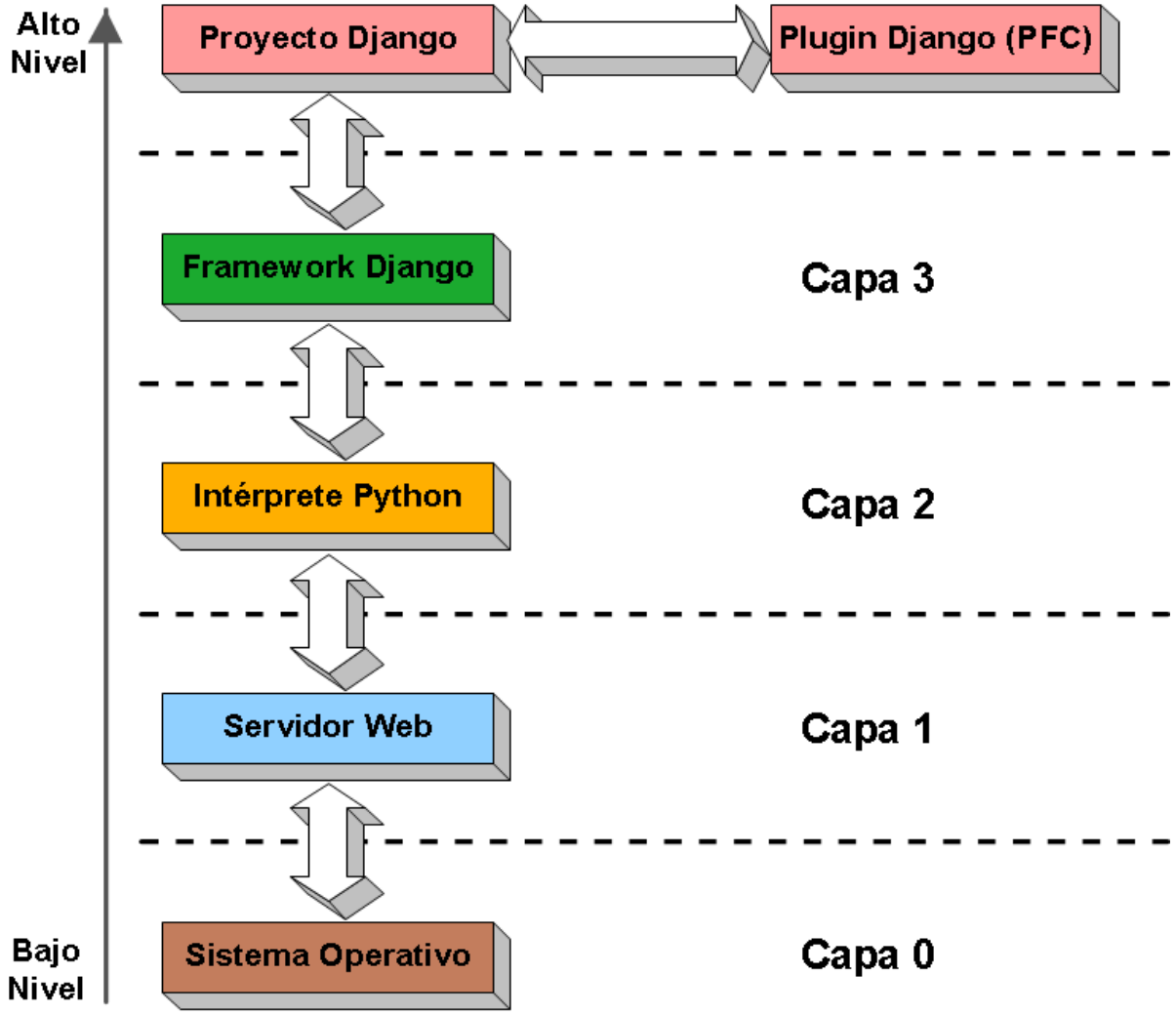


Figura 4.1: Arquitectura lógica de cada uno de los elementos software

Estructura del proyecto

Una vez que hemos visto como es la arquitectura lógica del proyecto, y en cada una de las capas en las que podríamos definir el mismo, de forma que cada una de estas interactúen únicamente con aquellas que están inmediatamente por encima o por debajo de ella misma, toca describir en mayor profundidad la capa situada más arriba de la jerarquía, la cual se corresponde a la aplicación que vamos a desarrollar.

Toda aplicación contenida en un proyecto Django, posee una estructura común, de forma que puedan identificarse cada uno de los elementos que componentes de forma inmediata. Esta

estructura se puede apreciar a modo orientativo, en la Figura 4.2, donde se muestran cada uno de los componentes principales de una aplicación. A continuación se describen la finalidad de cada uno de estos apartados:

- **Models:** se trata de un fichero llamado *models.py* o un directorio llamado *models*, el cual contendrá diferentes ficheros python, en donde se describirán cada uno de los modelos que componen a la aplicación Django.
- **Forms:** se trata de un fichero llamado *forms.py* o un directorio llamado *forms*, el cual contendrá diferentes ficheros python, en donde se describirán cada uno de los diferentes tipos de formularios Django.
- **Views:** se trata de un fichero llamado *views.py* o un directorio llamado *views*, el cual contendrá diferentes ficheros python, donde se describirán cada uno de los modelos que componen a la aplicación Django.
- **Templates:** se trata de un directorio común para todo el proyecto Django, en el cual se contendrán cada una de las plantillas las cuales se usarán por las vistas para mostrar la información en el formato especificado en la plantilla.
- **Static:** es un directorio donde se almacenan todos los elementos, tanto media, como estilos, javascript, etc. . . que se usará en dicha aplicación.
- **Url's:** es un fichero llamado *urls.py* o un directorio llamado *urls*, que contendrá una serie de ficheros python, donde se definirán cada una de las URIs de las que dispondrá nuestro proyecto, y se asociarán con las vistas que se activará al llamar a dicha URI.

Estos componentes que hemos comentado son solo los principales, ya que además de estos componentes, pueden existir muchos otros para otras finalidades.

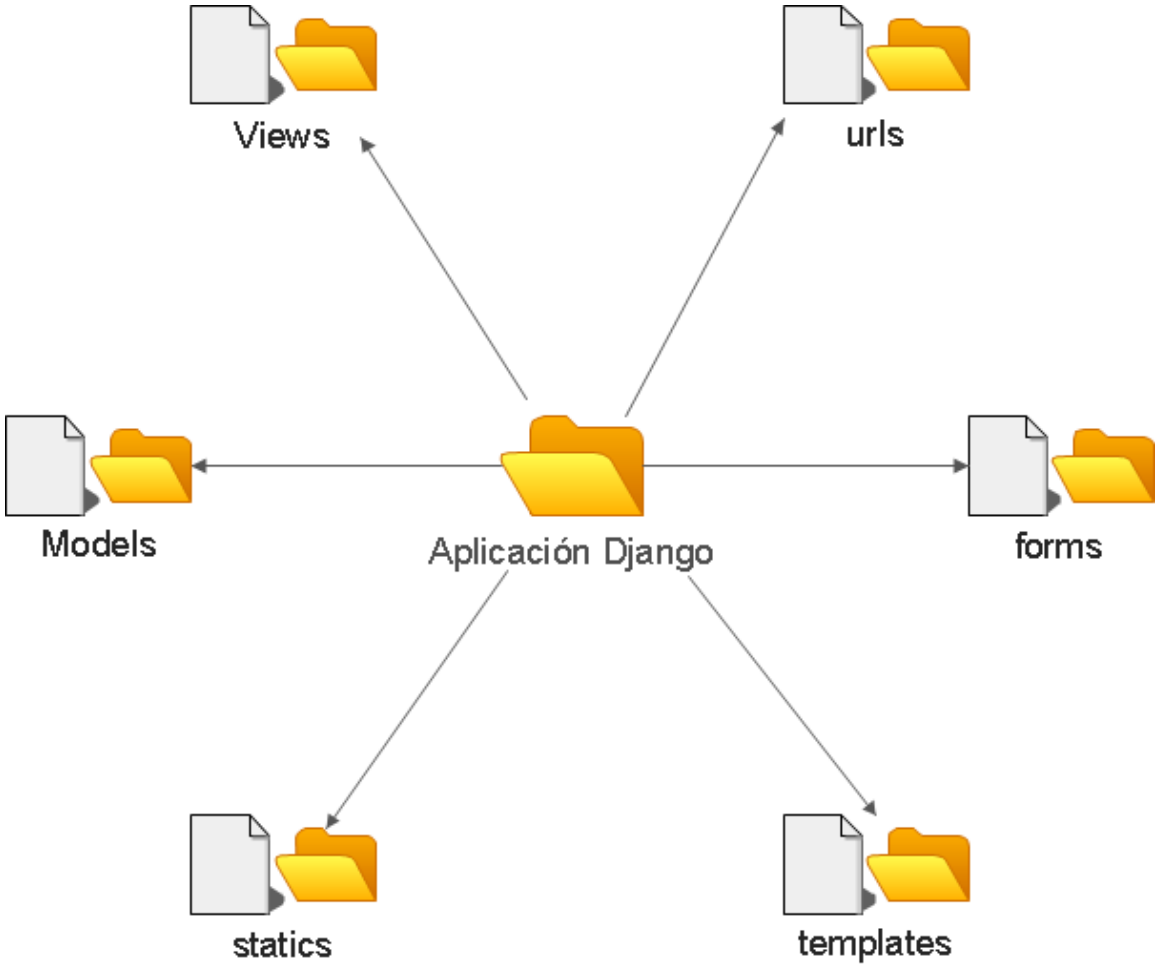


Figura 4.2: Arquitectura lógica de una aplicación Django

4.1.3. Arquitectura de diseño

La arquitectura de diseño especifica la forma en que los artefactos software de más bajo nivel, interactúan entre sí para lograr el comportamiento deseado en el sistema. Utilizaremos el patrón arquitectónico Layers (Capas), con el cual estructuramos el sistema en un número apropiado de capas, de forma que todos los componentes de una misma capa trabajan en el mismo nivel de abstracción y los servicios proporcionados por la capa superior utilizan internamente los servicios proporcionados por la capa inmediatamente inferior.

Es este caso, para la realización del proyecto, estamos utilizando Django, que se trata de un framework que aunque no lo sigue fielmente, está basado en el patrón de diseño MVC (Modelo-Vista-Controlador), de tal forma que en toda aplicación Django se hace una separación entre los distintos artefactos software, los cuales interactuando entre sí, conforman el comportamiento esperado del software. Por consiguiente, tal y como se aprecia en la figura de más abajo (Figura 4.3) podemos decir que toda la lógica de una aplicación Django se encuentra dividida en los

siguientes artefactos:

- Los modelos.
- Las URL's.
- Las views (o vistas).
- Los templates (o plantillas).

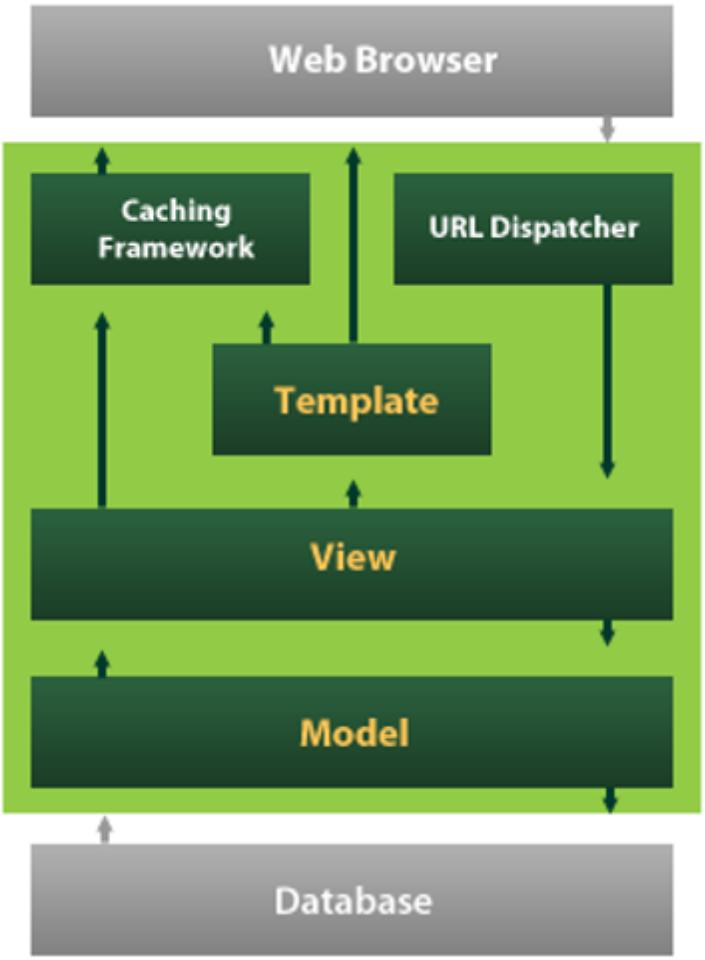


Figura 4.3: Patrón de diseño usado en el framework Django (fuente: Cambioderuta)

A continuación, vamos a relacionar cada uno de estos artefactos con la capa con la que se corresponderían cada uno: presentación, negocio o integración.

Capa de presentación

Este grupo de artefactos software conforman la capa de presentación del sistema, incluyendo tanto los componentes de la vista como los elementos de control de la misma.

En el framework Django, esta capa se correspondería con los conocidos como *templates*. Mediante el uso de plantillas en Django, se consigue separar la presentación de un documento, de la obtención y tratamiento de los datos. En las plantillas se definen los rellenos y alguna lógica de control básica para determinar como mostrar los datos. Mediante las plantillas de Django, aunque el resultado más común pueda ser HTML, se puede generar cualquier tipo de documento basado en texto, donde presentar los datos.

Capa de negocio

Este grupo de artefactos software conforman la capa de negocio del sistema, incluyendo los elementos del modelo de dominio y los servicios (operaciones del sistema).

En el caso del framework Django, la capa de negocio estaría fuertemente ligada con los views (o las vistas, que se corresponden con los controladores en el patrón MVC). En las vistas de Django, se lleva a cabo el tratamiento de los datos, se consultan y se realizan operaciones de los mismos, y opcionalmente estos se envían a la capa de presentación (los templates de Django) donde se muestran al usuario. Si nos fijamos en la figura 4.3, podemos ver, que los controladores hacen un poco como nexo de unión entre los templates y los modelos (entre la capa de presentación y de integración).

Capa de integración

Este grupo de artefactos software conforman la capa de integración del sistema, incluyendo las clases de abstracción para el acceso a datos (BD o sistema de ficheros) o a sistemas heredados.

En esta capa, entran en juego diferentes elementos del framework Django. Primeramente podríamos nombrar a los modelos, estos no son otra cosa, sino clases Python, las cuales son una representación de las tablas que hay almacenadas en la base de datos, junto con sus características (restricciones, valores iniciales, etc), mediante esto se simplifica mucho la representación de los datos para el usuario, ya que los tratará como objetos de una clase.

Por otro lado, aparecen los managers, estos elementos son una interfaz la cual poseen todos los modelos de un proyecto Django (son configurables, ya que se pueden crear interfaces que muestren solo ciertos datos), a través de la cual, permite al usuario realizar consultas (desde sencillas hasta complejas), usando un lenguaje propio de consulta del framework (evitamos tener que hacer uso de SQL, aunque también puede utilizarse este). Con esto, llegamos a la tercera característica, ya que al hacer uso de un lenguaje propio de consultas en Django, estas son válidas para cualquier sistema de bases de datos soportado por Django (este posee distintos backends

para distintos motores de bases de datos), de forma que una misma aplicación funcione con bases de datos PostgreSQL, Oracle, etc . . . sin necesidad de realizar cambios en las consultas.

Servicios transversales

Este grupo de artefactos software pueden ser usados por elementos de cualquiera de las capas del sistema y fundamentalmente proporcionan servicios relacionados con requisitos no funcionales (calidad).

Para nuestro caso, el framework Django posee determinadas características, las cuales ayudan al cumplimiento de determinados requisitos no funcionales de los que comentamos anteriormente (Sección 3.4), como son los siguientes:

- **Seguridad:** Django incorpora la gestión automática de sesiones, o la generación automática de formularios basados el modelos o definidos por el programador, incorporando a su vez medidas de seguridad (como por ejemplo contra ataques csrf).
- **Mantenibilidad:** la herramienta promueve a la escritura de un código fácil de mantener, a parte de estructurar las aplicaciones en partes bien diferenciables, de forma que se puedan localizar los distintos elementos fácilmente por cualquier desarrollador con algo de experiencia en Django, promueve además el uso de patrones de diseño (como puede ser el patrón DRY, Facade, etc).

  Además, por otro lado, el desarrollo del proyecto, promueve un compromiso de seguir unos códigos de buenas maneras definidos en [RW12].
- **Portabilidad:** el framework está desarrollado en Python, el cual se trata de un lenguaje multiplataforma, de tal manera que la aplicación será válida en cualquier sistema que soporte Python.

4.2. Diseño de la interfaz de usuario

En esta sección vamos a proceder a detallar las interfaces que existirán entre el sistema y el usuario de la aplicación, de forma que se pueda apreciar el aspecto de estas y el comportamiento que mostrarán las mismas. De forma que sea más fácil de apreciar para el lector, se van a realizar una serie de bocetos de cada una de las interfaces, y sobre estos bocetos, se definirá el comportamiento que tendrán.

Además, a continuación en la figura 4.4 se muestra el mapa de navegabilidad de la aplicación, del cual se realizarán bocetos de las siguientes interfaces:

- Listado de namespaces.
- Visibilidad de modelos y atributos.

- Visualización de los modelos.
- Mapeo de modelos.
- Mapeo de los atributos.

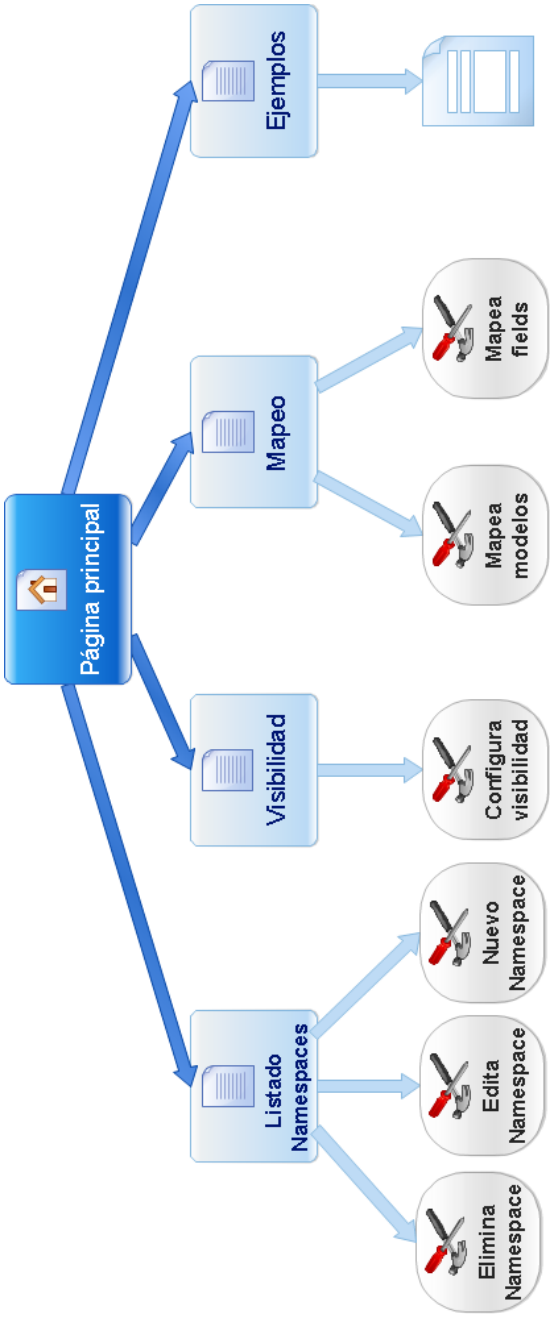


Figura 4.4: Mapa web de la aplicación

4.2.1. Listado de namespaces

A continuación en la figura 4.5 se muestra un boceto de la interfaz para el listado de namespaces de la aplicación.

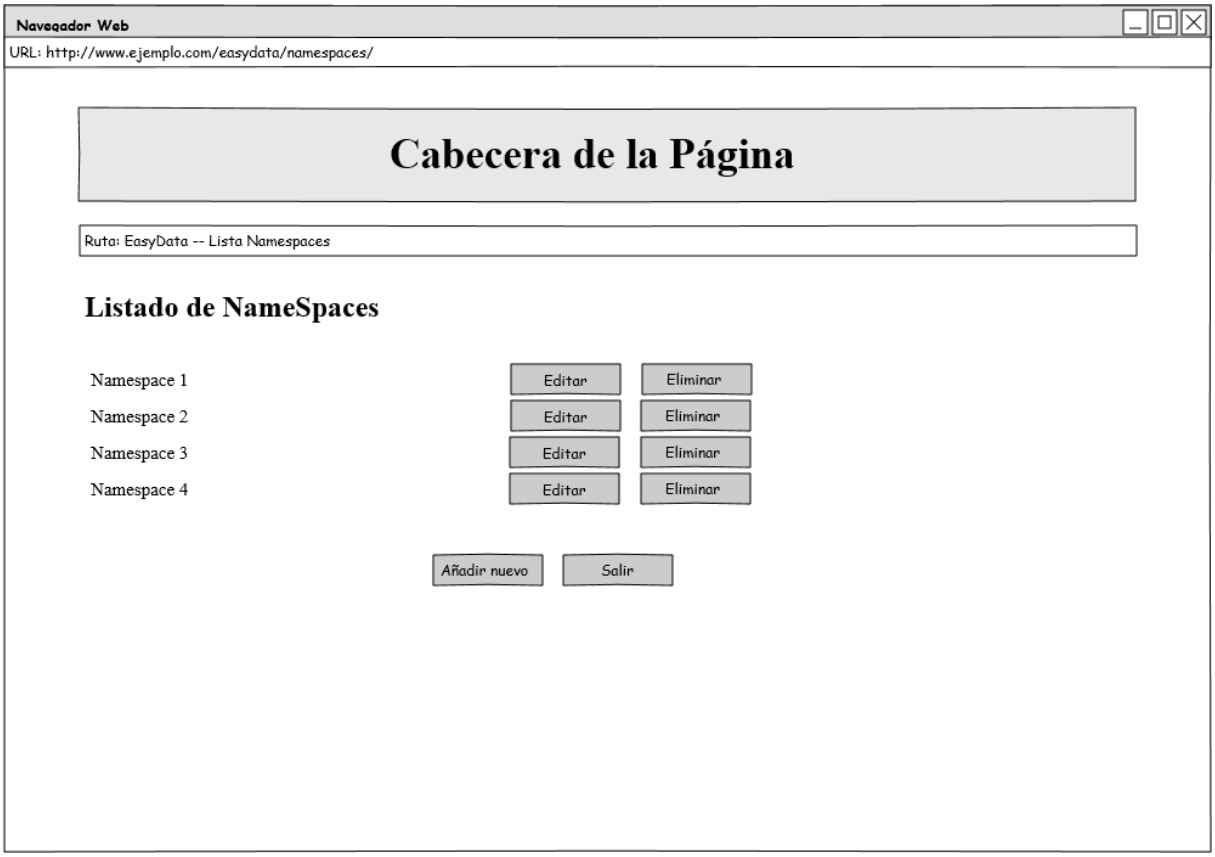


Figura 4.5: Boceto del listado de namespaces

Si observamos el boceto, podemos apreciar que se muestra un listado con cada uno de los namespaces existentes en la aplicación, los cuales pueden utilizarse para realizar la exportación de datos. Junto a cada uno de los nampesaces, aparecen dos botones, los cuales permiten editar las propiedades o actualizar la especificación de dicho namespace, o eliminar el mismo junto con todas sus entidades y propiedades.

4.2.2. Visibilidad de modelos y atributos

A continuación en la figura 4.6 se muestra un boceto de la interfaz para la configuración de la visibilidad de los modelos y los atributos.

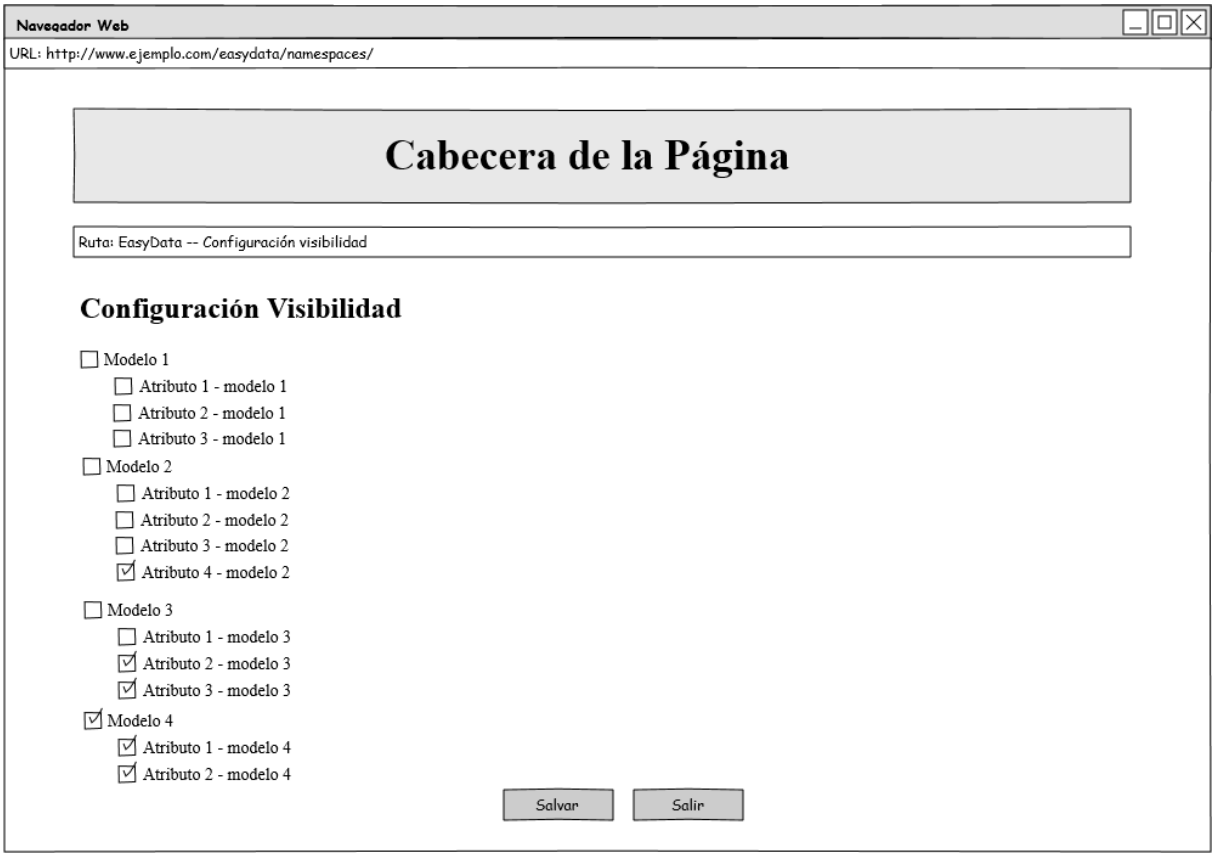


Figura 4.6: Boceto de la interfaz para la modificación de la visibilidad

Si observamos el boceto, la configuración de la visibilidad de los modelos y atributos del proyecto es bastante sencilla. Se muestra al usuario por pantalla un listado de todos los modelos y atributos que componen a cada uno de estos en forma de árbol. El usuario marcará aquellos modelos o atributos de estos que desee que sean visibles. De esta forma, si el usuario quiere que un modelo sea completamente visible, marcará la casilla del modelo, y además marcará todos los atributos del mismo. Por otro lado, si un usuario no desea que un determinado atributo de un determinado modelo no sea visible, únicamente deberá dejar la casilla correspondiente a dicho modelo sin marcar. Finalmente, si el usuario desea que un modelo no sea visible, bastará con marcar la casilla correspondiente al modelo, independientemente de los atributos del mismo que estén o no marcados.

4.2.3. Visualización Modelos

A continuación en la figura 4.7 se muestra un boceto de la interfaz donde se podrá visualizar la configuración realizada del mapeo de los diferentes modelos y atributos con los diferentes namespaces cargados en la aplicación.

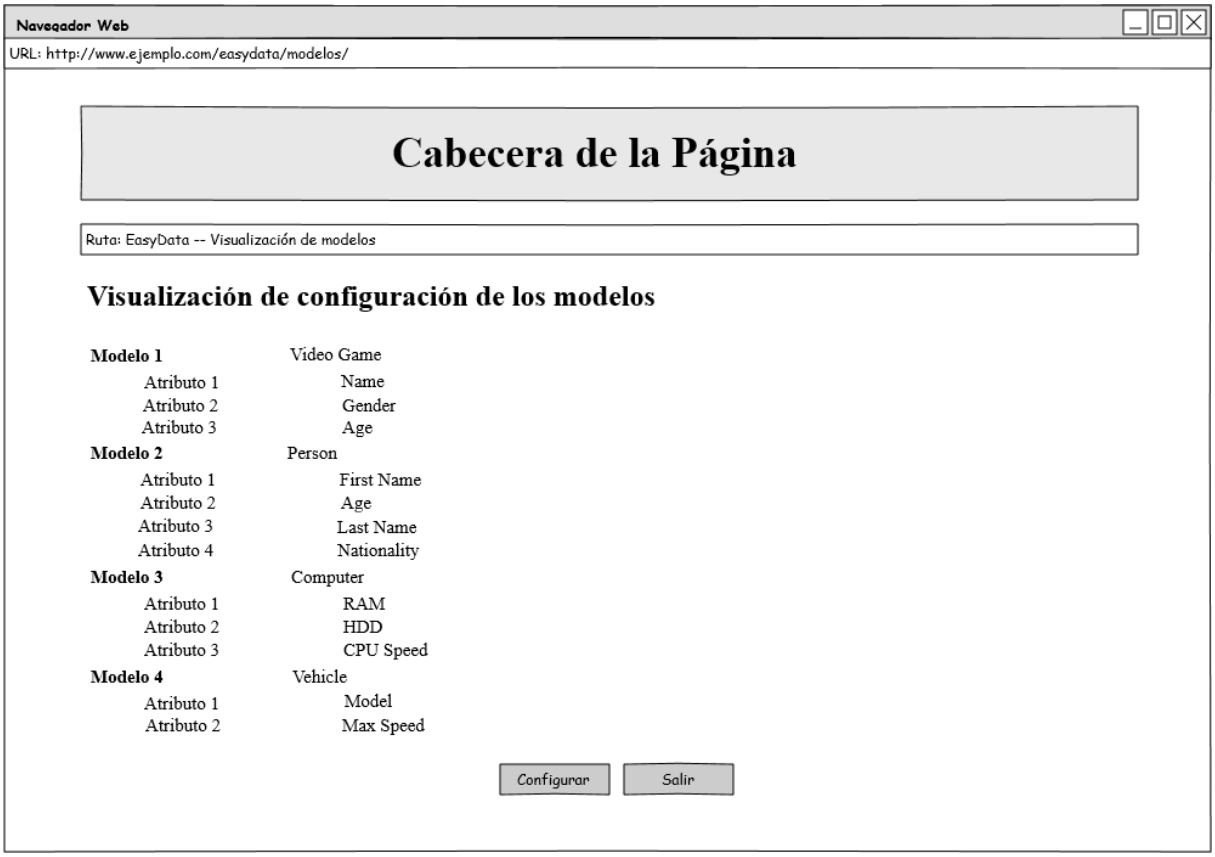


Figura 4.7: Boceto de la interfaz para visualización del mapeo de los modelos

Esta pantalla es plenamente informativa, dando al usuario una idea de la configuración que existe actualmente del mapeo entre los modelos que componen su proyecto y las entidades de los namespaces. Posteriormente, podrá modificarse la configuración existente, pulsando sobre el botón *Configurar*.

4.2.4. Mapeo de los modelos

En la figura 4.15 de abajo, se muestra un boceto de la interfaz para la configuración del mapeo de los modelos del proyecto, con las entidades de los diferentes namespaces que queramos hacer que se corresponda.

Figura 4.8: Boceto de la interfaz para el mapeo de los modelos

El usuario, cuando se encuentre en la pantalla del visualización de la configuración del mapeo de los modelos, tal y como se indica en el punto anterior, podrá acceder a configurar dicho mapeo, mostrándole la pantalla que se puede apreciar, donde para cada uno de los modelos que posee el proyecto donde se encuentra esta aplicación, mostrará las distintas entidades disponibles para cada uno de los namespaces. Tras hacer la correspondencia, que el usuario crea conveniente, solo tendrá que pulsar sobre el botón de salvar, para almacenar los cambios realizados.

4.2.5. Mapeo de los atributos

En la figura 4.16 que se muestra a continuación, podemos apreciar un boceto de la interfaz que se usará en la aplicación para hacer corresponder, cada uno de los atributos de los modelos de nuestra aplicación, con cada posible propiedad de la entidad con la que se ha relacionado el modelo al que pertenece el atributo, o con cualquiera de las propiedades de los namespaces cargados.

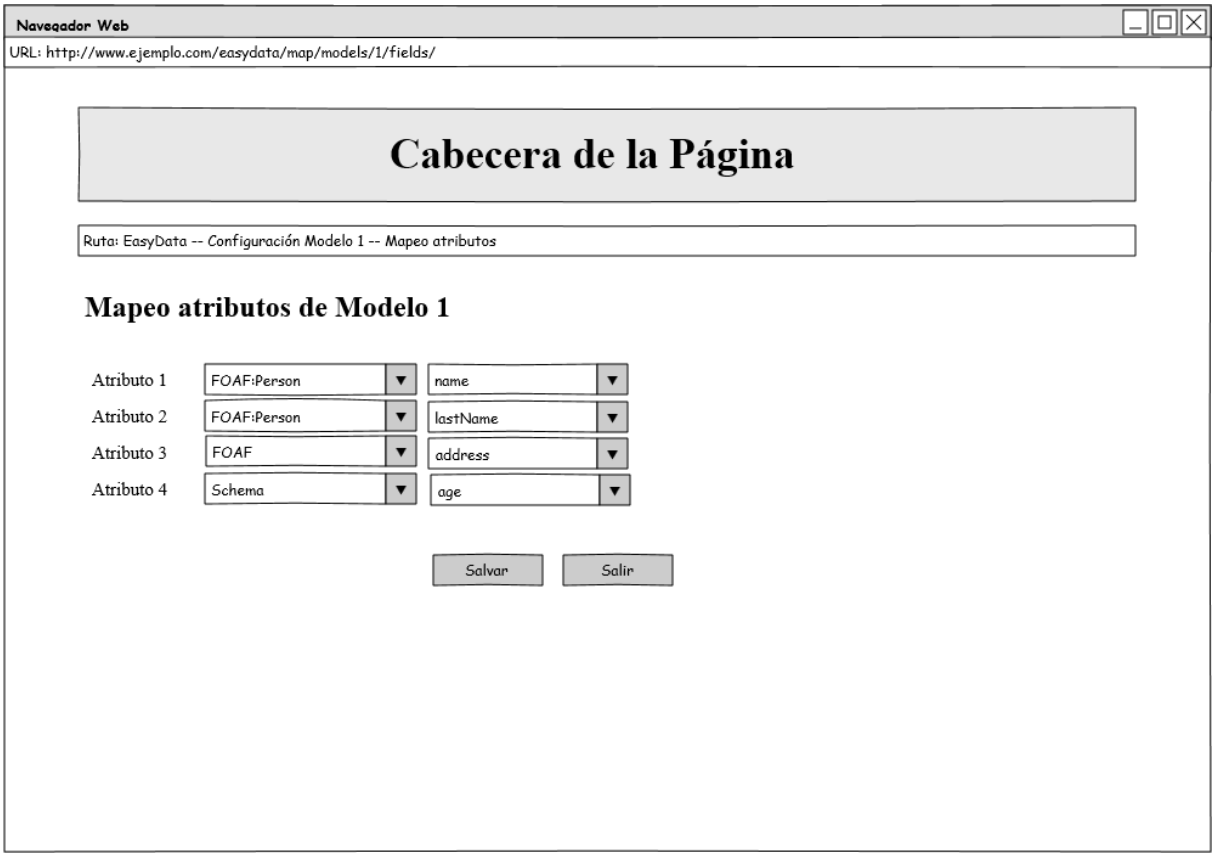


Figura 4.9: Boceto de la interfaz para el mapeo de los atributos

Si nos fijamos bien, aparece una entrada por cada uno de los atributos de los que dispone el modelo, y al lado de cada uno de estos, dos desplegables con cada uno de los diferentes namespaces y cada una de las posibles propiedades con las que podremos hacer corresponder al atributo. Una vez se haya realizado toda la correspondencia, solo habrá que almacenar los cambios realizados pulsando sobre el botón de salvar.

4.3. Diseño de datos

En esta sección definimos los datos que utilizará la aplicación, y como se estructuran los mismos dentro de la base de datos. Para plasmar la estructura de los datos en la base de datos, en la figura 4.10 se muestra el diagrama de datos, donde podemos apreciar cada una de las tablas y columnas de las mismas existentes en la base de datos, así como las funciones que implementan cada una de ellas.

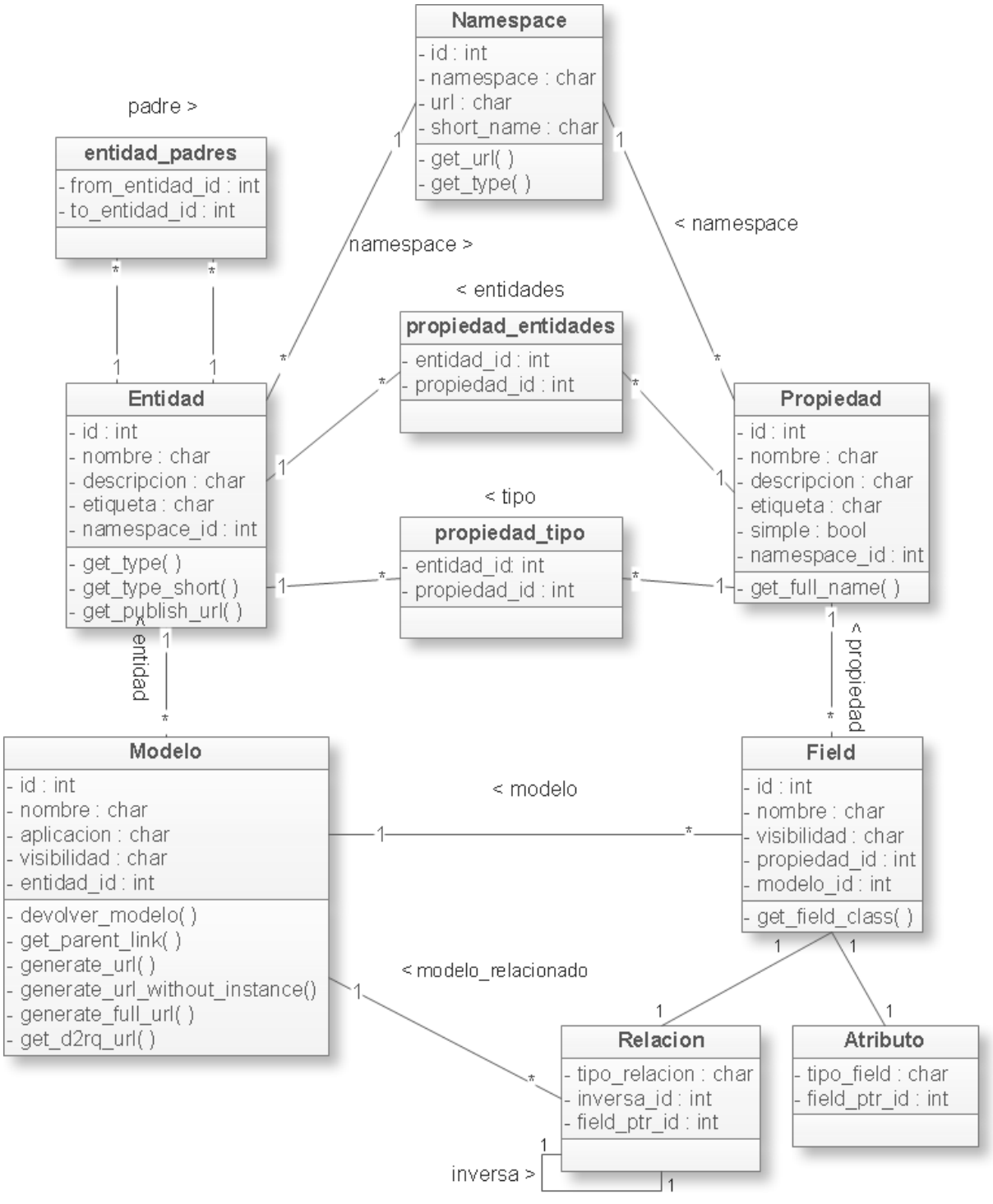


Figura 4.10: Diseño de los datos

A continuación se detallan cada una de las tablas que aparecen en el diagrama de diseño de los datos:

- **Namespace:** Esta tabla representa a un determinado namespace, almacenando el nombre del mismo y la url donde se encuentra el schema que describe cada una de sus entidades y propiedades.
- **Entidad:** Es una determinada entidad de un namespace. Almacena el nombre del mismo, y la etiqueta y descripción asociadas en el caso de que la posean. Está relacionada con cada una de las propiedades que lo componen, con el namespace al que pertenece y con los modelos con los que lo haga corresponder el usuario.
- **Propiedad:** Representa a una determinada propiedad de una entidad (no tiene que estar relacionado con una entidad forzosamente), almacenando el nombre y el tipo de la misma, así como una etiqueta y descripción que pueda tener asociados, al igual que en el caso de las entidades. Además, estará relacionada con la entidad y el namespace a la que pertenece y con los fields con los que lo haya hecho corresponder el usuario.
- **Modelo:** Es la representación de cada uno de los modelos de los que está compuesto el proyecto Django. Almacena el nombre del mismo, la aplicación del proyecto a la que pertenece y la visibilidad que tendrá el mismo respecto al exterior. Posee relaciones con la entidad con la que se haga corresponder y con los fields de los que está compuesto.
- **Field:** Esta tabla a su vez, representa a cada uno de los fields del proyecto, que componen a los modelos descritos anteriormente. Están compuestos por el nombre de estos y la visibilidad que tendrán respecto al exterior. Posee relaciones tanto con el modelo al que pertenecen, como con las propiedades con las que se haya hecho corresponder. Además, existen dos tablas más, llamadas Atributo y Relacion, con una relación OneToOne con la tabla Field (es la forma de expresar la herencia en Django), de tal forma que se indica si dicho field del modelo es un atributo o una relación.
- Las tablas entidad_padres, propiedad_entidades y propiedad_tipo son tablas auxiliares que se utilizan para implementar las relaciones Muchos-A-Muchos en la base de datos, y los datos que contienen únicamente son los punteros (id) a los elementos que relaciona.

4.4. Diseño de componentes

En esta sección se definen los componentes software necesarios para la implementación del sistema. Este sistema se implementa como una única aplicación Django, pero se puede descomponer en subsistemas en función de las diferentes funcionalidades que ofrece y de cómo se encontrará estructurado. Al estar haciendo uso para la realización del proyecto de un framework basado en el patrón MVC, vamos a diferenciar dentro del mismo, distintos módulos o componentes, como son los modelos, los controladores y las vistas (conocidos por modelos, vistas y plantillas respectivamente en Django). En la siguiente figura (4.11) se puede ver un diagrama que refleja perfectamente la estructura de los componentes que deberá de tener la aplicación.

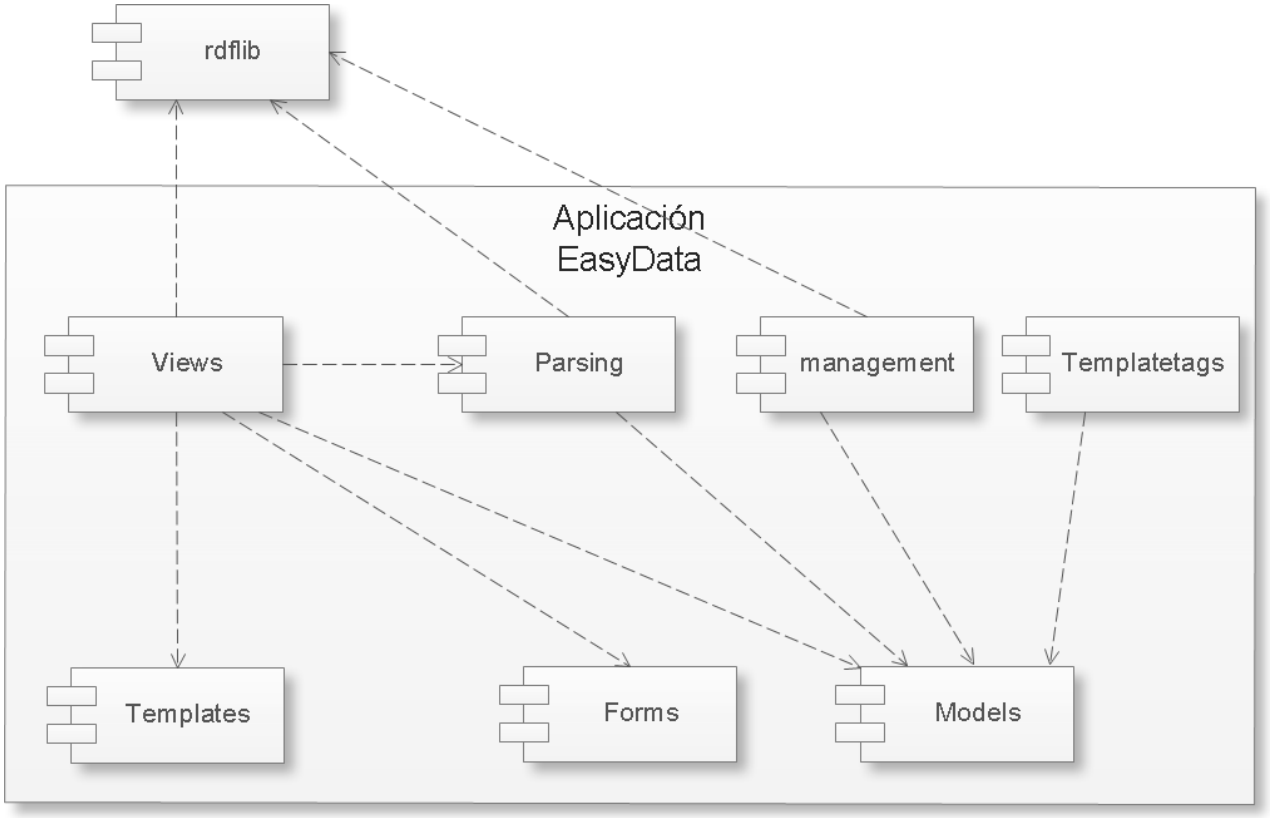


Figura 4.11: Estructura de los componentes de la aplicación

Donde cada uno de los componentes que se aprecian en la figura 4.11 son los encargados de realizar las siguientes funcionalidades:

- **Views:** Almacena todas las vistas de django (controladores en el patrón MVC) de los que estará compuesto la aplicación, y está dividido en submódulos según la funcionalidad que realiza cada uno de la siguiente forma:
  - **map:** almacena aquellas vistas encargadas de realizar el mapeo entre los modelos y entidades y entre los fields y propiedades.
  - **modelo:** almacena las vistas encargadas de configurar la visibilidad de los modelos y sus fields.
  - **namespaces:** contiene las vistas encargadas de la gestión de los namespaces, tanto la alta, edición como eliminado de un namespace.
  - **publish:** contiene aquellas vistas encargadas de realizar la publicación de los datos según el mapeo realizado de los mismos.
  - **sesiones:** contiene las vistas encargas de realizar tanto el login como el logout de la aplicación.
- **management:** contiene aquellos comandos que se añadirán al manage.py de Django. En este caso, se trata de loadmodels y de easydata_d2rq, que se encargará de realizar la

captación de los modelos y fields del proyecto Django y de generar el fichero de configuración para d2rq, respectivamente.

- **Models:** contiene cada uno de los modelos definidos para la aplicación EasyData. Cada uno de los módulos de los que está compuesto, contiene cada uno de los modelos con el mismo nombre que indica el módulo.
- **Template tags:** contiene aquellas funciones que se usarán en las plantillas Django para la publicación de información. Está separado en dos módulos easydata_microdata y easydata_rdfa, para la publicación en los formatos en microdata y rdfa respectivamente. Los template tags, hacen uso de los modelos para obtener los datos que van a publicar.
- **Parsing:** Contiene aquellas clases y funciones encargadas de realizar el parseo de los ficheros RDF (o cualquier otro formato que se desee añadir en un futuro) para la carga de nuevos namespaces.
- **Templates:** contiene todas las plantillas donde los controladores (vistas de Django), renderizan los datos generados. Estos estarán compuestos principalmente por plantillas html.
- **forms:** este componente o módulo de la aplicación contiene todos los formularios django que se usarán en las vistas.
- **rdflib:** es un módulo externo que se usará en el proyecto, el cuál ofrece funciones para el tratamiento de ficheros RDF.

Para plasmar la funcionalidad que tienen cada uno de los componentes que integran la aplicación, vamos a utilizar diagramas de interacción, donde se podrá observar como se combinan cada uno de estos para proporcionar una determinada funcionalidad. Estos elementos no se corresponderán únicamente con los datos especificados en el diagrama anterior, sino que además podrán verse el resto de componentes software que intervienen en la prestación de una determinada funcionalidad. Estas funcionalidades, se corresponderán con aquellas que hemos descrito en apartados anteriores.

4.4.1. Diagramas de interacción

A continuación mostramos los diagramas de interacción correspondientes a cada una de las funcionalidades que hemos descrito, las cuales debe de cumplir el proyecto.

Carga de modelos del proyecto

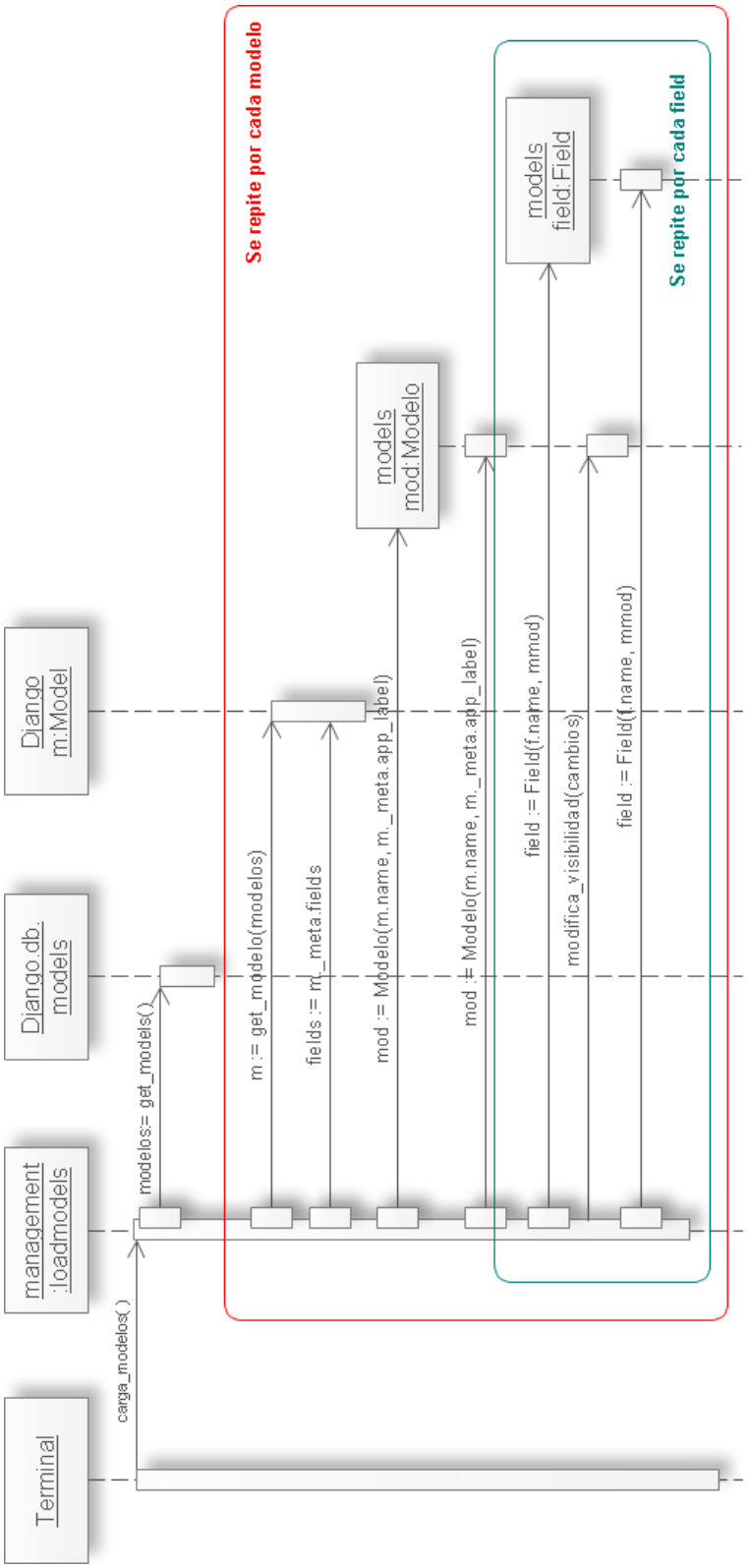


Figura 4.12: Diagrama de interacción: carga de modelos del proyecto

Carga de namespaces

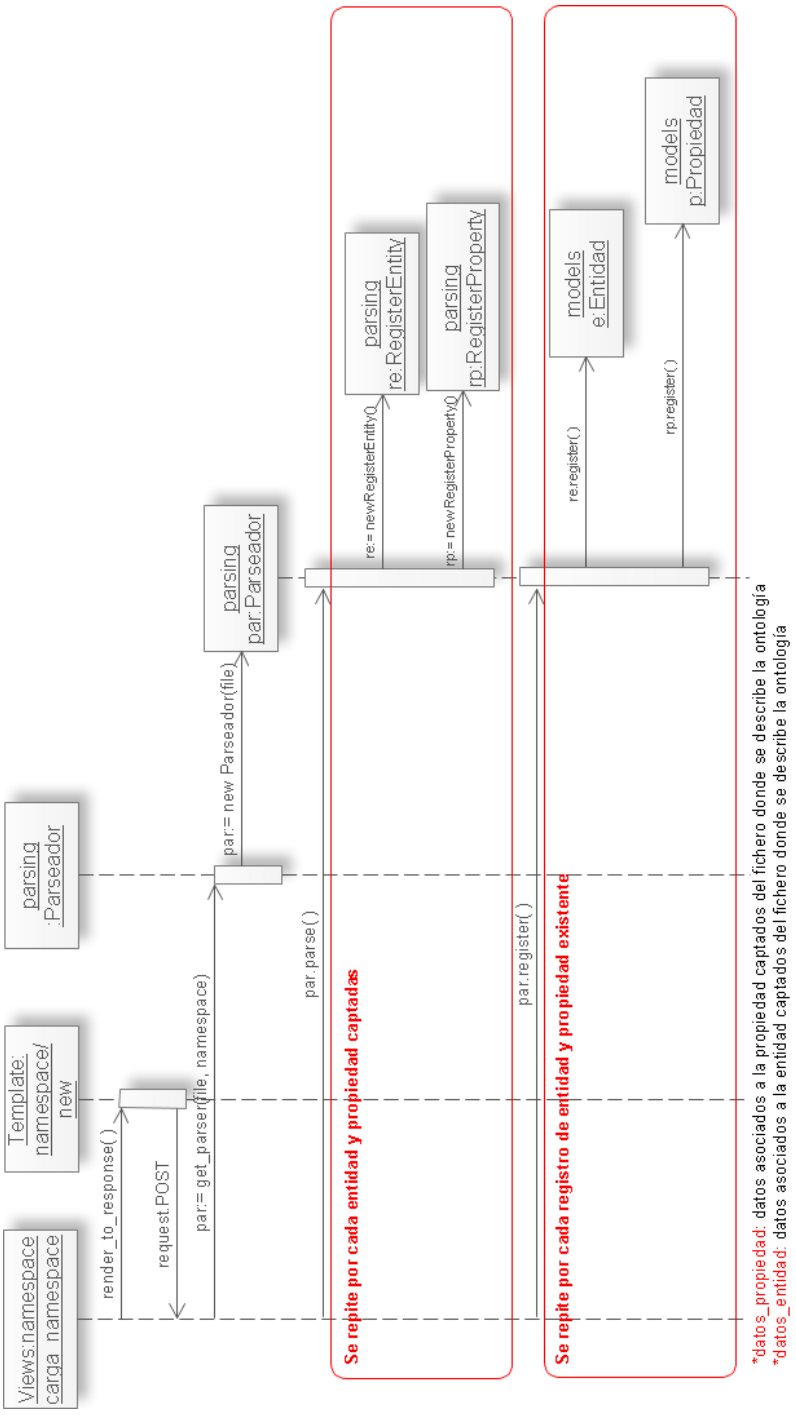


Figura 4.13: Diagrama de interacción: carga de namespaces

Configuración de la visibilidad

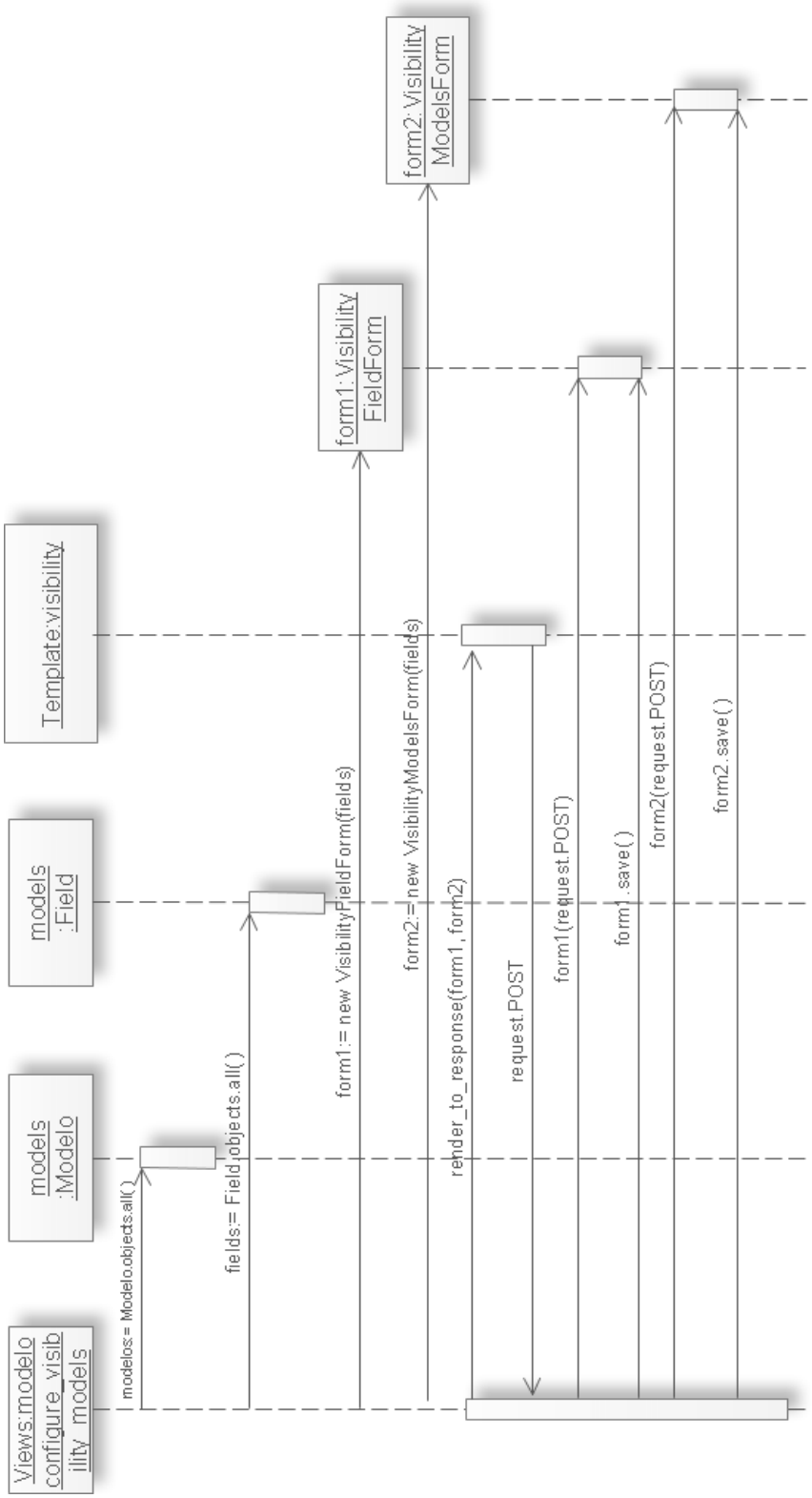


Figura 4.14: Diagrama de interacción: configuración de la visibilidad

Mapeo de los modelos del proyecto

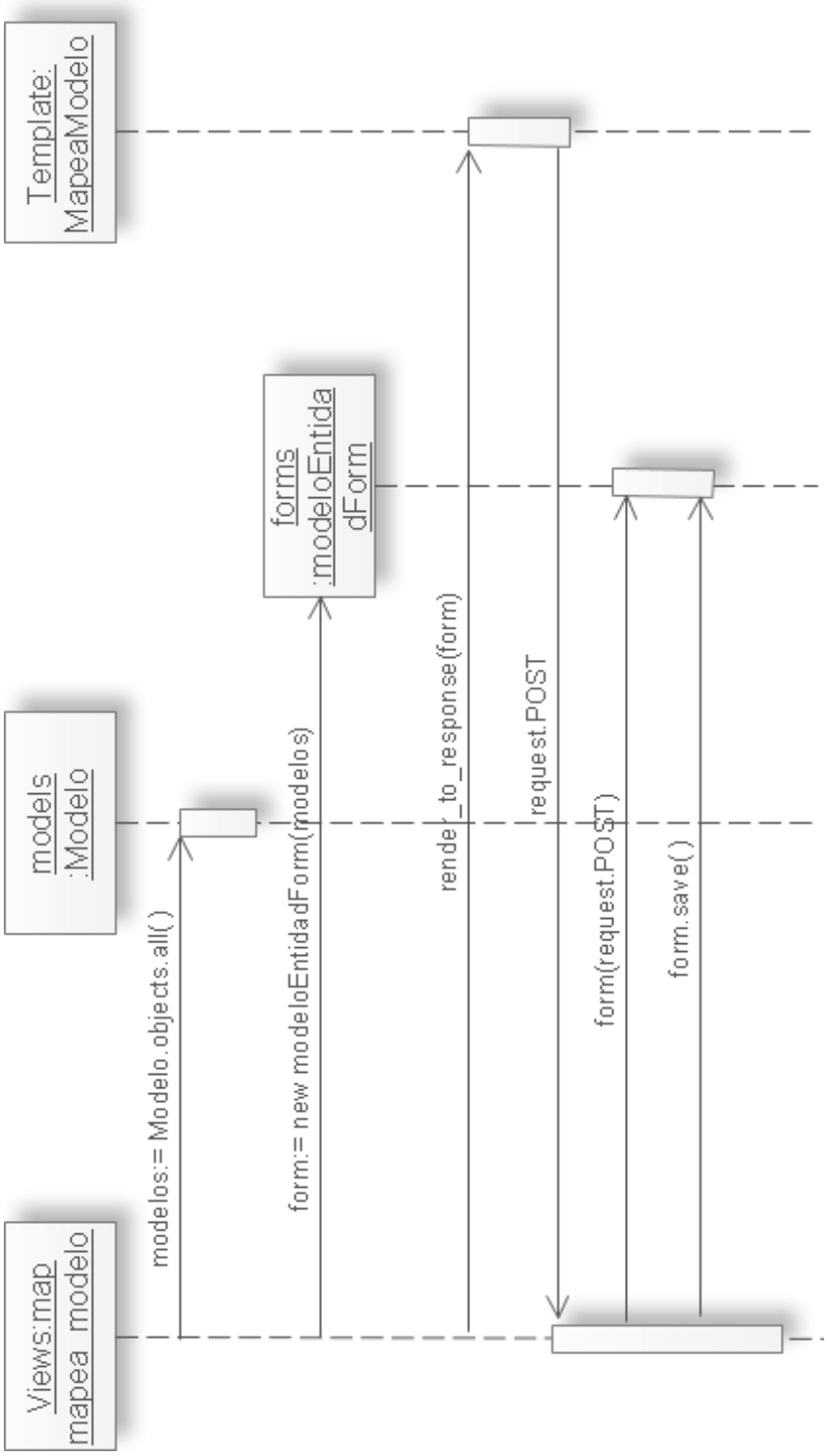


Figura 4.15: Diagrama de interacción: mapeo de los modelos del proyecto

Mapeo de los atributos de los modelos

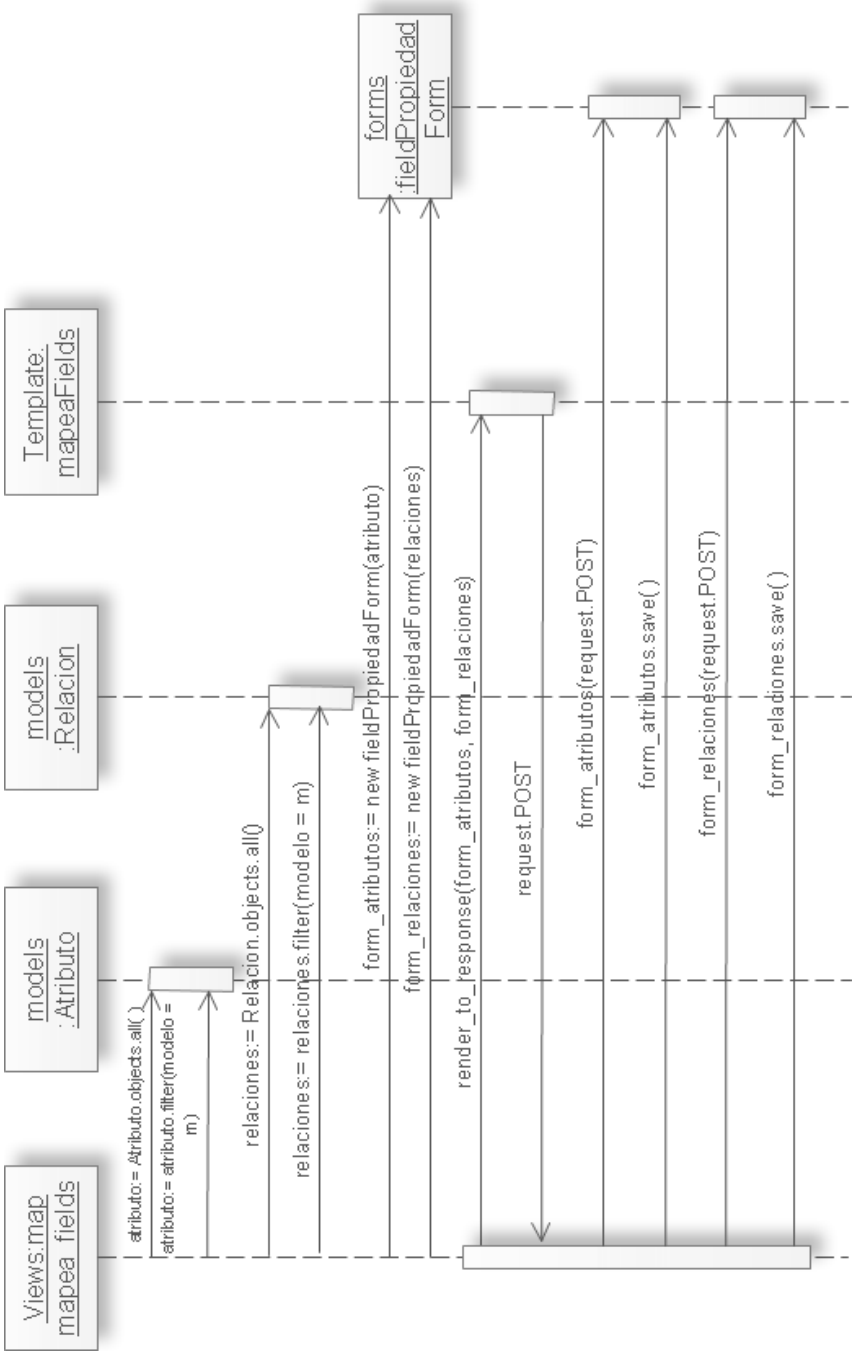


Figura 4.16: Diagrama de interacción: mapeo de los atributos de los modelos

Publicación de datos

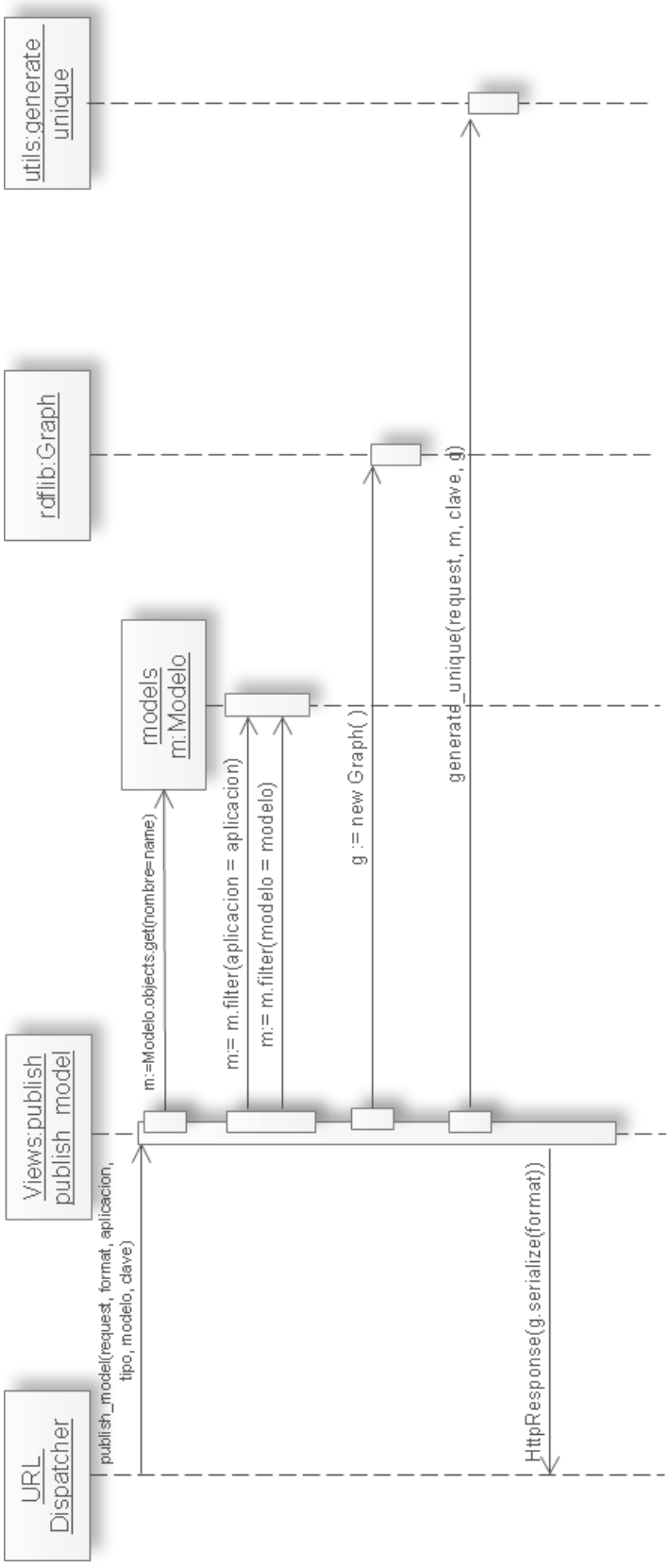


Figura 4.17: Diagrama de interacción: publicación de datos para un elemento determinado

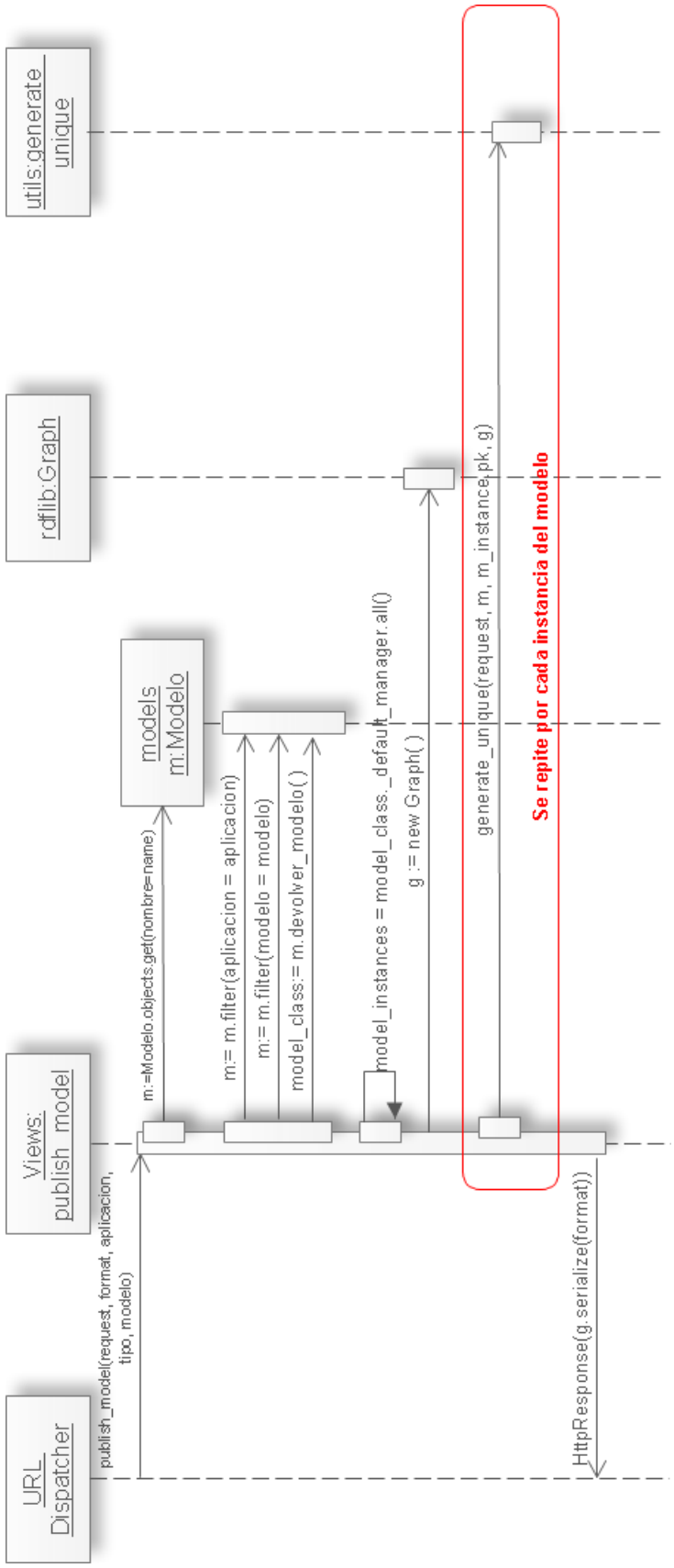


Figura 4.18: Diagrama de interacción: publicación de datos para todos los elementos de un modelo

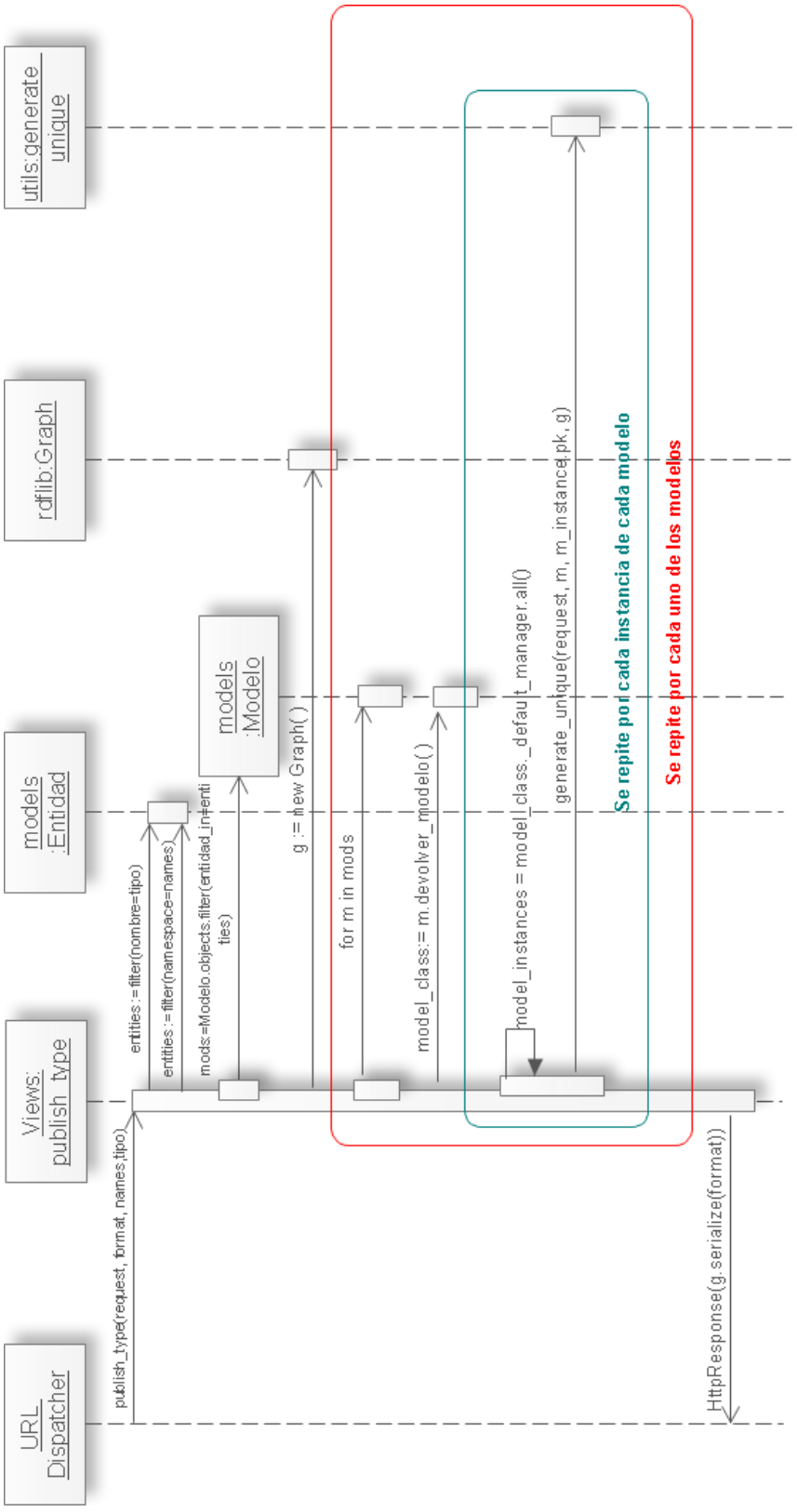


Figura 4.19: Diagrama de interacción: publicación de datos para todos los elementos de un tipo

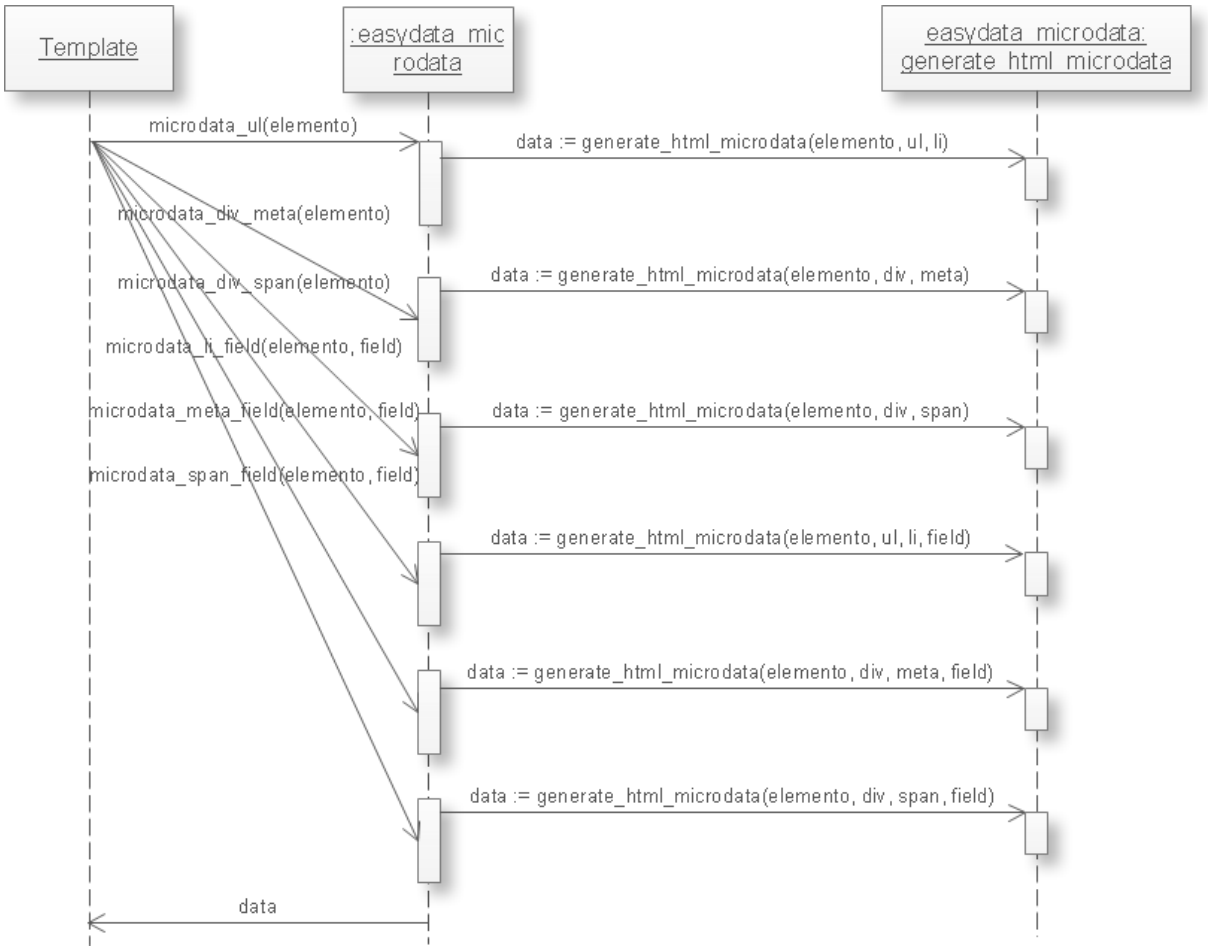


Figura 4.20: Diagrama de interacción: publicación de datos mediante microdata

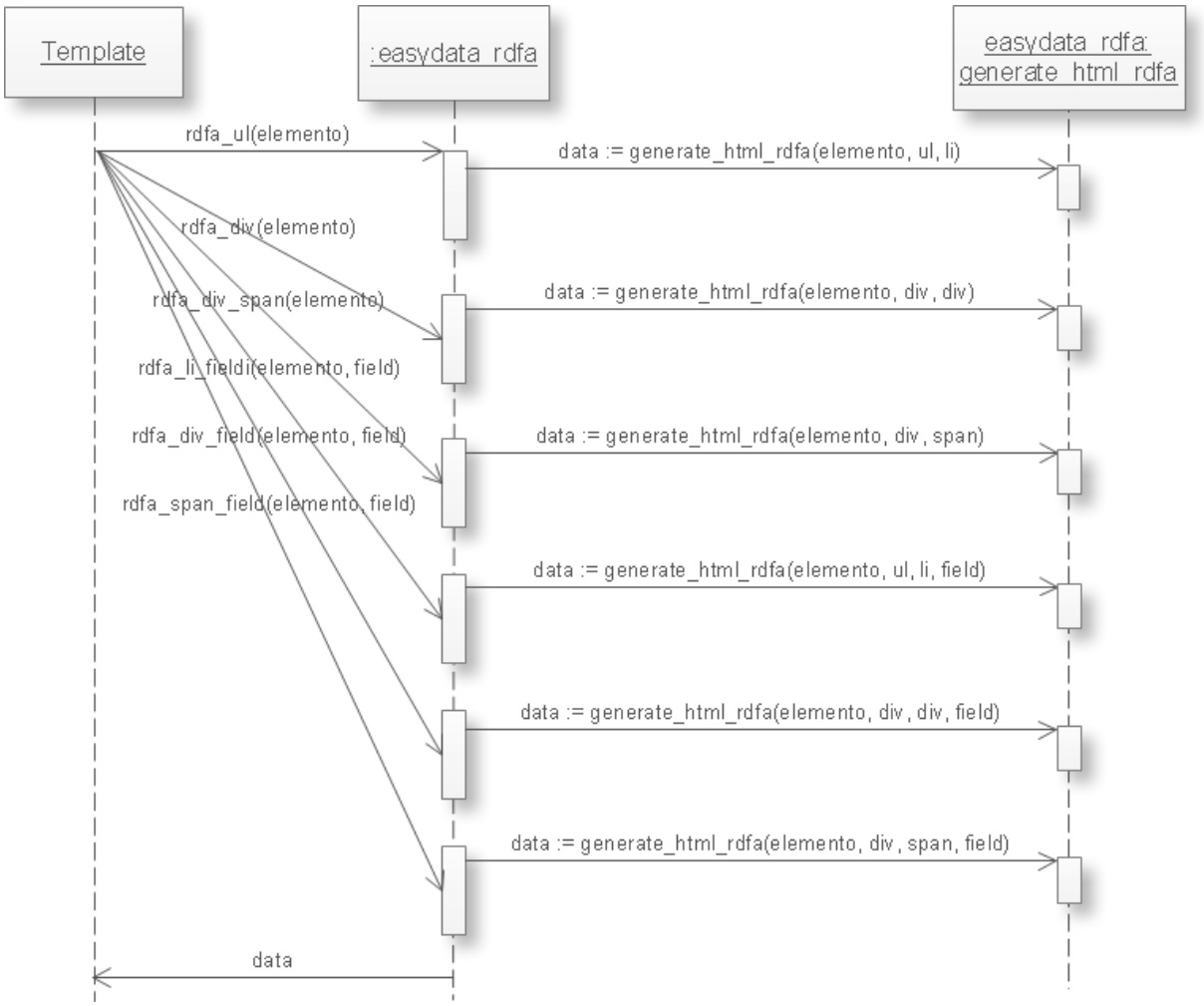


Figura 4.21: Diagrama de interacción: publicación de datos mediante rdfa

Generación fichero D2Rq

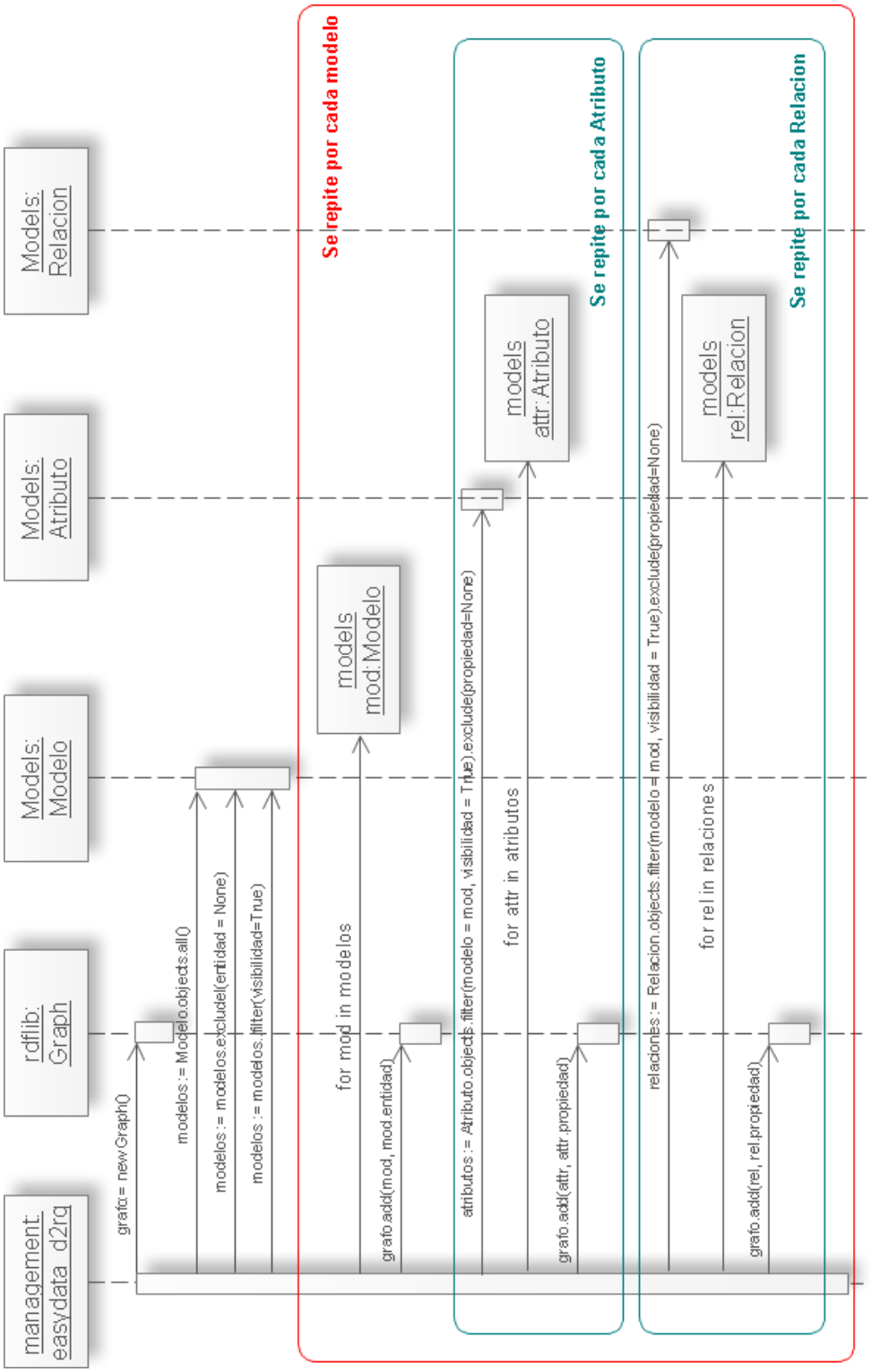


Figura 4.22: Diagrama de interacción: generación de fichero d2rq

Capítulo 5

Implementación del Sistema

Este capítulo trata sobre todos los aspectos relacionados con la implementación del sistema en código, haciendo uso de un determinado entorno tecnológico.

5.1. Entorno tecnológico

En esta sección se indica el marco tecnológico utilizado para la construcción del sistema, como son:

- Entorno de desarrollo (IDE).
- Lenguaje de programación, bibliotecas y frameworks de desarrollo empleados.
- Herramientas de ayuda a la construcción y despliegue.
- Control de versiones y repositorio de componentes.

El lenguaje de programación utilizado para la elaboración del proyecto ha sido Python, usándose multitud de bibliotecas de todas las que ofrece el lenguaje, pero en especial, se ha hecho uso de la biblioteca *rdflib*, la cual ha permitido tanto el parseo de fichero rdf, como la generación del rdf para la exportación de datos, en diferentes formatos.

Dentro del lenguaje Python, se ha utilizado el framework *Django*, ya que el proyecto básicamente consiste en el desarrollo de un plugin para dicho framework. Además del framework Django, también se ha hecho uso de otros frameworks, los cuales facilitasen tanto la escritura de estilos CSS como de código JavaScript. Estos frameworks auxiliares utilizados son *jQuery* [jQu13] y *Bootstrap* [Twi13].

Tanto en la importación de vocabularios para la aplicación como en la publicación de los datos, ha sido necesario utilizar formatos específicos, como son el caso de *RDF/XML*, *RDF/Ntriples*, *RDF/Turtle*, *RDFa* o *Microdata*. Además, en la importación de los datos de los vocabularios,

ha sido necesario utilizar el lenguaje de consultas *SPARQL*, que permite realizar consultas sobre conjuntos de datos definidos en ficheros RDF. También se ha hecho uso del software D2Rq, que ofrece al usuario a partir del esquema de la base de datos, múltiples utilidades para la publicación de sus datos en internet.

Como IDE de desarrollo, se ha utilizado un editor de texto simple, en este caso gEdit, con plugins para programación, y el resaltado de código python. A su vez, también se han utilizado herramientas proporcionadas por la comunidad Python para la depuración del código, tales como *iPython* y *ipdb*.

Para el despliegue del proyecto se ha utilizado el servidor de pruebas que ofrece el framework Django, y el servidor *Apache* junto con el intérprete de apache para código Python *mod_wsgi*.

Por último, para el alojamiento del código fuente y demás ficheros que componen el proyecto, se ha hecho uso del sistema de control de versiones *subversion*, integrado el mismo con la aplicación *Redmine* para la gestión de tareas y asignaciones de tiempo.

5.2. Código fuente

La organización del código fuente del proyecto, se ha llevado a cabo siguiendo las indicaciones del propio framework Django para la estructura de sus aplicaciones, de tal forma que en directorio raíz del proyecto cabría destacar los siguientes subdirectorios y ficheros:

- **forms:** en este directorio se encuentran alojados todos los formularios Django que se utilizan dentro de la aplicación. A su vez, este directorio está compuesto por el directorio modelforms, que contiene todos los formularios basados en modelos de Django.
- **decorators:** en este directorio se encuentran implementado el decorador utilizado en las vistas, para comprobar que el usuario que intenta acceder a ellas, se encuentra logueado y tiene permisos de superusuario.
- **locale:** en este directorio se encuentran las traducciones de la aplicación a los diferentes idiomas que soporta la misma. Inicialmente está soportado los idiomas inglés y español.
- **management:** este directorio contiene a su vez el directorio commands, el cual contiene todos los comandos desarrollados, a fin de que puedan ser ejecutados desde el manage.py de Django desde una terminal.
- **models:** este directorio contiene todos los modelos que componen a la aplicación Django. Contiene un fichero por cada uno de los modelos de la aplicación.
- **parsing:** este directorio no es común a todo proyecto Django, pero se encarga de agrupar todas las clases y métodos necesarios para el parseo y carga de ontologías en formato RDF.
- **static:** este directorio compuesto a su vez por un subdirectorio cuyo nombre es el mismo de la aplicación, se encarga de almacenar todos los elementos estáticos de la aplicación, ya sean imágenes, estilos, código JavaScript, etc.

- **templates:** este directorio compuesto también a su vez por un subdirectorio con el mismo nombre de la aplicación, se encarga de almacenar todas las plantillas del proyecto django.

- **templatetags:** este directorio contiene todos los template tags desarrollados que pueden ser usados en las plantillas django. Se divide en dos ficheros, easydata_microdata y easydata_rdfa, y cada uno de ellos contienen los template tags necesarios para exportar la información en formato microdata y rdfa respectivamente.

- **views:** este directorio contiene todos los ficheros python con las vistas que se usarán en la aplicación. Existen varios ficheros, y se encargan de agrupar las vistas por funcionalidades comunes.

- **urls.py:** contiene todas las asociaciones de urls con las distintas vistas de la aplicación.

- **utils.py:** contiene multitud de funciones auxiliares que se utilizan tanto para el mapeo de modelos, como para la exportación de los datos en distintos formatos.

5.2.1. Internacionalización y localización

A la hora de desarrollar el software EasyData/Django se ha tenido en cuenta que el mismo pueda ofrecer su contenido en diferentes idiomas en función de los diferentes usuarios.

Para preparar el software para que este pueda ser traducido a diferentes idiomas (internacionalización) se han seguido las recomendaciones de Django y se han hecho uso de las herramientas que proporciona, tanto para la traducción de cadenas en el código Python como para la traducción de las plantillas. De igual forma, se ofrecen las traducciones de la aplicación (localización) en los idiomas inglés y español.

5.3. Calidad de código

Para asegurar la calidad del código fuente y por consiguiente, su fácil mantenibilidad, además de estructurar la aplicación tal y como recomienda Django, se ha seguido la guía de estilos para escribir código Python llamada PEP8. La guía de estilos PEP8 proporciona una serie de convenciones para la codificación en Python. De esta forma, además de proporcionarme una forma de trabajo y de escribir un código claro, nos aseguramos en cierta medida, que el código será fácilmente entendible por cualquier programador que esté acostumbrado a utilizar esta guía de estilos. A parte del manual PEP8 que se encuentra publicado en la web oficial de Python, existe una herramienta para diferentes sistemas operativos, con el mismo nombre que la guía de estilos, que comprueba directamente si nuestro código cumple con las especificaciones de pep8.

Además de la guía de estilos PEP8, también se ha utilizado la herramienta PyLint que proporciona una métrica que permite obtener una idea de en qué medida el código de la aplicación es de calidad. Para obtener dicha métrica se basa en diferentes como puede ser:

- Cantidad de comentarios de código (ni poco comentado ni sobrecomentado), que no existan elementos como clases, funciones o modulos sin comentar.
- Que se cumplan patrones de diseño como el patron DRY (Don't repeat yourself), de tal forma que no existan abundantes sentencias de código repetidas.
- Que el proyecto se encuentre correctamente estructurado.
- Cumplimiento de convenciones en la escritura del códgio fuente, como en el nombre de variables, clases, interlineado, etc...
- Que no existan errores en el código.

Una vez ejecutada la prueba de calidad de código con PyLint, se ha obtenido una puntuacion de 9,3 sobre 10 para el código de la aplicación, haciendo uso de un fichero de configuración de PyLint específico para proyectos Django. Además solo existe un 1,345 % de líneas duplicadas (89 líneas) de entre mas de 5000 líneas de código que ha abarcado la aplicación. El resultado es el que se puede apreciar en la figura 5.1.

La finalidad de realizar estas pruebas de código, además de que el código escrito cumpla con un mínimo de calidad, sirve a título personal para la adquisición de buenas prácticas a la hora de programar, las cuales se puedan poner en práctica en futuros proyectos.

```
Global evaluation
-----------------
Your code has been rated at 9.30/10


Duplication
-----------

+-------------------------+------+---------+-----------+
|                         |now   |previous |difference |
+=========================+======+=========+===========+
|nb duplicated lines      |89    |89       |=          |
+-------------------------+------+---------+-----------+
|percent duplicated lines |1.345 |1.345    |=          |
+-------------------------+------+---------+-----------+


Raw metrics
-----------

+----------+-------+------+---------+-----------+
|type      |number |%     |previous |difference |
+==========+=======+======+=========+===========+
|code      |3465   |57.07 |3465     |=          |
+----------+-------+------+---------+-----------+
|docstring |1164   |19.17 |1164     |=          |
+----------+-------+------+---------+-----------+
|comment   |839    |13.82 |839      |=          |
+----------+-------+------+---------+-----------+
|empty     |604    |9.95  |604      |=          |
+----------+-------+------+---------+-----------+
```

Figura 5.1: Resultado de pruebas de validación de PyLint

Capítulo 6

Pruebas del Sistema

En este capítulo se documentan los diferentes tipos de pruebas que se han llevado a cabo, ya sean de carácter manual o automatizadas mediante software específico de pruebas.

6.1. Pruebas unitarias y de integración

Para el desarrollo de las pruebas unitarias y de integración de los distintos artefactos software desarrollados, se han realizado una serie de pruebas automatizadas mediante el framework de Python para el desarrollo de pruebas PyUnit, el cual está basado en el framework de pruebas JUnit de Java.

Mediante las pruebas automáticas desarrolladas, se ha probado que tanto los modelos y métodos de que disponen los mismos realizan las funciones para las que han sido desarrollados correctamente. Además, también se ha probado la integración entre los mismos, así como las distintas funciones auxiliares desarrolladas, donde se combinan el uso de los distintos modelos, así como sus métodos.

Para lanzar La ejecución de las distintas pruebas automáticas diseñadas, se hará de la siguiente forma, desde una terminal de escritorio en el directorio donde se encuentre el fichero manage.py de nuestro proyecto Django:

```
$> python manage.py test easydata
```

Esto muestra un informe de los distintos tests ejecutados, así como los posibles errores que hayan podido surgir en los mismos si alguna de las funciones no se llevado a cabo tal y como se esperaba. El resultado de la validación debe de ser similar al que puede apreciarse en la siguiente figura (6.1).

```
proyecto@pfc ~/Escritorio/PFC/codigoFuente $ make test
PYTHONPATH="$LOCALPYTHONPATH" python project/manage.py test easydata
Creating test database for alias 'default'...
..........................
----------------------------------------------------------------------
Ran 26 tests in 0.279s

OK
Destroying test database for alias 'default'...
proyecto@pfc ~/Escritorio/PFC/codigoFuente $
```

Figura 6.1: Resultado de validación PyUnit

Adicionalmente, al tratarse de una aplicación la cual se instalará en proyectos Django ya existentes, se ha probado su integración con aplicaciones disponibles en el *Python Package Index* desarrolladas por otros autores. Más concretamente, para el desarrollo de las pruebas que se muestran a continuación, se ha utilizado la aplicación django-blog-zinnia, que como su nombre indica, se trata de un blog escrito en Python para Django.

6.2. Pruebas de sistema

En este apartado se describen las pruebas de sistema de modo que se asegure que el sistema cumple con todos los requisitos establecidos al comienzo del proyecto, bajo un entorno específico para pruebas.

Para ello se han realizado pruebas manuales, donde se ha comprobado que se cumplen los requisitos funcionales especificados inicialmente, accediendo a los distintos apartados de:

- **Captación de modelos y fields del proyecto Django:** se ha probado que la carga de modelos y fields del proyecto funciona correctamente, así como la actualización de los datos antes posibles cambios en los modelos.
- **Carga de nuevos namespaces:** se ha probado que la carga de nuevos namespaces u ontologías funciona correctamente, además de la actualización de la especificación de los mismos ante posibles cambios. Para ello se ha probado con multitud de namespaces diferentes suministrados por distintas organizaciones.
- **Configuración de la visibilidad de los modelos y fields:** se ha probado que tanto el proceso de configuración de la visibilidad funcione correctamente, como que a la hora de publicar los datos, aquellos marcados como no visibles no se publiquen a los usuarios utilizando las diferentes herramientas de publicación de datos.
- **Mapeo de modelos y fields:** se ha probado que el mapeo de los datos funciona correctamente utilizando varios namespaces de forma individual o simultánea para un mismo

modelo. Posteriormente, se ha comprobado que los datos se publican haciendo uso de las etiquetas especificadas en este apartado de configuración.

- **Publicación de datos en RDF, RDFa y Microdata:** Una vez realizada tanto la carga de modelos y fields, como la configuración de los mismos comentada anteriormente, se ha probado que estos se publican correctamente según los criterios especificados.
- **Herramienta de generación de D2Rq:** se ha probado que una vez realizada la configuración de los modelos y fields, se genera el fichero con la configuración para el software D2Rq y este funciona correctamente con la aplicación, probando consultas SPARQL y comprobando que los resultados sean satisfactorios.
- **Generación de diagrama con configuración:** una vez realizada la configuración de los modelos y los fields, se ha probado que la generación del diagrama con la configuración mediante Graphviz, se corresponde con la configuración realizada.

Además de la comprobación de que se han satisfacido cada uno de los requisitos funcionales descritos en el apartado de análisis del proyecto, se han realizado pruebas de validación, de los datos exportados.

A continuación se muestra un resumen de las pruebas de validación realizadas sobre los datos publicados por la aplicación EasyData/Django.

6.2.1. Pruebas no funcionales

Pruebas de validación de RDF

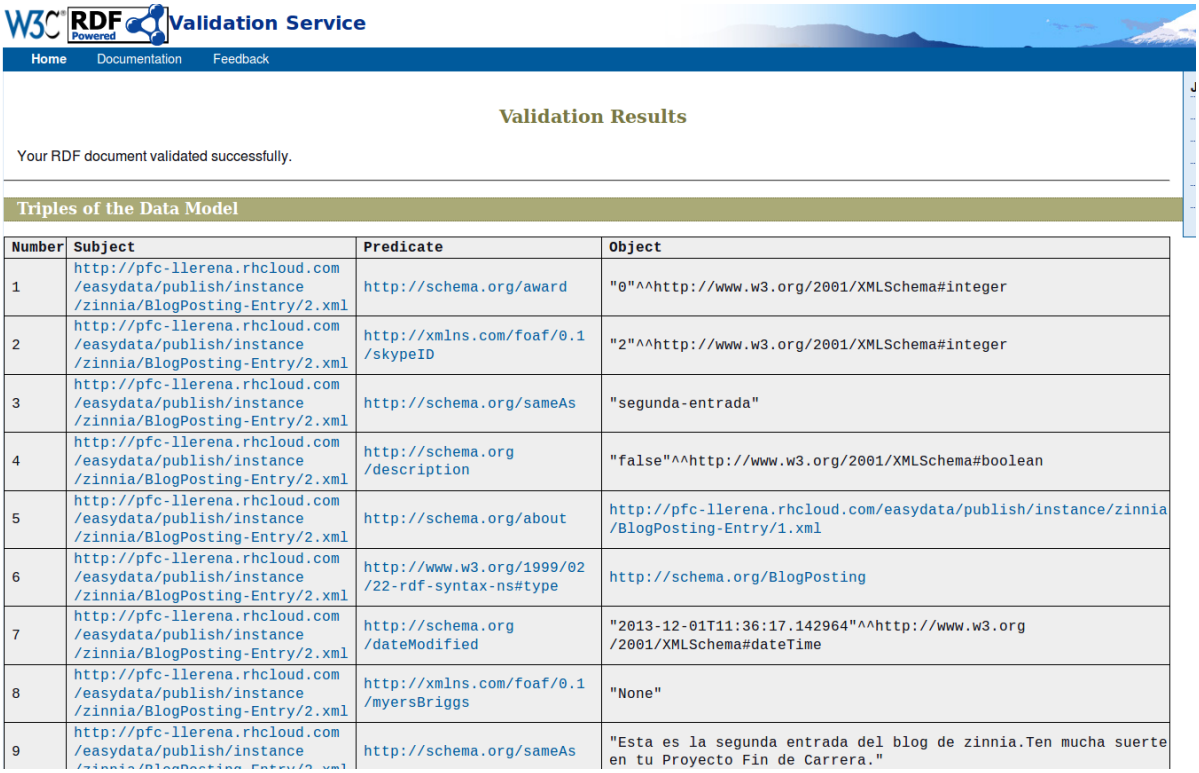
La validación del código RDF generado por la aplicación EasyData/Django, se ha hecho uso del validador suministrado por el W3C (http://www.w3.org/RDF/Validator/). Para la realización de la prueba, se suministró al validador del W3C diferentes salidas proporcionadas por la aplicación EasyData/Django, comprobando en cada caso que el resultado de la validación fuese satisfactorio. De esta forma, obtenemos la seguridad de que los datos RDF publicados, están perfectamente generados y cumple estrictamente los estándares impuestos por el W3C.

A continuación, se muestra un ejemplo de una salida generada por la aplicación EasyData/Django en formato RDF/XML, para una entrada de blog de la aplicación anteriormente comentada, donde se pueden apreciar las marcas que se han añadido en función del mapeo realizado.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <rdf:RDF
3     xmlns:foaf="http://xmlns.com/foaf/0.1/"
4     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5     xmlns:schema="http://schema.org/"
6 >
7   <rdf:Description rdf:about="http://pfc-llerena.rhcloud.com/easydata/publish/
        instance/zinnia/BlogPosting-Entry/2.xml">
8     <schema:sameAs>Esta es la segunda entrada del blog de zinnia.Ten mucha suerte
          en tu Proyecto Fin de Carrera.</schema:sameAs>
```

```
9     <schema:dateModified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
          >2013-12-01T11:36:17.142964</schema:dateModified>
10    <foaf:skypeID rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">2</foaf:
          skypeID>
11    <schema:text>&lt;p&gt;Esta es la segunda entrada del blog de zinnia.&lt;/p&gt
          ;&lt;p&gt;Ten mucha suerte en tu Proyecto Fin de Carrera.&lt;/p&gt;</
          schema:text>
12    <schema:description rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
          false</schema:description>
13    <schema:keywords></schema:keywords>
14    <foaf:myersBriggs>None</foaf:myersBriggs>
15    <rdf:type rdf:resource="http://schema.org/BlogPosting"/>
16    <schema:dateCreated rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
          >2013-12-01T11:35:33</schema:dateCreated>
17    <schema:datePublished rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
          >2013-12-01T11:36:11</schema:datePublished>
18    <schema:headline>Segunda entrada</schema:headline>
19    <foaf:page>entry_detail.html</foaf:page>
20    <schema:sameAs>segunda-entrada</schema:sameAs>
21    <schema:award rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">0</
          schema:award>
22    <schema:about rdf:resource="http://pfc-llerena.rhcloud.com/easydata/publish/
          instance/zinnia/BlogPosting-Entry/1.xml"/>
23  </rdf:Description>
24 </rdf:RDF>
```

En la figura 6.2 se puede apreciar una captura de pantalla con el resultado de la validación devuelto por el validador del W3C, donde se puede ver que el código RDF cumple con los estándares impuestos por el W3C.



W3C RDF Powered Validation Service

Home Documentation Feedback

Validation Results

Your RDF document validated successfully.

Triples of the Data Model

Number	Subject	Predicate	Object
1	http://pfc-llerena.rhcloud.com /easydata/publish/instance /zinnia/BlogPosting-Entry/2.xml	http://schema.org/award	"0"^^http://www.w3.org/2001/XMLSchema#integer
2	http://pfc-llerena.rhcloud.com /easydata/publish/instance /zinnia/BlogPosting-Entry/2.xml	http://xmlns.com/foaf/0.1 /skypeID	"2"^^http://www.w3.org/2001/XMLSchema#integer
3	http://pfc-llerena.rhcloud.com /easydata/publish/instance /zinnia/BlogPosting-Entry/2.xml	http://schema.org/sameAs	"segunda-entrada"
4	http://pfc-llerena.rhcloud.com /easydata/publish/instance /zinnia/BlogPosting-Entry/2.xml	http://schema.org /description	"false"^^http://www.w3.org/2001/XMLSchema#boolean
5	http://pfc-llerena.rhcloud.com /easydata/publish/instance /zinnia/BlogPosting-Entry/2.xml	http://schema.org/about	http://pfc-llerena.rhcloud.com/easydata/publish/instance/zinnia /BlogPosting-Entry/1.xml
6	http://pfc-llerena.rhcloud.com /easydata/publish/instance /zinnia/BlogPosting-Entry/2.xml	http://www.w3.org/1999/02 /22-rdf-syntax-ns#type	http://schema.org/BlogPosting
7	http://pfc-llerena.rhcloud.com /easydata/publish/instance /zinnia/BlogPosting-Entry/2.xml	http://schema.org /dateModified	"2013-12-01T11:36:17.142964"^^http://www.w3.org /2001/XMLSchema#dateTime
8	http://pfc-llerena.rhcloud.com /easydata/publish/instance /zinnia/BlogPosting-Entry/2.xml	http://xmlns.com/foaf/0.1 /myersBriggs	"None"
9	http://pfc-llerena.rhcloud.com /easydata/publish/instance	http://schema.org/sameAs	"Esta es la segunda entrada del blog de zinnia.Ten mucha suerte en tu Proyecto Fin de Carrera."

Figura 6.2: Resultado de validación RDF

Pruebas de validación de RDFa

Al igual que para el caso de RDF, también se ha realizado una validación del código RDFa generado por la aplicación EasyData/Django, haciendo uso de igual forma del validador suministrado por el W3C (http://www.w3.org/2012/pyRdfa/Validator.html). Para la realización de las pruebas, se suministró al validador del W3C diferentes salidas proporcionadas por la aplicación EasyData/Django, comprobando en cada caso que el resultado de la validación fuese satisfactorio. Así de esta forma, nos aseguramos también que el código RDFa generado por la aplicación, están perfectamente generado y cumple de igual forma estrictamente los estándares impuestos por el W3C.

A continuación, se muestra un ejemplo de una salida generada por la aplicación EasyData/Django en formato RDFa, para una entrada de blog de la aplicación anteriormente comentada, donde se pueden apreciar las marcas que se han añadido en función del mapeo realizado al lenguaje HTML utilizando la notación RDFa.

```
1 <div about="http://pfc-llerena.rhcloud.com/easydata/publish/instance/zinnia/
    BlogPosting-Entry/2.xml" typeof="schema:BlogPosting" prefix="schema: http://
    schema.org/ foaf: http://xmlns.com/foaf/0.1/">
2     <div content="2" property="foaf:skypeID"></div>
3     <div content="Segunda entrada" property="schema:headline"></div>
4     <div content="segunda-entrada" property="schema:sameAs"></div>
5     <div content="2013-12-01 11:36:11" property="schema:datePublished"></div>
```

```
6       <div content="None" property="foaf:myersBriggs"></div>
7       <div content="2013-12-01 11:35:33" property="schema:dateCreated"></div>
8       <div content="2013-12-01 11:36:17.142964" property="schema:dateModified"></div
            >
9       <div content="&lt;p&gt;Esta es la segunda entrada del blog de zinnia.&lt;/p&gt
            ;&lt;p&gt;Ten mucha suerte en tu Proyecto Fin de Carrera.&lt;/p&gt;"
            property="schema:text"></div>
10      <div content="0" property="schema:award"></div>
11      <div content="Esta es la segunda entrada del blog de zinnia.Ten mucha suerte
            en tu Proyecto Fin de Carrera." property="schema:sameAs"></div>
12      <div content="" property="schema:keywords"></div>
13      <div content="False" property="schema:description"></div>
14      <div content="entry_detail.html" property="foaf:page"></div>
15      <link resource="http://pfc-llerena.rhcloud.com/easydata/publish/instance/
            zinnia/BlogPosting-Entry/1.xml" rel="schema:about">
16  </div>
17
18  <div about="http://pfc-llerena.rhcloud.com/easydata/publish/instance/zinnia/
        BlogPosting-Entry/1.xml" typeof="schema:BlogPosting" prefix="schema: http://
        schema.org/ foaf: http://xmlns.com/foaf/0.1/">
19      <div content="1" property="foaf:skypeID"></div>
20      <div content="Saludo inicial" property="schema:headline"></div>
21      <div content="saludo-inicial" property="schema:sameAs"></div>
22      <div content="2013-12-01 11:01:17" property="schema:datePublished"></div>
23      <div content="None" property="foaf:myersBriggs"></div>
24      <div content="2013-12-01 11:00:40" property="schema:dateCreated"></div>
25      <div content="2013-12-01 11:01:36.367237" property="schema:dateModified"></div
            >
26      <div content="&lt;p&gt;Esto es una entrada de prueba para el blog de zinnia
            para probar junto con EasyData.&lt;/p&gt;" property="schema:text"></div>
27      <div content="0" property="schema:award"></div>
28      <div content="Esto es una entrada de prueba para el blog de zinnia para probar
            junto con EasyData." property="schema:sameAs"></div>
29      <div content="easydata" property="schema:keywords"></div>
30      <div content="False" property="schema:description"></div>
31      <div content="entry_detail.html" property="foaf:page"></div>
32      <link resource="http://pfc-llerena.rhcloud.com/easydata/publish/instance/
            zinnia/BlogPosting-Entry/2.xml" rel="schema:about">
33  </div>
```

En la figura 6.3 se puede apreciar una captura de pantalla con el resultado de la validación producido por el validador del W3C, donde se puede ver que el código RDFa cumple con los estándares impuestos por el W3C.

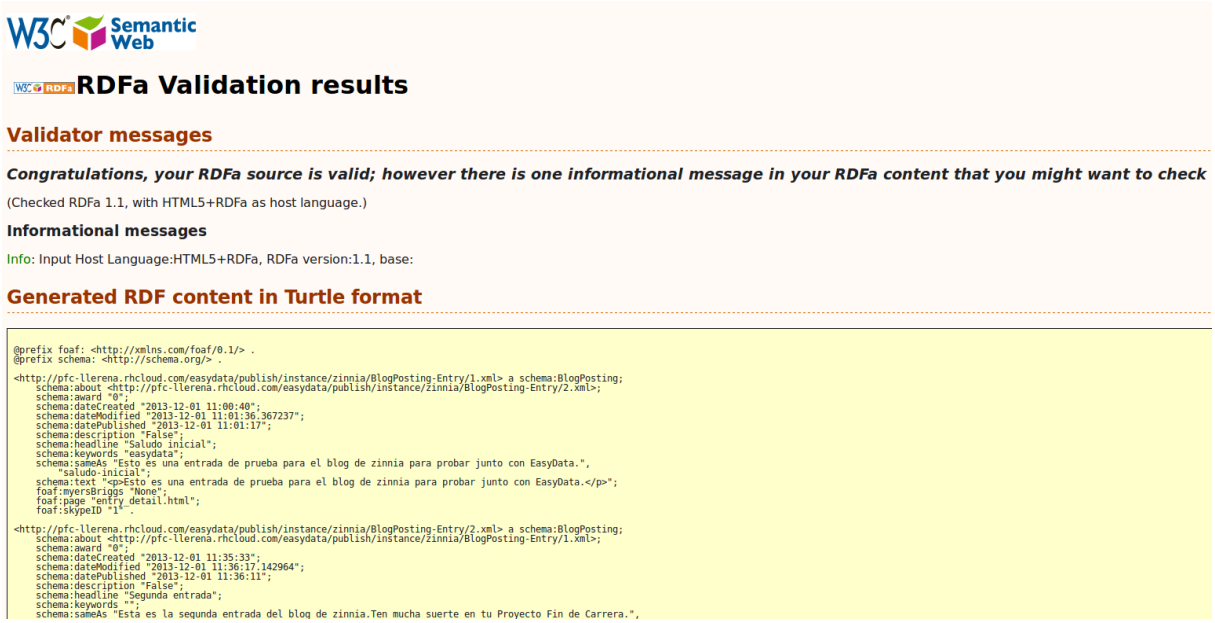


Figura 6.3: Resultado de validación RDFa

Pruebas de validación de Microdata

Por último, al igual que en los casos anteriores, también se ha realizado una validación del código Microdata generado por la aplicación EasyData/Django, haciendo uso de una herramienta suministrada por Google (http://www.google.com/webmasters/tools/richsnippets), la cual busca metadatos en una determinada URI que se le suministre. Para la realización de la prueba, se suministró a la herramienta de Google diferentes salidas proporcionadas por la aplicación EasyData/Django, comprobando en cada caso que los datos captados por la herramienta fuesen los correctos. Así de esta forma, nos aseguramos también que el código Microdata generado por la aplicación, está correctamente construido.

A continuación, se muestra un ejemplo de una salida generada por la aplicación EasyData/Django en formato Microdata, para una entrada de blog de la aplicación anteriormente comentada, donde se pueden apreciar las marcas que se han añadido en función del mapeo realizado al lenguaje HTML utilizando la notación Microdata.

```
1 <div itemid="http://pfc-llerena.rhcloud.com/easydata/publish/instance/zinnia/
    BlogPosting-Entry/2.xml" itemtype="http://schema.org/BlogPosting" itemscope >
2     <span itemprop="http://xmlns.com/foaf/0.1/skypeID">2</span>
3     <span itemprop="http://schema.org/headline">Segunda entrada</span>
4     <span itemprop="http://schema.org/sameAs">segunda-entrada</span>
5     <span itemprop="http://schema.org/datePublished">2013-12-01 11:36:11</span>
6     <span itemprop="http://xmlns.com/foaf/0.1/myersBriggs">None</span>
7     <span itemprop="http://schema.org/dateCreated">2013-12-01 11:35:33</span>
8     <span itemprop="http://schema.org/dateModified">2013-12-01 11:36:17.142964</
        span>
9     <span itemprop="http://schema.org/text"><p>Esta es la segunda entrada del blog
        de zinnia.</p><p>Ten mucha suerte en tu Proyecto Fin de Carrera.</p></
```

```
        span>
10      <span itemprop="http://schema.org/award">0</span>
11      <span itemprop="http://schema.org/sameAs">Esta es la segunda entrada del blog
            de zinnia.Ten mucha suerte en tu Proyecto Fin de Carrera.</span>
12      <span itemprop="http://schema.org/keywords"></span>
13      <span itemprop="http://schema.org/description">False</span>
14      <span itemprop="http://xmlns.com/foaf/0.1/page">entry_detail.html</span>
15      <link href="http://pfc-llerena.rhcloud.com/easydata/publish/instance/zinnia/
            BlogPosting-Entry/1.xml" itemprop="http://schema.org/about">
16  </div>
17
18  <div itemid="http://pfc-llerena.rhcloud.com/easydata/publish/instance/zinnia/
        BlogPosting-Entry/1.xml" itemtype="http://schema.org/BlogPosting" itemscope >
19      <span itemprop="http://xmlns.com/foaf/0.1/skypeID">1</span>
20      <span itemprop="http://schema.org/headline">Saludo inicial</span>
21      <span itemprop="http://schema.org/sameAs">saludo-inicial</span>
22      <span itemprop="http://schema.org/datePublished">2013-12-01 11:01:17</span>
23      <span itemprop="http://xmlns.com/foaf/0.1/myersBriggs">None</span>
24      <span itemprop="http://schema.org/dateCreated">2013-12-01 11:00:40</span>
25      <span itemprop="http://schema.org/dateModified">2013-12-01 11:01:36.367237</
            span>
26      <span itemprop="http://schema.org/text"><p>Esto es una entrada de prueba para
            el blog de zinnia para probar junto con EasyData.</p></span>
27      <span itemprop="http://schema.org/award">0</span>
28      <span itemprop="http://schema.org/sameAs">Esto es una entrada de prueba para
            el blog de zinnia para probar junto con EasyData.</span>
29      <span itemprop="http://schema.org/keywords">easydata</span>
30      <span itemprop="http://schema.org/description">False</span>
31      <span itemprop="http://xmlns.com/foaf/0.1/page">entry_detail.html</span>
32      <link href="http://pfc-llerena.rhcloud.com/easydata/publish/instance/zinnia/
            BlogPosting-Entry/2.xml" itemprop="http://schema.org/about">
33  </div>
```

En la figura 6.4 se puede apreciar una captura de pantalla con el resultado de la validación producido por el validador de Google, donde se puede ver que que se extraen correctamente los datos del código Microdata.

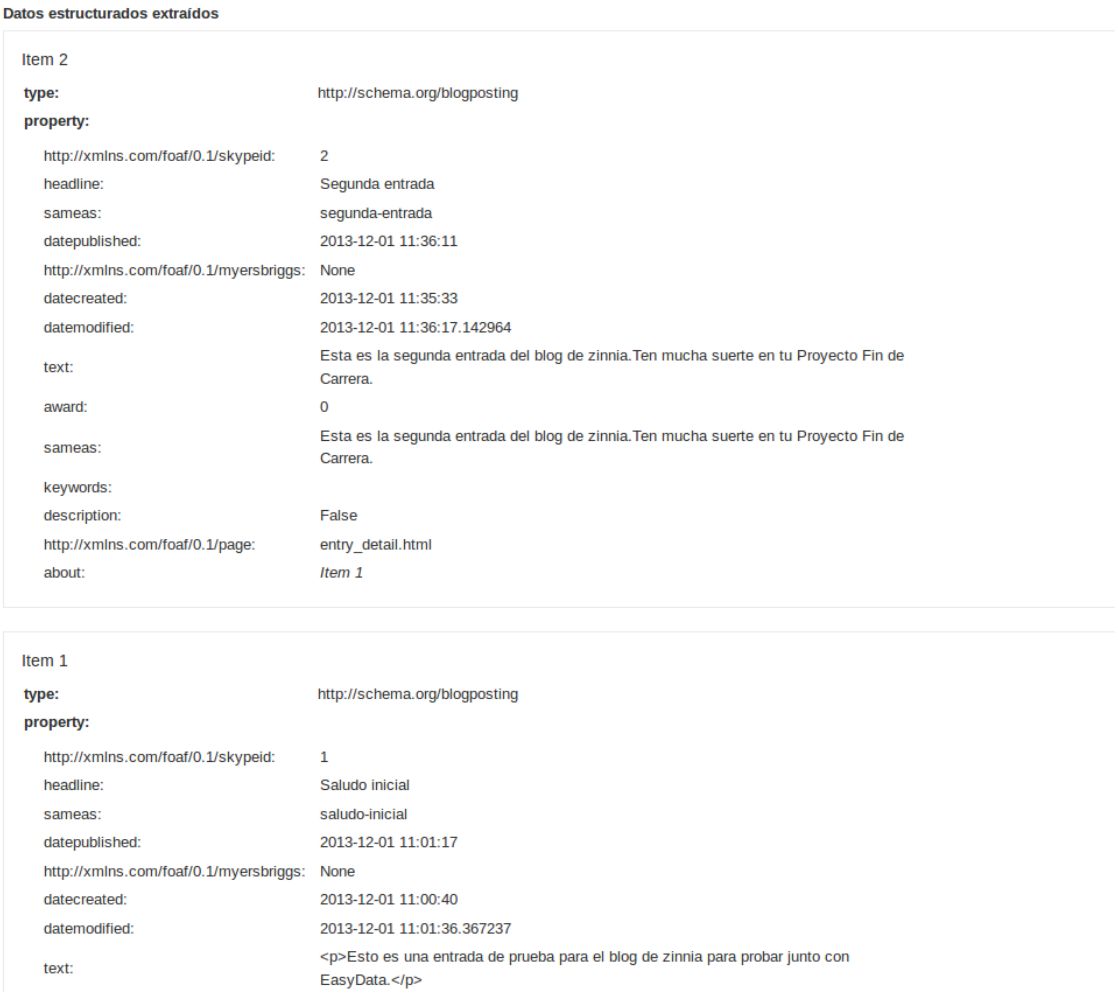


Figura 6.4: Resultado de validación Microdata

Resto de pruebas no funcionales

Por otro lado, para llevar a cabo el resto de pruebas no funcionales, se ha realizado una validación manual, asegurándonos de que se cumplen el resto de requisitos no funcionales impuestos para la aplicación, como son:

- **Seguridad:** comprobando que las vistas son únicamente accesibles para usuarios administradores de Django, y que únicamente se publican aquellos datos que el usuario ha marcado como visibles.
- **Portabilidad:** habiéndose usado siempre herramientas compatibles con python, o soportadas por múltiples plataformas.
- **Entorno tecnológico:** se ha desarrollado una aplicación perfectamente compatible con las versiones iguales o superiores a Django 1.4.

- **Mantenibilidad:** se trata de una aplicación desarrollada como un paquete python, el cual cumple con las reglas impuestas por el PEP8 para la escritura de código y habiéndose realizado una evaluación de la calidad del código fuente con la herramienta PyLint.

Parte III

Epílogo

Capítulo 7

Manual de instalación y explotación

A continuación, se detallan la instrucciones para la correcta instalación y explotación de la aplicación EasyData/Django para la apertura de datos en proyectos Django.

7.1. Introducción

El software EasyData/Django, ofrecerá al usuario que incorpore la aplicación a su proyecto facilidades a la hora de la publicación de los datos de su base de datos, pudiendo elegir únicamente aquellos que estime conveniente el usuario, haciendo uso de las diferentes ontologías disponibles en internet (como pueden ser, Schema, FOAF, etc.), y en diferentes formatos ampliamente utilizados (como pueden ser RDF, RDFa o microdata).

De esta forma, las principales funcionalidades que ofrece el software al usuario son las siguientes:

- Captación de los distintos modelos y fields de los mismos, de los que se encuentra compuesto el proyecto Django donde se instale la aplicación EasyData/Django.
- Carga de diferentes namespaces en formato RDF disponibles en internet.
- Configuración de los datos que serán visibles y aquellos que no se considerarán privados y no se mostrarán al exterior.
- Correspondencia de los modelos y fields del proyecto con los distintos elementos de las ontologías que se hayan cargado en el proyecto.
- Publicación de los datos en los diferentes formatos disponibles.

7.2. Requisitos previos

Para el correcto funcionamiento de la aplicación EasyData/Django, los únicos requisitos hardware existentes son los propios de cualquier aplicación Python y proyecto Django.

Por otro lado, en referencia a los requisitos software, como requisito base será necesario un sistema operativo el cual permita le ejecución de código Python y que cumpla con los requisitos software para la ejecución proyectos Django. Además, como requisitos específicos existen los siguientes:

- Al tratarse de una aplicación para proyectos Django, será necesario que el proyecto donde se vaya a incluir la aplicación EasyData/Django funcione bajo una versión igual o superior a la 1.4 de Django.
- Así mismo, también será necesario que se encuentre instalado el paquete *rdflib* de Python (a ser posible en su versión 4.0.1, ya que no se ha probado para versiones anteriores y no se puede asegurar su correcto funcionamiento), ya que esta aplicación hace uso de él para ofrecer la mayoría de sus funcionalidades descritas anteriormente.
- Además, para la generación de los gráficos mediante Graphviz de la configuración de los modelos del proyecto, será necesario el paquete pydot.

7.3. Inventario de componentes

Junto con la aplicación EasyData/Django se incluyen los siguientes paquetes software necesarios para el correcto funcionamiento de la aplicación:

- Framework jQuery en su versión 1.9.1.
- Plugin jQuery-UI para el framework jQuery en su versión 1.10.3.
- Framework CSS Bootstrap en su versión 3.0.0.

7.4. Procedimientos de instalación

A continuación se detallan cada uno de los pasos necesarios para la correcta instalación y explotación de la aplicación EasyData/Django dentro de un proyecto Django.

Lo primero que hay que hacer es instalar los paquetes necesarios para que la aplicación EasyData/Django funcione dentro de nuestro sistema, por lo que deberemos de instalar/actualizar la versión de Django de nuestro sistema a una versión igual o superior a la 1.4 y asegurarnos que nuestro proyecto Django es compatible con esta versión. Lo primero podemos hacerlo mediante el comando pip:

```
$> pip install django>=1.4
```

O bien, entrando en la web oficial de Django y descargando los ficheros fuente del repositorio de Django y siguiendo las instrucciones de instalación. Esto se encuentra correctamente explicado en la documentación oficial de Django.

Para la instalación del paquete rdflib de Python, podremos proceder exactamente de la misma forma, instalando el mismo desde pip de la siguiente forma:

```
$> pip install rdflib==4.0.1
```

O bien desde el repositorio oficial GIT del proyecto, el cual se puede encontrar en GitHub disponible para su descarga y perfectamente documentado.

De igual forma, para la instalación del paquete pydot de Python, podemos hacerlo a través de la herramienta pip de la siguiente forma:

```
$> pip install pydot
```

Para la instalación de los paquetes comentados anteriormente, también podemos hacer uso de la herramienta easy_install, la cual es bastante similar a pip, de la siguiente forma:

```
$> easy_install django
$> easy_install rdflib
$> easy_install pydot
```

Una vez hemos instalado los paquetes necesarios para la ejecución de la aplicación EasyData/Django, únicamente nos faltará descargar la aplicación EasyData/Django y copiar el contenido de la aplicación dentro de nuestro *PYTHONPATH* o dentro del propio directorio del proyecto Django, junto con el resto de aplicaciones que componen a dicho proyecto Django.

También existe la posibilidad de descargar la aplicación EasyData/Django desde el Python Package Index, como hemos hecho con las aplicaciones anteriores de las que hace uso el proyecto. Además, la aplicación EasyData/Django tiene especificados los requisitos necesarios para su ejecución, por lo que al instalarla desde el pip, el propio instalador se encargará de instalar las aplicaciones de las que depende, ahorrándonos los pasos anteriores. Para ello procederemos de igual forma que en los casos anteriores introduciendo uno de los siguientes comandos:

```
$> easy_install django-easydata
```

```
$> pip install django-easydata
```

Una vez tengamos la aplicación EasyData/Django, el siguiente paso será incluir dicha aplicación dentro del proyecto Django donde queremos hacer uso de la misma. Para ello, solo tendremos que añadir la aplicación en el settings.py de nuestro proyecto dentro de la variable INSTALLED_APPS, quedando de la siguiente forma:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin',
    'easydata',
)
```

Cuando hayamos incluido la aplicación *EasyData/Django* en nuestro proyecto Django, lo siguiente que deberemos hacer será sincronizar la base de datos, ya que la aplicación *EasyData/Django* hace uso de sus propias tablas para almacenar la información respecto a los namespaces, modelos y fields existentes. Para ello, como debería de hacerse con cualquier aplicación, deberemos ejecutar el siguiente comando en el directorio donde se encuentre el manage.py de nuestro proyecto:

```
$> python manage.py syncdb
```

El siguiente paso de la instalación de la aplicación EasyData/Django en nuestro proyecto Django, será añadir las urls propias de la aplicación a las urls de nuestro proyecto, de forma que el proyecto pueda servir dichas urls. Para realizar esto, añada la siguiente línea al fichero url.py de su proyecto:

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

El nombre asignado de easydata para la url es opcional, de tal forma que se puede modificar por otro que se adecue mejor a las necesidades del usuario.

Como último paso, la aplicación EasyData/Django usa staticfiles para servir los ficheros estáticos de la aplicación, por lo que en nuestro proyecto deberá de estar soportado. Para cargar los ficheros estáticos de la aplicación en nuestro proyecto, se debe de lanzar el siguiente comando desde nuestro directorio del proyecto:

```
python manage.py collectstatic
```

Para más información acerca de los ficheros estáticos y de cómo se sirven estos en un proyecto Django, visite la documentación oficial.

Por último, se informa al usuario que la aplicación EasyData/Django hace uso del sistema de usuarios y de roles de Django para el control de permisos a las vistas de configuración de la publicación de datos, restringiendo el acceso a todo aquel usuario que no posea los permisos de super usuario de Django. Por lo que si desea crear un usuario superusuario de Django, deberá hacerlo mediante el siguiente comando en la terminal:

```
python manage.py createsuperuser
```

El asistente se encargará de solicitarle los datos necesarios para la creación del usuario. Si por otro lado, desea poder utilizar un usuario existente que no está marcado como superusuario, únicamente deberá marcar a True los campos *is_staff* y *is_superuser*.

7.4.1. Otras configuraciones

Con los pasos explicados anteriormente sería suficiente para que la aplicación pudiera funcionar correctamente, pero existen otros parámetros de configuración que pueden modificarse, para personalizar la misma a las necesidades del usuario.

Cabecera de URLs

La aplicación EasyData utiliza el framework Sites de Django para obtener la cabecera de las URLs (dominio), pero si el usuario lo desea, puede especificar una función alternativa para calcular las cabeceras. Para ello, deberá implementar una función propia que devuelva una cadena de texto con la cabecera de las URLs, bajo el nombre *url_header* y especificar en el settings de Django el módulo donde se encuentra dicha función mediante la variable *EASYDATA_URL_HEADER*, de manera que EasyData sepa donde buscar dicha función.

URIs de modelos

Por otro lado, la aplicación EasyData de Django, genera una URI única para cada una de las instancias de los modelos del proyecto Django donde ha sido instalado, la cual hace referencia a el fichero RDF con los datos de dicha instancia con el mapeo configurado.

En vez de usar la URI que hace referencia al fichero RDF con los datos de la instancia, al usuario le podría interesar que se utilizase otro tipo de URI para identificar a las instancias de un determinado modelo (como por ejemplo la URL donde de la web donde se lista la información de dicha instancia), por lo que existe la posibilidad de que el usuario especifique la URL para un determinado modelo implementando el método *easydata_generate_url*, el cual devolverá el path absoluto (excepto la cabecera de la URL que utilizará la función explicada en 7.4.1).

URIs para D2Rq

La aplicación EasyData/Django generará una URI por defecto para el fichero de configuración de D2Rq, aunque también será posible configurar esta URI para que utilice una especificada por el usuario. Para modificar la URI que utilizará la aplicación para D2Rq, se deberá crear en el modelo que se desee modificar la URI una variable de nombre *easydata_url_d2rq* la cual sea de tipo string y contenga en el formato utilizado por D2Rq la plantilla que se utilizará para generar las URI del modelo.

Limitar ficheros RDF

La aplicación EasyData/Django por defecto limitará a un número máximo de 50 instancias a la hora de exportar los datos de estas a través de RDF, de forma que se evite sobrecargar al servidor debido a modelos con grandes volúmenes de datos. De igual forma, este número máximo de instancias puede ser configurado por el usuario administrador, añadiendo en el fichero settings del proyecto la variable *EASYDATA_PUBLISH_LIMIT*, el cual recibirá un número entero que hará referencia al número máximo de instancias a generar en los ficheros RDF.

Creación de nuevos template tags para RDFa y Microdata

A la hora de realizar la publicación de los datos en formato HTML, se utiliza en la aplicación los template tags creados para tal fin, donde cada uno de ellos utilizan diferentes etiquetas HTML para realizar la publicación. Puede que estas etiquetas no sean precisamente las que el usuario necesita para realizar la publicación, por ello, se ofrece al usuario la posibilidad de crear nuevos template tags que usen etiquetas HTML diferentes.

Los template tags para publicar los datos, únicamente deben de devolver el resultado de invocar a las funciones *generate_html_rdfa* y *generate_html_microdata*, ubicadas en los ficheros easydata_rdfa.py y easydata_microdata.py respectivamente, del directorio templatetags del proyecto. Estas funciones tienen las siguientes cabeceras:

```
generate_html_rdfa(instance, tag1, tag2, content)

generate_html_microdata(instance, tag1, tag2, content)
```

Donde cada uno de los atributos que reciben las funciones hacen referencia a:

- **instance**: es la instancia del modelo de la que se desea generar el HTML con los datos.
- **tag1**: es la etiqueta HTML que se desea usar para incluir la información principal de la instancia.
- **tag2**: es la etiqueta HTML que se desea usar para incluir cada uno de los datos de la instancia.
- **content**: recibe True o False, lo que indica que para mostrar los datos se haga uso o no, del atributo content de las etiquetas.

7.5. Procedimientos de operación y nivel de servicio

No existen procedimientos necesarios que deban de llevarse a cabo para asegurar el correcto funcionamiento de la aplicación, a excepción de los descritos anteriormente para la puesta en

marcha del mismo. Si bien, como ocurre con cualquier proyecto software, se recomienda la realización periódica de back-ups, para prevenir la pérdida de datos, y en el caso de esta aplicación, la perdida de la configuración del mapeo de los modelos.

Por otro lado, se advierte al usuario que se está trabajando con una aplicación para la publicación de datos de forma controlada de bases de datos de proyectos Django, lo cual significa, que debe de prestarse excesivo cuidado a la hora de indicar los datos que van a mostrarse públicamente, en referencia a los problemas que pudiesen acarrear la publicación de ciertos datos sensibles de a ser publicados. Esta responsabilidad recae únicamente sobre el usuario administrador de la aplicación, el cual deberá de prestar especial atención al apartado de visibilidad de los datos.

7.6. Pruebas de implantación

Una vez haya realizado todos los pasos descritos en el apartado 7.4, para comprobar que la aplicación ha sido instalada correctamente dentro de su proyecto Django, deberá de acceder a la siguiente url, donde se le mostrará una pantalla de bienvenida a la aplicación. Sino consigue acceder a la aplicación, revise los pasos descritos anteriormente.

http://base_url/easydata/

Donde:

- Deberá sustituir el protocolo http, por cualquier otro protocolo diferente que utilice, como por ejemplo comunicación segura mediante SSL.
- base_url hace referencia a la dirección base de su proyecto. Si ejecuta por defecto runserver de django, esta dirección sería 127.0.0.1:8000.
- El nombre de easydata es el que hemos asignado en el paso anterior del tutorial, si se hubiera asignado otro distinto, entonces este debería modificarse por el nombre asignado en el fichero urls.py.

Suponiendo que estemos usando el protocolo http, con la dirección IP 127.0.0.1:8000 y el nombre para la url indicado en la documentación, la dirección a la que deberíamos de acceder, sería la siguiente:

http://127.0.0.1:8000/easydata/

Capítulo 8

Manual de usuario

A continuación se detallan las instrucciones de uso de la aplicación EasyData/Django, las cuales deberá seguir para un correcto funcionamiento de la aplicación.

8.1. Introducción

El presente capítulo es un manual de usuario, el cual le permitirá tener los conocimientos necesarios para poder utilizar la aplicación EasyData/Django. Esta aplicación le permitirá realizar un mapeo entre cada uno de los modelos de su proyecto Django donde instale la aplicación, y diferentes ontologías existentes en la web, para posteriormente realizar la publicación de sus datos en función al mapeo que ha realizado con dichas ontologías, de tal forma que el contenido de su aplicación, sea perfectamente entendible por cualquier entidad que acceda.

Haciendo uso de las ontologías anteriormente citadas, al tratarse de espacios de nombres establecidos, estará marcando su contenido web de forma que los datos que aquí figuren sean mucho más comprensibles, ya que esta dotando a los datos que publica de un mayor contenido semántico.

Siga este manual, una vez haya concluido con la instalación de la aplicación EasyData/Django, tal y como se detalla en el apartado 7, de la presente memoria.

8.2. Características

Las características o funcionalidades principales que le ofrece la aplicación EasyData/Django, son las que se especifican a continuación:

- Captación de los modelos y fields que componen su proyecto Django.

- Carga de namespaces u ontologías.
- Configuración de la privacidad de los datos.
- Mapeo de modelos y fields de su proyecto con las entidades y propiedades de las distintas ontologías.
- Publicación de datos en diferentes formatos.
- Generación de fichero D2Rq con la configuración realizada en la aplicación.
- Generación de gráfico con la configuración del mapeo.

8.3. Requisitos previos

Los requisitos previos que debe cumplir la máquina donde vaya a instalarse la aplicación EasyData/Django son los siguientes:

- Sistema operativo con soporte para la ejecución de código python.
- Versión de python 2.6 o 2.7.
- Versión de Django igual o superior a la 1.4.
- Paquete rdflib de python en su versión 4.0.1.

8.4. Utilización

A continuación, se va a explicar como realizar cada una de las funcionalidades comentadas en el apartado 8.2. La explicación de las mismas, se realiza en el mismo orden en que deberán de llevarse a cabo para que todo funcione correctamente.

8.4.1. Captación de modelos y fields

La captación de modelos y fields por introspección de su proyecto Django es bastante sencilla. Este procedimiento se lleva a cabo desde el manage.py de su proyecto Django, ya que la aplicación EasyData/Django añade una nueva opción al manage.py, que le permitirá realizar este procedimiento con toda facilidad.

Para ello únicamente deberá de dirigirse al directorio de su proyecto Django donde se encuentra el fichero manage.py, y desde una terminal de escritorio, deberá ejecutar el siguiente comando:

```
$> python manage.py loadmodels
```

Una vez el proceso de carga de los modelos y fields haya concluido, se le informará mediante un mensaje. En la figura 8.1 puede ver el resultado de como sería el proceso de captación.

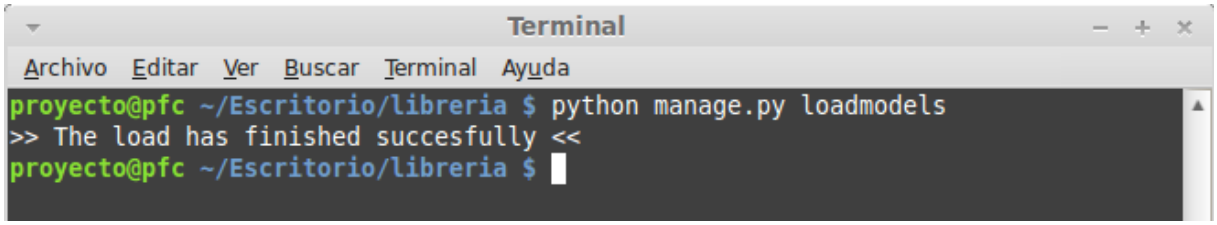


Figura 8.1: Resultado de captación de modelos y fields

8.4.2. Carga de namespaces

El siguiente paso imprescindible para poder realizar el mapeo de sus modelos con las ontologías, es tener alguna cargada en la aplicación. La carga de ontologías o namespaces en la aplicación, se hace desde dentro de la aplicación, en el apartado llamado Namespace.

Cuando acceda a este apartado, le aparecerá una lista con todos los namespaces disponibles (Figura 8.2) para ser usados (inicialmente la lista estará vacía). Desde este apartado, podrá tanto añadir nuevos namespaces (Figura 8.3), como editarlos/actualizarlos (Figura 8.4), o incluso eliminarlos[1].

[1]Tenga especial cuidado a la hora de borrar los namespaces, ya que perderá también la configuración que tenga entre sus modelos y el namespace. A la hora de borrar un determinado namespace, se advertirá al usuario de que esté seguro de realizar esta acción.

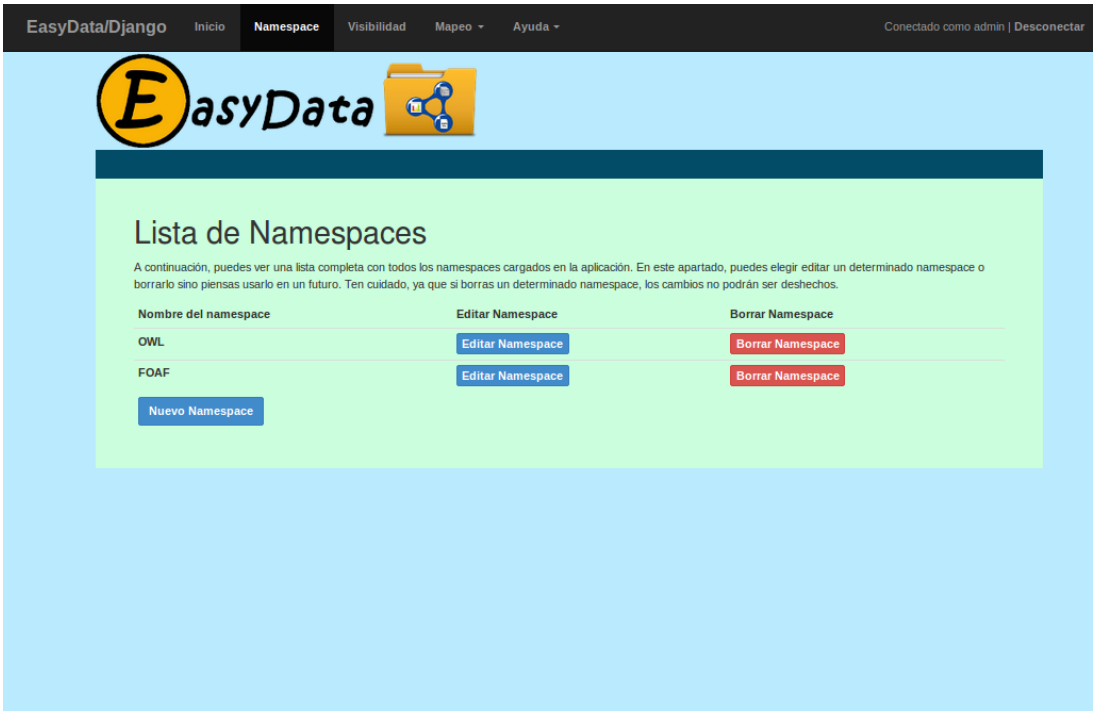


Figura 8.2: Listado de namespaces cargados en la aplicación

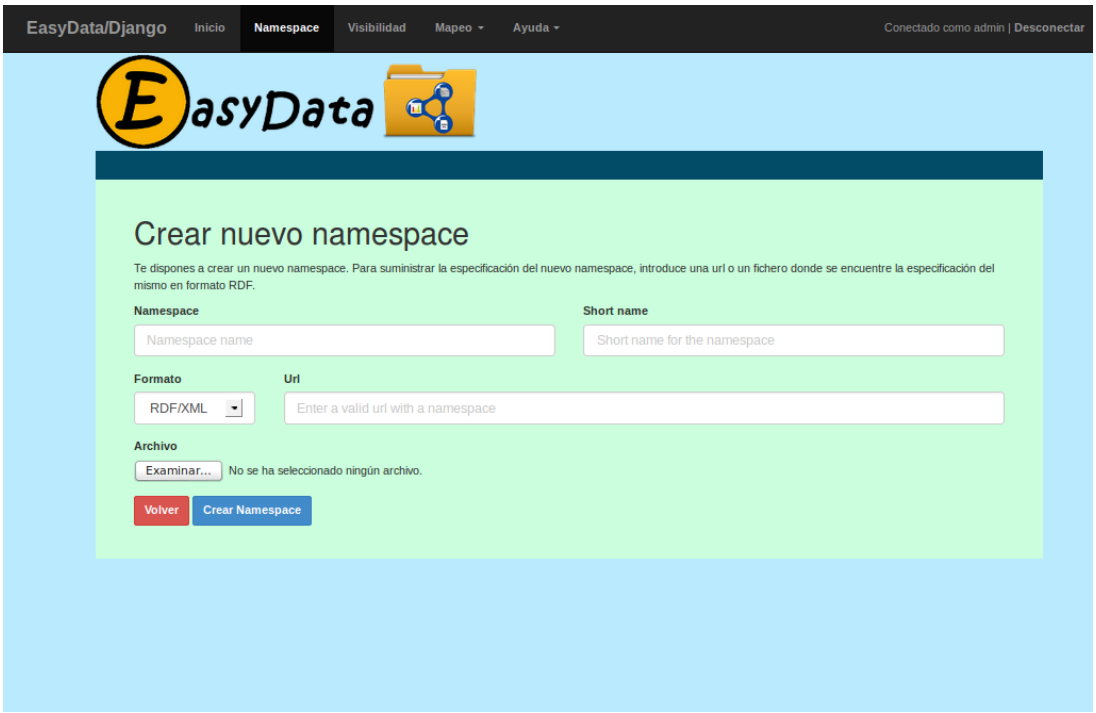


Figura 8.3: Formulario de creación de un nuevo namespace

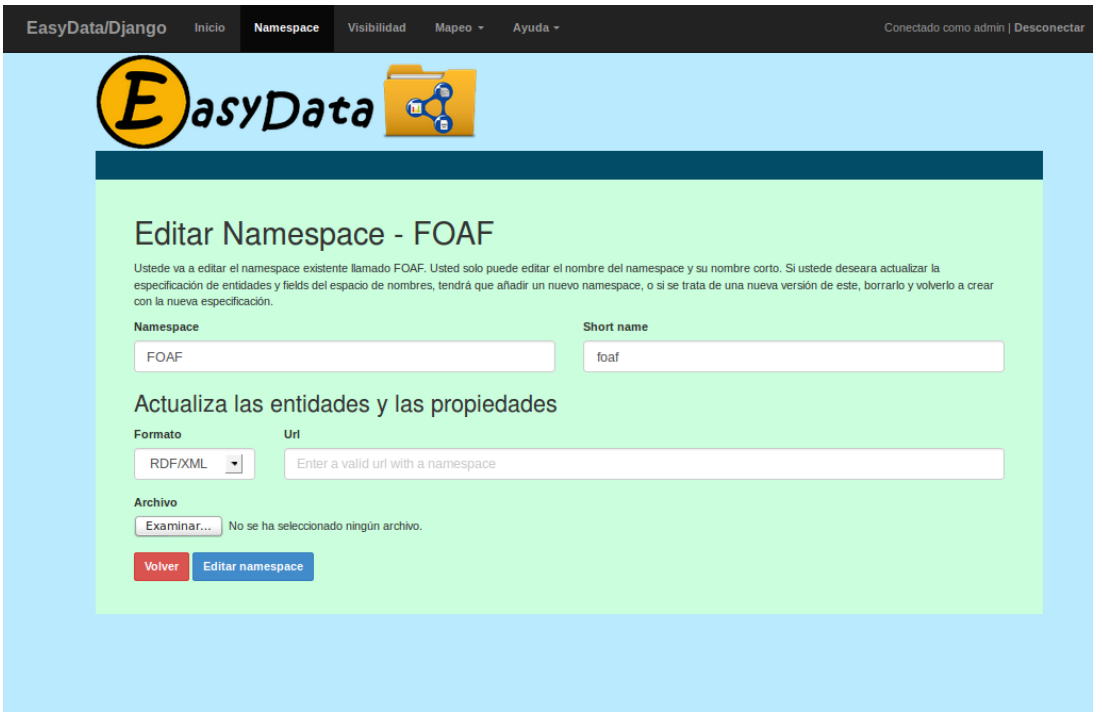


Figura 8.4: Pantalla de edición de un namespace existente

Para cargar un nuevo namespace, pulse sobre el botón azul de *Nuevo Namespace*, y le aparecerá un formulario donde se le solicitará la siguiente información:

- **Namespace:** esto es un nombre identificativo que recibirá el namespace, el cual no influye en el mapeo. Este dato es único y no puede repetirse. Tiene como máximo 40 caracteres.
- **Short name:** este dato hace referencia a un nombre corto que represente el namespace, el cual se utilizará a la hora de generar tanto el RDF como RDFa, etc... Este dato es único y no puede repetirse con otros ya existentes. Tiene como máximo 15 caracteres.
- **Formato:** es el formato del fichero que se suministra con la especificación de la ontología. Actualmente están disponibles los formatos RDF/XML y RDF/Ntriples.
- **Url:** Es una url donde se encuentra el fichero con la especificación del namespace. Este dato no es obligatorio y solo debe suministrarse la url o el archivo.
- **Archivo:** Es un fichero donde se encuentra la especificación del namespace. Este dato no es obligatorio y solo debe suministrarse la url o el archivo.

Una vez introducidos los datos, pulse en *Crear Namespace*, para cargar el mismo en el sistema. Este proceso puede tardar un poco, en función del tamaño del namespace, ya que se debe de leer toda la especificación y cargarla en la base de datos.

Una vez se haya creado el namespace, se le redirigirá a la vista principal de namespaces, y podrá observar que aparece en la lista el nuevo que acaba de crear. Si por el contrario hubo algún error, se le informará y deberá de corregir el formulario.

A la hora de editar/actualizar un namespace, le aparecerá un formulario similar al de creación de un nuevo namespace, donde los campos nombre y short name, aparecerán rellenos con los datos del namespace, de tal forma que pueda editar los mismos. Por otro lado, también encontrará los apartados de formato, url y archivo, donde para este caso, la utilidad de los mismos difiere un poco al del caso anterior, ya que esta vez se cargará la especificación de las entidades y propiedades del namespace, para añadir nuevas entidades o propiedades nuevas que pudiesen haber sido incluidas posteriormente en la especificación, o para actualizar las relaciones existentes entre entidades y propiedades del namespace, con el resto de entidades y propiedades de otros namespaces existentes en la aplicación, que no existiesen anteriormente.

Tanto a la hora de cargar un nuevo namespace como de actualizar uno existente, esto se hará como bien se puede apreciar en el formulario, haciendo uso de ficheros en formato rdf. Para que la aplicación pueda reconocer tanto las entidades como las propiedades y las relaciones entre los mismos, estos deben de estar especificados haciendo uso de las siguientes etiquetas defindas en los namespaces RDFS y OWL:

- Para la identificación de clases:
  - **rdfs:Class**.
  - **owl:Class**.
- Para la identificación de propiedades:
  - **rdf:Property**.
  - **owl:ObjectProperty**.
  - **owl:DatatypeProperty**.
  - **owl:AnnotationProperty**.
- Para la obtención de características de las clases y propiedades:
  - **rdfs:comment**: para captar una descripción de la entidad o propiedad.
  - **rdfs:label**: para captar la etiqueta de la entidad o propiedad.
  - **rdfs:domain**: indica a qué entidades pertenece una determinada propiedad.
  - **rdfs:range**: indica con qué entidades se relaciona una determinada propiedad.
  - **rdfs:subClassOf**: indica de qué entidad es hija una determinada entidad.

8.4.3. Configuración de la privacidad

Un punto muy importante a la hora de exportar nuestros datos, es decidir qué datos queremos que se publiquen y qué otros no, ya que en muchos casos, nos será interesante evitar que ciertos datos puedan ser accedidos por terceras personas, porque se traten de datos sensibles, como

contraseñas, números de cuentas bancarias, etc... o sencillamente porque no deseemos que estos puedan verse.

La aplicación le permite la posibilidad de decidir qué datos se ocultaran a los usuarios a la hora de publicar los mismos. Esto se hará en el apartado de *Visibilidad* (Figura 8.5). En este apartado, le aparecerá un listado con cada una de las aplicaciones instaladas en su proyecto Django, y pulsando en configurar visibilidad de una determinada aplicación, se le mostrará una nueva pantalla (Figura 8.6) con una pestaña por cada uno de los modelos que contiene la aplicación.

Figura 8.5: Pantalla de configuración de la privacidad - Aplicaciones

En cada una de las pestañas de los modelos, podrá elegir si este modelo es visible o privado completamente al exterior, o definirlo por fields en concreto, indicando la visibilidad de los mismos.

Por defecto, para prevenir que inicialmente algún dato sea publicado por error, todos los modelos están marcados como privados, de tal forma que será tarea del usuario decidir cuáles de ellos se harán visibles.

Figura 8.6: Pantalla de configuración de la privacidad - Modelos

8.4.4. Mapeo de modelos y fields

El siguiente y último paso antes de la publicación de los datos, es indicarle a la aplicación EasyData/Django, con qué entidades y propiedades de los distintos namespaces van a estar relacionados nuestros modelos. Este paso se conoce como el *Mapeo de los datos*, y por consiguiente se llevará a cabo en el apartado de la aplicación de *Mapeo*.

Si accedemos a este apartado, nos aparecerá una lista desplegable (Figura 8.7). Cada uno de los elementos de la lista, representan a cada una de las aplicaciones de nuestro proyecto. Si seleccionamos una de ellas, para que se expanda dicho elemento, nos aparecerá cada uno de los modelos que componen a dicha aplicación con dos elementos select. En esta parte realizaremos el mapeo de los modelos, donde en el primero de los selects, seleccionaremos el namespace que queremos utilizar de los que tenemos cargados en la aplicación, y el siguiente select se cargará con las distintas entidades que componen al namespace. De esta forma deberemos de mapear cada uno de los modelos que nos interesen (no es necesario realizar el mapeo de todos, solo aquellos que nos vaya a interesar publicar). Una vez realizado el mapeo de los modelos, pulsaremos sobre el botón de *Salvar cambios*.

Figura 8.7: Pantalla de mapeo de los modelos

Una vez hemos realizado el mapeo de los modelos y hemos salvado los cambios, se activará el botón verde posicionado a la derecha de cada modelo que hemos realizado su mapeo. Esto significa que podemos pasar ahora a realizar el mapeo de los fields de dichos modelos, pulsando en dicho botón.

Cuando pulsemos sobre dicho botón, se nos llevará a un apartado parecido al anterior, donde aparecerán por separado los atributos y relaciones del modelo (Figura 8.8). Al igual que en el caso anterior, cada uno dispondrá de dos selects, uno para elegir el namespace, y otro para elegir la propiedad con la que deseamos mapear el field. Por defecto, aparece seleccionada la entidad del namespace con que hemos mapeado el modelo, pero se puede modificar para utilizar otras propiedades del mismo namespace u otro namespace diferente.

Una vez se haya concluido con el mapeo de los field, igual que anteriormente con los modelos, pulsaremos sobre el botón de *Salvar cambios*, para que estos queden registrados en la aplicación.

Figura 8.8: Pantalla de mapeo de los fields de un determinado modelo

8.4.5. Publicación de los datos

Una vez llegados a este punto, donde ya tenemos cargados los namespaces que vamos a utilizar, tenemos configurada la privacidad de nuestros datos y el mapeo de los mismos con los namespaces, solo queda publicar los datos de los mismos.

Para la publicación de los datos de los modelos de nuestro proyecto, existen distintas posibilidades, según el formato y la vía a través de la cual los queramos publicar:

- A través de url generadas para instancias en concreto de los modelos, para todas las instancias de un modelos, o para todos los modelos mapeados con una determinada entidad de un namespace.

- Mediante código HTML, usando los formatos para el marcado de etiquetas HTML, como son RDFa y Microdata.

- A través de la plataforma D2Rq haciendo uso del endpoint SPARQL que ofrece.

Mediante URLs

Como hemos comentado anteriormente, una de las posibles formas que hay para publicar los datos de su aplicación es a través del uso de URLs. Estas URLs servirán sus datos en formatos como puede ser RDF/XML, RDF/Ntriples o RDF/Trutle.

Para los ejemplos, supondremos que nuestro proyecto se encuentra instalado en la URL 127.0.0.1:8000.

La aplicación EasyData/Django creará automáticamente tres tipos diferentes de URLs:

- URLs específicas para instancias concretas de cada uno de los modelos. Estas URLs están compuestas de la aplicación, el tipo (la entidad), modelo y clave primaria de la instancia, de la siguiente forma:

  *http://127.0.0.1:8000/easydata/publish/instance/**aplicación/tipo-modelo/pk**.(xml|nt|ttl)*

  Donde las palabras marcadas en negrita, se sustituirán por los datos comentados anteriormente, y la clave primaria irá seguida del formato en el que deseamos recibir los datos.

- También se crearán URLs concretas para cada modelo de nuestro proyecto, de tal forma que nos devolverá un fichero RDF con cada una de las instancias de dicho modelo. Dicha URL será de la siguiente forma:

  *http://127.0.0.1:8000/easydata/publish/model/**aplicación/tipo-modelo**.(xml|nt|ttl)*

  Donde las palabras marcadas en negrita, se sustituirán por los nombres de la aplicación y del modelo, seguido el modelo del formato en el que deseamos recibir los datos.

- Por último, también se crearán URLs concretas para cada una de las entidades de los namespaces, de tal forma que nos devolverá un fichero RDF con cada una de las instancias de los modelos mapeados con dicha entidad. Dicha URL tendrá el siguiente formato:

  *http://127.0.0.1:8000/easydata/publish/type/**namesapace/entidad**.(xml|nt|ttl)*

  Donde la palabra namespace, se sustituirá por el nombre del namespace y la palabra entidad por el nombre de la entidad de la que queremos obtener los datos en RDF, seguido del formato RDF en el que queremos que nos muestre los datos.

Para obtener información mucho mas concreta sobre todo lo comentado anteriormente, puede consultar los apartados *Modelos configurados* y *Entidades relacionadas* de la ayuda de la aplicación, donde se le muestra todo lo comentado anteriormente, con ejemplos sobre datos reales de su proyecto.

Mediante RDFa y Microdata

Para la inclusión de los datos en formato HTML en nuestras plantillas de Django, se hará uso de los template tags de Django, que no son mas que funciones, que pueden ejecutarse desde las plantillas.

Estos template tags, son funciones que reciben una determinada instancia de un modelo, y generarán el html con las etiquetas RDFa o Microdata (siempre que el modelo de la instancia haya sido mapeado previamente), de tal forma que cuando una máquina acceda a nuestra aplicación, sea capaz de interpretar dichas marcas como si de un fichero RDF se tratara y pueda interpretar la información de la misma.

A continuación se indican los template tags disponibles, tanto para la generación de código RDFa como Microdata.

Microdata

Para poder hacer uso de los template tags para la generación de código Microdata, se deberá de cargar en la plantilla Django el módulo donde se encuentran los template tags, incluyendo el siguiente código:

{ % load easydata_ microdata %}

Una vez tenemos cargados los template tags para microdata, únicamente tendremos que hacer uso en las plantillas de ellos. Los template tags para microdata disponibles son los siguientes:

En el caso de que queramos generar el código microdata de una determinada instancia de un modelo y que muestre todos los fields visibles y mapeados de este:

{ % microdata_ ul instance %}
{ % microdata_ div_ meta instance %}
{ % microdata_ div_ span instance %}

En el caso de que queramos generar el código microdata de una instancia, indicándole el field en concreto que deseamos que muestre:

{ % microdata_ li_ field instance “field_ name” %}
{ % microdata_ meta_ field instance “field_ name” %}
{ % microdata_ span_ field instance “field_ name” %}

Cuando se hace uso de los templatetags que genera información de un determinado field de la instancia, solo se genera la etiqueta HTML con el tipo del dato y el dato en cuestión, por lo

que no se incluye información del tipo de la instancia a la que pertenece el dato, y que deberá incluir el usuario. Para ello, existe un template tag de bloque que indicándole la instancia y el tipo de etiqueta que queremos usar, nos genera dicha etiqueta de apertura y cierre HTML con la información Microdata del tipo de la instancia. La estructura de este template tag es la siguiente:

{ % microdata_ open_ tag instance "tag" %}
Resto de etiquetas de la instancia
{ % microdata_ end_ tag %}

A su vez, puede que una determinada instancia de un modelo de datos, contenga relaciones con otra instancia de otro modelo, que por defecto al generar el código HTML con Microdata, lo que se hará será generar una etiqueta con la referencia a la URI que representa a la instancia con la que se relaciona. Puede que sea de interés para el usuario, que en vez de dicha referencia, se generen determinados datos de la instancia indicando la relación de esta (anidar instancias). Para ello, existe también un template tag de bloque similar al anterior, donde se indican las instancias relacionadas, el nombre del field y la etiqueta mediante la cual relacionarlos. La estructura de este template tag es la siguiente:

{ % microdata_ open_ tag_ interno instancia_ padre instancia "field" "tag" %}
Resto de etiquetas de la instancia
{ % microdata_ end_ tag_ interno %}

RDFa

Para poder hacer uso de los template tags para la generación de código RDFa, se deberá de cargar en la plantilla Django el módulo donde se encuentran los template tags, incluyendo el siguiente código:

{ % load easydata_ rdfa %}

Una vez tenemos cargados los template tags para RDFa, únicamente tendremos que hacer uso en las plantillas de ellos. Los template tags para RDFa disponibles son los siguientes:

En el caso de que queramos generar el código RDFa con una instancia y que muestre todos los fields visibles y mapeados de este:

{ % rdfa_ ul instance %}
{ % rdfa_ div instance %}
{ % rdfa_ div_ span instance %}

En el caso de que queramos generar el código RDFa con instancias e indicándole el field en concreto que deseamos que muestre:

{ % rdfa_ li_ field instance “field_ name” %}
{ % rdfa_ div_ field instance “field_ name” %}
{ % rdfa_ span_ field instance “field_ name” %}

Cuando se hace uso de los templatetags que genera información de un determinado field de la instancia, solo se genera la etiqueta HTML con el tipo del dato y el dato en cuestión, por lo que no se incluye información del tipo de la instancia a la que pertenece el dato, y que deberá incluir el usuario. Para ello, existe un template tag de bloque que indicándole la instancia y el tipo de etiqueta que queremos usar, nos genera dicha etiqueta de apertura y cierre HTML con la información RDFa del tipo de la instancia. La estructura de este template tag es la siguiente:

{ % rdfa_ open_ tag instance “tag” %}
Resto de etiquetas de la instancia
{ % rdfa_ end_ tag %}

A su vez, puede que una determinada instancia de un modelo de datos, contenga relaciones con otra instancia de otro modelo, que por defecto al generar el código HTML con RDFa, lo que se hará será generar una etiqueta con la referencia a la URI que representa a la instancia con la que se relaciona. Puede que sea de interés para el usuario, que en vez de dicha referencia, se generen determinados datos de la instancia indicando la relación de esta (anidar instancias). Para ello, existe también un template tag de bloque similar al anterior, donde se indican las instancias relacionadas, el nombre del field y la etiqueta mediante la cual relacionarlos. La estructura de este template tag es la siguiente:

{ % rdfa_ open_ tag_ interno instancia_ padre instancia “field” “tag” %}
Resto de etiquetas de la instancia
{ % rdfa_ end_ tag_ interno %}

Para obtener información más detallada sobre el uso de los template tags, puede dirigirse al apartado de *Uso en plantillas* de la ayuda de la aplicación, donde podrá observar todos los template tags disponibles, así como ejemplos de uso de cada uno y resultado generado, sobre datos reales de sus modelos.

Enlace en HTML

Además de los template tags comentados anteriormente, dispone de un template tag más, el cual a partir de una instancia de un modelo, genera el código HTML de un enlace “<a>” con la URI donde se encuentra la información RDF de la instancia que se le indicó.

Para hacer uso de este template tag, debe de importarlo de la siguiente forma:

{ % load easydata_ links %}

Y por último, el template tag se usará dentro de la plantilla Django de la siguiente forma:

{ % easydata_ include_ link instance %}

8.4.6. Generación de fichero D2Rq

La plataforma D2RQ de código abierto, es un sistema para el acceso a los datos de bases de datos relacionales, como si se tratasen de grafos RDF virtuales de solo lectura. De esta forma, ofrece a los usuario acceso mediante RDF al contenido relacional de las bases de datos, sin la necesidad de tener que replicar estos sobre un almacenamiento en formato RDF. Resumiendo, haciendo uso de D2RQ podrás:

- Hacer consultas sobre una base de datos que no sea de tipo RDF haciendo uso de SPARQL.
- Acceder al contenido de la base de datos, como Linked Data sobre la web.
- Crear volcados personalizados de la base de datos en formato RDF.
- Acceder a información de una base de datos que no sea de tipo RDF, haciendo uso de la API Apache Jena.

Para realizar las tareas anteriormente citadas, el software D2Rq necesita de un fichero de configuración, donde se plasme el mapeo entre las tablas y columnas de la base de datos y las etiquetas de los distintos namespaces, de forma que este software pueda procesar las consultas SPARQL y crear el contenido RDF. Este fichero, puede ser generado en la aplicación EasyData/Django, haciendo uso del mapeo realizado en la propia aplicación.

Para realizar dicho mapeo, al igual que cuando realizó la resolución de los modelos y fields del proyecto, deberá ejecutar desde la terminal el siguiente comando del manage.py de Django.

```
$> python manage.py easydata_d2rq
```

Este comando le mostrará por pantalla la configuración en formato *Turtle* (ttl) realizada por usted en la aplicación lista para ser usada en la aplicación D2Rq. Si desea guardar la configuración en un fichero, solo tiene que redirigir la salida hacia un fichero de la siguiente forma:

```
$> python manage.py easydata_d2rq > configuracion.ttl
```

Una vez generado el fichero, deberá editarlo mediante cualquier editor compatible con este formato de fichero, para introducir las opciones de conexión a la base de datos (nombre de usuario, contraseña, host, controlador, etc...). Además, este se trata de un documento base con la estructura que hay definida de la base de datos dentro de la aplicación, el cual usted podrá editar para añadir nuevos tipos de relaciones, propiedades, etc...

Tenga en cuenta que el software D2Rq es bastante complejo y ofrece al usuario una gran cantidad de posibilidades a la hora de trabajar con los datos y realizar el mapeo de los mismo.

Además, este software trabaja directamente sobre la estructura de la base de datos y esta en los proyectos Django se encuentra estructurada en función de la forma de trabajar del ORM del framework, de tal forma que no siempre se corresponderán los modelos con las tablas existentes en la base de datos, principalmente cuando existan herencias y relaciones complejas. Por estas dos razones, el resultado final del fichero puede no ser definitivo, sino que servirá como base para que el usuario no tenga que trabajar desde un fichero completamente en blanco y pueda aprovechar gran parte del trabajo realizado sobre la aplicación, pero este requerirá finalmente de la intervención del usuario.

Finalmente, una vez tenga el fichero de configuración para D2Rq podrá realizar multitud de acciones, como desplegar un servidor donde los usuarios puedan consultar los datos publicados. Este servidor podrá ser desplegado directamente desde la línea de comandos (`http://d2rq.org/d2r-server#command-line`), o bien como una aplicación web J2EE dentro de un servidor Apache Tomcat o Jetty (`http://d2rq.org/d2r-server#servlet-container`).

Para más información acerca de cómo realizar consultas, modificar los ficheros de configuración D2Rq, ejecución del servidor web, y el resto de utilidades y herramientas que le proporciona la plataforma D2Rq, consulte la página oficial [Ber13] donde se le explica todo esto mucho más detallado.

8.4.7. Generación de gráfico

A la vez que vaya realizando el mapeo de sus modelos y fields con las correspondientes entidades y propiedades de los namespaces, podrá obtener un grafo donde podrá visualizar dicha configuración. En ella se apreciarán cada uno de los modelos con sus atributos, así como las relaciones existentes entre modelos, y junto a cada uno de ellos, las entidades o propiedades de los namespaces con las que están mapeados.

Para generar dicho grafo, deberá de hacer click sobre el apartado de *Mapeo*, y en el menú desplegable que le aparece, pulsar sobre la opción de *Generar grafo*. Automáticamente, la aplicación le devolverá un fichero .png con el grafo indicado.

En caso de que el sistema no disponga de la aplicación GraphViz para la generación de la imagen png, se le devolverá un fichero en formato .dot, a partir del cual, podrá generar la imagen haciendo uso de la aplicación graphviz. Para generar la imagen desde la interfaz gráfica de la aplicación graphviz, o desde una terminal de escritorio, donde tendrá que ejecutar el siguiente comando:

```
$> dot grafo.dot -Tpng -o nombre_grafo.png
```

Capítulo 9

Conclusiones

En este último capítulo se detallan las lecciones aprendidas tras el desarrollo del presente proyecto y se identifican las posibles oportunidades de mejora sobre el software desarrollado.

9.1. Objetivos

A continuación, se recoge una valoración de los objetivos planteados inicialmente en la sección 1.3 que debería de recoger el proyecto y los objetivos que se han alcanzado una vez finalizado el proyecto:

- Se han implementado una serie de modelos de Django los cuales se encargan de almacenar tanto la información de los distintos namespaces cargados en la aplicación junto con las entidades y propiedades que componen a estos, como la información acerca de los diferentes modelos que componen a las aplicaciones del proyecto Django junto con los fields de los mismos. De esta forma se satisface el primer subobjetivo planteado en el objetivo **OBJ-001**.
- Se ha desarrollado un script y agregado este al manage.py de Django, que se encarga de realizar la resolución de los distintos modelos que componen al proyecto, así como los fields (atributos y relaciones) de cada uno de estos. Con esto se satisface el segundo subobjetivo planteado en el objetivo **OBJ-001**.
- Además, el procedimiento anterior, también permite actualizar la especificación de los modelos y sus fields una vez que estos ya han sido cargados, cuando se realicen cambios en los mismos. Como por ejemplo, que se agregue o elimine algún modelo o field. De esta forma, se amplía la funcionalidad descrita por el objetivo **OBJ-001**.
- Se ha desarrollado un apartado donde el usuario puede añadir nuevos namespaces u ontologías a la aplicación. Además, dicho apartado también permite eliminar los namespaces incluidos, y actualizar la especificación de los mismos. De esta forma, se satisface el primer subobjetivo del objetivo **OBJ-002**.

- Se ha desarrollado otro apartado donde se permite al usuario realizar la correspondencia entre los distintos modelos y fields de nuestro proyecto, con los diferentes vocabularios que se han cargado en la aplicación. De esta forma se satisface los subobjetivos segundo y tercero del objetivo **OBJ-002**.

- También se ha desarrollado un apartado donde el usuario puede realizar la configuración de la visibilidad de cada uno de los modelos y fields que existen en el proyecto Django, de forma que no se permita publicar información no deseada. Con esto se satisface el cuarto subobjetivo del objetivo **OBJ-002**.

- Se han creado además, dos nuevos apartados donde se puede generar un fichero de configuración para utilizar con la plataforma D2Rq y también permite descargar un gráfico con la configuración que se ha realizado sobre los modelos de nuestro proyecto. Con esto se mejora la especificación del objetivo **OBJ-002**.

- Se han desarrollado herramientas que permiten la publicación de los datos de los modelos del proyecto Django, en multitud de formatos, como pueden ser RDF/XML, RDF/Ntriples, RDF/Turtle, RDFa o Microdata, además de herramientas para la generación de enlaces a los datos de las instancias de los modelos e inserción de información en plantillas Django. De esta forma, se satisface el objetivo **OBJ-003**.

9.2. Lecciones aprendidas

A continuación se detallan las buenas prácticas adquiridas, tanto tecnológicas como procedimentales. Por ello, podemos decir que a nivel tecnológico, se ha adquirido un mayor dominio y conocimiento del framework de desarrollo Django, de su funcionamiento interno y de las diferentes herramientas que este presta al usuario. Además, también se han obtenido nuevos conocimientos en tecnologías como SPARQL, D2Rq, RDF y web semántica. En general, al haber sido realizado todo bajo el lenguaje de programación Python, se ha obtenido un mayor dominio de este lenguaje, se han conocido nuevas librerías disponibles en el *Python Package Index* y características que ofrece el lenguaje, como puede ser la introspección. Además, en el ámbito del desarrollo web, también he tenido que hacer uso de otras tecnologías, ya conocidas anteriormente, pero que me ha ayudado a ampliar mis conocimientos sobre ellas, así como el uso de otros frameworks, para el contenido tanto CSS como JavaScript.

Además de todos los conocimientos tecnológicos adquiridos, también se han obtenido buenas prácticas a la hora de programar y organizar el trabajo. Esto se ha conseguido siguiendo la guía de estilos PEP8 y la herramienta PyLint a la hora de realizar la implementación del proyecto, lo cual es importante, ya que asegura que el código será mantenible y comprensible por cualquier usuario que esté habituado a trabajar siguiendo estas normas de estilo.

También se han puesto en práctica los conocimientos adquiridos tanto en mis estudios de Ingeniería Técnica en Informática de Sistemas como en Ingeniería Informática, ya que estos me han sido necesarios para la elaboración del proyecto, tanto en el apartado de análisis y diseño de requisitos, prototipado, análisis de riesgos, como en el apartado de implementación, haciendo uso de las buenas prácticas adquiridas, como por ejemplo el uso de patrones de diseño.

Por último, a nivel organizativo, se ha seguido una guía metodológica haciendo uso de un lenguaje de modelado de procesos, donde se describen y priorizan cada una de las etapas del proyecto. Se ha realizado un análisis y diseño del proyecto, así como una planificación de riesgos y temporal del proyecto. En dicho estudio temporal del proyecto, el cual se puede ver el resultado en el diagrama de Gantt del apartado de análisis (Figura 2.1), se le asignaba al proyecto una duración estimada de 12 meses, comenzado el desarrollo del mismo en Octubre de 2012, teniendo en cuenta que la dedicación al mismo no sería total y el resto de riesgos que pudiesen surgir. De esta forma, se ha aprendido la importancia de la gestión de los proyectos, de la estimación temporal y del esfuerzo de los mismos y de la realización de un análisis y diseño del mismo, de tal forma que se verá reflejado en una mejor administración de los recursos y cumplimiento de plazos.

9.3. Trabajo futuro

Una vez concluido el proyecto y obtenida la primera versión del software EasyData/Django, a raíz de la experiencia obtenida a lo largo del desarrollo del mismo, se han podido observar una serie de mejoras o ampliaciones, en cuya dirección podría ir orientado el trabajo futuro para la aplicación. Estas ampliaciones aportarían a la aplicación un mayor valor para el usuario, aumentando las funcionalidades de la misma.

A continuación se muestra una lista de posibles mejoras en las que podría ir encaminado el trabajo futuro:

- Posibilidad de poder realizar más de una configuración simultánea de los modelos de la aplicación, con los namespaces cargados en la misma. De este modo se ofrecería al usuario mayor versatilidad a la hora de publicar sus datos en la web.
- Incluir un endpoint de SPARQL propio, que funcione desde la misma aplicación de Django, que sustituya al fichero D2Rq que se genera actualmente, quedando todo integrado en la misma aplicación. De igual forma, si se desarrollase este módulo, se podría considerar la posibilidad de que no sólo pudiesen realizarse consultas, sino también inserciones y modificaciones de los datos.
- Ampliar los formatos de exportación de datos respecto de los que pueden usarse actualmente (RDF, RDFa y Microdata), usando otros formatos existentes como por ejemplo JSON-LD además de ampliar los existentes.
- Ampliar la disponibilidad de la aplicación EasyData (actualmente existente para Ruby on Rails y Django) a otros frameworks de desarrollo web que sean ampliamente utilizados actualmente.

Si bien, todas las posibles mejoras no requieren un mismo esfuerzo, ya que mejoras como las de la creación de un módulo propio para la realización de consultas haciendo uso del lenguaje SPARQL es bastante ambicioso y complejo, el cual requiere de un gran trabajo, tanto de investigación como de desarrollo, ya que habría que estudiar primeramente como funciona el ORM

de Django, para posteriormente adaptar las consultas SPARQL a este. Además, si se tratase la posibilidad de realizar inserciones y modificaciones en la base de datos, habría que gestionar de alguna forma los permisos de los usuarios para realizar las mismas.

Por otro lado, mejoras como la exportación de la aplicación a otros frameworks de desarrollo web, habría que estudiar entre las posibilidades de realizar una nueva aplicación para el framework en concreto, o la posibilidad de adaptar la existente a cualquier framework. O al menos agrupar estos por lenguajes de programación, como por ejemplo, en el caso de Python, existen varios frameworks de desarrollo web además de Django, como puede ser Web2Py, Zope, etc... por lo que también se podría estudiar la posibilidad de desarrollar un paquete de Python que sirviese para la mayoría de los frameworks web de Python.

Apéndice A

Tablas del apartado de análisis

En este apartado se van a plasmar cada una de las tablas que se han desarrollado para el apartado de análisis de requisitos del proyecto.

A.1. Tablas de requisitos funcionales

A continuación se muestran cada una de las tablas donde se indican cada uno de los requisitos de funcionales que debe de implementar el proyecto.

A.1.1. Carga de modelos

UC-001	**Carga de modelos**
Autor	José Manuel Llerena Carmona
Descripción	Carga de modelos y fields del proyecto Django donde se va a usar la aplicación.
Actores	Administrador del sistema
Precondición	Se ha configurado la aplicación en el proyecto Django, la aplicación posee una base de datos y se han sincronizado las modelos de la aplicación con la base de datos de la aplicación para que se creen las tablas necesarias.
Postcondición	Se han cargado en las tablas de la base de datos correspondiente a la aplicación los modelos y fields de estos.

Continúa en la siguiente página

Continuación de la tabla

UC-001	**Carga de modelos**
Secuencia normal	1. El administrador accede a través de la línea de comandos de la terminal al path donde se encuentra el proyecto Django. 2. El administrador ejecuta el script que se encarga de cargar los modelos y fields existentes en la aplicación Django. 3. El sistema indica al administrador que los datos se han cargado correctamente.
Excepciones	**3.1** Existen modelos ya cargados en la base de datos, por lo que el sistema actualiza dichas tablas y añade las que no existan. **3.2** La aplicación no tiene configurada una base de datos o sus modelos no están configurados correctamente. La aplicación aborta el procedimiento de carga de modelos y muestra mensaje de error.

Tabla A.1: Caso de Uso - 001 - Carga de modelos

A.1.2. Carga de espacios de nombres

UC-002	**Carga de espacios de nombres**
Autor	José Manuel Llerena Carmona
Descripción	Se debe de realizar una serie de procedimientos los cuales permitan a la aplicación cargar la estructura de los diferentes namespaces existentes, además de permitir actualizar y eliminar estos.
Actores	Administrador del sistema
Precondición	Se ha configurado la aplicación en el proyecto Django, la aplicación posee una base de datos y se han sincronizado las modelos de la aplicación con la base de datos de la aplicación para que se creen las tablas necesarias.
Postcondición	Se han cargado en las tablas de la base de datos correspondientes a la aplicación las distintas entidades que componen al namespaces y los atributos de cada una de estas entidades.

Continúa en la siguiente página

Continuación de la tabla

UC-002	**Carga de espacio de nombres**
Secuencia normal	1. El administrador sistema accede al listado de namespaces disponibles. 2. El sistema muestra el listado de namespaces existentes en la aplicación. 3. El administrador del sistema selecciona crear un nuevo namespace. 4. El sistema muestra al administrador un formulario para crear un nuevo namespace. 5. El administrador del sistema indica los datos del nuevo namespace, así como el fichero o la URL donde se encuentra la especificación de dicho namespace. 6. El sistema carga la especificación del namespace en la base de datos, y muestra por pantalla el resultado de la carga.
Excepciones	***.1** El administrador del sistema puede cancelar en cualquier momento el procedimiento de carga/actualización de los namespaces. ***.2** El usuario no tiene permisos de superusuario. **3.1** El usuario indica editar un namespace, para actualizar su nombre o actualizar la especificación mediante un fichero o URL. **6.1** La URL o fichero especificado por el usuario no es correcta, y no se encuentra en ella una estructura de namespaces, o dichos datos no son correctos. **6.2** Los nombres suministrados para el namespace ya están siendo usados. El sistema vuelve a solicitar un nombre correcto.

Tabla A.2: Caso de Uso - 002 - Carga de espacios de nombres

A.1.3. Mapeo de modelos

UC-003	**Mapeo de modelos**
Autor	José Manuel Llerena Carmona

Continúa en la siguiente página

Continuación de la tabla

UC-003	**Mapeo de modelos**
Descripción	Este caso de uso describe el procedimiento mediante el cual el administrador del sistema, describe en la aplicación la relación de cada uno de los modelos del proyecto Django, con las entidades de los distintos namespaces que existen en la aplicación.
Actores	Administrador del sistema
Precondición	Se ha cargado en la aplicación los modelos y atributos existentes en el proyecto, y además existe algún namespace cargado en la aplicación, de forma que se dispongan entidades y propiedades con el que mapear los modelos.
Postcondición	Uno o varios de los modelos del proyecto Django de los cuales se quieren publicar sus datos, se encuentran relacionados con una de las entidades de los namespaces existentes.
Secuencia normal	1. El administrador del sistema accede al listado de modelos del sistema disponibles para mapear. 2. El sistema muestra el listado de modelos y dos selectores por cada uno de los modelos, donde se pueda elegir el namespace con el que se desea mapea y la entidad concreta con la que se desea realizar el mapeo. 3. El administrador del sistema introduce la configuración de aquello modelos que crea oportunos y salva los cambios realizados. 4. El sistema muestra un mensaje con el resultado del mapeo.
Excepciones	***.1** El administrador del sistema puede cancelar en cualquier momento la operación de mapeo de los modelos. ***.2** El usuario no tiene permisos de superusuario.

Tabla A.3: Caso de Uso - 003 - Mapeo de modelos

A.1.4. Mapeo de los atributos

UC-004	**Mapeo de los atributos**
Autor	José Manuel Llerena Carmona
Descripción	Este caso de uso describe el procedimiento mediante el cual el administrador del sistema, describe en la aplicación la relación de cada uno de los atributos de los modelos del proyecto Django, con las propiedades de las entidades de los distintos namespaces que existen en la aplicación.

Continúa en la siguiente página

Continuación de la tabla

UC-004	**Mapeo de los atributos**
Actores	Administrador del sistema
Precondición	Se ha cargado en la aplicación los modelos y atributos existentes en el proyecto, además existe algún namespace cargado en la aplicación, con el que mapear la relación con los atributos y además, se ha realizado la correspondencia del modelo con alguna entidad del namespace que deseamos configurar.
Postcondición	Alguno de los atributos del modelo del proyecto Django que se ha configurado, ha sido relacionado con alguna de las propiedades de la entidad con la que está relacionada el modelo.
Secuencia normal	1. El administrador del sistema accede al listado de modelos del sistema disponibles para mapear. 2. El sistema muestra el listado de modelos. 3. El administrador del sistema elige configurar los atributos de un determinado modelo que ya se encuentra mapeado con alguna entidad. 4. El sistema una pantalla con todos los atributos del modelo y dos selectores con los posibles namespaces y las propiedades disponibles del namespace seleccionado disponibles para mapear dicho atributo. 5. El administrador del sistema, introduce la configuración de aquellos atributos que desee y salva los cambios. 6. El sistema muestra un mensaje con el resultado del mapeo.
Excepciones	***.1** El administrador del sistema puede cancelar en cualquier momento la operación de mapeo de los atributos. ***.2** El usuario no tiene permisos de superusuario. **3** El modelo no se encuentra mapeado en el sistema, por lo que previamente deberá de realizar el mapeo del mismo antes de configurar el mapeo de sus atributos. **Include:** UC-003.

Tabla A.4: Caso de Uso - 004 - Mapeo de los atributos

A.1.5. Establecer visibilidad de modelos

UC-005	**Establecer visibilidad de modelos**
Autor	José Manuel Llerena Carmona
Descripción	Este caso de uso describe el procedimiento que debe de seguir el administrador del sistema, para especificar aquellos modelos del proyecto Django de los que podrán publicarse los datos.
Actores	Administrador del sistema
Precondición	Se ha configurado la aplicación en el proyecto Django, la aplicación posee una base de datos y se han sincronizado las modelos de la aplicación con la base de datos de la aplicación para que se creen las tablas necesarias donde se almacenará la estructura de los namespaces.
Postcondición	Se han marcado aquellos modelos que queremos que se puedan mostrar sus datos y los que no queremos que se puedan mostrar como visibles y no visibles, respectivamente.
Secuencia normal	1. El administrador del sistema accede al apartado de configuración de visibilidad de modelos. 2. El sistema muestra una lista con todos los modelos y fields de los mismos existentes en el proyecto Django. 3. El administrador del sistema marca como visibles o no visibles, los modelos y fields que desea que se puedan ver o no. 4. El administrador del sistema una vez ha terminado, salva los cambios en el sistema. 5. El sistema indica al administrador que los datos se han cargado correctamente.
Excepciones	***.1** El administrador del sistema puede cancelar en cualquier momento la operación de configuración de visibilidad de los modelos. ***.2** El usuario no tiene permisos de superusuario.

Tabla A.5: Caso de Uso - 005 - Establecer visibilidad de modelos

A.1.6. Establecer visibilidad de los atributos

UC-006	**Establecer visibilidad de los atributos**
Autor	José Manuel Llerena Carmona
Descripción	Este caso de uso describe el procedimiento que debe de seguir el administrador del sistema, para especificar aquellos atributos de los modelos del proyecto Django de los que podrán publicarse los datos.

Continúa en la siguiente página

Continuación de la tabla

UC-006	**Establecer visibilidad de los atributos**
Actores	Administrador del sistema
Precondición	Se ha configurado la aplicación en el proyecto Django, la aplicación posee una base de datos y se han sincronizado las modelos de la aplicación con la base de datos de la aplicación para que se creen las tablas necesarias donde se almacenará la estructura de los namespaces.
Postcondición	Se han marcado aquellos atributos de los modelos que queremos que se puedan mostrar sus datos y los que no queremos que se puedan mostrar como visibles y no visibles, respectivamente.
Secuencia normal	1. El administrador del sistema accede al apartado de configuración de visibilidad de modelos. 2. El sistema muestra una lista con todos los modelos existentes en el proyecto Django. 3. El administrador del sistema selecciona en configuración de los atributos de un determinado modelo. 4. El sistema muestra al usuario la lista de atributos del modelo seleccionado. 5. El administrador del sistema marca como visibles o no visibles, los atributos de los modelos que desea que se puedan ver o no, respectivamente. 6. El administrador del sistema una vez ha terminado, salva los cambios en el sistema. 7. El sistema indica al administrador que los datos se han cargado correctamente.
Excepciones	***.1** El administrador del sistema puede cancelar en cualquier momento la operación de configuración de visibilidad de los modelos. ***.2** El usuario no tiene permisos de superusuario.

Tabla A.6: Caso de Uso - 006 - Establecer visibilidad atributos

A.1.7. Publicación de los datos

UC-007	**Consulta de datos**
Autor	José Manuel Llerena Carmona

Continúa en la siguiente página

Continuación de la tabla

UC-007	**Consulta de datos**
Descripción	Muestra los datos por pantalla usando alguno de los estándares existente para la publicación de datos en internet.
Actores	Usuario externo
Precondición	Se ha configurado la aplicación en el proyecto Django, la aplicación posee una base de datos y se han sincronizado las modelos de la aplicación con la base de datos de la aplicación para que se creen las tablas necesarias. Además se ha realizado el mapeo el modelo junto con sus atributos que se desea visualizar y también se ha configurado su visibilidad.
Postcondición	Se ha mostrado por pantalla usando el formato deseado los datos referente a la instancia(s) del modelo seleccionado.
Secuencia normal	1. El usuario externo accede a la URI correspondiente a la dirección dinámica existente para los datos que desea consultar. 2. El sistema muestra al usuario externo los datos del modelo o los modelos , haciendo uso del estándar y el namespace solicitado.
Excepciones	***** El usuario externo puede cancelar la operación en cualquier momento. **2.1** El sistema muestra los datos de la instancia concreta en el formato solicitado. **2.2** El sistema muestra los datos de la entidad concreta en el formato solicitado.

Tabla A.7: Caso de Uso - 007 - Consulta de datos

A.1.8. Inicio de sesión

UC-008	**Inicio de sesión**
Autor	José Manuel Llerena Carmona
Descripción	Realiza el login del usuario dentro de la aplicación web.
Actores	Usuario externo
Precondición	El usuario externo no se encuentra logueado en la aplicación.
Postcondición	Se ha iniciado sesión en la aplicación y tiene permisos de superusuario.

Continúa en la siguiente página

Continuación de la tabla	
UC-008	**Inicio de sesión**
Secuencia normal	1. El usuario externo accede a la dirección de inicio de sesión. 2. El sistema muestra al usuario externo un formulario donde se le solicita el usuario y contraseña. 3. El usuario introduce las credenciales. 4. El usuario y contraseña son válidos y el sistema redirige al usuario a la aplicación.
Excepciones	* El usuario externo puede cancelar la operación en cualquier momento. **4.1** El usuario o contraseña son incorrectos. Vuelve a solicitar las credenciales al usuario. **4.2** El usuario no posee credenciales de superusuario.

Tabla A.8: Caso de Uso - 008 - Inicio de sesión

A.1.9. Consulta con SPARQL

UC-009	**Consulta con SPARQL**
Autor	José Manuel Llerena Carmona
Descripción	Debe de permitirse al usuario realizar consultas sobre los datos utilizando el lenguaje SPARQL.
Actores	Usuario externo
Precondición	Los modelos y fields se han cargado en la aplicación. Se han cargado también algún namespace. Se ha realizado el mapeo de alguno de los modelos del proyecto.
Postcondición	Se devuelven aquellos datos que se corresponde con la consulta SPARQL indicada.
Secuencia normal	1. Un usuario externo accede al apartado de consultas SPARQL e introduce una consulta. 2. El sistema devuelve al usuario el resultado de ejecutar la consulta sobre los datos almacenados en la base de datos y que cumplan las relaciones indicadas en la consulta.
Excepciones	No existen excepciones para este caso de uso.
Continúa en la siguiente página	

Continuación de la tabla	
UC-009	**Consulta con SPARQL**

Tabla A.9: Caso de Uso - 009 - Generación fichero D2Rq

A.2. Tablas de requisitos de información

A continuación se muestran cada una de las tablas donde se indican cada uno de los requisitos de información que posee el proyecto.

IRQ-001	**Información de los namespaces**
Autor	José Manuel Llerena Carmona
Descripción	Se debe almacenar la información acerca de los distintos namespaces que almacena la aplicación, de tal forma que se tenga la información acerca del namespace, de las entidades que componen a este, y de las propiedades que componen a la entidad.
Dependencias	De este requisito de información depende que se pueda almacenar la información relativa a los distintos namespaces, realizar el posterior mapeo de los datos y finalmente la publicación de los mismos, usando dichos namespaces.

Continúa en la siguiente página

Continuación de la tabla

IRQ-001	**Información de los namespaces**
Datos específicos	▪ Datos de la clase *Namespace*: • Nombre: nombre identificativo del namespace. • url: es la dirección donde se encuentra la especificación del namespace. • short_name: nombre corto que se representará al namespace en los ficheros rdf. ▪ Datos de la clase *Entidad*: • nombre: es el nombre identificativo de la entidad. • padres: indica si la entidad hereda de una entidad padre, y en caso de ser cierto, indica cuáles son. • namespace: indica el namespace al que pertenece dicha entidad. • descripción: descripción asociada a la entidad. • etiqueta: etiqueta asociada a la entidad, la cual representa al elemento. ▪ Datos de la clase *Propiedad*: • nombre: es el nombre identificativo de la propiedad. • tipo: en caso de que el tipo de dato no sea simple, indica las entidades a las que hace referencia la propiedad. • entidades: son las entidades a las que pertenece dicha propiedad. • descripción: descripción asociada a la propiedad. • etiqueta: etiqueta asociada a la propiedad, la cual representa a dicha característica. • simple: indica si la propiedad es un tipo de dato simple o si por el contrario hace referencia a otra entidad. • namespace: indica el namespace al que pertenece la propiedad.

Tabla A.10: IRQ - 001 - Información de los namespaces

IRQ-002	**Información de los modelos**
Autor	José Manuel Llerena Carmona

Continúa en la siguiente página

Continuación de la tabla

IRQ-002	**Información de los modelos**
Descripción	Se debe almacenar la información acerca de los distintos modelos que componen al proyecto Django, de tal forma que se tenga la información acerca del modelo, y de los fields que componen al modelo.
Dependencias	De este requisito de información, depende que se puedan almacenar la estructura de los modelos del proyecto Django donde se ha instalado la aplicación, su posterior mapeo con los distintos namespaces y su publicación final.

Continúa en la siguiente página

Continuación de la tabla

IRQ-002	**Información de los modelos**
Datos específicos	▪ Datos de la clase *Modelo*: • Nombre: nombre identificativo del Modelo dentro del proyecto Django. • aplicacion: es la aplicación dentro del proyecto Django al que pertenece dicho modelo. Se utiliza para poder localizarlo más adelante, cuando se necesite captar sus datos. • Entidad: se trata de una relación muchos-muchos con cada una de las entidades con las que se ha configurado su mapeo. • visibilidad: indica si este modelo es visible o no al exterior. ▪ Datos de la clase *Field*: • nombre: es el nombre identificativo del field dentro del modelo. • visibilidad: indica si este field es visible o no al exterior. • propiedad: es una relación muchos-muchos donde la cual almacena el mapeo de dicho field con las distintas propiedades. • modelo: indica el modelo del proyecto Django al que pertenece dicho field. Esta estará diferenciada a su ver, mediante herencia en dos clases diferentes: • **Atributo:** se encargará de representar a las relaciones existentes entre los distintos modelos. Posee los siguientes atributos: ◦ tipo de field: almacena el nombre del tipo de atributo al que representa. • **Relacion:** se encargará de representar a las relaciones existentes entre los distintos modelos. Posee los siguientes atributos: ◦ Tipo de relación: indica si la relación es 1:1, 1:N o N:M. ◦ Modelo relacionado: indica el modelo al que hace referencia la relación. ◦ inversa: indica la relación inversa, es decir, la que posee la navegabilidad en el sentido contrario.

Tabla A.11: IRQ - 002 - Información de los modelos

Bibliografía

[Ber13] Freie Universität Berlin, *Documentación oficial D2Rq*, http://d2rq.org/, 2013.

[DuC11] Bob DuCharme, *Learning sparql*, O'Reilly, 2011.

[FOA10] Project FOAF, *FOAF*, http://www.foaf-project.org/, 2010.

[Fou13a] Django Foundation, *Documentación oficial de Django*, https://www.djangoproject.com/, 2013.

[Fou13b] Python Software Foundation, *Documentación oficial de Python*, http://www.python.org, 2013.

[GMY12] Google-Microsoft-Yahoo!, *Schema*, http://schema.org/, 2012.

[HB11] Tom Heath and Christian Bizer, *LinkedDataBook*, http://linkeddatabook.com/editions/1.0/, 2011.

[Hep12] Martin Hepp, *Good Relations*, http://www.heppnetz.de/projects/goodrelations/, 2012.

[Inf13] InfoJobs, *Salarios promedios InfoJobs*, http://plandecarrera.infojobs.net/, 2013.

[jQu13] jQuery, *Documentación oficial jQuery*, http://api.jquery.com/, 2013.

[MAG12] MAGERITV3, *Ministerio de Hacienda y Administraciones Públicas de España*, http://administracionelectronica.gob.es/?_nfpb=true&_pageLabel=PAE_PG_CTT_General&langPae=es&iniciativa=magerit, 2012,
Método formal para investigar los riesgos que soportan los Sistemas de Información y para recomendar las medidas apropiadas que deberían adoptarse para controlarlos.

[Mic13] W3C Microdata, *Microdata*, http://www.w3.org/TR/microdata/, 2013.

[OWL04] W3C OWL, *OWL*, http://www.w3.org/TR/owl-ref/, 2004.

[RDF04a] W3C RDF, *RDF*, http://www.w3.org/TR/rdf-primer/, 2004.

[RDF04b] W3C RDFS, *RDFS*, http://www.w3.org/TR/rdf-schema/, 2004.

[RDF13] W3C RDFa, *RDFa*, http://www.w3.org/TR/rdfa-syntax/, 2013.

[RW12] Guido Van Rossum and Barry Warsaw, *Style Guide for Python Code*, http://www.python.org/dev/peps/pep-0008/, 2012.

[Twi13] Twitter, *Bootstrap V3*, http://getbootstrap.com/, 2013.

Información sobre Licencia

El software del presente proyecto se ha desarrollado bajo la licencia de software libre GNU GPL (GNU General Public License). De esta forma se garantiza que los usuarios finales (personas u organizaciones) tengan libertad total de usar, compartir y modificar el presente software.

Además, la presente documentación se encuentra bajo la licencia GFDL (GNU Free Documentation License), que se detalla a continuación:

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.