Universidad Autónoma de Madrid

Escuela Politécnica Superior

Proyecto fin de grado

UN GENERADOR AUTOMÁTICO DE SITIOS WEB

Grado en Ingeniería Informática

Juan Martínez Palazón

Tutor: Juan de Lara Jaramillo

7 de julio de 2014

UN GENERADOR AUTOMÁTICO DE SITIOS WEB

AUTOR: Juan Martínez Palazón
TUTOR: Juan de Lara Jaramillo

Dpto. de Ingeniera Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
7 de julio de 2014

Resumen

Internet, con la llegada de la web 2.0, ha dejado de ser un medio de mera consulta para ser un lugar donde el usuario genera el contenido y lo hace además de manera constante. En este nuevo paradigma, la importancia de una marca viene intrínsecamente ligada a su presencia en la red y esto denota la importancia de tener su propio sitio web.

A día de hoy, infinidad de metodologías permiten la producción de páginas web, cada una de ellas plantea diferentes soluciones al problema mediante técnicas diversas, pero en muchos casos realizan tareas similares. Esta falta de homogeneidad provoca en los desarrolladores la necesidad de conocer múltiples tecnologías para distintos contextos.

La herramienta para la generación automatizada de sitios web presentada en este proyecto se ha basado en el desarrollo guiado por modelos y el uso de generadores de código, que permiten un alto nivel de abstracción y hacen que los detalles técnicos asociados a la utilización de diferentes tecnologías sean transparentes al diseñador.

En este caso, la solución que se presenta se ha especializado en la producción de sitios web destinados a grupos de investigación.

Para ello, se han implementado numerosos componentes web que encapsulan funcionalidades como la conexión a bases de datos externas, la comunicación con APIs o la representación de gráficos.

Como caso de estudio, se ha validado la solución diseñada con la creación del sitio Web del Grupo de Aprendizaje Automático de la Universidad Autónoma de Madrid.

Palabras clave

Desarrollo Web, MDE, MetaDepth, deep, meta-modelado, multi-nivel, generación de código, dominio específico.

Abstract

Web 2.0 arrival few years ago has changed completely the way we use the Internet. It is no longer a system just to search for information since it has become a place where the users constantly generate new contents. With this new paradigm, the importance of a brand is directly related to its presence on the Internet and therefore, having its own Web site is an essential aspect.

Nowadays, there are lots of methodologies that allow us to create web pages and they each propose different solutions using diverse techniques, but in most cases they carry out similar tasks. This lack of homogeneity makes the developers have to know a big variety of technologies to be used in different contexts.

The tool to automatically generate websites presented in this project is based on model-driven development and code generators. These techniques entail a high abstraction layer making the technical details transparent to the designer.

In this case, the solution proposed has been focused on producing websites for researcher groups. To this end, a lot of different Web components able to connect to external data bases, to communicate to API's, and to represent graphs have been implemented.

As study case, the tool has been validated creating the Web Site for the Machine Learning Group of the Universidad Autónoma de Madrid.

Keywords

Web Development, MDE, MetaDepth, deep, meta-modelling, multi-level, code generation, specific domain.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En 2013 se contabilizaron más de 730 millones de páginas Web en Internet y esta cifra crece en decenas de millones cada año. Parece indiscutible la necesidad de que cada empresa u organización, independientemente del tamaño que tenga, posea su propio sitio Web en el que publicitarse, vender sus productos, proporcionar servicios, o realizar trámites administrativos. La variedad de estas empresas, la aparición de nuevas funcionalidades para el usuario y las múltiples y diferentes fuentes de datos (BBDD, APIs, etc) ha añadido complejidad a un problema que ya no está acotado y que exige al desarrollador el estudio de todas estas novedades.

1.1. Motivación

La creación de un sitio Web puede no ser una tarea sencilla. Si bien es cierto que en los últimos años se han popularizado distintas plataformas capaces de construir dinámicamente sitios en línea de manera personalizada y sencilla, estas soluciones no consiguen satisfacer a todos los usuarios dado que no permiten el diseño más allá de las plantillas predefinidas por la empresa que proporciona el servicio.

Si fuese requerido, por ejemplo, el acceso a información almacenada en bases de datos externas, se necesitara la visualización de gráficos complejos en tiempo real o la interacción con servicios de terceros mediante APIs no contempladas en las plantillas se hace evidente la necesidad de una herramienta diferente que permita eliminar este tipo de restricciones.

Además, la distinta naturaleza de los proyectos a los que vaya dirigido el sitio Web puede requerir diferentes especializaciones de los elementos que conforman las páginas. Puede ocurrir que para satisfacer las necesidades de un contexto particular estas especializaciones supongan el inicio de un proceso de desarrollo nuevo. Esta situación puede evitarse gracias a la reutilización de modelos generales y el refinamiento de los mismos cada vez más aplicados a plataformas específicas.

Por otro lado, la inclusión de otras funcionalidades como; conexiones a bases de datos

empleando diferentes gestores como Oracle, DB2, MySQL, la interacción con APIs de diferentes servicios en línea o el uso de distintos estándares como SOAP o WSDL con el objetivo de intercambiar datos entre aplicaciones, hacen necesario el conocimiento de tecnologías muy desiguales. De manera añadida, el uso de múltiples lenguajes, HTML, PHP, JavaScript, etc, incrementan las dificultades descritas.

1.2. Objetivos

En este punto, el trabajo que se presenta propone una solución a esta problemática. El principal objetivo de este proyecto es el diseño de un lenguaje genérico para crear sitios web y que éste sea adaptable a distintos dominios, como por ejemplo, grupos de investigación.

Para ello se estima necesario que el generador de páginas automatice prácticamente todo el proceso, eximiendo a los diseñadores de las tareas repetitivas dentro de la implementación.

Si bien la práctica totalidad del código del sitio web es generado dinámicamente se espera que, si fuera necesaria la intervención de desarrolladores en el proceso de creación de las páginas, sea únicamente para refinar ciertos detalles que no hayan podido ser modelados en alto nivel.

Por otro lado, como se ha indicado en la sección anterior, el uso de múltiples técnicas y lenguajes dificulta la tarea de desarrollo de un sitio Web y es por eso que la herramienta presentada debe incorporar un conglomerado de modelos que permita hacer transparente al programador, los aspectos relacionados con la tecnología de implementación.

De manera añadida, la utilización de un modelado multinivel [4] con un número no prefijado de niveles de profundidad hará factible un diseño que se asemeje lo máximo posible a la concepción de la lógica de negocio que tiene el experto en el ámbito para el que se desarrolla la aplicación Web y permite, además, la reutilización de los metamodelos aplicados a un contexto específico de manera que puedan rediseñarse nuevos sitios Web de temática similar con un proceso de desarrollo simple y rápido.

1.3. Estructura del documento

Para la comprensión del trabajo realizado, se presenta, en primer lugar, el estudio previo llevado a cabo sobre las distintas tecnologías y metodologías de programación en las que se ha basado el proyecto. En la sección 3 se expone el estado del arte analizando trabajos con objetivos similares a este y el punto en el que se encuentran, para conocer el contexto en el que nace el generador de sitios Web presentado en este documento. En la sección 4 se explica el funcionamiento del generador Web, su arquitectura, el conjunto

de modelos que se han diseñado y los generadores de código empleados. En la siguiente sección se describe el proceso de pruebas sobre componentes individuales del proyecto y se muestra de forma detallada la implementación del sitio Web que, a modo de validación, se ha creado para el Grupo de Aprendizaje Automático de la EPS-UAM. Por último se añade un análisis general de proyecto en el apartado de conclusiones, seguido de la presentación del trabajo propuesto para el futuro.

Capítulo 2

Estudio previo

El coste del desarrollo de software puede causar grandes impactos en la economía de la compañía que lo produce y es por ello que siempre se ha intentado incrementar la productividad en el proceso.

A finales de los noventa se había instaurado la filosofía de programación orientada a objetos. El estándar Unified Modeling Language (UML) planteaba una notación basada en round-trip (sincronización de artefactos software, relacionados entre sí, para evitar inconsistencias) que permitía un paso ágil de los modelos UML al código. Esta relación entre modelo y código entra en un proceso de degradación gradual a medida que se va implementando la aplicación, debido a que los desarrolladores cada vez van ajustando más el programa al contexto en el que se utilizará y a las tecnologías sobre las que se construye, es decir, el producto software se está especializando y perdiendo por tanto la abstracción que conseguía el modelo.

Modificar el modelo en tiempo de desarrollo de manera que se mantenga consistente y sincronizado con el código que se está implementando es una tarea laboriosa y que a menudo se entiende como una pérdida de tiempo[6]. Este aspecto es determinante cuando el tiempo y los recursos asignados a un proyecto son escasos y desemboca en muchas ocasiones en el abandono de la tarea de modificación y cuidado de la documentación, descartando uno de los objetivos principales del desarrollo basado en modelos, que es el de poder resumir de manera abstracta y de fácil comprensión la funcionalidad del sistema.

En este contexto es dónde aparece el Desarrollo de Software Dirigido por Modelos, parte fundamental de este proyecto.

2.1. Desarrollo de Software Dirigido por Modelos

El Desarrollo Dirigido por Modelos, más conocido en inglés como Model Driven Development (MDD) se trata de un paradigma de programación en el que los artefactos

principales del desarrollo son los modelos (no los programas) y la transformación de estos modelos.

MDD consigue mejorar la productividad gracias a que los modelos se van refinando, mediante transformaciones, de manera automatizada, desde un nivel de abstracción máximo hasta la transformación última al código fuente. Este aspecto favorece la creación de una aplicación mas robusta ya que los modelos contienen menos detalles accidentales que el código. Además, haciendo uso de esta filosofía es posible integrar con mayor facilidad nuevos desarrollos en un sistema ya existente, así como realizar incorporaciones futuras de terceros.

El Object Management Group (OMG) diseñó la Arquitectura dirigida por modelos (Model Driven Architecture o MDA) que consiste en la estructuración del sistema mediante unas guías que permitan un modelado basado en la Model Driven Engineering (MDE), es decir, crear un estándar bajo el desarollo dirigido por modelos.

OMG basa el MDE en cuatro principios fundamentales:

- A efectos de conseguir una comunicación fluída entre elementos del sistema y un fácil entendimiento del mismo, los modelos deben estar expresados con una notación bien definida, UML y perfiles UML
- Las especificaciones del sistema deben estar definidas en torno a un conjunto de modelos y asociadas a las transformaciones que implentan las relaciones entre los mismos. De esta manera se permite la organización del diseño basada en un framework multicapa.
- Los modelos deben ser diseñados de acuerdo a un conjunto de metamodelos. Esto facilita la integración entre ellos y la automatización mediante el uso de herramientas software.
- Este framework de modelado debe aumentar la aceptación de MDE y fomentar la competencia entre proveedores de herramientas de modelado.

En la Model Driven Architecture se definen los niveles CIM, PIM, PMS y el código, así como las transformaciones entre ellos.

- **Computation Independent Model (CIM):** Es una representación del sistema desde un punto de vista completamente independiente de la computación. Este tipo de modelos permite al experto conocedor del dominio diseñar el sistema que dará solución a sus requerimientos en un lenguaje familiar y sin la necesidad de conocer los aspectos técnicos de la estructura del sistema.
- **Platform Independent Model:** Modelo con alto nivel de abstracción. Ignora la plataforma en la que se va a desarrollar el artefacto software, no tiene en cuenta el sistema operativo o lenguajes de programación.

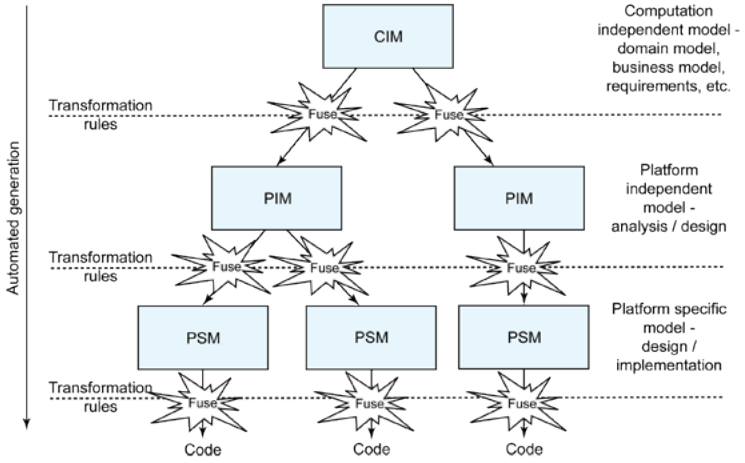

Figura 2.1: Esquema de la Arquitectura MDA.[10]

- **Platform Specific Model:** Un PIM se transforma en uno o más PSMs. Cada PSM se entiende como la aplicación de un PIM en una plataforma determinada y con una tecnología específica.
- **Código:** Una vez que cada PSM está orientado a una plataforma específica se asocia con un código orientado a ese dominio.
- **Transformaciones:** Procesos que realizan la especialización y refinamiento de los modelos hasta llegar al código. Es necesario que estas transformaciones estén estandarizadas con el objetivo de que más tarde sea modificable y escalable. Un ejemplo de estas transformaciones puede observarse en la figura 2.2

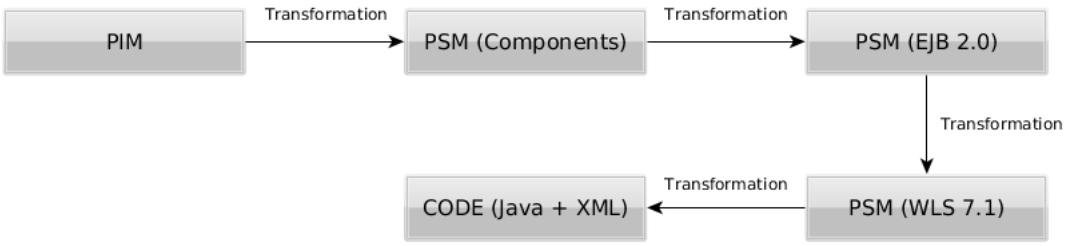

Figura 2.2: Transformaciones de PIM a código.

Actualmente, la gran mayoría de los desarrollos de software emplean modelos durante la fase de diseño. En esta práctica, además del inconveniente de la separación gradual entre la visión abstracta de los modelos y el código, que se mencionó anteriormente, existe el problema de la dinamicidad y aparición de nuevas tecnologías, característica intrínseca de la industria informática que no puede ser obviada por diferentes razones, por ejemplo, la exigencia de nuevas instalaciones por parte de los clientes, la necesidad de resolver determinados problemas que únicamente son posible solventar haciendo uso

de estas nuevas técnicas o incluso la decisión de los proveedores de herramientas de dejar de dar soporte a versiones antiguas de su software.

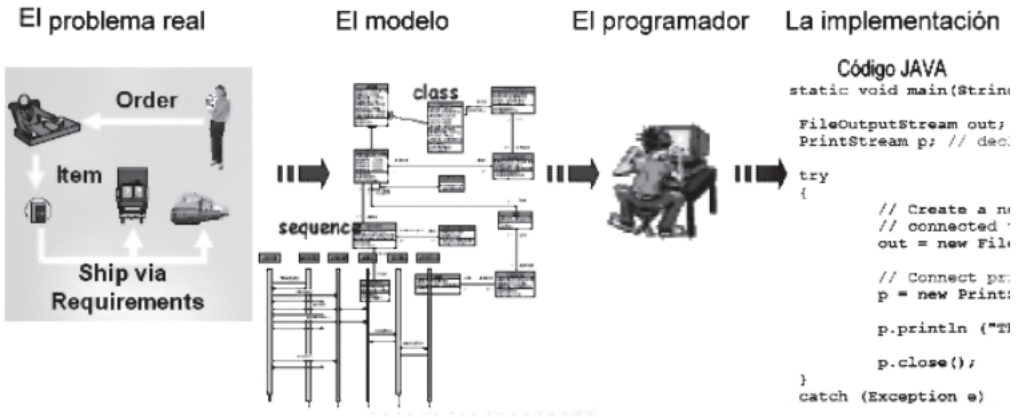

Figura 2.3: Desarrollo de software basado en modelos[2].

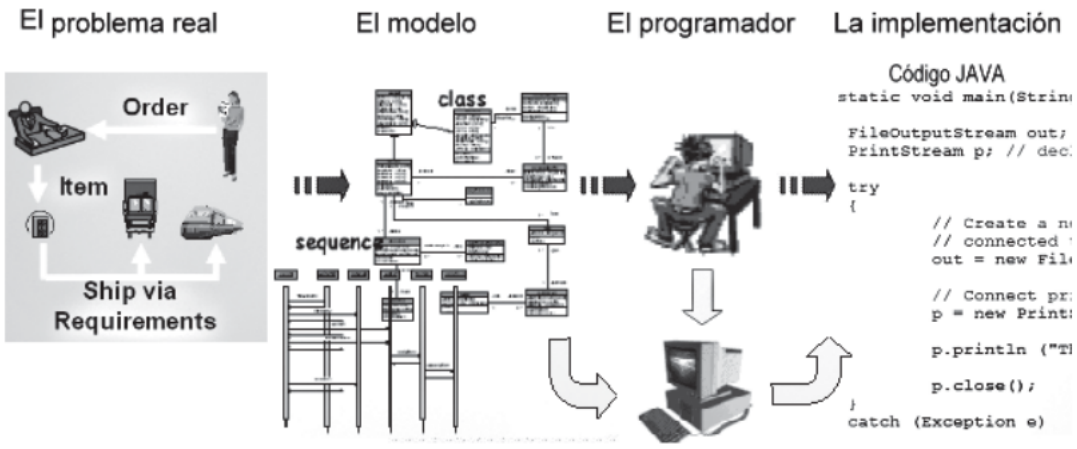

Figura 2.4: Desarrollo de software dirigido por modelos[2].

El desarrollo dirigido por modelos, sin embargo, es capaz de resolver estas cuestiones permitiendo una mayor abstracción en la fase de modelado, disminuyendo la distancia entre el experto conocedor del modelo de negocio y los desarrolladores que implementarán la solución. Con el alto nivel de estandarización y el modelado modularizado se consigue, además, una fácil integración de nuevas tecnologías que puedan aparecer. Los modelos asociados a generadores de código toman por tanto una nueva función en el diseño del sistema y juegan un papel más productivo, permitiendo a los programadores, gracias a la generación semi-automática del código, no tener que intervenir en esa fase y únicamente tener que implementar determinadas características específicas del contexto para el que se está desarrollando.

2.2. Meta-modelado multinivel

Dentro de una metodología fundamentada en el desarrollo guiado por modelos es crucial proporcionar un sistema de creación de meta-modelos apropiado para resolver un problema de un dominio particular y que éste pueda ser transformado automáticamente en soluciones reales para una plataforma seleccionada, lo que hace patente la necesidad de un framework de modelado extensible.

En los procesos de desarrollo de software guiado por modelos el nivel de meta-modelos es, comunmente, diseñado mediante lenguajes de propósito general como; MOF, Ecore o Emfatic, entre otros.

Con estos lenguajes se describen los meta-modelos que después se instancian, es decir, en este paradigma se diferencian dos dimensiones, una que hace las veces de elementos de tipado y la del nivel inferior en el que se encuentran los objetos puros que obligatoriamente deben ser de un tipo definido en su nivel superior y cumpliendo las restricciones que imponga el meta-modelo.

Sin embargo, en la introducción del modelado multi-nivel a la hora de crear esos meta-modelos, el número de niveles de los que se compone el diseño es variable. Esto quiere decir que existirán diferentes profundidades controladas por el concepto de potencia que se explicará con más detalle en la sección siguiente. En el nivel superior se siguen encontrando, al igual que en el enfoque de dos meta-niveles, los modelos que no son instancia de otro superior y que por tanto únicamente son *tipo* de un modelo inferior. En el nivel más bajo se encuentran, por tanto, los modelos completamente instanciados, los objetos. Los elementos que encontramos en los meta-niveles intermedios pueden ser denominados clabjets ,unión de las palabras en inglés class y object, término que describe la naturaleza dual de estos modelos que se entienden como objetos instanciados del nivel superior, pero a su vez tipos de los modelos de niveles inferiores.

En la figura 2.5 extraída del artículo [3] se representa el proceso de definición de modelado de un lenguaje de meta-modelado para dos dominios específicos diferentes, un sistema de gestión educacional para el control de clases (b), prácticas y evaluaciones y otro para la definición de un lenguaje de modelado de procesos de ingeniería de software (d).

El modelo (b) muestra una definición de una versión instanciada del modelo descrito en el nivel superior (a). Los números que siguen al símbolo @ determinan el valor de la potencia en ese nivel. La potencia puede tomar valores enteros positivos o cero y se va decrementando cada vez que se hace efectiva una instanciación, una potencia con valor cero implica la imposibilidad de seguir instanciando el modelo y la obtención, por tanto, de un objeto puro.

Con el objetivo de simplificar las explicaciones se han obviado los modelos que deberían instanciar del meta-modelo *performers* (*students* y *professors*) y los del meta-modelo *artefacts* (*course*, *material* y *grades*).

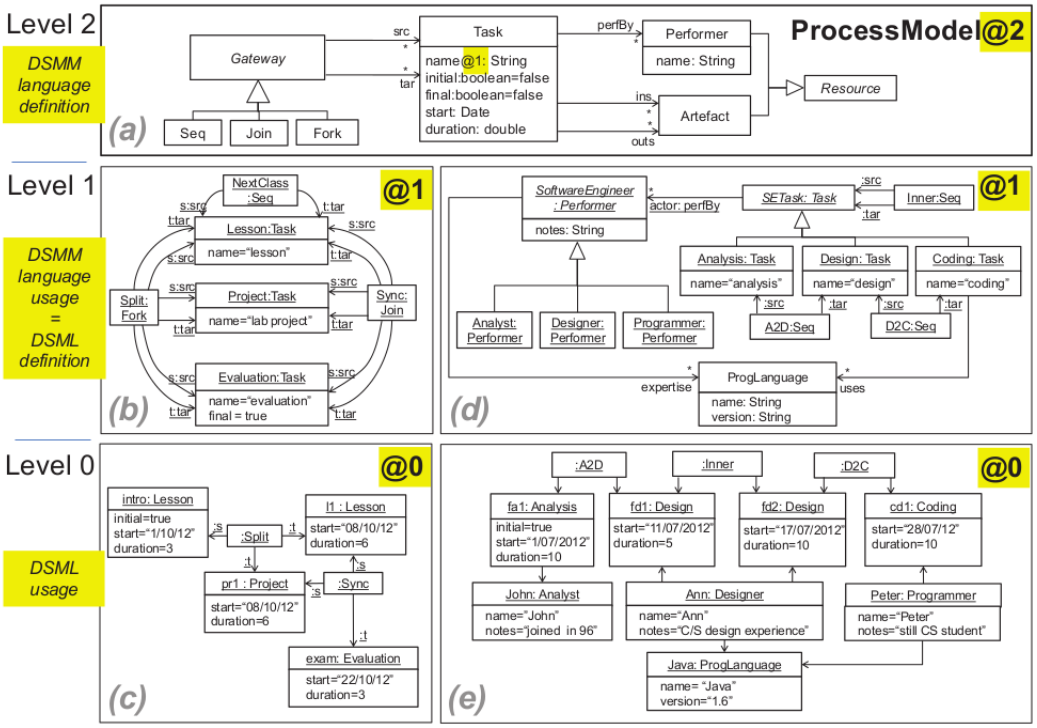

Figura 2.5: Definición del proceso de modelado de un lenguaje DSMM.

El modelo de proceso define un conjunto de posibles tareas como son; *lessons*, *projects* y *evaluations*. Se incluyen además una serie de elementos para gestionar la planificación; *Nextclass* para definir procesos secuenciales y que es una instancia de *Seq*, *Fork*, para separar procesos y *Join* para unirlos. Estos últimos son instancias de *Split* y *Sync* respectivamente.

Como se puede comprobar, esta especialización del meta-modelo respeta la estructura definida en el mismo.

Por otro lado, aunque un lenguaje de dominio específico pueda no tener el potencial de uno de propósito general, el uso de DSML permite la creación de aplicaciones de una manera más natural ya que se establece una relación directa entre la lógica de negocio y los modelos. Sin embargo en un lenguaje como MOF no podríamos usar símbolos que fueran empleados en nuestro ámbito, en este ejemplo estudiado; lecciones, clases, libros, etc.

El mismo DSMM definido en el nivel (a) puede ser empleado, mediante una relación de instanciación, para producir otros modelos que, esta vez, se ajusten a un contexto de definición de un lenguaje de modelado de procesos de ingeniería de software.

Para este ejemplo, representado en el diagrama (d), se han modelado distintos conceptos comunes al contexto, como son los distintos roles que llevan a cabo los actores integrantes del proyecto de ingeniería del software; *Analyst, Programmer o Designer*. Además se describen en el modelo las distintas tareas relevantes, task, en un proceso de

este tipo. Como en el ejemplo anterior, con el objetivo de simplificar la representación, se ha omitido el modelado de los productos finales, que serían una instancia de *Artefacts*.

Ambos contextos tenían en común determinados aspectos que han podido ser abstraídos y condensados en el meta-modelo definido en el nivel (a).

Nótese que además, el proceso de especialización de los modelos, ha conllevado la inclusión de nuevos atributos (por ejemplo, *notes* para los *clabjets Permorfers*) que se han requerido en cada contexto, pero que al no ser comunes a (b) y (d) no se representan en el meta-modelo (a).

La especialización del DSMM en los DSML (a) y (b) permiten la creación de diferentes aplicaciones en el contexto de la educación (c) y en el dominio de los procesos de desarrollo en ingeniería del software (e).

2.3. MetaDepth

MetaDepth [4] es una herramienta de meta-modelado multi-nivel desarrollada bajo el proyecto METEORIC.

Esta herramienta permite el diseño de modelos de manera textual y haciendo uso de múltiples niveles. Los modelos, además, pueden ser caracterizados por un atributo *potencia* que permite determinar el número de instanciaciones máximas que pueden llevarse a cabo sobre el modelo.

Código 2.1: Modelo Website implementado en metaDepth

```
1  Model WebSite@2 imports WebComponents {
2
3     Node Site {
4        title : String;
5        pages : PageType[*];
6        codeGenerator@1 : String[0..1] = "webSiteGen.egl";
7     }
8
9     Node PageType {
10       title : String;
11       indexable : boolean;
12       compData : WebComponent[*];
13       compLayout : WebComponent[*];
14       codeGenerator@1 : String[0..1] = "htmlGen.egl";
15       type : String;
16    }
17
18 }
```

En el Código 2.1 se muestra un modelo WebSite de potencia 2 y sus nodos que, al no especificar un valor distinto, heredan el mismo número de instanciaciones máximas.

El término Node en MetaDepth es una palabra reservada que determina elementos sin un tipo ontológico, esto es, que no son instancia de un elemento superior.

Los atributos de cada elemento deben ser declarados con un tipo y un nombre. De manera adicional se podrá incluir la potencia asociada a ese atributo y que puede diferir a la del modelo padre. También puede añadirse un valor inicial o la multiplicidad del atributo indicándola entre corchetes.

En el ejemplo el campo codeGenerator del Node PageType es declarado de tipo String y con una potencia de valor 1, además se ha iniciado su valor a "htmlGen.egl". El cero y uno entre corchetes [0..1] indican que el atributo codeGenerator podrá tomar o no valor, sin embargo, el asterisco en compLayout expresa que pueden encontrarse cero o más WebComponents.

Para hacer posible la generación de código, MetaDepth integra Epsilon Generate Language (EGL) [7] que es un lenguaje "Modelo-a-texto"que permite generar texto a partir de un modelo.

EGL escribe en un fichero de salida el contenido del template salvo el código introducido entre los símbolos [% y %] que determinan el espacio de código que será ejecutado y que permite el acceso a los datos de los modelos.

En el ejemplo 2.1 se muestra una colección de elementos WebComponents a los que se desea acceder para colocar sus datos en un fichero HTML o PHP. En la fase en la que se diseña el generador de código no es posible conocer qué elementos contendrá el modelo, pero, gracias a la integración de EGL en MetaDepth podemos acceder a la colección haciendo uso de las etiquetas [% y %] e iterar sobre ella.

Conseguimos con esto la construcción de un generador de código para el cual resulten transparentes los elementos que conformen el modelo y pueda adaptarse sin problemas a posibles cambios en el diseño.

Además, la posibilidad de enlazar un modelo a un generador en cualquiera de los meta-niveles hace posible la elaboración de un template final que se haya ido formando a medida que se ha ido especializando el diseño, es decir, para un modelo de potencia 2, asociado a un template, se le añade el código del template asociado al modelo de potencia 1 y por último, el de potencia 0, consiguiendo combinaciones finales que se adecuen a distintos modelos requeridos.

Capítulo 3

Estado del Arte

Como ya se ha señalado, en el desarrollo Web se encuentran importantes deficiencias que se deben subsanar. Existe una enorme variedad de tecnologías que pueden ser empleadas para realizar prácticamente la misma tarea pero que se aprecian dispares por el uso de su propia notación, las técnicas para realizarlas y el uso de modelos propietarios.

Este aspecto dificulta la interoperabilidad entre metodologías y hace difícil encontrar soluciones a problemas que realmente no son tan complejos.

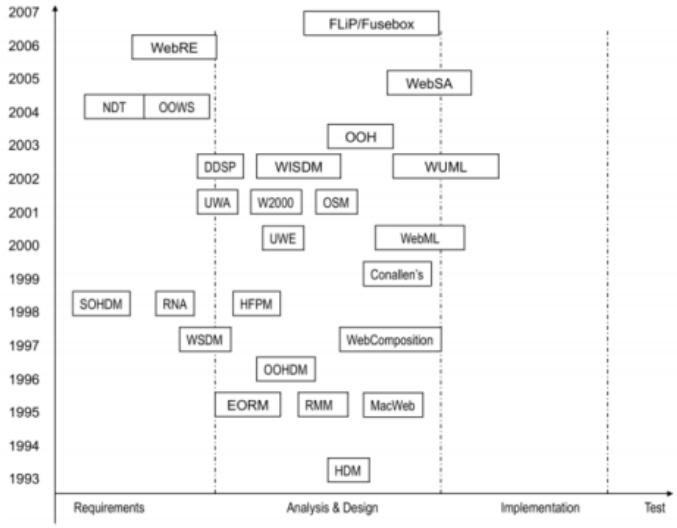

Figura 3.1: Evolución y cobertura de las metodologías de desarrollo web más conocidas [5].

Tal y como se muestra en la figura 3.1 no existe una metodología que abarque todas las fases del ciclo de vida del desarrollo Web y esto implica que deben emplearse diferentes enfoques, aspecto que puede ser un problema ya que, como acabamos de mencionar, existe una falta de compatibilidad entre técnicas.

Otro inconveniente de utilizar un sistema heterogéneo de herramientas y tecnologías

es la separación y falta de comunicación entre el proceso de desarrollo y las metodologías. Es decir, el uso de múltiples tecnologías no permite que se establezca una relación entre la técnica y cada fase del ciclo de vida software, dificultando esto el proceso.

La Ingeniería Web Dirigida por Modelos, o MDE aplicada al desarrollo Web, propone la representación abstracta de conceptos a través de artefactos software independientes de la plataforma para la que se va a implementar la solución, los ya mencionados meta-modelos.

Estos meta-modelos dan la posibilidad de expresar la lógica de negocio sin tener que prestar atención a distintas terminologías o estándares, permitiendo por tanto enfocar los esfuerzos a la resolución del problema que motiva el desarrollo.

Definir estos meta-modelos no es una tarea trivial ya que se requiere la capacidad de encontrar la forma en la que agrupar diferentes conceptos y funcionalidades de manera que se consiga, realmente, que estos sean independientes de cualquier tipo de metodología o contexto.

A continuación se presentan distintos métodos de desarrollo Web basado en el paradigma MDE.

3.1. WebML

Web Modeling Language (WebML) es una metodología para especificar el contenido, composición y navegación de aplicaciones Web complejas con una notación visual.

Los objetivos principales del proceso de diseño WebML son [16]

- Expresar la estructura de una aplicación Web con una descripción alto nivel, la cual puede ser empleada para realizar consultas, desarrollos futuros y mantenimiento.
- Proporcionar diferentes formas de visualizar un mismo contenido.
- Separar la información a mostrar de la composición de la página, la navegación entre las mismas y la presentación. Definiendo estos aspectos de manera independiente.
- Almacenar en repositorios la meta-información que ha sido recabada durante la fase de diseño. Esta información puede usar a lo largo de todo el ciclo de vida de la aplicación con el objetivo de generar dinámicamente páginas Web.
- Permitir especificar las operaciones de manipulaciones de datos para poder actualizar el contenido del sitio o interactuar con él con servicios externos.

Esta metodología hace uso de cinco modelos: Estructura, composición, derivación, navegación y presentación. Cada uno de estos modelos es desarrollado en un proceso interactivo.

Una de las contribuciones más interesantes de WebML es la herramienta WebRatio, una aplicación de Ingeniería de Software Asistida por Computadora, también denominada herramienta CASE, que permite la aplicación de la metodología propuesta por WebML de manera sistemática.

3.2. OOHDMDA

OOHDMDA [1] es una extensión del famoso OOHDM, una de las metodologías de ingeniería Web más importantes. Su principal atractivo, compartido por otros enfoques que también se presentan en esta sección, es la separación del diseño Web en tres ámbitos; El nivel conceptual, el nivel de navegación y el de interfaz.

El funcionamiento está basado en la conversión mediante transformaciones de artefactos PIM a PSM basados en servlets. Para ello, proporciona una herramienta de diseño basado en UML que haciendo uso de los modelos conceptuales y de navegación de OOHDM genera un modelo en un fichero XMI, estándar del OMG para el intercambio de metadatos vía Extensible Markup Language (XML).

En OOHDMDA se definen las clases para los modelos de comportamiento de los que se extrae la semántica de comportamiento, que, añadida a la información del fichero XMI construyen un fichero PIM XMI que a su vez es mutado mediante transformaciones definidas para la especialización del artefacto en un fichero PSM-XMI adaptado a una tecnología servlet específica.

En secciones anteriores se explicaba cómo MDE favorecía la integración de diferentes enfoques priorizando la interoperabilidad entre tecnologías y OOHDMDA es, precisamente, un buen ejemplo de ello ya que esta misma tecnología es una extensión de OOHDM a la cual se han incorporado nuevos conceptos para la generación de artefactos PSM.

3.3. UWE

UWE (UML Web Engineering)[15] es una de las técnicas más importantes en desarrollo Web y una de las primeras en integrar el paradigma del desarrollo dirigido por modelos.

Esta metodología proporciona una notación de domino específico y una herramienta para el desarrollo de aplicaciones Web.

Una de sus ventajas principales es la utilización de modelos basados en UML. Esta característica facilita enormemente la tarea de modelado gracias a la gran aceptación de la que goza el lenguaje de modelado UML y reduce de manera considerable la curva de aprendizaje para los desarrolladores Web.

Para el desarrollo de aplicaciones Web UWE hace una distinción entre los siguientes ámbitos: el contenido a modelar, la estructura de navegación y la representación gráfica. La especificación del contenido se hace mediante diagramas de clase UML, los cuales contienen clases, atributos, asociaciones, herencia y otros elementos del modelo UML.

Para el mapa de navegación entre páginas se construye un diagrama de navegación, esto es, un conjunto de nodos conectados por enlaces. Los tipos de nodos que pueden emplearse son clases de navegación, que vienen a ser los elementos a los que se puede acceder usando otros flujos de navegación, típicamente una clase de navegación puede ser entendida como una página o sección. Otros tipos de nodos serían menús, índices o consultas.

Por último, en el ámbito de la presentación de los datos, es requerido el diseño de otro diagrama de clases, esta vez, que presente de manera simplificada qué componentes conforman cada pantalla del sitio Web.

3.4. WebDSL

WebDSL[8] es un lenguaje de dominio específico (Domain Specific Language, DSL), es decir, un lenguaje que ha sido diseñado para un contexto en particular y que proporciona un alto nivel de abstracción al usuario y están por tanto destinados a la utilización por expertos en el dominio que puedan ser capaces de sintetizar y comprender problemas comunes que suelan existir en su ámbito.

La función de este lenguaje es la de emplearse en el desarrollo de sitios Web con un modelo de datos complejo. WebDSL consiste en un conglomerado de sub-lenguajes que agrupan diferentes técnicas de dominios dispares pero que en su conjunto conforman la funcionalidad de una aplicación Web. Con estos sub-lenguajes se pueden definir los modelos de datos y las páginas para la visualización y edición de estos modelos.

A estos modelos de datos, descritos usando un modelado textual, es posible asignarles un tipo o incluirles unos atributos.

Con una diferenciación menos clara que en UWE, WebDSL expresa la navegación del sitio mediante formas navigate que se incluyen en los modelos antes mencionados.

WebDSL sigue el paradigma de generación de código como el método para realizar las transformaciones entre modelos. Se han desarrollado las transformaciones modelo-a-modelo, modelo-a-código y código-a-código. El generador de código de WebDSL divide estas mutaciones en diferentes etapas, partiendo desde un modelo cercano al núcleo de los sub-lenguajes WebDSL para ir hacia una construcción más especializada y contextualizada.

3.5. Creadores/Editores Online

Actualmente se han popularizado multitud de herramientas en línea que permiten la creación y edición de páginas Web con gran facilidad y sin, prácticamente, requerir conocimientos de desarrollo Web.

En este capítulo se incluye un breve resumen de su funcionamiento, aunque es importante remarcar que estas metodologías no siguen la filosofía de Ingeniería Web Dirigida por Modelos.

Estos servicios proveen al usuario de una librería de templates configurables desde el propio navegador que están preparados para únicamente tener que añadir el contenido que se desea publicar en las páginas; imágenes y texto, además de permitir pequeños cambios en el layout.

Entre los más populares actualmente pueden encontrarse los proveedores Wix[18], 1and1[9], Jimdo[13] y Webnode[17]. El funcionamiento de los editores de sitios de cada uno de ellos es muy similar. Eligiendo un diseño predefinido entre una galería comúnmente separada por categorías en función del tema principal de la página que se quiere diseñar, el usuario puede incluir diferentes componentes desde una interfaz gráfica tales como imágenes, tablas, vídeos incrustados, etc.

Con estas herramientas es posible la creación de un sitio Web sencillo de cuidado diseño gráfico en muy poco tiempo y con gran facilidad.

El principal inconveniente de estas plataformas reside en el tipo de páginas que se pueden llegar a producir, de carácter simple, con poca posibilidad de interacción con el usuario final, y a menudo, con la incapacidad de incorporar funcionalidad compleja como conexión con APIs de otros servicios, acceso a bases de datos externas, etc.

3.6. Perspectiva General

Como se ha descrito, las tecnologías basadas en desarrollo Web dirigido por modelos proponen la representación del diseño Web mediante modelos. Para ello es necesaria la creación de un lenguaje de modelado y se espera que ese lenguaje pueda ser definido por el desarrollador si no existe una alternativa que cubra los requerimientos extraídos durante el análisis del producto que se desea implementar. Si por ejemplo se quisiera desarrollar una aplicación para la gestión de una clínica sanitaria, sería interesante que el lenguaje de modelado que se empleara en el proceso de creación tuviera una semántica adaptada a este contexto y nos permitiera definir símbolos que, típicamente, aparecerían durante la implementación del software destinado a fines médicos.

Para hacer posible la creación de estos lenguajes de modelado es necesaria la creación de los meta-modelos, que, como se ha presentado en el capítulo anterior, permiten que cada concepto general dependiente del contexto tenga una representación abstracta.

Las tecnologías que se han estudiado en este capítulo coinciden en la necesidad del uso de meta-modelos durante el ciclo de desarrollo y en particular determinan la importancia de la diferenciación entre el modelo de navegación por las páginas Web que forman el sitio, la estructura de la información a mostrar y la representación gráfica de la misma.

De las diferentes metodologías analizadas, exceptuando las de creación/edición online por no basarse en la filosofía MDE, se ha realizado una comparativa presentada en la tabla 3.1 que muestra la relación de las diferentes técnicas con los niveles de la arquitectura MDA. En esta tabla extraída de [5] se incluye, además, la comparación con el Generador Web, del cual su diseño es objeto de este proyecto.

	CIM	PIM	PSM	Code
WebML		X	X	X
OOHDMDA		X	X	
UWE	X	X	X	
WebDSL		X	X	X
Generador Web		X	X	X

Tabla 3.1: Relación entre enfoques y arquitectura MDA.

Todas las técnicas estudiadas incluyen transformaciones entre modelos independientes y dependientes de la plataforma, sin embargo, otros como OOHDMDA o UWE no incluyen en su proceso la generación del código final.

El generador de sitios web desarrollado en este proyecto está orientado a la creación de páginas para el dominio de grupos de investigación y es por eso que no está presente en el nivel de Modelos Independientes del Contexto (CIM).

Por otro lado, los creadores de sitios online presentados en este capítulo como Wix, 1and1, etc, si bien son capaces de dotar a los usuarios de la capacidad de construir una página Web sencilla con suma facilidad, en un tiempo reducido y con un diseño de la interfaz gráfica cuidada, encontramos múltiples desventajas asociadas a este tipo de sistemas para el diseño web desde el navegador y encontradas en todos los casos de estudio que aquí se han presentado. El tipo de páginas que pueden ser construidas con estos servicios suelen ser aplicaciones web simples que muestren texto e imágenes pero no requiera la inclusión de componentes que realicen tareas complejas como por ejemplo conexiones a APIs de terceros, uso de servlets o cualquier otra funcionalidad avanzada que no haya sido inicialmente incluida en el template creado por el proveedor del servicio.

Capítulo 4

Generador web

En este capítulo se describe el generador automático de sitios web que se ha desarrollado y cuyo objetivo es el de proporcionar una herramienta software capaz de construir sitios web con facilidad, haciendo uso, para ello, de una librería de componentes web típicamente empleados en el desarrollo de páginas web.

En este proyecto se presenta una solución particularizada en el contexto de sitios web para grupos de investigación.

4.1. Arquitectura

La arquitectura del generador se basa, como se muestra en la figura 4.1, en la separación de los modelos, los generadores de código y las librerías, que se ensamblan con la herramienta metaDepth para generar los ficheros .html y .php que darán lugar a la página web final.

Para la generación completa del sitio, inicialmente, es necesaria la utilización de modelos implementados en MetaDepth que definan de manera abstracta los requerimientos de la página que desea construirse.

En este caso se proporciona un conjunto de metamodelos y modelos orientados al diseño web y a la generación de sitios para grupos de investigación en particular.

Partiendo de un metamodelo inicial "WebSite"se inicia un proceso de especialización del mismo en función de los requisitos que determina el producto que se desea conseguir. Cada modelo puede llevar asociado unos generadores de código que serán ejecutados secuencialmente y en un orden específico que, apoyado en las librerías adjuntas, implementarán los ficheros finales que conforman la página web.

La organización interna de un proyecto de Generación Web es la siguiente:

- **Modelos:** Se describen en los ficheros MetaDepth. Contienen únicamente código MetaDepth e incluyen los enlaces con los ficheros de generación de código.

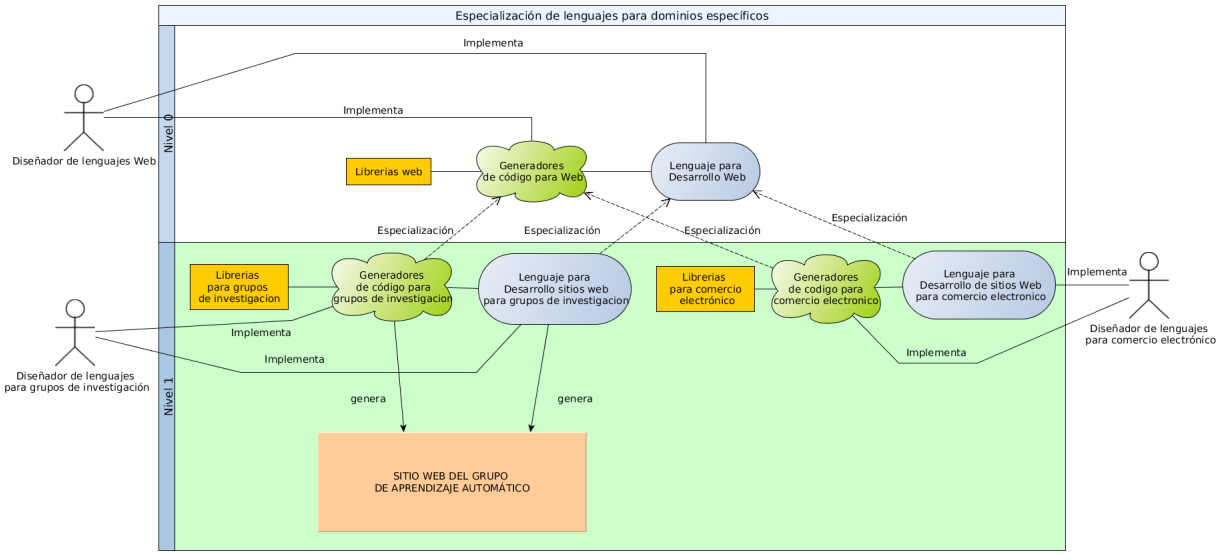

Figura 4.1: Proceso de especialización de lenguajes para dominios específicos.

- **Generadores de código:** Implementados en los archivos egl. Haciendo uso de Epsilon Generate Language se codifican estos ficheros que en función del diseño construído en los ficheros de modelado escriben en los ficheros de salida el código HTML, JAVASCRIPT y PHP necesario para construir la funcionalidad requerida.
- **Librerías:** Conjunto de los distintos ficheros fuente que son necesarios para la generación de determinada funcionalidad, como por ejemplo, APIs externas para el diseño de grafos u otros componentes web complejos así como gran cantidad de estilos CSS para hacer posible la personalización de la interfaz gráfica de las páginas.
- **Binario metaDepth.jar:** Software desarrollado bajo el proyecto METEORIC que a partir de los ficheros mdepth realiza las llamadas a generadores de código y devuelve los ficheros html y php.

4.2. Modelos

El punto de partida de la fase de modelado es el Meta-modelo Website. Éste implementa la idea básica de que un sitio web es un conjunto de páginas web. Además se le asocia un generador de código al sitio general y otro a cada página.

Por un lado debe generarse el sitio general que servirá de nexo de unión de todas las páginas y por otro lado distintos generadores asociados a distintas páginas podrán especializarlas según el contenido que desee publicarse en ellas.

Esto puede verse con facilidad, por ejemplo, en el sitio web de un grupo de música. Para este tipo de sitios es previsible que sea necesaria una página con la relación de próximos eventos, una galería fotográfica con imágenes de los conciertos, otra con la historia del

grupo y posiblemente un formulario de contacto para poder realizar contrataciones. La distinta naturaleza de cada una de estas secciones hace patente la necesidad de asociar distintos generadores de código, uno preparado para conformar listados, otro para mostrar una secuencia de imágenes y un último capaz de mostrar un elemento *form* HTML.

Sin embargo, es posible que todas estas secciones tengan en común una cabecera con el nombre del grupo, un menú que permita la navegación entre las páginas o un footer a pie de página en el que aparezcan los datos de contacto.

Es interesante, entonces, la diferenciación de elementos comunes a todas las páginas como elementos que conforman el layout de elementos que muestran el contenido.

Siguiendo esta filosofía se incluye entonces a cada Modelo *Página* la posibilidad de contener elementos que muestren datos y elementos que definan la interfaz.

Llegados a este punto es fácil deducir que también será necesario especializar estos nuevos modelos para dar forma a diferentes funcionalidades: Elementos que recojan información de una base de datos, componentes que muestren información en forma de tablas, desplegables o gráficas, o elementos que determinen la apariencia de la interfaz del sitio como los menús de navegación, las cabeceras y la maquetación en general de toda la información.

4.2.1. Especialización de los modelos

Con el objetivo de orientar un modelo al contexto para el que va destinado el producto final es necesaria la especialización gradual de los modelos y los lenguajes de dominio específico, proceso mostrado en la figura 4.1 de la sección anterior.

En el objeto de estudio de este proyecto, sitios web para grupos de investigación, el meta-modelo con potencia más alta y que no es herencia de ningún modelo anterior es WebSite. De aquí, en función del ámbito al que vaya dirigida la web se especializará en diferentes modelos, ArtistWebsite, OnlineShopWebSite, o en este caso, ResearchGroupWebsite. Esta especialización comprende una serie de modificaciones e incorporación de componentes que se consideran interesantes para este contexto pero que no lo serían para otro.

Un modelo cuyo objetivo fuera la generación del código que construya una plataforma de pago haciendo uso de la API de PayPal, no tendría sentido, a priori, que fuera incluído en el sitio web de un grupo de investigación, sin embargo, es primiordial mostrar una relación de las publicaciones que ha llevado a cabo el grupo.

Tras analizar numerosos sitios web de diferentes grupos de investigación (GAA [11], MISO [14] y GINA [12], entre otros) se encuentran multitud de funcionalidades similares que permiten la automatización del proceso de creación de las páginas; Presentación del equipo de investigación, listado de miembros, listado de publicaciones y de patentes, publicación de eventos relacionados con sus estudios,etc.

Por tanto, para este proyecto, se plantea la creación de un Meta-modelo *Site* que contiene Modelos *Pages* de los cuales pueden ser instancia:

- **PresentationPage:** Para la presentación del grupo en la web. De este modelo típicamente heredará la página index donde puede aparecer una breve descripción de los objetivos del grupo, un mensaje de bienvenida y una relación de noticias o publicaciones destacadas.
- **LargeContentPage:** Páginas destinadas a almacenar componentes complejos como listados de publicaciones dinámicos, gráficos o formularios.
- **ListPage:** Para páginas cuyo fin es presentar mediante una lista rápida una enumaración de elementos.
- **MembersPage:** El objetivo principal de estas páginas es la de mostrar un listado o fichas de integrantes del grupo de investigación.

Estos modelos, a pesar de asociarse de manera predefinida a unos componentes, en posteriores instanciaciones se pueden añadir elementos que no hayan sido incluídos en esta preselección y que puedan dotar a la página de una funcionalidad que no haya sido prevista para grupos de investigación.

La diferencia entre los diferentes modelos son los componentes predefinidos, tanto de layout como de datos que se les asocia y que se describen en las siguientes secciones.

4.2.2. Librería de componentes web. WebComponents

En este proyecto se proporciona una librería de meta-modelos y modelos reutilizables que permiten la fácil y rápida construcción de un sitio web para un grupo de investigación. Estos elementos son los llamados WebComponents.

Un modelo Page puede contener elementos WebComponents que le permiten construir una funcionalidad muy específica del sitio. Los componentes se asocian según su utilidad final, si están destinados a la configuración de la interfaz gráfica de la web se agrupan en el atributo compLayout del modelo Page, si por el contrario su función principal es otra, recopilar datos, crear formularios, construir gráficas o cualquier otra operación que no vaya única y exclusivamente destinada al maquetado, se almacenan en el atributo compData.

Para la formación de la interfaz se han implementado los siguientes elementos:

- **Header:** Componente con atributos título y subtítulo. Es la cabecera del sitio. Suele ser un elemento común a todas las páginas y en este caso puede emplearse para colocar el nombre del grupo de investigación para el que se está diseñando la web.

- **Menu:** Relación de hipervínculos dispuestos linealmente que permiten el acceso a diferentes páginas del sitio web. En cada modelo Page se ha incorporado un atributo *indexable* que con valor *TRUE* indica al generador de código que debe ser incluído en el componente Menu, si por el contrario no se requiere el acceso a esa página el atributo debe colocarse con valor *FALSE* o no instanciarse.

- **Intro:** Contiene un bloque de texto y una imagen de gran tamaño. Su función es hacer de mensaje de bienvenida o introducción al sitio. Puede incorporar además un enlace a otra página para mostrar más información, por ejemplo, sobre las funciones principales del grupo de investigación.

- **Content:** Este elemento hace, únicamente, las veces de contenedor de otros componentes. Conforma el cuerpo principal de las páginas y es quizás el elemento más dinámico de todos los que se agrupan bajo la familia de artefactos que definen la interfaz. En la siguientes secciones se describen los componentes que pueden ser incluídos en un contenedor Content y qué pueden aportar al sitio web.

- **Footer:** Por último, otro de los puntos en común de todos los sitios visitados durante el análisis de sitios de grupos de investigación, es el de la inclusión, al final de las páginas, de una sección con datos de contacto, enumeración de licencias que aplican sobre el trabajo del grupo y el diseño de la web, etc.

Además se ha dotado a cada uno de estos elementos de un atributo "style"que puede tomar valores enteros y que para cada uno de ellos asigna una plantilla de estilos CSS diferente, de manera que facilita la creción de sitios completamente diferentes realizando combinaciones entre los distintos componentes.

Los componentes asociados al atributo compData son bastante más diferentes entre sí ya que pueden realizar tareas muy dispares.

A continuación se enumeran los más relevantes implementados durante el desarrollo de este proyecto:

- **DataSchema** Este componente sirve de nexo de unión entre uno o varios componentes DataVisualizer y un DataCollector ambos descritos más adelante. Su función es, una vez conseguido una serie de datos a través de alguna fuente de información añadirles la posibilidad de ser mostrados en una página asociándoles uno o más visualizadores previamente definidos.

  Para entender su funcionamiento es necesario estudiar estos elementos en profundidad.

- **DataCollector** Implementa la recolección de información de un almacén de datos determinado que puede ser un fichero de texto, una base de datos o cualquier otra fuente externa. El objetivo aquí es dotar al diseñador de un elemento capaz de

tomar datos de fuentes heterogéneas de manera transparente y que puedan ser tratados posteriormente, permitiéndose así la realización de este procesado de manera automática.

Ejemplos interesantes serían:

- **ORCIDCollector** Realiza la conexión a las bases de datos de ORCID. Open Researcher and Contributor ID (ORCID) proporciona un identificador digital único para cada investigador que se registre en su plataforma y permite asociar su ID a un listado de trabajos y publicaciones realizadas por el mismo. Toda esta información se almacena en bases de datos de acceso público y para el cual facilitan una API que permite mediante peticiones por URL extraer datos para un ID determinado.

  El componente contiene un atributo en el que almacenar un número de identificación ORCID y una petición, ya sea el perfil del investigador, el listado de sus publicaciones o ambas cosas.

  Al modelo se le ha asociado un generador de código programado en PHP que con la integración de EGL es capaz de extraer estos datos y realizar la consulta especificada.

  Para ello, se abre una conexión con la base de datos mediante la herramienta cURL que permite la transferencia de ficheros usando una sintáxis URL. Para versiones de PHP superiores a 4.0.2 se dispone de la libería libcurl que implementa la conexión y comunicación con diferentes tipos de servidores y a través de múltiples protocolos. Esta librería permite la inicialización de un manipulador cURL al que pueden añadirse las opciones de conexión que determina la API de ORCID para establecer la conexión y poder tener acceso a la información de sus bases de datos.

  Este componente debe instanciarse con los datos del investigador del que se desea descargar la información y el tipo de consulta que desea realizarse. Es necesario crear un objeto puro de este componente por cada investigador de que se quiera extraer el listado de publicaciones o su perfil.

  Por si solo este elemento no aporta información visible a las páginas web y únicamente sirve como herramienta para recabar datos que otro tipo de componente se dedicará a mostrar.

  El resultado de las consultas devueltas por la API se almacena en un fichero XML que desde el propio generador de código es parseado y copiado a un array asociativo multidimensional de PHP para un acceso sencillo y transparente al resto de componentes de visualización.

- **BibtexCollector** Este modelo encapsula el proceso de conversión de un fichero en formato Bibtex a código HTML. Para su implementación se ha utilizado la librería open source phpBibLib que codifica esta funcionalidad.

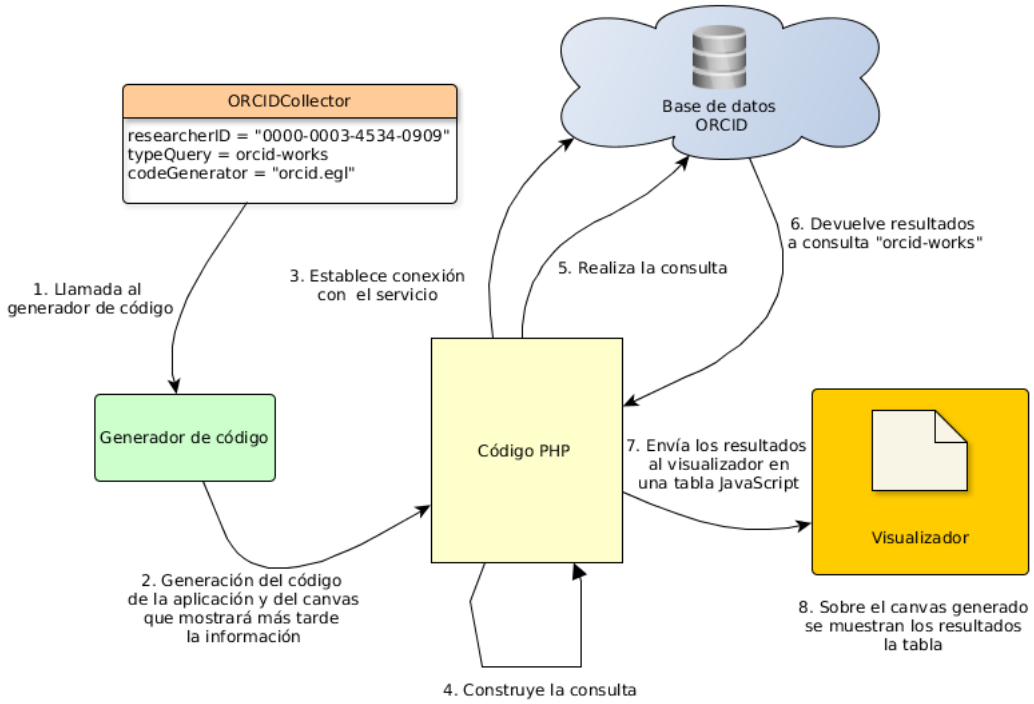

Figura 4.2: Proceso de extracción de datos con un ORCIDCollector

En este caso el generador de código accede al atributo source del componente que almacena el nombre del fichero .bib que contiene los datos Bibtex, esta información es procesada en PHP llamando a las primitivas de la API y realizando la conversión a código HTML.

- **DataVisualizer** Una vez se ha recabado la información que desea mostrarse es necesario determinar la forma en la que la misma será mostrada en las páginas web. Es cierto que típicamente los elementos visualizadores más utilizados sean simples labels de texto o la inclusión de una imagen pero, como se ha descrito en el apartado anterior, existen elementos DataCollector que nos devuelve información que para poder extraer todo su valor es necesario mostrar de una manera determinada. Los elementos que se han implementado para estos casos son:

  - **TableSchema** Tabla simple con posibilidad de añadir títulos de columnas y scroll automático si se desea. Permite mostrar un listado de gran tamaño sin ocupar excesivamente el espacio vertical del sitio web.
  - **ChartComponent** Para datos que sean más interesante mostrar en forma de gráfica se han adaptado los elementos Chart de Google que permiten, con una simple incrustación de funciones JavaScript, representar la información tabulada tal y como requiere la API. Un ejemplo de la utilización de estos elementos se ha llevado a cabo para presentar los resultados de las consultas realizadas por los DataCollector a las bases de datos de ORCID. Con ello se

pueden conseguir gráficos circulares, lineales, de barras, etc, que muestren, por ejemplo, el incremento de publicaciones llevadas a cabo por un investigador, qué tipos de publicaciones suelen realizar determinado grupo de investigación, y otros aspectos relacionados con la actividad investigadora.

- **GraphComponent** De la misma manera, gracias a los datos encontrados por el DataCollector sobre una base de datos con una relación de publicaciones, es posible extraer información como la de los autores que han colaborado en una investigación determinada. Con esta información, asociando el componentente al generador de código pertinente, se ha implementado en PHP una funcionalidad que permite crear una red de enlaces entre los investigadores. Esta red, es almacenada en una variable que se envía a las funciones de la librería D3JS. Esta librería posibilita la creción de gráficos SVG interactivos para su posterior incrustación en un fichero PHP o HTML.
- **MapComponent** Otro funcionalidad que nos permite Google a través de sus APIs es la de posicionar elementos, con componentes geográficos entre sus atributos, en un mapa. Con estas características podemos asignar a distintos componentes de la web la propiedad *geoposicionable* con el objeto de mostrarlos luego sobre el plano de una ciudad o un mapamundi. De esta forma, si por ejemplo se estuviera diseñando el sitio web de un grupo de investigación con miembros residentes en paises diferentes, podemos mostrar sobre un mapa la procedencia de las publicaciones.
- **WordCloudComponent** En muchos casos las páginas muestran gran cantidad de información y puede ser dificil, a simple vista, distinguir los temas de los que se está tratando. Para facilitar esta tarea se ha diseñado el elemento gráfico WordCloudComponent cuya tarea es parsear los atributos de los componentes seleccionados y extraer de ellos las palabras con mayor ocurrencia, de manera que pueda generar de manera automatizada una nube de palabras clave que ayuden a resumir la temática de la página. Este componente también se ha implementado haciendo uso de JavaScript.

4.3. Generadores de Código

Como se ha venido explicando en las secciones anteriores, una parte fundamental del desarrollo dirigido por modelos y de este proyecto en particular es el uso de generadores de código.

Para su utilización en este generador de páginas web, el código para los generadores se ha emplazado en los ficheros de formato EGL. En el contenido se encuentra código de diferentes lenguajes; PHP, JavaScript, HTML que son directamente impresos en el fichero de salida y, por supuesto, EGL que permite el acceso, de forma dinámica, a los modelos.

Código 4.1: Modelo que incluye un generador de código.

```
Node Site {
   title : String;
   arrayElementos : Elementos[*];
   codeGenerator@1 : = "gen.egl";
}
```

En el ejemplo mostrado en la figura 4.1 se muestra un elemento descrito en MetaDepth al que se le ha asociado el generador *gen.egl*. Como se describió en el capítulo 2 los símbolos [*] tras el atributo *arrayElementos* y que determinan la cardinalidad del atributo, indican que se trata de una colección de componentes *Elementos*.

Si estuviéramos interesados en listar el nombre de estos elementos en un fichero HTML podríamos escribir un generador de código de la siguiente manera:

Código 4.2: Generador de código que recorre una colección.

```
[% for (elemento in component.arrayElementos){%]
        Name of the element :  [%=elemento%]
[%}%]
```

Como se explicó anteriormente, el código entre [% y %] es interpretado por EGL y en este caso se ha empleado para iterar sobre el array de elementos del componente *component*, para obtener el nombre de la variable elemento por cada iteración.

En este otro ejemplo se muestra una generación más compleja en la que se incluye código HTML incrustado en el fichero EGL y que generará una tabla con las etiquetas table y poblada por el nombre de los *subElement* que componen la colección *arrayElementos*.

Código 4.3: Generador de código realiza un bucle anidado.

```
<html>
<head>
<title>Page Title</title>
</head>
<body>
 <table>
  <tr>
  [% for (element in component.arrayElements){%]
   <td>
        [% for (subElement in component.arrayElements){%]
        [%=subElement%]
        [%}%]
   </td>
  [%}%]
  </tr>
```

```
16          </body>
17          </html>
```

Estos ficheros EGL nos permiten, con la compenetración de distintos lenguajes, la construcción de páginas web sin conocer a priori el contenido que se va a mostrar. Es decir, se hace posible el diseño de templates que se vayan a completar en tiempo de compilación por el contenido que determinen los modelos implementados en código MetaDepth.

Además, y como se ha comentado anteriormente, el uso de varios templates en cascada como se muestra en el siguiente ejemplo, permiten la elaboración de una página final que puede ser fruto de múltiples combinaciones en función de los modelos que hayan sido diseñados.

Código 4.4: Generador de código de primer nivel.

```
1
2  <html>
3          <head>
4                  <title> First Generation -Two levels </title>
5          </head>
6          <body>
7  [%
8    for (component in page.references('comps')) {
9      for (c in site.value(component)) {
10       var t : Template := TemplateFactory.load(basePath+c.
            codeGenerator);       //Obtenemos el generador de codigo
             del componente
11
12       t.populate('output', output); //Propagamos valor de
            diferentes variables
13       t.populate('site', site);
14       t.populate('component', c.getValue());
15       t.populate('basePath', basePath);
16
17       t.process(); //Llamamos al generador de codigo de cada
            componente
18     }
19   }
20 %]
21         </body>
22 </html>
```

En el código anterior se muestra como se recorre una colección de componentes asociadas a un modelo page y se van ejecutando de manera secuencial los generadores de código de cada elemento de la colección comps.

Código 4.5: Generador de código de segundo nivel.

```
<p> Hello </p>
```

Código 4.6: Generador de código de segundo nivel.

```
<p> World! </p>
```

En el ejemplo, cada componente tiene únicamente la impresión de las palabras Hello y World. Por tanto unir el template padre con la generación de los dos elementos de la colección el resultado HTML será:

Código 4.7: Resultado de la generación haciendo uso de tres ficheros EGL.

```
<html>
        <head>
                <title> First Generation -Two levels </title>
        </head>
        <body>
                <p> Hello </p>
                <p> World! </p>
        </body>
</html>
```

Capítulo 5

Pruebas y caso de estudio

En este apartado se describe el conjunto de pruebas al que se ha sometido el software desarrollado en este proyecto y el proceso de creación, mediante el generador Web, de la página del grupo de investigación Machine Learning Group que desarrolla su actividad principalmente en la Escuela Politécnica Superior de la Universidad Autónoma de Madrid.

5.1. Pruebas por componentes

En la sección 4.2.2 se explicaba el funcionamiento de la librería de componentes que se proporciona en este proyecto.

El objetivo de este apartado es comprobar el funcionamiento de los componentes más complejos que han sido diseñados para la creación de sitios Web y de los cuales es interesante mostrar los resultados de su generación de manera independiente por no haber sido incluidos en la página del Grupo de Aprendizaje Automático.

5.1.1. ChartComponent

A lo largo del documento se ha descrito el proceso de diseño de componentes capaces de recoger datos de distintas fuentes; ficheros, bases de datos, etc. Para que estos datos puedan ser visualizados, tal y como se explicaba anteriormente, se hace uso de los componentes DataSchema que asocia un DataCollector con un DataVisualizer.

En este apartado se analiza el resultado de la generación de un ChartComponent, instancia de DataVisualizer, al que se le ha enviado la información recogida tras enviar una consulta a la base de datos pública a la que da acceso ORCID. En la consulta se envía el identificador de un investigador y la petición de su relación de publicaciones, con el objetivo de contar el número de entradas de la respuesta de la base de datos y acumularlo por años.

La forma seleccionada para visualizar los resultados, en esta ocasión, ha sido un gráfico circular. Esta funcionalidad se encuentra encapsulada en el componente ChartComponent, en este caso de tipo PieChart (gráfico de tarta).

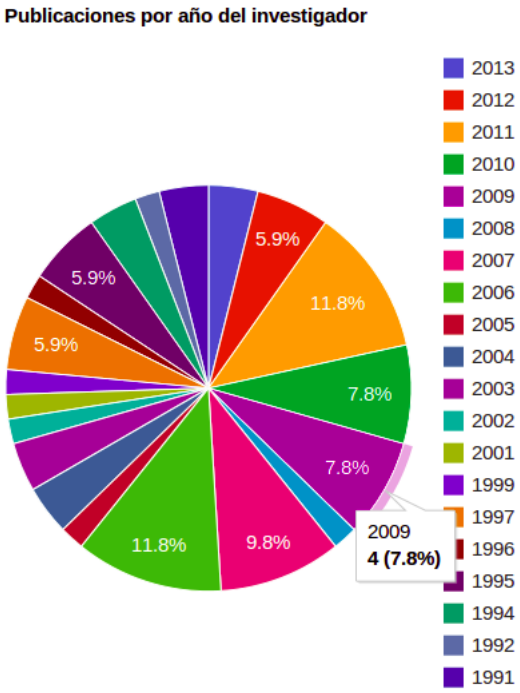

Figura 5.1: Gráfico generado por ChartComponent, muestra publicaciones por año de un investigador.

El resultado obtenido es un diagrama JavaScript interactivo que permite seleccionar diferentes sectores del círculo y obtener información más detallada en forma de tooltip.

Se puede observar en la figura 5.1 como cada sector de diferente color representa un año y sobre éste aparece el porcentaje de trabajos que fueron publicados en ese año.

Esta misma información podría ser interesante mostrarla en otro tipo de gráficos, como por ejemplo un diagrama de barras. En este caso, y gracias al componente BarChart, instancia también de ChartComponent, se puede indicar al visualizador que habrá diferentes series de datos. Una serie de datos, en este ejemplo que nos ocupa, podría ser el tipo de publicación al que se refiere cada barra, es decir, si es un artículo, un libro, etc.

El resultado de esta generación puede observarse en la figura 5.2.

Para este caso, el componente crea un visualizador JavaScript en el que se diferencian las series de datos por colores que asigna arbitrariamente.

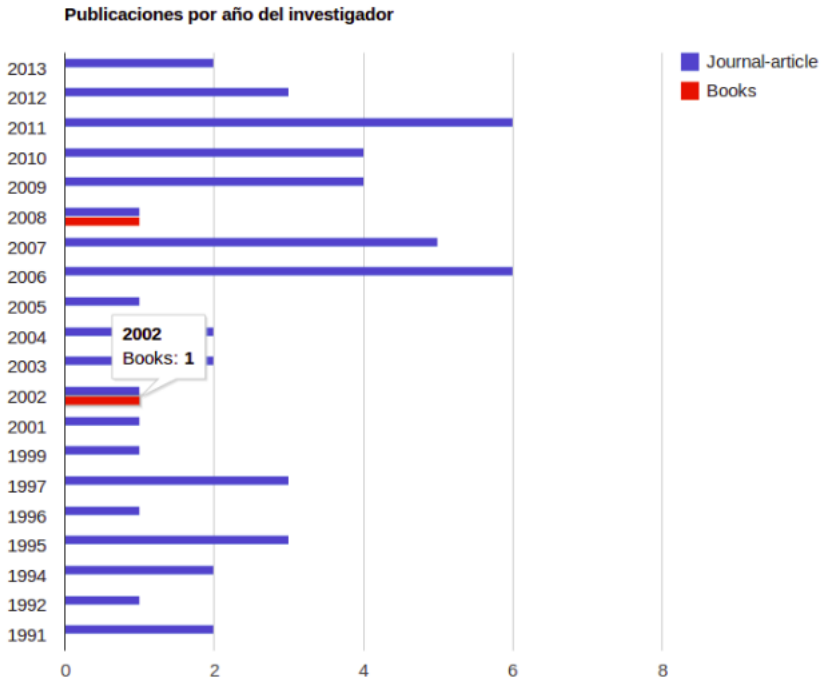

Figura 5.2: Gráfico generado por ChartComponent, muestra publicaciones por año de un investigador.

5.1.2. MapComponent

Otra de la funcionalidad que se ha explicado con anterioridad es la de modelar diferentes elementos como geolocalizables.

Para ello, se incluye al objeto que se quiera situar sobre un mapa los atributos latitud y longitud.

Figura 5.3: Mapa generado por MapComponent. En el se sitúan diferentes elementos geolocalizables.

En la figura 5.3 se muestra el resultado de la llamada al generador de código para un MapComponent al que se le ha enviado una tabla de 9 objetos geolocalizables, en este caso, investigadores.

5.1.3. GraphComponent

Otra de las posibles representaciones gráficas que se pueden obtener de los datos extraídos de las bases de datos de ORCID es la relación entre los diferentes colaboradores de un grupo de investigación o departamento.

En la siguiente figura 5.4 se muestra el gráfico JavaScript resultante de la ejecución de los generadores de código asociados a los componentes de tipo GraphComponent.

Esta imagen interactiva permite seleccionar nodos para comprobar el nombre del elemento, en este caso del investigador, y arrastrarlos para aislarlos del resto del grupo de nodos y poder las relaciones de un colaborador en particular.

Figura 5.4: Gráfico generado por GraphComponent, marca las relaciones entre un investigador y su entorno de colaboradores.

Para la creación de este gráfico se ha incrustado la librería D3JS en el generador de páginas Web desarrollado en este proyecto.

5.1.4. WordCloudComponent

Por último, otro de los elementos que se describían en la sección 4.2.2 eran las nubes de tags. En este caso, una vez más, se ha generado el gráfico sobre información extraída por un DataCollector configurado para utilizar la API de ORCID.

En la figura se muestra el resultado del cálculo de palabras más usadas en los títulos de publicaciones resultantes de una consulta.

Figura 5.5: Gráfico generado por WordCloudComponent, muestra los palabras más empleadas en un texto.

5.2. Sitio Web para Machine Learning Group

A modo de validación del software diseñado se ha implementado un sitio Web real y que actualmente se encuentra activo en la URL [11]

Tras el análisis de las páginas antiguas del grupo se comprueba que el esquema de navegación y conjunto de secciones se asemeja a lo analizado hasta ahora en otros casos de estudio y que, por tanto, sin prácticamente modificaciones sobre el diseño, puede generarse en su totalidad, de manera automatizada, el código HTML y PHP que conforma el sitio sin requerirse una edición manual posterior.

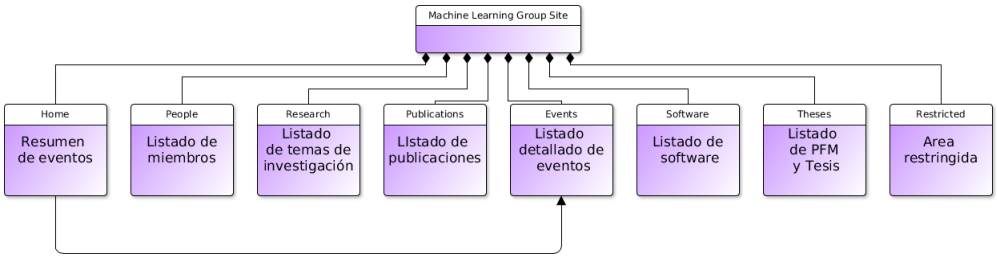

Figura 5.6: Esquema de Navegación Web MLG

El esquema de navegación del sitio Web, hasta ahora, era el mostrado en la figura 5.6. Las relaciones entre las distintas secciones no se muestran ya que todas se relacionan con todas por un menú de enlaces común. Siendo así, se ha querido remarcar el vínculo entre la página Home y Events ya que, además de los enlaces mediante el menú, cada resumen de evento publicado en la página principal Home contiene un link a la sección de Events.

Se concertaron tres reuniones con el actual Webmaster encargado del mantenimiento del sitio y se detectaron los principales requerimientos en los que debe basarse la nueva página.

Hacían hincapié en tres aspectos:

- La renovación del diseño gráfico del sitio pero manteniendo la estructura de navegación actual.
- La no repetición de código o información, esto es, si algo está descrito en una página y vuelve a aparecer en otra que no sea necesario duplicar esos datos, si no que, ambas páginas hagan referencia a la misma información.
- Y por último, automatizar el proceso de inclusión de referencias bibliográficas al sitio Web. Si en alguna sección de la página se va a incluir un listado, por ejemplo, de publicaciones, que no haya que introducir este texto de forma manual y pueda cargarse directamente de un fichero en formato Bibtex.

Para el primero de los requisitos únicamente se han incorporado hojas de estilo diferentes asociadas a los modelos, al ser ficheros CSS independientes, se consigue que pueda ser alterado el diseño gráfico de la página de manera transparente a los modelos.

El caso de información redundante en el sitio Web actual del Grupo de Aprendizaje Automático venía ocurriendo, principalmente, en el proceso de publicación de nuevos eventos. Hasta ahora, el encargado del mantenimiento de la página debía subir una información resumida de la noticia a la página principal y otra más detallada en la sección Events.

Para solventar esto se han diseñado unos modelos que contienen toda la información del evento y que, de manera dinámica, puede ser consultada por distintas páginas.

En el último requerimiento, simplemente hubo que insertar componentes BibtexCollector asociados a los modelos de listas de publicaciones de manera que, desde distintos ficheros .bib, se recabaran las distintas referencias y se incrustaran en la sección correspondiente. Un ejemplo de esto es la instanciación del modelo gaaMembers de tipo Members, que a su vez es instancia de WebComponent, este proceso se representa en el diagrama de la figura 5.7

5.2.1. Modelado

El primer paso para la generación automatizada de este sitio Web es el diseño de sus modelos en código metaDepth. Tener en cuenta la navegación que se desea en el resultado final es crucial ya que establece las bases de los primeros modelos a implementar.

En este caso, por petición del grupo de investigación, el esquema se ha mantenido prácticamente intacto y es por ello que la decisión del diseño es un modelo GroupSite que agrupa los Page para cada una de las secciones: Home, Research, People, etc. En la figura 5.7 se muestra el proceso de especialización de estos modelos.

Tal y como se explica en la sección 4.2.1, del modelo Page heredan PresentationPage, LargeContentPage, ListPage y MembersPage. Cada uno orientan Page a un uso específico y facilita su generación mediante la inclusión por defecto de determinados WebComponents que pueden ser utilizados o no en la página final.

Analizando el esquema 5.6 se puede ver rápidamente que hay un gran número de secciones que únicamente mostrarán un listado y que deberían ser instancia del modelo ListPage. Además serían necesarios un objeto PresentationPage para el Home y otro MembersPage para la sección Members. Por último, las secciones que incluyen componentes de otros tipos heredarán del modelo LargeContentPage.

Quedando, este diseño principal, de la siguiente manera:

Código 5.1: Modelado simplificado de un grupo de páginas.

```
1         GroupSite GaaSite {
```

```
2           groupName = "Machine Learning Group";
3           indexPage = index;
4           sitePages = [people,research,publications, events,
               software, theses];
5         }
6         PresentationWebPage index {
7       title = "Home";
8     }
9         MembersPage people {
10      title = "People";
11        }
12        ListPage publications {
13      title = "Publications";
14        }
15        ListPage research {
16      title = "Research";
17        }
18        LargeContentPage events{
19          title = "Events";
20        }
21        LargeContentPage software{
22      title = "Software";
23    }
24        ListPage theses{
25      title = "Theses";
26        }
```

En este punto ya es posible empezar a armar las páginas con su contenido. Para ello puede resultar más sencillo colocar primero los elementos que definen el layout de la página, explicados en la sección 4.2.2.

Como ejemplo se muestra la edición del componente Intro:

Código 5.2: Modelo de un componente Intro.

```
1         Intros gaaIntros{
2                 Intro gaaIntro{
3                         texto1 = "About Us";
4                         texto2 = "Welcome to the Machine Learning
                              Group Website of Universidad Aut&
                             oacute;noma de Madrid";
5                         textoBoton = "More Information";
6                         style = 1;
7                 }
8         }
```

El atributo texto1 permite asignar el valor a un texto principal bajo el cual se coloca el texto2 como mensaje de bienvenida. Además es posible crear un botón, en este caso con el texto *More Information*.

Por último, el atributo style nos deja asociar diferentes hojas de estilo al componente Intro.

El resultado final de estos componentes puede observarse en el apartado de generación que se presenta en las siguientes páginas.

Una vez se han diseñado los componentes de layout que aparecerán en cada página, éstos deben incluirse en el modelo, instancia de Page, correspondiente.

Para la página Web del Grupo de Aprendizaje Automático se han implementado diferentes componentes de layout que han sido, cada uno de ellos, instancia directa de los meta-modelos proporcionado en el paquete para ResearchGroupWebSite.

Esta especialización de los modelos, llevada a cabo para los componentes que tienen relación con la configuración del layout, no es demasiado compleja. Sin embargo, el resto de componentes sí requieren la implementación de cambios mayores respecto de los proporcionados en el paquete de modelos de ResearchGroupWebsite.

Vamos a analizar ahora un elemento que se repite en este caso práctico, las listas. La página de publicaciones contendrá una enumeración de los trabajos del grupo de investigación.

Para ello se emplea un elemento ListPage, al que se le asigna el título 'Publications' y se le activa la propiedad indexable a True para que esta página sea referenciada desde el menú que se generará de manera automática.

Además incorporamos los elementos de tipo ResearchHeader, ResearchMenu, ResearchFooter y ResearchContent antes mencionados para construir la interfaz gráfica. Estos modelos se incluyen para todas las páginas orientadas a un grupo de investigación genéricos. Para el Grupo de Aprendizaje Automático es necesaria una instanciación más, obteniendo los objetos puros gaaCabecera, gaaMenu, gaaFooter y gaaContAcordeon.

Código 5.3: Modelo de un componente ListPage.

```
1    ListPage publications {
2       title = "Publications";
3       indexable = true;
4       cabe = gaaCabecera;
5       men = gaaMenu;
6       foot = gaaFooter;
7       cont = [gaaContAcordeon];
8    }
```

En este caso, los tres primeros se han especializado únicamente modificando textos a mostrar, como el nombre del grupo, datos para el pie de página y otros cambios menores.

Para el contenedor gaaContAcordeon se ha realizado una instanciación más compleja.

El objetivo de este contenedor es el de almacenar una lista por cada año que a su vez contenga todos los trabajos publicados por el grupo. A estas listas por año se les quiere añadir una funcionalidad adicional, la posibilidad de colapsarse y volver a desplegarse pulsando sobre el título de la misma. Este efecto no se implementaba en el paquete de modelos para grupos de investigación y es por eso que es interesante el estudio de este WebComponent ya que sirve para ilustrar cómo añadir código para cubrir nuevas necesidades que no hayan sido tenidas en cuenta a la hora de desarrollar el paquete ResearchGroupWebsite.

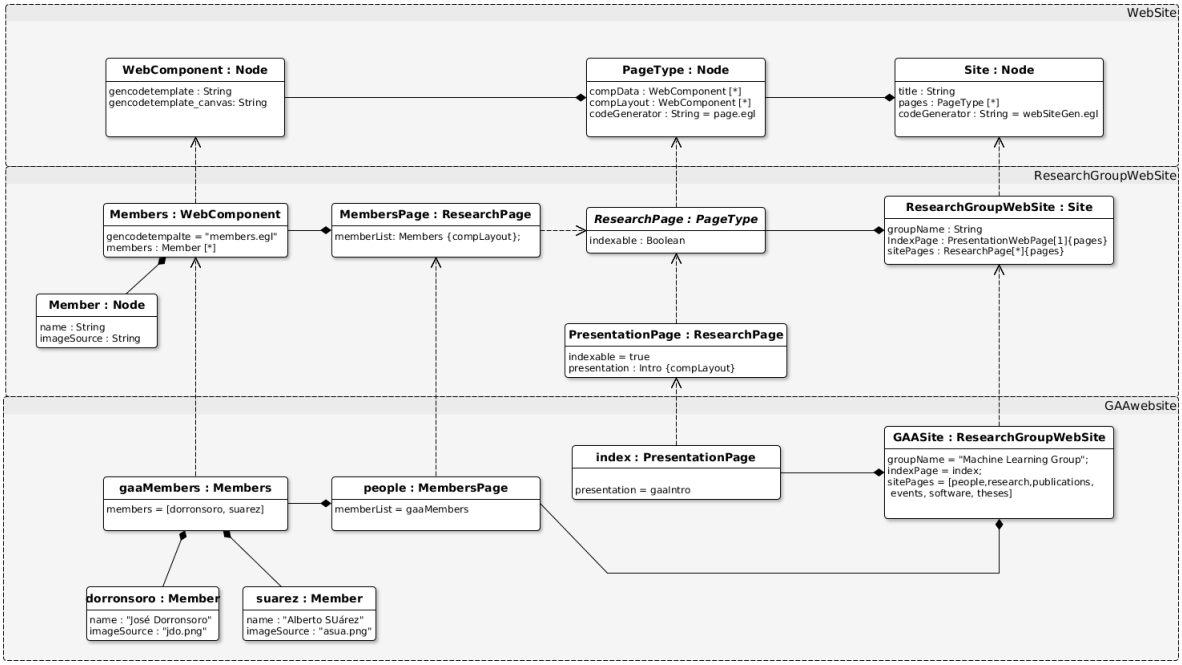

Figura 5.7: Diagrama simplificado de la especialización de modelos para la página del grupo GAA.

En primer lugar se han instanciado las listas. En la fase de análisis de requisitos se decidió incorporar componentes capaces de cargar información de un fichero Bibtex para incluir las publicaciones en la Web, entonces, es necesario que la lista esté formada por clabjects BibtexCollector y cada uno de ellos estará referenciado al fichero .bib del que se desea obtener la información.

Para hacer posible la inclusión de esta funcionalidad y los efectos gráficos sobre las listas es necesario que el nuevo generador de código para el componente gaaContAcordeon coloque en cada elemento de la lista el resultado de la generación de código del componente BibtexCollector.

5.2.2. Generación

Una vez completada la fase de modelado se ejecuta el script encargado de invocar a la aplicación MetaDepth y que además se encargará de copiar las librerías necesarias a la carpeta de destino donde se alojarán los ficheros HTML y PHP resultantes tras el proceso de generación.

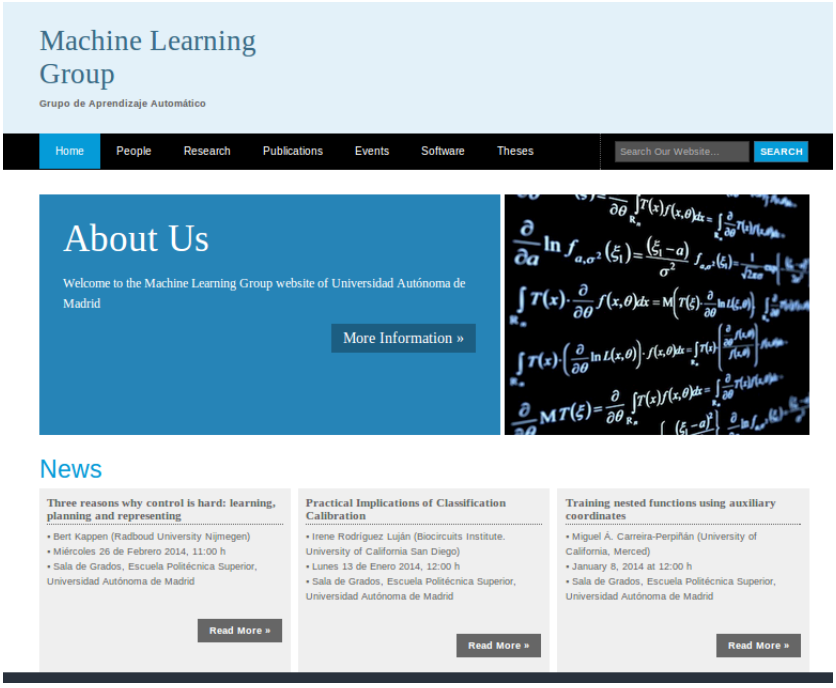

Figura 5.8: Web GAA. Captura de pantalla de la página principal.

Para este sitio, la aplicación devuelve un total de 10 ficheros HTML y la carpeta contenedora de las hojas de estilo seleccionadas.

Su instalación en servidor supone, únicamente, copiar todo el contenido de la carpeta de salida de MetaDepth a la carpeta de destino deseada y ya estará disponible el acceso al sitio web desde la URL correspondiente.

En las figuras 5.8, 5.9 y 5.10 se muestran los resultados de toda la fase de generación del sitio Web para el Grupo de Aprendizaje Automático.

En estas imágenes se puede ver la visualización de los componentes de tipo evento para la página principal (figura 5.8) y como, del mismo objeto, se ha mostrado otra información más detallada en la página de la figura 5.9, cumpliéndose el requisito de no duplicar esta información de manera que no haya que sincronizar constantemente las dos páginas de forma manual.

Se incluye también una captura de pantalla de la sección de publicaciones en la que se puede observar la lista de trabajos con la sublista 2014 desplegada y el resto de años plegados, sin mostrar el contenido.

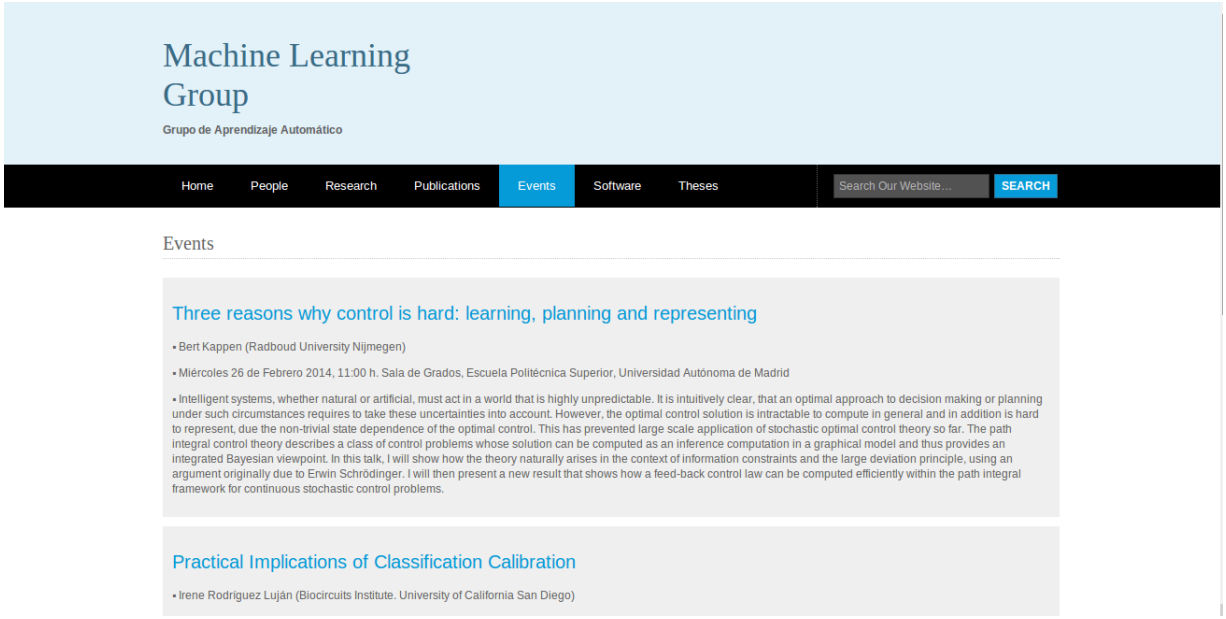

Figura 5.9: Web GAA. Captura de pantalla de la página de eventos.

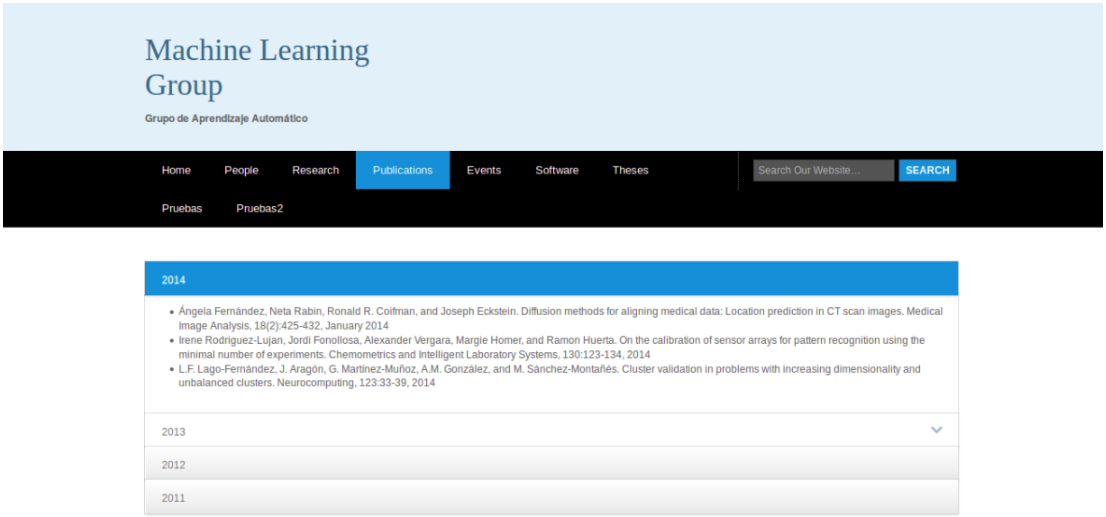

Figura 5.10: Web GAA. Captura de pantalla de la página de publicaciones.

Con esto se cumplen todas las especificaciones que se determinaron durante la fase de análisis junto con el administrador de la página del grupo y se consigue un nuevo sitio Web que permite un mantenimiento más cómodo y que cubre las necesidades del Grupo de Aprendizaje Automático.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

Tras el análisis de diferentes sitios Web de grupos de investigación de numerosas áreas y habiendo encontrado elementos comunes a todos ellos, se ha diseñado, en este proyecto, un generador automatizado de sitios Web contextualizado en este ámbito de páginas para investigadores, con el principal objetivo de facilitar la creación y mantenimiento de estos sitios.

El estudio de diferentes tecnologías que ya aplican el paradigma de diseño dirigido por modelos ha acotado el entorno donde nace este trabajo, que además, incluye el meta-modelado multi-nivel gracias al cual se ha conseguido el diseño de modelos más ajustados a la lógica de negocio que se pretende implementar.

De manera añadida, y como se ha enfatizado a lo largo del documento, esta filosofía ha permitido la reutilización de meta-modelos de carácter más general y su refinamiento para su posterior aplicación a sitios Web de temática determinada.

En el software que se presenta se ha conseguido también, haciendo uso de la generación de código, automatizar diferentes tareas en el proceso de implementación de las páginas Web, ejemplo de ello sería, en el caso de estudio detallado en la sección 5.2, la creación de componentes que permiten generar listados de publicaciones científicas haciendo uso de ficheros Bibtex.

Con otros componentes, tales como ORCIDCollector, se ha conseguido encapsular el laborioso proceso de conexión a APIs externas y el uso de tecnologías y lenguajes muy dispares, haciendo transparente al diseñador toda su implementación y facilitando la utilización de estos servicios de terceros.

Por otro lado, la creación del sitio Web a partir de modelos ha hecho posible la no repetición de código e información en donde antes sí ocurría. Ejemplo de ello han sido los componentes diseñados para eventos en la página Web del Grupo de Aprendizaje Automático.

6.2. Trabajo Futuro

Como se ha demostrado en la sección 5.2, el generador automatizado de sitios Web que se presenta en este documento puede generar, al completo, la página de un grupo de investigación. Sin embargo, hay otra funcionalidad que podría ser interesante poder incorporar por defecto y que no se ha implementado en este proyecto. Por tanto, la primera línea de trabajo para abordar en el futuro sería la ampliación de las librerías de componentes, de manera que se proporcione al desarrollador Web una serie de elementos que permitan implementar todos los requisitos del sitio Web que se pretende diseñar.

Algunos de estos requisitos, que ya se han contemplado, pero que están aun sin finalizar son la modelización del acceso a bases de datos locales, esto es, su creación, edición y mantenimiento desde componentes reutilizables y extensibles a diferentes gestores de bases de datos o la posibilidad de hacer más interactivos los gráficos JavaScript que se muestran en las páginas, permitiendo al usuario final poder elegir qué datos quiere mostrar en la gráfica.

Otro de los aspectos principales a tener en cuenta en la continuación de este proyecto es facilitar la tarea de la fase de modelado, abandonando la interfaz textual y añadiendo un gestor gráfico que permita crear los meta-modelos y modelos de las páginas, consiguiendo así que no sea requerido un perfil técnico para poder diseñarlos y haciendo accesible este software a usuarios que no tengan conocimientos en desarrollo Web.

Continuando con la filosofía de hacer más accesible el generador que se presenta, se plantea la posibilidad de abrir el conjunto de contextos a los que se dirige y proporcionar meta-modelos para su posterior aplicación en otros ámbitos como el de páginas para comercio en línea, sitios para artistas musicales o gráficos, páginas Web corporativas, etc.

Glosario

- **MDE:** Model-driven engineering
- **MDA:** Model-driven architecture
- **MDD:** Model-driven development
- **OMG:** Object Management Group
- **UML:** Unified Modeling Language
- **EGL** Epsilon Generation Language
- **WebML:** Web Modeling Language
- **OOHDMDA:** MDA approach to Object Oriented Hypermedia Design Method
- **UWE:** UML-based web engineering
- **WebDSL:** Web domain-specific language
- **CIM:** Computation-Independent Model
- **PIM:** Platform-Independent Model
- **PSM:** Platform-Specific Model

Bibliografía

[1] Schmid H. A. and Donnerhak O. Oohdmda – an mda approach for oohdm. *Web Engineering*, 2005.

[2] Pons C., Giandini R., and Pérez G. *Desarrollo de Software Dirigido por Modelos. Conceptos teóricos y su aplicación práctica*. McGraw-Hill, 2010.

[3] de Lara J., Guerra E., and Sánchez Cuadrado J. Model-driven engineering with domain-specific meta-modelling languages. *Software and Systems Modeling*, 2013.

[4] de Lara J., Guerra E., Cobos R., and Moreno-Llorena J. Extending deep meta-modelling for practical model-driven engineering. *The Computer Journal*, 2012.

[5] Aragon G., Escalona M. J., Lang M., and Hilera J. R. An analysis of model-driven web engineering methodologies. *International Journal of Innovative Computing, Information and Control*, 2013.

[6] Sommerville I. *Ingeniería del Software*. Pearson Addison Wesley, 2006.

[7] Rose L. M., Paige R. F., Kolovos R. F., and Polack F. The epsilon generation language. *Model Driven Architecture – Foundations and Applications*, 2008.

[8] E. Visser. Webdsl: A case study in domain-specific language engineering. *Generative and Transformational Techniques in Software Engineering II*, 2008.

[9] 1and1. http://www.1and1.es/.

[10] Comparing and merging uml models in ibm rational software architect: Part 10. realign your models after migration or transformation. ibm. http://www.ibm.com/developerworks/rational/tutorials/realign-your-models-after-migration-or-transformation/index.html?ca=dat.

[11] Grupo de aprendizaje automático. arantxa.ii.uam.es/ gaa/.

[12] Grupo de investigación de navegación aérea. http://gina.infra.upm.es/.

[13] Jimdo. http://es.jimdo.com/.

[14] Miso group. http://www.miso.es/.

[15] Uwe -uml-based web engineering. http://uwe.pst.ifi.lmu.de/.

[16] The web modelling language. webml. http://www.webml.org.

[17] Webnode. http://www.webnode.es/.

[18] Wix. http://es.wix.com/.