Universidad de Salamanca

Máster Universitario en Ingeniería Informática

VNiVERSiDAD D SALAMANCA

CAMPUS OF INTERNATIONAL EXCELLENCE

Trabajo de Fin de Máster

APLICACIÓN DE INGENIERÍA DE DOMINIO PARA LA GENERACIÓN DE DASHBOARDS PERSONALIZADOS

Andrea Vázquez Ingelmo

Tutores

Francisco J. García Peñalvo

Roberto Therón Sánchez

Salamanca, 2018

El Dr. D. Francisco José García Peñalvo, Catedrático de Universidad, y el Dr. D. Roberto Therón Sánchez, Profesor Titular de Universidad, del Área de Ciencia de la Computación e Inteligencia Artificial del Departamento de Informática y Automática de la Universidad de Salamanca

CERTIFICAN

Que el trabajo titulado "APLICACIÓN DE INGENIERÍA DE DOMINIO PARA LA GENERACIÓN DE DASHBOARDS PERSONALIZADOS" ha sido realizado por Dª Andrea Vázquez Ingelmo, con DNI 70918831X y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Máster de la titulación Máster Universitario en Ingeniería Informática de esta Universidad.

Salamanca, a 22 de junio de 2018

Dr. D. Francisco José García Peñalvo
Dpto. Informática y Automática
Universidad de Salamanca

Dr. D. Roberto Therón Sánchez
Dpto. Informática y Automática
Universidad de Salamanca

Resumen

Los paneles de información (*dashboards*, en inglés), juegan un papel clave en el proceso de análisis y visualización de datos sobre un tema o dominio específico. En esencia, los *dashboards* muestran información y permiten a los usuarios generar conocimiento y llegar a conclusiones para poder realizar una toma de decisiones con una consistente base informativa. Sin embargo, los usuarios finales pueden presentar una serie significativa de necesidades que difieren entre sí, incluyendo la información mostrada, características de diseño o incluso funcionalidades. Aplicar un enfoque de ingeniería de dominio (dentro del paradigma de las líneas de productos *software*) trae consigo valiosos beneficios, permitiendo producir *dashboards* personalizados y adaptados a los requisitos particulares de cada usuario (o grupo de usuarios) implicado mediante la identificación de similitudes y puntos de variabilidad de cada producto que podría ser parte de la línea. A través de la parametrización de características y la configuración de los componentes de presentación y fuentes de datos, es posible obtener una línea de productos *software* de paneles de control, donde podrán irse variando los diversos componentes que conforman el panel, así como sus funcionalidades o fuentes de datos. La creación de esta línea de productos puede llegar a incrementar la productividad, la mantenibilidad y la trazabilidad en cuanto a la evolución de los requisitos de los *dashboards*, junto a otros beneficios. Para validar esta aplicación, se ha realizado un caso de estudio en el contexto del Observatorio de Empleabilidad y Empleo Universitarios, donde los usuarios (universidades españolas y administradores), podrán controlar sus propios *dashboards* para explorar datos sobre la empleabilidad y el empleo de sus graduados. Dichos *dashboards* serán generados automáticamente a través de un lenguaje específico de dominio (DSL), donde se podrán especificar los requisitos de cada usuario, y un generador de código basado en plantillas.

Palabras clave: Líneas de productos software; *Dashboards*; Paneles de control; Generación de código; Visualización de datos; Sistemas de información.

Abstract

Information dashboards play a key role in analyzing and visualizing data about a specific topic or domain. In essence, dashboards display information and allow users to reach insights and make informed decisions during decision-making processes. However, end users can present several necessities that differ from each other, including the displayed information itself, other design features and even functionalities. That is why applying a domain engineering approach (within the software product line paradigm) is beneficial in this domain, in order to produce customized dashboards adapted to particular requirements of every involved user (or group of users) by the identification of commonalities and singularities of every product that could be part of the software product line. By the parameterization of features and the configuration of presentation components and data sources, it is possible to obtain a software product line of information dashboards where the different components can be combined to build a particular dashboard. The creation of a product line would increase productivity, maintainability and traceability regarding the evolution of the dashboards' requirements, among other benefits. To validate this approach, a case study of its application in the context of the Spanish Observatory of Employability and University Employment ecosystem has been executed, where users (Spanish universities and administrators) will control their own dashboards to reach insights about the employability of their graduates. These dashboards will be automatically generated through a domain specific language (DSL), which will provide the syntax to specify the requirements of every user, and through a template-based code generator.

Keywords: Software product lines; Dashboards; Code generation; Data Visualization; Information systems.

Tabla de contenido

Índice de figuras

1. Introducción

La incorporación de los sistemas de información en gran diversidad de actividades como herramientas de soporte y automatización de tareas ha provocado un incremento exponencial del volumen de datos generados, jugando estos un papel central en infinitud de tareas, sean estas triviales o complejas. A su vez, debido a la propia evolución de las tecnologías, este incremento es continuo tanto en volumen como en complejidad, provocando que se hagan necesarias herramientas más potentes para poder administrar los datos recolectados. Sin embargo, la recolección y almacenamiento en sí mismos de grandes volúmenes de datos no reporta beneficios; los datos realmente adquieren valor cuando son analizados y explotados con el objetivo de extraer conocimiento de los mismos [1]. En los últimos años esta tendencia ha ganado gran importancia en diversos sectores [2, 3] con la visión de lograr que la toma de decisiones estratégicas esté arropada por una consistente base informativa [4], y provocando que los sistemas dirigidos por datos (*data-driven systems*, en inglés) se hayan extendido para ofrecer una gestión eficiente y eficaz de los mismos [2].

Así pues, para obtener información útil, es necesario explotar los datos en bruto a través del análisis y el cálculo de métricas. La información permite responder preguntas sobre lo que está sucediendo (por ejemplo: "hay un decremento de visitas a una determinada web"). Gracias a esa información podrá generarse posteriormente conocimiento a través de la asociación e identificación de patrones en dichas métricas, lo cual podrá ayudar a la toma de decisiones y a responder a una pregunta más compleja: "cómo actuar" [5].

Es importante, sin embargo, tener en cuenta que en el proceso de toma de decisiones pueden intervenir actores con diversos perfiles, encontrando un contexto en el que se pueden llegar a generar brechas de comunicación entre ellos, al no poseer los mismos niveles de conocimientos específicos en sus campos de actividad [6]. Para evitar dichas brechas y conseguir que la toma de decisiones sea un proceso eficiente y no problemático, es de vital importancia que los individuos implicados en el proceso se encuentren en el "mismo nivel", en referencia al hecho de que todos los sujetos deben comprender y tener claras las implicaciones que tienen los datos recolectados y los resultados derivados del análisis de los mismos. Pero alcanzar dicha situación no es una tarea trivial; las variables influyentes en la toma de decisiones pueden llegar a ser muy complejas y necesitar de soporte adicional que facilite la aprehensión total de dichos resultados por parte de todos los perfiles involucrados.

Uno de los productos *software* más empleados para la extracción de conocimiento y explotación de conjuntos de datos lo constituyen los cuadros de mandos o paneles de información (también conocidos como *dashboards*) [7], los cuales suelen componerse de una serie de recursos gráficos [8, 9] (generalmente interactivos) que muestran métricas obtenidas tras analizar los datos, con el objetivo de mostrar la información de una forma más comprensible [10] y permitir la identificación de patrones o indicadores relevantes para la toma de decisiones [11], obteniendo nuevas perspectivas sobre la información mostrada. Los *dashboards* facilitan a sus usuarios la tarea de establecer relaciones entre los datos, permitiéndoles aplicar su experiencia para extraer conocimiento de la situación y transmitir los resultados de forma comprensible de tal forma que los encargados de la toma de decisiones actúen en función de los mismos [6]. A través de la analítica visual [12, 13], es posible combinar la visualización de datos con su propio análisis, pudiendo razonar y comprender diversos aspectos de conjuntos de datos pertenecientes a cualquier dominio [14-19].

Es necesario tener en cuenta que, debido a la continua evolución de estos datos, a su constante crecimiento en cuanto a cantidad y a su incremento en complejidad (jerarquías o anidamientos profundos en los datos, relaciones complejas entre entidades y atributos, etc.), las herramientas que tratan con ellos deben ser versátiles y adaptarse a estos cambios de manera ágil, de lo contrario el mantenimiento de las mismas puede convertirse en una tarea costosa. Con el paso del tiempo pueden ser necesarias nuevas métricas o técnicas de visualización (o cualquier otra nueva funcionalidad necesaria en las visualizaciones ya implementadas) para explorar y analizar nuevas dimensiones informativas, siendo necesario reaccionar y modificar los requisitos de los paneles de información. La introducción de los nuevos requisitos debe ser relativamente sencilla y tomar efecto rápidamente en el sistema, con el fin de evitar la ralentización del proceso de obtención de conocimiento, la toma de decisiones y, en consecuencia, el alcance de los objetivos establecidos.

Añadido a este problema, se tiene que la variedad de requisitos que puede llegar a tener cada perfil de usuario supone un reto en cuanto al diseño e implementación de paneles de información, de forma que tenerlos todos en cuenta puede llegar a ser inviable o consumir demasiados recursos. A su vez, cabe la posibilidad de que ciertos requisitos sean excluyentes entre sí, existiendo un conflicto a la hora de introducirlos en el diseño. Es evidente que, aunque sea una solución para este dilema, el diseño e implementación de un *dashboard* específico para cada perfil contemplado implicaría un gran consumo de recursos, tanto durante el desarrollo de los mismos como en los procesos de mantenimiento.

Es por ello que los *dashboards* (además de ofrecer un diseño visual e interactivo adecuado para los usuarios objetivo) han de ser flexibles, mantenibles y escalables en cuanto a su contenido y su propio diseño, de manera que puedan evolucionar de forma fluida a lo largo del tiempo. Ciertos paradigmas propios de la ingeniería del *software* [20] ofrecen soluciones viables para abordar esta variedad de requisitos de forma flexible. Campos de estudio como las familias o líneas de productos *software* (*Software Product Lines, SPLs*) [21, 22] o la arquitectura dirigida por modelos (*Model Driven Architecture, MDA*) [23] constituyen soluciones potencialmente aplicables y beneficiosas para el tipo de producto *software* en el que se enmarcan los paneles de control.

Concretamente, el enfoque de las familias *software* (en conjunción con el desarrollo dirigido por modelos) constituye una buena estrategia para plantear la generación automática de *dashboards* versátiles por una serie de razones; la ingeniería de dominio analiza los puntos comunes y diferencias que existen entre los productos de un ámbito específico [22]. De esta forma es posible identificar las partes del sistema genérico que necesitarán ser configurables para adaptarse a los requisitos de un producto específico de la familia [24]. En conclusión, las líneas de productos *software* promueven un marco de trabajo donde la reutilización y configuración de componentes a través de puntos de variabilidad son la clave de sus potenciales beneficios.

La implementación de los productos específicos, una vez identificados los componentes comunes y las características variables, se realiza a través de la composición y configuración de activos *software* (*core assets*) previamente implementados para ser reutilizados [25].

La elaboración de un repositorio de *core assets* establece una base de componentes para un desarrollo de productos finales de forma sistemática y eficiente, al basarse en configuraciones válidas del modelo de características, pudiendo añadir nuevos activos (o modificar los activos existentes) al repositorio y asegurando una evolución fluida de dichos productos.

Los paneles de control pueden considerarse como una combinación de componentes visuales con diversas funcionalidades y requisitos dependiendo del usuario que lo utilizará, la fuente de datos que alimenta los recursos gráficos del panel o cualquier otra característica concreta. El problema de crear *dashboards* flexibles para una cantidad significativa de usuarios puede abordarse mediante la creación de una familia de productos, donde se introducirán puntos de variabilidad que permitirán adaptar los productos finales a los requisitos y necesidades de los usuarios objetivo.

A este problema se enfrenta el sistema que da soporte al Observatorio de Empleabilidad y Empleo Universitarios, en cuyo proyecto se contextualiza este Trabajo de Fin de Máster, debido a, como se verá en las próximas secciones, la variedad de consumidores de sus datos y servicios, así como a la continuidad de sus estudios

El resto de esta memoria está estructurada de la siguiente forma: en la sección 2 se describe la motivación de la aplicación de ingeniería de dominio para la generación de *dashboards* en el contexto del Observatorio de Empleabilidad y Empleo Universitarios. En la sección 3 se detallan los materiales utilizados y la metodología seguida para la implementación de la línea de productos de *dashboards*. La sección 4 describe el estado del arte de la ingeniería de dominio y las prácticas utilizadas para la implementación de las mismas, así como la aplicación de dicho paradigma sobre *dashboards*. En la sección 5 se detalla la propuesta de arquitectura de la línea de productos, seguida de los resultados obtenidos (sección 6) y la discusión de los mismos (sección 7). Para cerrar la memoria, en la sección 8 se presentan las conclusiones alcanzas y las líneas de investigación y trabajo futuras.

2. Contexto y motivación

Como se ha introducido en la anterior sección, la aplicación de ingeniería de dominio para crear una línea de productos de paneles de control es un enfoque potencialmente beneficioso debido a las características comunes que pueden presentar los *dashboards* en un determinado contexto. La motivación principal de este trabajo surge en el contexto del Observatorio de Empleabilidad y Empleo Universitarios (OEEU, https://oeeu.org/). Como se explicará en las siguientes secciones, esta organización se enfrenta a una serie de retos en cuanto a la gestión de los datos que recolecta y analiza.

2.1. El Observatorio de Empleabilidad y Empleo Universitarios

El Observatorio de Empleabilidad y Empleo Universitarios (OEEU) se constituye como una organización compuesta por investigadores y técnicos que trabajan conjuntamente desde diversas partes de España con una metodología unificada, con el objetivo de producir, analizar y difundir información sobre la empleabilidad y el empleo de los titulados y las tituladas universitarias en España. El Observatorio está bajo la dirección de la Cátedra UNESCO de Gestión y Política Universitaria, y cuenta con el asesoramiento de un Consejo de Expertos, formado por académicos y expertos universitarios nacionales e internacionales.

Debido a su misión y, en especial, a su visión de convertirse en una referencia para entender las variables que afectan a la empleabilidad y empleo universitarios, el Observatorio debe contar con herramientas útiles para los consumidores de sus servicios (universidades españolas, usuarios generales, administradores, etc.), de manera que puedan explotarse los datos recolectados y sus bancos de conocimiento para obtener resultados que faciliten la comprensión y el efecto de dichas variables.

Para alcanzar los anteriores objetivos, el Observatorio desarrolla, implementa y explota una serie de productos dirigidos por datos (*data-driven*) [2, 3]. El enfoque *data-driven* ayuda al Observatorio a obtener conocimiento de los datos recolectados, puesto que todos sus productos necesitan de datos para ser desarrollados. Uno de los principales productos del Observatorio son los barómetros de empleabilidad y empleo universitarios, los cuales cuentan actualmente con dos ediciones: el Barómetro de empleabilidad y empleo de los universitarios en España, 2015 (Primer informe de resultados) [26] y el Barómetro de empleabilidad y empleo universitarios (Edición Máster 2017) [27].

Las actividades del OEEU están soportadas por un ecosistema tecnológico que permite recolectar, procesar, analizar y diseminar datos sobre empleabilidad y empleo [28] mediante diversos componentes conectados a través de flujos de datos. Con este enfoque, el Observatorio es capaz de ofrecer un mecanismo para que las universidades envíen datos administrativos de sus estudiantes, así como un sistema de cuestionarios sobre empleabilidad y empleo para recolectar las respuestas de dichos estudiantes y analizarlas posteriormente.

A través de estos mecanismos de recogida de información se obtiene una serie de variables sobre la empleabilidad de los titulados y la vinculación entre la formación universitaria que recibieron y sus empleos (en caso de que obtuviesen algún empleo). Estas variables pueden ser demográficas, relacionadas con la satisfacción con los estudios cursados, relacionadas con la cualificación en los empleos, salarios, criterios de selección de empleo, métodos de búsqueda de empleo, etc.

En total, para el primer Barómetro de empleabilidad y empleo de los universitarios en España [26] realizado en 2015, se recogieron más de 500 variables de 13.006 estudiantes de grado que terminaron sus estudios en el año 2009/2010. En el caso del Barómetro de empleabilidad y empleo universitarios (Edición Máster 2017) [27], se recogieron más de 370 variables de 6.738 estudiantes de máster que terminaron sus estudios en el año 2013/2014.

Una vez analizados los datos recogidos, se obtiene un conjunto de resultados a diseminar. Estos resultados ofrecen una serie de perspectivas en cuanto a la empleabilidad y empleo de los egresados, lo que permite percibir la situación general de estos dos campos en España.

Sin embargo, debido a la cantidad de datos recolectados, la tarea de diseminar los resultados globales de manera comprensible supone un reto. Añadido a esto, se encuentra el hecho de que las propias universidades que participan en los estudios son usuarios principales del sistema, y están interesadas en consultar los resultados particulares cosechados a partir de las respuestas de sus propios estudiantes.

2.2. El problema de la visualización de datos

El sistema del Observatorio ofrece un portal público donde los resultados generales pueden consultarse a través de una serie de visualizaciones interactivas. A este portal tiene acceso cualquier persona interesada en el estudio (https://datos.oeeu.org/).

Figura 1. Portal público del OEEU para la Edición Máster 2017 (https://datos.oeeu.org)

Cada categoría de resultados muestra una serie de estadísticas obtenidas a partir de las respuestas de los egresados a los cuestionarios diseñados por el Observatorio, para obtener una visión general de las principales variables recogidas en una categoría de resultados concreta.

Los informes [26, 27] y las visualizaciones del portal público permiten la diseminación de los resultados de los estudios de una forma gráfica. No obstante, estas visualizaciones, aunque ofrecen una visión general, no ofrecen patrones de interacción avanzados para explorar de manera más profunda los resultados.

Por otra parte, el ecosistema del OEEU ofrece a las universidades una intranet para consultar los resultados obtenidos de sus propios estudiantes. En la primera edición del barómetro, a las universidades se las proveía de un panel de control básico, con un pequeño conjunto de estadísticas sobre la participación de los estudiantes en el cuestionario, así como la opción de descarga de dichos datos en bruto (Figura 3).

Estos paneles fueron mejorados replicando el portal público para cada universidad, con la diferencia de que las visualizaciones muestran exclusivamente los datos de la universidad en cuestión, en vez de los datos agregados de todos los egresados participantes (Figura 4). Esto es, fue añadida una capa básica de personalización en los *dashboards* a nivel de datos, puesto que solo se muestran los de la universidad en concreto, pero en caso de necesitar añadir una nueva capa personalizada a nivel de diseño, estructura y/o funcionalidad, habría que implementarlos de forma específica para cada universidad.

Figura 2. Ejemplo de una de las categorías de los resultados obtenidos (https://datos.oeeu.org/metodos-aprendizaje)

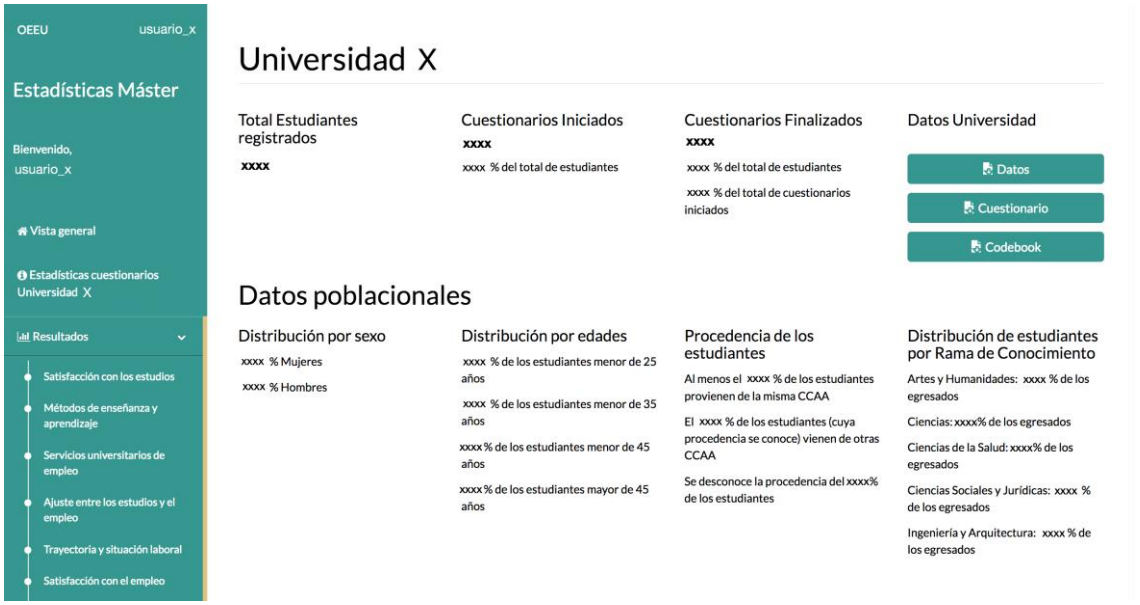

Figura 3. Ejemplo de los datos que pueden consultar las universidades a través de la intranet del Observatorio

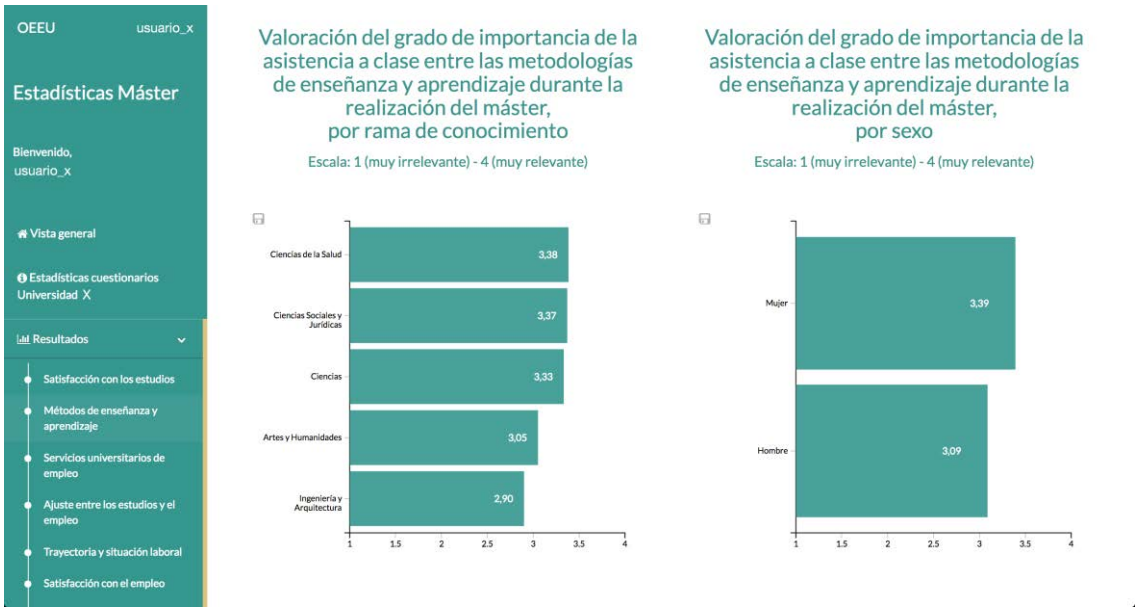

Figura 4. Replicación de las visualizaciones del portal público adaptadas para mostrar los datos concretos de cada universidad

2.3. Motivación

Debido a la gran cantidad de categorías de resultados y variables recolectadas, la extracción de conocimiento de los datos se convierte en una tarea complicada. Ciertas universidades pueden estar más interesadas en categorías de resultados o variables concretas, siendo más irrelevantes otras dimensiones de los datos para sus requisitos de información particulares.

Esta cuestión puede resolverse mediante la personalización de los paneles de control de cada universidad según sus requisitos concretos. Sin embargo, la personalización de los *dashboards* de cada universidad se convierte en un reto debido al significativo número de participantes en el estudio. Diseñar e implementar un *dashboard* específico y personalizado para cada una de ellas es una potencial solución para un sistema con pocos usuarios y requisitos muy estáticos. Sin embargo, el

Observatorio cuenta con una cantidad significativa de usuarios que pueden requerir diversas funcionalidades o necesitar restricciones, además de que sus estudios tienen continuidad en el tiempo, con lo cual, el enfoque de crear *dashboards* hechos a medida para cada universidad conllevaría un gran consumo de recursos y, especialmente, de tiempo.

Añadido a este hecho principal, se obtendría una serie de *dashboards* a mantener de forma individual a lo largo del tiempo; ya no solo por la continuidad de los estudios del Observatorio, sino porque los requisitos de cada universidad pueden ser dinámicos, necesitando añadir, modificar o eliminar funcionalidades y/o elementos de sus paneles de control.

Los *dashboards*, aunque diferentes en cuanto a requisitos, cuentan con una base común y una serie de componentes que podrían ser reutilizados para diferentes universidades, en el caso del Observatorio. El enfoque idóneo se basaría en contar con una serie de elementos reutilizables y configurables para componer los paneles de control de cada usuario en función de sus requisitos. Esto no solo supondría un mantenimiento mucho más sencillo de los paneles de control, sino un marco de trabajo que permitiría la producción automatizada de *dashboards* escalables en cuanto a usuarios, funcionalidades y datos, propagando las actualizaciones y los ajustes realizados en cada componente individual a lo largo de todos los *dashboards* que los utilizan.

Como se ha introducido anteriormente y como se verá en los siguientes apartados, dicho enfoque es potencialmente abordable a través de la aplicación del paradigma de las líneas o familias de productos *software*.

2.4. Objetivos

El objetivo principal de este trabajo es aportar al Observatorio de Empleabilidad y Empleo Universitarios una forma sistemática de producir *dashboards* personalizados para conseguir afrontar de manera eficiente el dinamismo y variabilidad en los requisitos de los usuarios del ecosistema del Observatorio.

Para ello, se plantean los siguientes sub-objetivos:

- Identificación los requisitos potenciales y/o existentes de los usuarios de los *dashboards* (en este caso concreto, estos usuarios serán las universidades participantes en los estudios del Observatorio).
- Modelado de los paneles de control.
- Realización del *feature model* de la línea de productos.
- Implementación de los *core assets* de la línea de productos.
- Implementación los mecanismos para automatizar la configuración de productos específicos de la línea a través de un DSL (*domain specific language*).

Mediante estos objetivos secundarios se pretende obtener un mecanismo para mantener una serie de *dashboards* de forma adaptativa y escalable.

3. Líneas de productos software

En este apartado se describen las aportaciones que se han realizado en los principales temas que este proyecto abarca con el objetivo de contextualizarlo teóricamente. Para ello, se realiza una breve descripción tanto de las líneas de productos software como de la posibilidad de generar su código de manera automática. Además, se muestran aportaciones existentes en lo que a la aplicación del paradigma de líneas de productos *software* a la generación de paneles de control e interfaces de usuario se refiere.

3.1. Líneas de productos software (SPL)

Las líneas de productos *software* componen uno de los paradigmas de reutilización sistemática de código más extendidos y aplicados en la práctica. Otros paradigmas como el desarrollo dirigido por modelos (MDD, *Model Driven Development*) aún no cuentan con la suficiente robustez para ser llevados a la práctica de forma efectiva [23], aunque en un futuro podría suponer un gran avance en lo que al desarrollo *software* se refiere.

La viabilidad de las líneas de productos *software* coloca esta metodología como una potente herramienta para abordar el desarrollo de *software* personalizado de manera masiva, a través del análisis de similitudes y puntos de variabilidad en la denominada fase de ingeniería de dominio [22]. La fase de ingeniería de dominio es clave para la creación de líneas de productos *software*, en ella se abstraen las principales características (meta-modelado) que tendrán los productos pertenecientes a la familia y se formalizarán los *core assets* (componentes) necesarios para su realización.

Es en la fase de ingeniería de dominio donde se definen los denominados puntos de variabilidad, los cuales, como su nombre indica, son la forma de introducir, modificar o adaptar ciertas funcionalidades en función de los requisitos [22]. Gracias a la aplicación práctica de este paradigma, aparecen una serie de métodos para abordar el modelado de los puntos de variabilidad [29, 30].

Uno de los mecanismos más extendidos es el método FODA (*Feature-Oriented Domain Analysis*) [31], donde se identifican las diversas características (propiedades visibles) de los productos que formarán parte de la línea. Los *feature models* se introducen por primera vez en [31] como mecanismo formal de descripción de dichas características. A través de estos diagramas jerárquicos, se especificarían las características obligatorias, alternativas, opcionales, etc., de la línea de productos (esto es, sus puntos de variabilidad). Un ejemplo de la notación y uso de estos diagramas pueden verse en la
Figura 5.

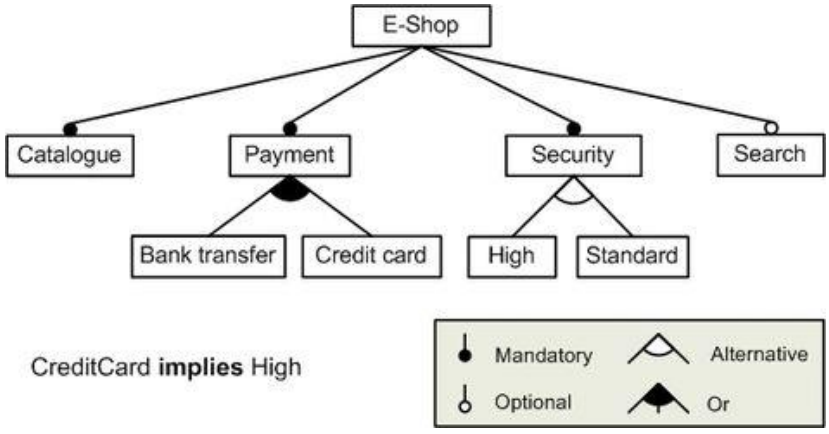

Figura 5. Ejemplo de un modelo de características para una tienda online (Fuente: Wikipedia)

Debido a su amplio uso, estos diagramas han sido extendidos en varias ocasiones para soportar nuevas dimensiones y aportar información más amplia sobre el dominio del problema. Los *Hyper Feature Models*, por ejemplo, tratan de capturar la variabilidad a lo largo del tiempo, esto es, las diversas versiones a las que se ha visto sometida la línea de productos en cuanto a sus características [32]. Por otro lado, los diagramas de características también han sido extendidos para soportar la especificación de multiplicidades, dado que es posible que una línea de productos pueda contar con diferentes instanciaciones de la misma característica [33].

Una vez completada la fase de ingeniería de dominio, se continuaría con la fase de ingeniería de aplicación, donde, haciendo uso del meta-modelo de la familia de productos obtenido en la fase de ingeniería de dominio, se crearán instancias concretas del mismo, seleccionando las características finales que conformarán el producto específico a construir [21]. En resumen, del meta-modelo se derivan modelos concretos asociados a productos finales de la SPL, lo que permite la trazabilidad de las características y requisitos.

En función de la configuración descrita en los modelos instanciados, se seleccionarían, configurarían y compondrían los componentes (*core assets*) necesarios para implementar el producto final. Es en esta fase donde se demuestra la potencia del paradigma de las líneas de productos *software*: los *core assets* ya habrían sido implementados y preparados para ser configurables en la fase de ingeniería de dominio, con lo cual solo sería necesario adaptarlos a los requisitos específicos.

3.2. Generación automática de código en SPL

A nivel de modelado, como se ha descrito, existen soluciones muy variadas. Sin embargo, uno de los potenciales beneficios de las SPL es la posibilidad de generar productos de la familia de forma automática. Este enfoque podría verse como una combinación del paradigma de las líneas de productos y el desarrollo dirigido por modelos [34].

Gracias a la división de los productos en componentes configurables y características bien definidas, es posible crear generadores de código que, a través de los modelos, sean capaces de seleccionar y configurar los *core assets* correspondientes para finalmente obtener el código fuente del producto final.

Para ello, es necesario materializar los puntos de variabilidad especificados en los modelos en el propio código fuente de los *core assets*; de esta forma, el generador de código solo tendría que encargarse de inyectar los parámetros de configuración en los componentes seleccionados para configurarlos acorde a los requisitos de los productos específicos. Para materializar los puntos de variabilidad en el código fuente se puede hacer uso de diversas técnicas [24], por ejemplo:

- Delegación de funcionalidades.
- Herencia.
- Parametrización.
- Sobrecarga de métodos o funciones.
- Bibliotecas de enlace dinámico (DLL).
- Compilación condicional.

Otra técnica muy extendida para la generación de código son las plantillas [35]. Las plantillas de código permiten definir partes estáticas (las cuales se corresponderían con las características base u obligatorias de la SPL) y partes dinámicas donde puede inyectarse código en función de una serie de reglas (características opcionales o alternativas y restricciones) [36, 37].

Combinando la especificación formal de la línea de productos con las técnicas de materialización de puntos de variabilidad, sería posible implementar generadores de código que, haciendo uso del modelo concreto de un miembro de la familia de productos, sean capaces de inyectar o seleccionar las funcionalidades o características requeridas en los *core assets* para obtener el código fuente del producto final.

3.3. Aplicaciones prácticas

Las líneas de productos han sido y son aplicadas en campos muy diversos, con el objetivo de ofrecer productos personalizados de forma masiva [22]. En el campo del *software* su aplicación ha supuesto la posibilidad de reutilizar activos *software* de forma sistemática para adaptarse a los requisitos de los clientes, disminuyendo el tiempo de desarrollo y aumentando la mantenibilidad de los productos de la familia [21].

Muchos dominios se han visto beneficiados por la aplicación del paradigma de las líneas de producto *software*:

- El dominio de las aplicaciones móviles, debido a la variedad de plataformas que pueden llegar a utilizar estos dispositivos .
- *Dashboards* adaptables a los requisitos de los usuarios [38].
- Generación de documentos con contenido variable [39].
- Sistemas de *eLearning* [40].

Uno de los retos a los que se enfrenta el paradigma de las líneas de productos aplicado al *software* es la generación de interfaces gráficas de usuario. Varios autores han analizado y tratado este problema en la literatura, al ser esencial de cara a los usuarios [41-44]. Si bien la funcionalidad en un producto *software* es clave, también es necesario que sea usable [45, 46] y tenga en cuenta los requisitos de los usuarios a nivel de interfaz, dado que es el elemento del producto al que están expuestos y con el que se comunican una vez realizado el producto.

4. Materiales y Metodologías utilizadas

Se va a aplicar el paradigma de ingeniería de dominio (SPL) para crear un DSL (*domain specific language*) propio que, a través de un generador de código basado en plantillas, se utilizará para configurar los *dashboards* específicos de la línea de productos y generar su código de forma automática gracias a los *core assets* previamente implementados con sus correspondientes puntos de variabilidad.

Para ello ha sido necesario contar con materiales y metodologías que ayudasen a resolver la serie de retos a abordar en lo que a la generación automática de *dashboards* se refiere:

- Diseño e implementación del DSL.
- Diseño de los componentes del *dashboard* (*core assets*).
- Implementación de los *core assets*.
- Introducción e implementación de los puntos de variabilidad de la SPL.
- Composición del *dashboard* a través de los *core assets* configurados de forma automática.
- Conexión del *dashboard* con los datos del Observatorio.

A lo largo de esta sección se describen las técnicas y materiales utilizados para alcanzar el objetivo final de este proyecto.

4.1. Modelado

Para especificar el dominio del problema a alto nivel se ha diseñado un meta-modelo. El meta-modelado permite separar la especificación de las operaciones, datos y requisitos de una solución de la plataforma en la que será finalmente implementada. A partir del mismo, que sirve como referencia para la línea de productos *dashboard* del Observatorio, se instancian los modelos específicos de *dashboard* [47]. El meta-modelo se utiliza para especificar el dominio a alto nivel de abstracción, pudiendo especificar también, a raíz del mismo, líneas de productos para dominios más específicos.

Para generar el meta-modelo se ha utilizado la herramienta EMF (*Eclipse Modelling Framework*, https://www.eclipse.org/modeling/emf/).

En cuanto a las SPL, existen diversas técnicas para analizar y modelar las características y similitudes entre los productos de la familia a desarrollar, como ya se comentó en la sección 3.1. El método utilizado para este proyecto han sido los modelos de características o *feature models*, dado que están muy extendidos en el ámbito de las SPL. Además, la sencillez y legibilidad que aportan gracias a su

estructura jerárquica son aspectos a considerar de cara a la puesta en práctica de los modelos en la implementación de productos específicos.

Debido a la "naturaleza" de los *dashboards*, donde podrían encontrarse diversas instancias de un mismo componente con distintas configuraciones (por ejemplo, un *dashboard* con dos diagramas de dispersión que consumen de fuentes de datos diferentes), se torna necesario poder expresar la multiplicidad de las características en los *feature models*. Originalmente, la multiplicidad no formaba parte de este tipo de modelos, pero debido a las aplicaciones que fueron surgiendo, aparecen extensiones donde se posibilita la especificación del número de ocurrencias de cierta características [33].

Así pues, el modelo que se utilizará en este proyecto para especificar las similitudes y posibles variaciones en la línea de productos será el *feature model*.

La herramienta utilizada para modelar este tipo de diagramas ha sido la aplicación web *Glencoe* (https://glencoe.hochschule-trier.de/) debido a sus funcionalidades: análisis de la línea de productos, variedad de visualizaciones del árbol de características, requisitos y exclusiones entre características, exportación en formatos estructurados, etc., esto es, las características básicas necesarias para el contexto de este proyecto.

4.2. *Domain specific language* (DSL)

A la hora de definir la línea de productos, es necesario establecer el método o formato utilizado para especificar las características y posibilidades de configuración. En el contexto de este trabajo, al centrarse en la generación automática de productos específicos, se vuelve prioritario buscar un formato estructurado que pueda procesarse y sirva de nexo entre los *feature models* y la implementación final. El método de especificación de la línea de productos debe soportar las posibles restricciones que puedan encontrarse en el proceso de modelado de las características. Una vez especificada la línea de productos, se ha obtenido un lenguaje de dominio (DSL) que ha permitido representar posibles configuraciones de los productos, convirtiéndose en la columna vertebral del generador de código.

Se ha elegido la tecnología XML como método para representar las restricciones y características modeladas a través el *feature model*. La tecnología XML permite especificar, de forma estructurada, tanto las características anteriormente modeladas como aquellas que no tenían cabida en los *feature models* (como es el caso del *layout* o estructura visual del *dashboard*, como se verá en las secciones siguientes). A su vez, las reglas sintácticas o estructurales de los ficheros XML pueden definirse a través de un esquema XML (*XML Schema Definition*, XSD [48] en su versión 1.0, dado que, por razones de compatibilidad y soporte, no es posible utilizar la versión 1.1), lo que proporciona una capa de validación a la hora de configurar productos específicos, al permitirse únicamente la estructura, elementos y valores previamente especificados en el esquema. Tanto los expertos en el dominio como el propio generador de código se benefician del uso de tecnología XML, debido a que los ficheros de configuración son más cercanos al dominio y el tratamiento y extracción de características es directo y eficiente, respectivamente.

Por último, tanto los *feature models* como los ficheros XML comparten la propiedad de tener una estructura jerárquica, con lo que se simplifica la correspondencia entre ambos paradigmas, aumentando la trazabilidad de las características representadas en ambos aun estando a distintos niveles de abstracción.

En esencia, el DSL textual para generar los *dashboards* se ha implementado mediante XML y XSD al proporcionar una metodología robusta, relativamente sencilla y ya utilizada previamente en la configuración automática de SPL [49].

4.3. Implementación de la SPL

La implementación de la línea de productos es uno de los puntos clave en este proyecto. Si bien la especificación de la misma es esencial, es necesario que ésta pueda ser llevada a la práctica de manera viable. El hecho de introducir automatización en la generación de productos específicos de la línea añade complejidad a este proceso, aunque existen diversas técnicas para gestionar la "materialización" de los puntos de variación modelados en el código final de los *core assets*.

A lo largo de esta sección se describen los diversos enfoques, mecanismos y bibliotecas utilizadas para hacer posible la generación de *dashboards* personalizados de manera automática.

4.3.1. Sistema base

Es importante recalcar que el sistema en el que se quiere implantar el generador de *dashboards* personalizados está basado en Django [50], un *framework* web para Python. Debido a que la línea de productos se ha integrado en dicho sistema, ha sido necesario tener en cuenta la estructura y materiales utilizados previamente en el sistema base [28].

Como puede observarse en la Figura 6, el ecosistema del OEEU está conformado por un colector de datos, un analizador, un componente de almacenamiento permanente y una serie de componentes de presentación; todos ellos haciendo uso de flujos de datos para comunicarse entre sí, a través de un componente de interoperabilidad que abstrae las características de cada componente [28].

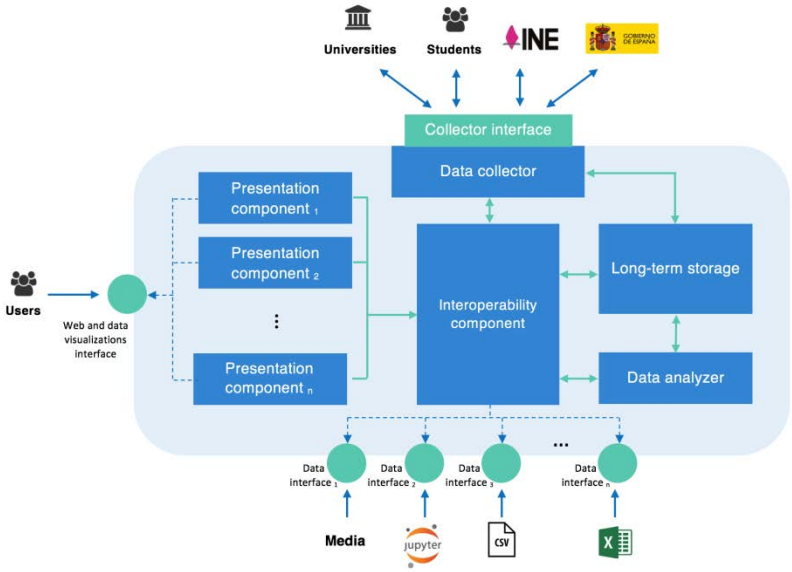

Figura 6. Esquema del ecosistema tecnológico del Observatorio de Empleabilidad y Empleo Universitario

En la Figura 7 se observa una vista más técnica del ecosistema tecnológico, mostrando las tecnologías utilizadas en cada componente.

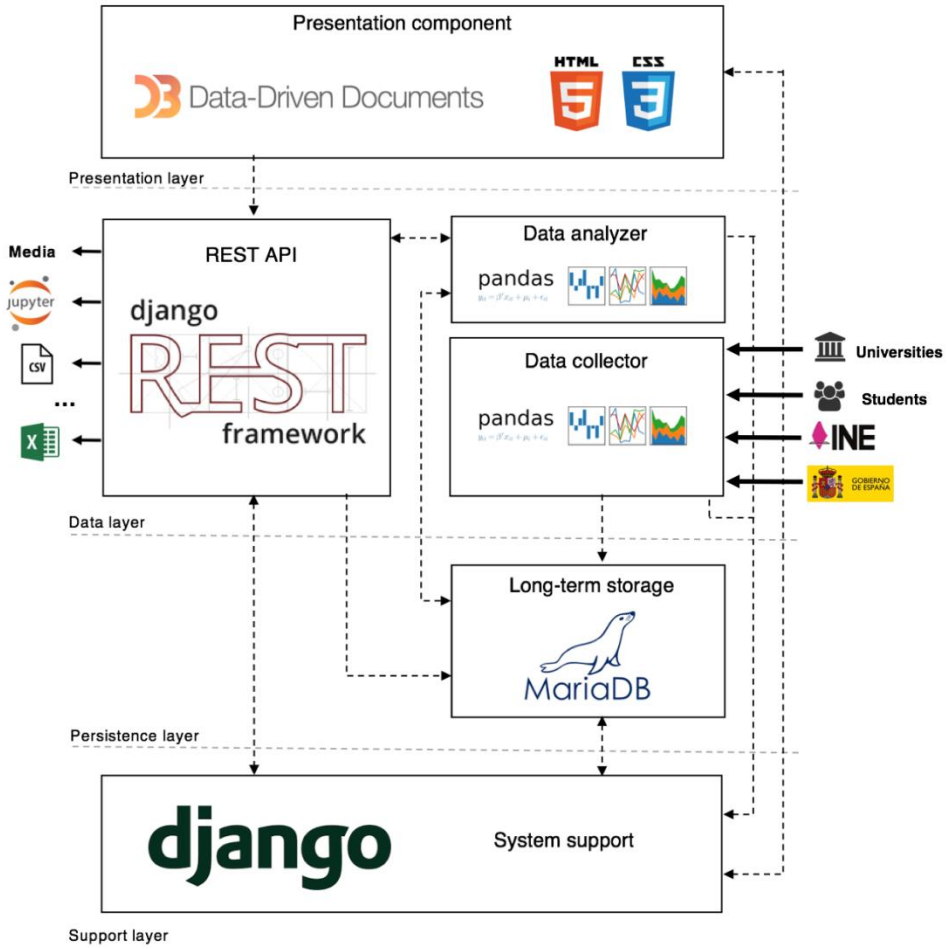

Figura 7. Tecnologías utilizadas en el ecosistema del OEEU

Para implementar los *dashboards*, se ha hecho uso de dos componentes principales: los componentes de presentación y el componente de interoperabilidad. Los primeros son los encargados de mostrar a los usuarios los recursos gráficos que componen el *dashboard*, mientras que, el segundo, se encarga de generar flujos de datos con el analizador para que los componentes de presentación consuman y muestren información del banco de conocimiento del Observatorio. Para este proyecto, el componente de interoperabilidad utilizado se ha basado en GraphQL en vez de en REST, como se comentará en la sección 4.3.3.

En esencia, el contexto de este trabajo se enmarca en la generación del código JavaScript (haciendo uso de la biblioteca D3.js en su versión 4 para la creación de visualizaciones [51]) y HTML de los componentes de presentación que dan soporte a los *dashboards*.

4.3.2. Generación de código

Como ya se avanzó en la sección 3.2, existen diversas formas de inyectar variabilidad en los componentes base o activos *software* de la línea de productos. Para este proyecto se ha decidido utilizar plantillas como método de generación de código una serie de razones:

- Es un método viable para las líneas de productos software [35, 37].

- Es un método ampliamente utilizado en el desarrollo web [52-54].
- Existen soluciones integrables en el sistema base (Django) [50, 55], lo cual evitaría contar con aplicaciones independientes a la plataforma para generar el código.
- Permiten la introducción de sentencias condicionales y bucles, haciendo sencilla la materialización de los puntos de variabilidad a partir de las reglas indicadas en los modelos.

El generador de código toma la configuración del *dashboard* previamente especificada a través del DSL diseñado para el propósito de este trabajo, e inyecta las propiedades descritas en las plantillas de código correspondientes (Figura 8). Dichas plantillas conforman los *core assets* de la línea de productos de *dashboards*.

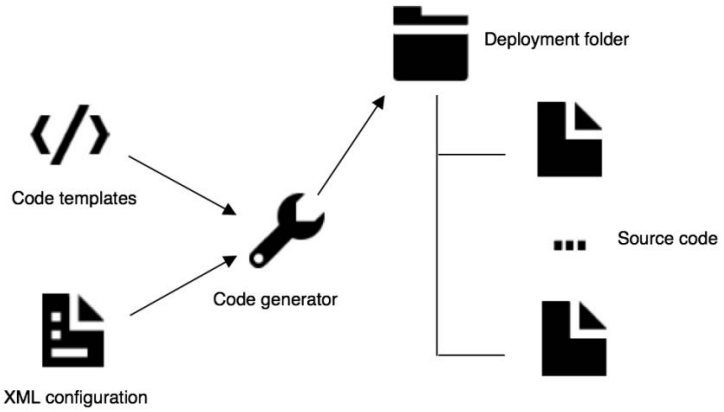

Figura 8. Esquema del método utilizado para la generación automática de código

Para la implementación de las plantillas de código se consideró utilizar el sistema de plantillas integrado en el propio *framework* que proporciona Django [56]. Sin embargo, se desechó la idea por estar extremamente ligado a dicho *framework* e impedir la portabilidad del código a otras plataformas. Además, sus funcionalidades se encontraban ligeramente limitadas para el alcance de los objetivos de este proyecto.

Por ello, las plantillas han sido realizadas con el lenguaje de plantillas Jinja2 [55]. Este método es independiente al marco de desarrollo, y, aunque deba ser utilizado junto al lenguaje de programación Python, es posible inyectar información externa y generar cualquier tipo de archivos [57]. A su vez, Jinja2 conforma un lenguaje de plantillas más potente que el proporcionado por Django, ofreciendo convenientes funcionalidades como la declaración de macros, muy útiles para ordenar el código y delimitar las características básicas de cada componente.

Cada componente gráfico de los *dashboards* ha sido programado a través de plantillas Jinja2, inyectando, mediante el uso de macros y sentencias condicionales, las funcionalidades y/o características especificadas en los ficheros XML de configuración.

4.3.3. Interoperabilidad

Un aspecto esencial en el dominio de los paneles de control son las fuentes de datos de las que se consume la información a mostrar. Las fuentes de datos son una de las características que pueden variar entre los miembros de la propia familia, por lo que es necesario desligar la lógica base de los

componentes del proceso de recuperación de datos, de tal forma que dicho proceso sea considerado como un punto de variabilidad.

Es por estas razones por las que los *dashboards* del Observatorio requieren una alta interoperabilidad respecto a las fuentes de datos, de tal forma que cada componente pueda evolucionar de forma independiente, disminuyendo el impacto negativo de los cambios y aumentando la mantenibilidad.

El Observatorio ya contaba con un componente de interoperabilidad basado en REST [58] que permitía recuperar y analizar los datos de su banco de conocimiento a través de llamadas a su API. Sin embargo, aunque este método dotaba al ecosistema de altos niveles de interoperabilidad, no constituía una solución suficiente para el alcance de este proyecto.

Por ello, se ha optado por construir una API en GraphQL [59, 60] que facilite el intercambio de información entre los componentes de presentación y el banco de datos del OEEU. GraphQL es un lenguaje de consulta para construir API que permitan llamadas más flexibles e incluso más eficientes que las implementadas con REST [61, 62]. El lenguaje de consulta proporciona una sintaxis parametrizable y flexible, muy beneficioso en términos de mantenibilidad de las aplicaciones cliente.

La API GraphQL del Observatorio ha sido implementada con Graphene (http://graphene-python.org/), una biblioteca que permite la construcción de APIs basadas en GraphQL para sistemas Django. Junto a la API de GraphQL se ha utilizado Pandas [63-65], otra biblioteca de Python para el análisis eficiente de cantidades significativas de datos. Se pretende, mediante este enfoque, conseguir el cálculo de métricas a demanda de los usuarios, a través de peticiones GraphQL ejecutadas en el *front-end*.

4.3.4. Patrones de reutilización en D3.js

Como se ha adelantado anteriormente, los componentes gráficos para los *dashboards* del OEEU se han implementado con la biblioteca D3.js en su versión 4 [51]. D3 permite la creación de visualizaciones interactivas dirigidas por datos [66] haciendo uso de tecnologías web (HTML, CSS y JavaScript).

El enfoque que se ha aplicado al diseño de los *dashboards* del OEEU requería que los componentes visuales fuesen reutilizables, configurables y tengan bien definidas sus funcionalidades para poder utilizar la generación de código por plantillas de manera efectiva. Debido a que D3.js tiene una complejidad significativa y no se trata de una biblioteca que genera gráficos predefinidos (como podría ser el caso de otras bibliotecas para Python como *Matplotlib [67]* o bibliotecas para JavaScript como *Google Charts, https://developers.google.com/chart/*), ha sido necesario tener en cuenta ciertos patrones de diseño a la hora de soportar la reutilización de los componentes implementados.

Las visualizaciones implementadas en D3.js en este proyecto han seguido las pautas especificadas por Michael Bostock (desarrollador de D3) para la implementación de gráficos reusables y configurables [68]. Estas pautas han permitido implementar *core assets* muy flexibles y hacer una composición relativamente trivial de visualizaciones y funcionalidades en los *dashboards* según las necesidades y restricciones de cada usuario.

5. Propuesta de la línea de productos software

A lo largo de esta sección se propone la línea de productos de paneles de control para el Observatorio de Empleabilidad y Empleo Universitarios. Para ello, se describen todos los elementos desarrollados, tanto a nivel de modelado como de implementación, necesarios para construir la línea de productos de *dashboards* con soporte para la generación automática de productos específicos que se presenta en este proyecto.

5.1. Meta-modelo de un *dashboard*

Para especificar la línea de productos se ha realizado, como primer paso, un meta-modelo del dominio de los *dashboards*. Los meta-modelos se caracterizan por ser abstractos y no depender de ningún tipo de plataforma, lenguaje o *framework* específico, permitiendo especificar y describir a alto nivel el dominio del problema enfrentado. A partir del meta-modelo, finalmente, se pueden instanciar modelos específicos, que serán los finalmente implementados en plataformas específicas.

Así pues, se ha propuesto el meta-modelo para *dashboards* expuesto en la Figura 9. Este meta-modelo debe ser sencillo y abstracto para evitar caer en características muy concretas y poder adaptarse a problemas más específicos del domino de estudio.

Se han especificado las siguientes entidades y relaciones en el meta-modelo propuesto:

- **Usuario.** Cada uno de los *dashboards* derivados del meta-modelo va a pertenecer a un usuario (o grupo de usuarios) concreto con una serie de requisitos.

- ***Dashboard.*** Se trata de la entidad abstracta que se quiere abordar en este dominio de estudio. El *dashboard* va a estar compuesto por una serie de elementos básicos comunes a todas las posibles instanciaciones del meta-modelo.

- **Página.** Es el elemento principal del *dashboard*. Todo *dashboard* va a estar compuesto por una o varias páginas que contendrán los elementos gráficos a mostrar. Estas páginas tendrán un título y una referencia que permitirá la navegación entre las mismas en el caso de existir más de una página en el *dashboard*.

- **Contenedor.** Es el elemento base que compone las páginas. Un contenedor puede ser de tipo **Fila** o **Columna** y cada contenedor puede contener uno o varios contenedores de forma recursiva. Con esta representación aseguramos la posibilidad de generar cualquier tipo de estructura (o *layout*) para la página.

- **Componente.** Dentro de cada uno de los elementos contenedores de la estructura del *dashboard* se encuentra un componente. A este nivel de abstracción, un componente puede ser cualquier elemento visual, de control o de diseño utilizado en el *dashboard*.

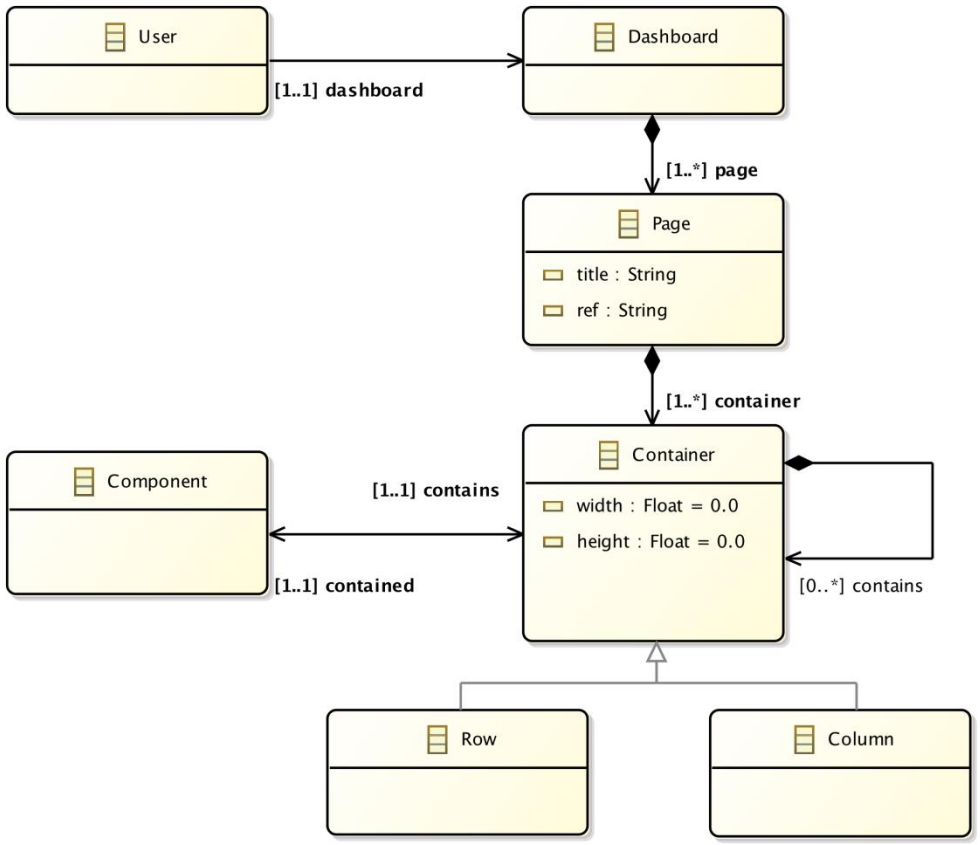

Figura 9. Meta-modelo propuesto para paneles de control (dashboards)

En las siguientes secciones se abordará el modelado de los componentes (a través del modelado de sus características específicas) que formarán parte de la familia de *dashboards* del Observatorio de Empleabilidad y Empleo Universitarios.

Búsqueda de necesidades

Como se ha introducido anteriormente, este trabajo se encuadra en un contexto donde los usuarios finales son los beneficiarios de la información recolectada. Sin embargo, es necesario clarificar las funcionalidades que los propios usuarios requieren, dado que estas pueden diferir en función de los objetivos específicos que tenga cada actor dentro del ecosistema.

En el contexto del OEEU, los usuarios principales del ecosistema son las universidades que participan en los estudios y los propios administradores. Si bien también existen usuarios generales que pueden consultar los resultados públicos, estos quedan fuera del alcance de este proyecto, aunque el enfoque planteado también sería aplicable a ellos una vez identificados sus requisitos. Así pues, para comenzar a conceptualizar los paneles de control, es ineludible contactar con los usuarios con el objetivo de obtener el conjunto de requisitos que demandan.

Las visualizaciones utilizadas para mostrar los resultados generales publicados en los informes [26, 27] y en el portal público obedecen al diseño de los estudios del Observatorio. Es posible que ciertas universidades quieran analizar y visualizar otras variables recolectadas o profundizar en algunos resultados según sus intereses concretos. El Observatorio permite que las universidades analicen libremente los datos en bruto de sus estudiantes poniendo a su disposición la descarga de los mismos en formato CSV, pero eso implica que ellos mismos tienen que realizar las tareas de análisis y visualización por su cuenta.

Lo que se pretende con la construcción de *dashboards* personalizados es tener en cuenta las necesidades específicas de cada usuario para facilitar la tarea de explotar y visualizar los datos de manera más libre y flexible, liberando a los mismos de la realización de tareas complementarias que podrían hacerles consumir una serie de recursos adicionales.

Los *dashboards* son herramientas muy potentes, siempre y cuando tengan un diseño adecuado [7]: no basta con construir una pantalla a base de una serie de gráficos que muestran métricas arbitrarias. Es indispensable tener en cuenta qué métricas pueden tener más peso en el proceso de toma de decisiones; lo que para un usuario puede ser una métrica o indicador esencial, para otro puede llegar a ser, incluso, irrelevante. Esta última es otra de las razones por la que la personalización de *dashboards* es un concepto potencialmente beneficioso si se cuenta con gran cantidad de usuarios y datos.

Así pues, la primera acción realizada para recolectar los diversos requisitos de las universidades participantes en los estudios del Observatorio fue la creación de un cuestionario en *Google Forms* con diversos ítems a valorar. En especial, estos ítems se centraron en cuantificar la prioridad que las universidades dan a diversas dimensiones informativas y funcionalidades ofrecidas por el Observatorio, para así tener en cuenta qué información es más relevante para ciertos usuarios. Sin embargo, los resultados mostraron que las universidades priorizaban la mayoría de información que podía ser ofrecida, precisamente por el hecho de que el sistema del Observatorio puede ofrecerlo de forma directa sin necesidad de que las universidades gasten recursos propios. De esta forma, los responsables de las universidades puntuaron alto la mayoría de ítems del cuestionario, concluyendo que, si el Observatorio puede ofrecer ciertas funcionalidades o pre-procesamiento de datos, las universidades lo valorarán. Todos los resultados obtenidos en la encuesta pueden consultarse en la carpeta "Resultados Cuestionarios" (los resultados se corresponden a las respuestas del personal de un total de 16 universidades y 2 administradores del Observatorio), donde se encuentran tanto las respuestas de las universidades como las de los administradores del sistema del Observatorio.

Debido a esto, se optó finalmente por obtener información cualitativa a través de entrevistas personales a los responsables de la comunicación con el Observatorio de las universidades más implicadas en los estudios (esto es, universidades que utilizan más asiduamente los datos de los barómetros del Observatorio o que han expresado sus ideas o sugerencias para introducir en el sistema). Específicamente, se contactó con el personal encargado de la participación de la universidad en los estudios del Observatorio (generalmente, los encargados de los observatorios de empleo propios de cada universidad, los cuales se encargan generalmente de analizar los datos recolectados por el Observatorio). Estas entrevistas tuvieron como objetivo alcanzar una visión más amplia de las tareas y procesos de análisis que las universidades realizan con los datos recolectados por el Observatorio, de manera que dichas tareas pudiesen ser realizadas directamente por el ecosistema o ser facilitadas por el mismo para el beneficio del personal de las universidades.

Durante las entrevistas (realizadas por teléfono), los responsables encargados de gestionar la participación y los datos del Observatorio de cuatro universidades[1] respondieron a una serie de preguntas sobre las tareas de análisis que realizan, sobre si los datos se utilizan en el proceso de toma de decisiones y sobre cualquier funcionalidad que pudiese serles útil de cara a facilitar la explotación de los datos. Las diversas respuestas a las preguntas planteadas en las entrevistas pueden consultarse en el Apéndice A.

Tras finalizar las entrevistas, las conclusiones fueron claras: los datos del Observatorio se utilizan, en general, para poder comparar a nivel nacional los datos propios recolectados por los sistemas internos de las universidades. Es decir, se utilizan los resultados generales, porque a nivel de universidad cada una cuenta con sus propias unidades de calidad y sistemas de recolección de datos internos. Sí que se especificaron, en algunas de las entrevistas, el deseo de poder guardar en formato de imagen las visualizaciones de los paneles de control, así como la comparación con las medias nacionales directamente desde el *dashboard*. Es posible que una universidad sin unidades internas de calidad utilizase más asiduamente los datos recolectados por el Observatorio para obtener los resultados de sus propios egresados.

Estos resultados supusieron un reto, puesto que no existía una serie de claros indicadores a introducir en cada *dashboard*, con lo cual estos mismos iban a tener que componerse de métricas genéricas que podrían refinarse a lo largo del tiempo con una investigación más profunda de las necesidades concretas de cada universidad.

Al usar un enfoque "genérico", ha sido necesario buscar una forma de dar libertad a los usuarios para explotar los datos de una manera sencilla, dado que la cantidad de variables es grande y su visualización puede descontrolarse, afectando a la experiencia de usuario. Se ha optado, pues, por ofrecer toda la información disponible (teniendo en cuenta las restricciones de acceso a datos que tiene cada usuario) en un formato explorable a través de selectores de variables, filtros y otra clase de controles.

5.2. Componentes de la línea de productos de *dashboards* para el OEEU

Como se introdujo en el apartado anterior, para diseñar los componentes de la línea de productos (visualizaciones del *dashboard*), se ha optado por elaborar componentes que den libertad a los usuarios para explotar los datos del OEEU.

Para ello, ha sido necesario tener en cuenta qué tipo de visualizaciones pueden ser beneficiosas para los tipos de variables que almacena el Observatorio; estas variables pueden clasificarse en categóricas y numéricas. Las variables numéricas almacenan, por lo general, las puntuaciones en las escalas Likert que se muestran a los titulados en los cuestionarios (satisfacción con los estudios, niveles de competencias, importancia de determinadas metodologías educativas, etc.). Las categóricas, por otra parte, almacenan características de los titulados (nivel de cualificación en el último empleo, rama de conocimiento de los estudios cursados, sexo, comunidad autónoma de procedencia, rangos de salarios, etc.).

[1] Las universidades implicadas en las entrevistas fueron la Universidad Nacional de Educación a Distancia, la Universidad Politécnica de Madrid, la Universidad de Las Palmas de Gran Canaria y la Universidad de Castilla La-Mancha, a cuyos responsables de la participación en los estudios del OEEU se les agradece plenamente su colaboración en este proceso de mejora de los *dashboards* ofrecidos por la organización.

Se pretende, mediante el uso de paneles de mandos, que los usuarios puedan establecer relaciones entre las diferentes variables para obtener una visión más amplia de la empleabilidad y la vinculación entre la formación universitaria y el empleo de los titulados. Gracias al paradigma de las líneas de productos *software*, el número de componentes de los *dashboards* puede evolucionar a lo largo del tiempo, ya sea en cantidad o en funcionalidad, por lo que podrían añadirse nuevos tipos de visualizaciones a medida que se fuesen descubriendo nuevos requisitos por parte de los usuarios (o debido a la propia evolución de los datos).

Así pues, para esta primera aproximación de la línea de productos de *dashboards*, se han propuesto los siguientes componentes: un diagrama de dispersión, un mapa de calor y un diagrama de cuerdas. A dichos componentes se les ha aplicado la filosofía de la visualización interactiva enunciada por Ben Shneiderman: *"Overview first, zoom and filter, then details-on-demand"* [69], es decir, mostrar una vista general, proporcionar mecanismos de zoom y filtrado y finalmente, en función de la interacción de los usuarios, mostrar los detalles de los datos mostrados.

5.2.1. Diagrama de dispersión (*Scatter Plot*)

Un diagrama de dispersión puede servir para explorar correlaciones entre variables o identificar patrones entre ellas. Estos diagramas representan las dimensiones como puntos en el propio espacio de representación, compuesto de dos ejes donde se encuentran las variables a explorar. Es posible, incluso, visualizar una tercera variable a través del tamaño de los puntos que representan las dimensiones (Figura 10).

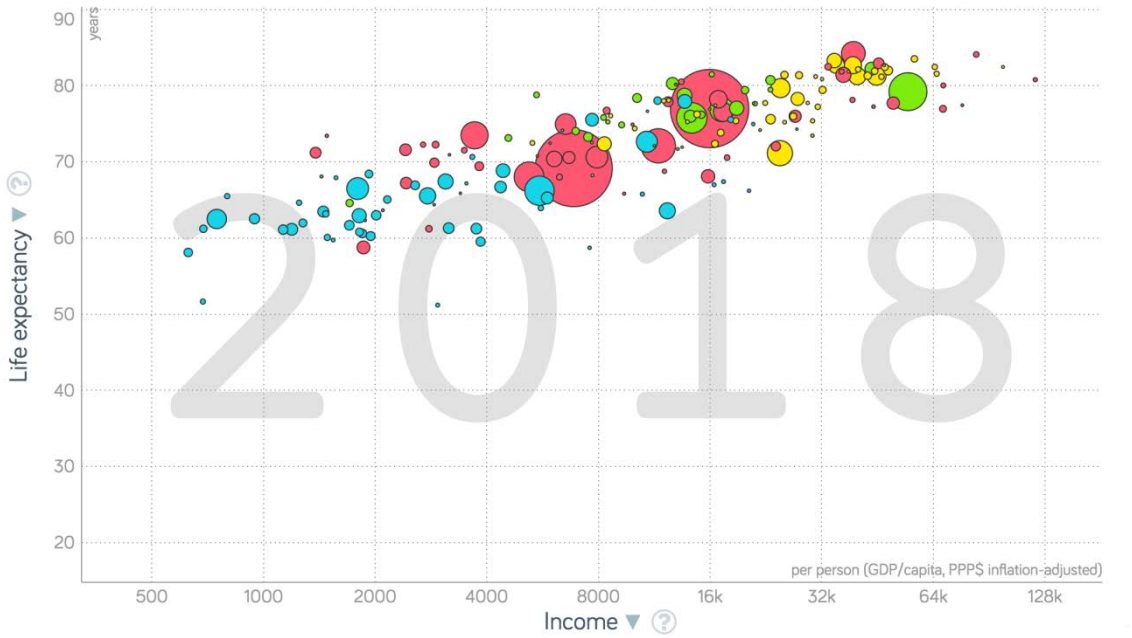

Figura 10. Ejemplo de un diagrama de dispersión que relaciona la esperanza de vida media en cada país con sus ingresos (Fuente: www.gapminder.org)

Una de las desventajas de los diagramas de dispersión viene derivada de las propias características de los datos a mostrar; por ejemplo, los valores podrían superponerse, haciendo más compleja la visualización e interpretación. Debido a este potencial problema, una de las funcionalidades que se propone proporcionar a este componente es la posibilidad de hacer *zoom* en el propio diagrama, para ofrecer más detalladamente los resultados obtenidos (bajo demanda del usuario).

Este componente de la línea de productos está inspirado por una de las herramientas de visualización ofrecidas por Gapminder (https://www.gapminder.org/). Se trata una herramienta a la que denominan *"Bubbles"*, la cual se compone de un diagrama de dispersión que permite la interacción del usuario a través de controles, así como el filtrado e incluso el cambio de las variables mostradas. Este enfoque permite a los usuarios decidir los datos que quieren visualizar de forma dinámica, ofreciéndoles libertad y flexibilidad.

Los diagramas de dispersión benefician al Observatorio al permitir explorar la relación de hasta tres variables numéricas simultáneamente. A su vez, debido a que los estudiantes pueden clasificarse a través de las variables categóricas recolectadas (rama de conocimiento de los estudios, sexo, procedencia, etc.), sería posible observar cómo se relacionan las diversas dimensiones con las variables seleccionadas. Estas características, combinadas con los controles del enfoque de la herramienta "*Bubbles*" de Gapminder, proporcionan un componente flexible y beneficioso para la explotación de los datos del Observatorio.

5.2.2. Mapa de calor (*Heat Map*)

El Observatorio almacena variables numéricas destinadas la puntuación de ciertas cuestiones; desde la satisfacción con diversos aspectos de los empleos que han tenido/tienen los titulados hasta el nivel de una serie de competencias que creen poseer o que se les han requerido en el último empleo a los mismos.

Especialmente, la visualización de las competencias que poseen los titulados se torna un aspecto muy importante, tanto a nivel de formación como a nivel de empleabilidad. El Observatorio analizó un total de 34 competencias genéricas en la edición del barómetro de 2015 y 26 en la edición de 2017. A su vez, dichas competencias se valoran desde tres perspectivas: el nivel que poseen los titulados, el nivel que les aporta la universidad y el nivel requerido en el último empleo. Es decir, aparece una cantidad significativa de competencias que, además, se valoran desde tres perspectivas. Encontrar una forma de visualizar esta cantidad de información de manera comprensible es esencial para poder alcanzar conclusiones respecto a las competencias de los titulados.

Una de las visualizaciones potencialmente aplicables para resolver este problema son los mapas de calor. Los mapas de calor permiten llamar la atención sobre elementos clave gracias a la codificación de valores mediante escalas de colores. Al puntuarse las competencias mediante escalas Likert, es posible aplicar este enfoque codificando el rango de valores de la escala a través de paletas de colores. Una aplicación de mapas de calor a la visualización de competencias se realiza en [70]; mediante este enfoque es posible observar a simple vista las competencias más requeridas según el ámbito del empleo (Figura 11).

Este método puede adaptarse al contexto del Observatorio: las competencias se colocarían en uno de los ejes, utilizando el eje restante para agregar los resultados según ciertas variables categóricas, permitiendo comparaciones de las competencias entre las diversas categorizaciones (ramas de

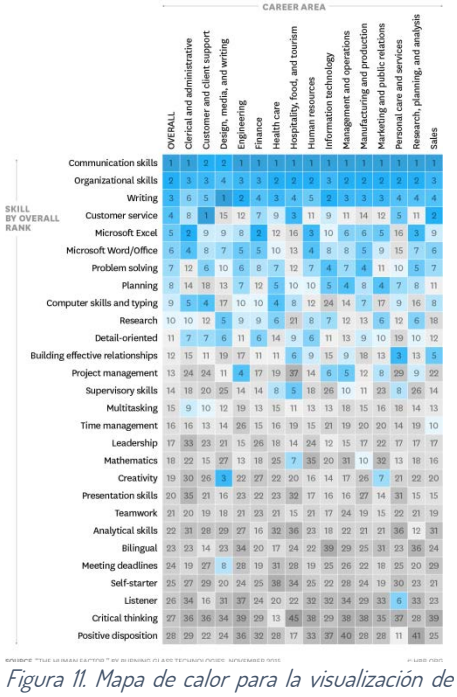

Figura 11. Mapa de calor para la visualización de competencias más solicitadas en ciertos empleos

conocimiento, sexo, etc.).

Añadido a esto, los mapas de calor pueden aplicarse a cualquier otro conjunto de variables recogidas por el Observatorio a través de escalas Likert, no solo a las competencias genéricas, proporcionando un mecanismo flexible para la visualización y comparación de puntuaciones a diversos aspectos.

5.2.3. Diagrama de cuerdas (*Chord Diagram*)

Como se ha mencionado anteriormente, el Observatorio almacena una significativa cantidad de variables categóricas. Las variables categóricas son variables sobre las que únicamente podemos obtener medidas de tipo nominal u ordinal, que nos permitirán categorizar o clasificar las diversas observaciones realizadas.

Debido a la naturaleza nominal de estas variables, no pueden aplicarse otros métodos de estadística descriptiva como el cálculo de medias, medianas o desviaciones estándar. Uno de los métodos más utilizado a la hora de relacionar variables categóricas entre sí son las tablas de contingencia: los datos se organizan en matrices en las que cada fila representaría los valores de un criterio de clasificación, aplicándose el mismo método en las columnas de la tabla para cualquier otro criterio de clasificación. Como resultado se obtendrían las frecuencias de distribución de las observaciones que cumplen cada uno de los criterios de clasificación, pudiendo analizar la relación de las variables categóricas. Puede verse un ejemplo en la Tabla 1.

Tabla 1. Ejemplo de una tabla de contingencia

Sexo/Nivel cualificación	Infra-cualificado	Cualificado	Sobre-cualificado
Hombre	44	10	23
Mujer	32	12	20

Sin embargo, ha sido necesario encontrar un mecanismo para visualizar las relaciones entre este tipo de variables gráficamente, el cual permita extraer la información de manera más sencilla.

Los diagramas de cuerdas (*chord diagrams*) toman como entrada matrices de contingencia, mostrando las frecuencias como arcos o flujos entre los diversos grupos (categorías) especificados. El grosor de dichos flujos o relaciones es directamente proporcional a las frecuencias mostradas en las tablas de contingencia.

Existen también aplicaciones exitosas de diagramas de cuerdas a datos sobre la vinculación entre la formación y el empleo para explorar la relación de dichas variables categóricas de manera interactiva (Figura 12).

Los diagramas de cuerdas aportan al Observatorio un mecanismo para explorar sus variables categóricas y facilitar el proceso de identificación de patrones y relaciones entre las mismas.

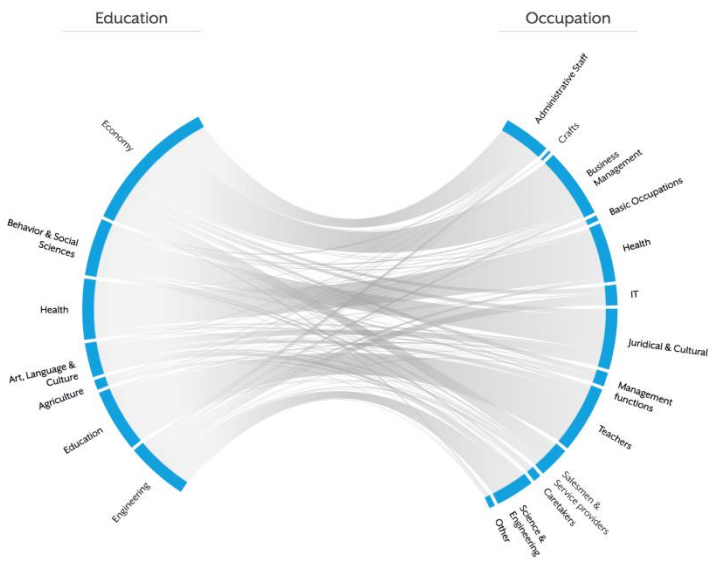

Figura 12. Ejemplo de un diagrama de cuerdas

(Fuente: https://www.visualcinnamon.com/2015/08/stretched-chord.html)

5.3. *Feature model* de la línea de productos *dashboard* del OEEU

Para esta primera aproximación de línea de productos de *dashboards*, se ha elaborado el siguiente diagrama de características en función de los componentes propuestos para el Observatorio de Empleabilidad y Empleo Universitarios en la sección 5.2. Cada uno de los componentes, a su vez, puede tener una serie de características específicas que varíen su funcionalidad e incluso los datos a mostrar, como se explicará a continuación. Como puede observarse, este modelado de características entra en el dominio del problema específico al que se enfrenta el Observatorio de Empleabilidad y Empleo Universitarios respecto a sus datos y sus usuarios, pero sigue el meta-modelo propuesto en 5.1.

A continuación, se muestra el diagrama de características dividido en secciones debido a su extensión, describiendo cada una de las características propias de cada componente especificado. La raíz del árbol es la línea de productos de *dashboards* diseñada para el Observatorio de Empleabilidad y Empleo Universitarios, que tiene dos características obligatorias: el sistema base común a todos los usuarios del OEEU (sección 4.3.1) y al menos una página.

Diagrama de dispersión (*Scatter Diagram*)

El diagrama de dispersión (Figura 13) cuenta con dos características obligatorias, comunes a todas las instancias que puedan configurarse de este componente: la **lógica base** y los **datos iniciales**. Es evidente que, para poder implementar un diagrama de dispersión, es necesario contar con una lógica común que permita representar la estructura básica de este tipo de diagramas (ejes, espacio de coordenadas, etc.). También es obligatorio contar con un conjunto de datos a representar, de lo contrario el diagrama se mostraría vacío y carecería de sentido. En los datos iniciales es obligatorio especificar las variables a representar en los ejes X e Y, y es opcional especificar la variable representada por el radio de los elementos del diagrama (en caso de no especificarlo se utiliza un radio por defecto), así como la agregación o categorización de los datos (en caso de no especificarlo, los datos no se agregan y se muestran los resultados totales del estudio).

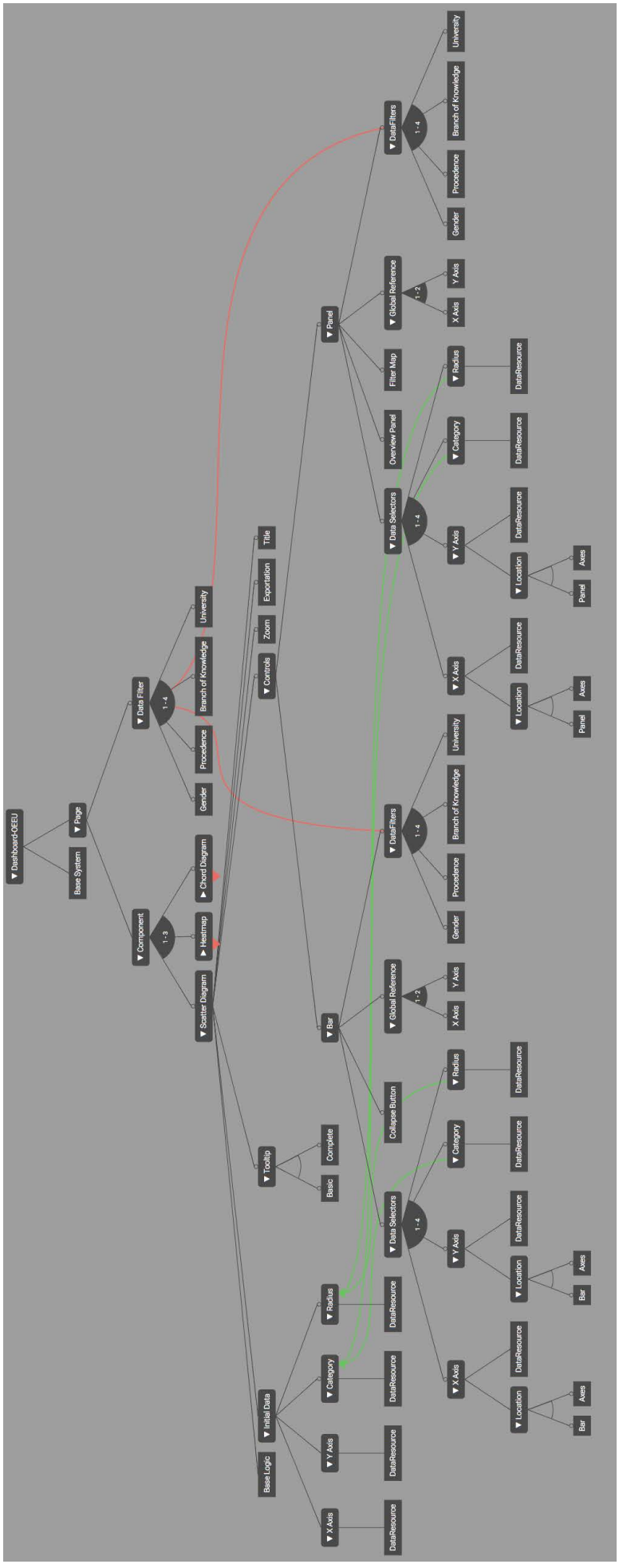

Figura 13. Feature model para los dashboards del OEEU (diagrama de dispersión)

En cuanto a las características opcionales se encuentran las siguientes alternativas:

- ***Tooltip:*** el diagrama puede contar opcionalmente con un *tooltip* que muestre en detalle los datos con los que interactúa el usuario. Este *tooltip* puede configurarse para ser **básico** y mostrar información muy resumida del elemento con el que se está interactuando. La otra opción, mutuamente excluyente con la anterior, consiste en que el *tooltip* sea **completo**, y muestre toda la información posible de los elementos.

- **Controles:** el diagrama de dispersión puede contar con una serie de controles que permitan interactuar con los datos mostrados. Esta característica cuenta con dos variantes mutuamente excluyentes:

  - **Barra:** los controles se muestran como una barra superior en el diagrama. Esta barra podría mostrarse y ocultarse a través de un control (***Collapse Button***). Entre los controles se encuentran los selectores de datos de las diversas variables del diagrama, los cuales permitirían dar libertad al usuario para elegir qué variable representar en cada momento.

    Los controles para los ejes X e Y pueden encontrarse dentro de la propia barra o localizados en los ejes correspondientes. Es necesario que en los datos iniciales se hayan especificado las variables que categorizan los datos o que representan el radio de los elementos para poder contar con los controles que permiten manipular dichas variables (restricción representada mediante líneas de color verde en la Figura 13).

    Por otra parte, también se permite el control de una referencia global (***Global Reference***) que mostrará en el diagrama una referencia desagregada de los datos representados, permitiendo comparar la información con los valores globales del estudio (uno de los requisitos expresados por las universidades). Al igual que los selectores de variables para los ejes X e Y, este control puede localizarse tanto en la propia barra como en el eje sobre el que toma efecto.

    Finalmente, dentro de estos controles puede añadirse una serie de filtros (***Data Filters***) que permiten filtrar los datos por género, procedencia, rama de conocimiento o universidad (siendo esta última opción seleccionable únicamente por usuarios administradores). Esta característica solamente puede seleccionarse si no se ha seleccionado previamente la característica ***Data Filter*** global para la página (componente que se explica al final de esta sección), esto es, son características mutuamente exclusivas.

  - **Panel:** en este caso los controles se contendrían en un panel junto al diagrama. Se aplica la misma explicación para las características ***Data Selectors***, ***Global Reference*** y ***Data Filters*** que en el caso inmediatamente anterior. El representar los controles como un panel permite contar con más espacio para su representación, por ello aparecen dos nuevas características para este tipo de controles:
    - ***Filter Map***: esta característica permite filtrar los datos según la procedencia de los estudiantes a través de un mapa interactivo de España.
    - ***Overview Panel***: esta característica muestra en detalle los datos representados cuando el usuario interactúa con ellos.

- ***Zoom***: se da opción para que el diagrama cuente con una funcionalidad de *zoom*. Esta funcionalidad puede ser de utilidad para explorar datos solapados, sin embargo, para

usuarios más inexpertos, podría entorpecer la interacción con el diagrama, con lo cual se mantiene como una característica opcional.

- **Exportación:** otra característica que se solicitaba en alguna entrevista con los responsables de las universidades era la posibilidad de exportar como un archivo de imagen las visualizaciones de los *dashboards*. Por ello mismo se especifica la característica opcional de exportación.

- **Título:** opcionalmente los diagramas pueden tener un título que permita especificar su funcionalidad o incluso diferenciarlos en el caso de tener varias instanciaciones del mismo.

Mapa de calor (*Heatmap*)

El mapa de calor (Figura 14) cuenta también con dos características obligatorias, comunes a todas las instancias que puedan configurarse de este componente: la **lógica base** y las **dimensiones** del diagrama. La lógica base tiene la misma función que la especificada en el caso del diagrama de dispersión, solo que adaptada a este tipo de visualizaciones. A su vez, un diagrama de calor mostrará al menos una dimensión, refiriéndonos como dimensión a la representación de los valores relacionados con dos variables.

Esta característica surge debido al estudio del propio dominio del problema específico del Observatorio, ya que cuenta con varias dimensiones de muchas de sus variables (por ejemplo, en el caso de las competencias, existen tres dimensiones: contribución de la universidad, nivel requerido en el último empleo y nivel que poseen actualmente). Cada una de las dimensiones a representar en el mapa de calor debe contar con unos datos iniciales a representar, por la misma razón que la expresada anteriormente, de lo contrario la visualización no mostraría ningún tipo de información.

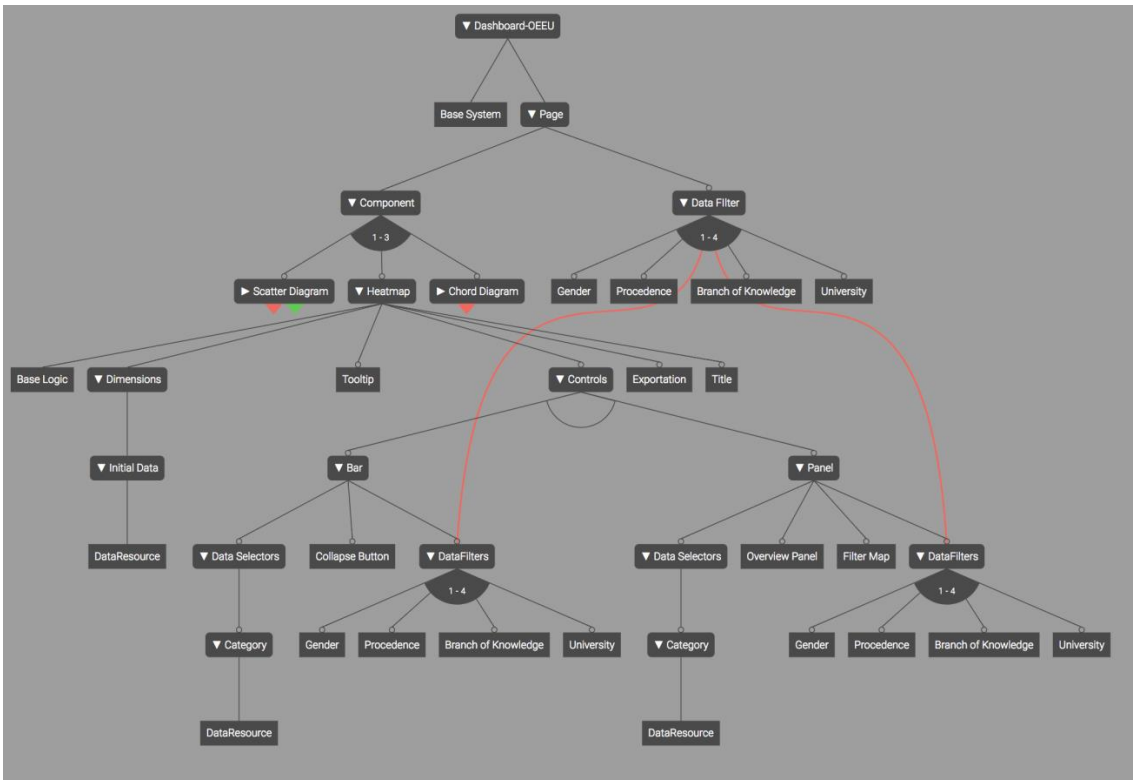

Figura 14. Feature model para los dashboards del OEEU (mapa de calor)

En cuanto a las características opcionales se encuentran las siguientes alternativas:

- ***Tooltip:*** opcionalmente, se mostrará un *tooltip* con información referente a cada uno de los elementos informativos del diagrama cuando se interactúe con los mismos.

- **Controles:** esta característica, de nuevo, cuenta con dos variantes mutuamente excluyentes:

  - **Barra:** los controles se muestran como una barra superior en el diagrama. Esta barra podría mostrarse y ocultarse a través de un control (***Collapse Button***). Entre los controles se encuentran los selectores de datos de las variables del diagrama, en este caso, se permite agregar y desagregar los valores según diversas variables categóricas. Respecto a la característica ***Data Filters*** se aplica exactamente la misma explicación que en el diagrama de dispersión.

  - **Panel:** los controles se contendrían en un panel junto al diagrama. Se aplica la misma explicación que en el diagrama de dispersión para las posibles características de este tipo de controles.

- **Exportación:** opcionalmente se da la posibilidad de descargar en formato de imagen el diagrama representado.

- **Título:** opcionalmente los mapas de calor pueden tener un título que permita especificar su funcionalidad o incluso diferenciarlos en el caso de tener varias instanciaciones del mismo.

Diagrama de cuerdas (*Chord Diagram*)

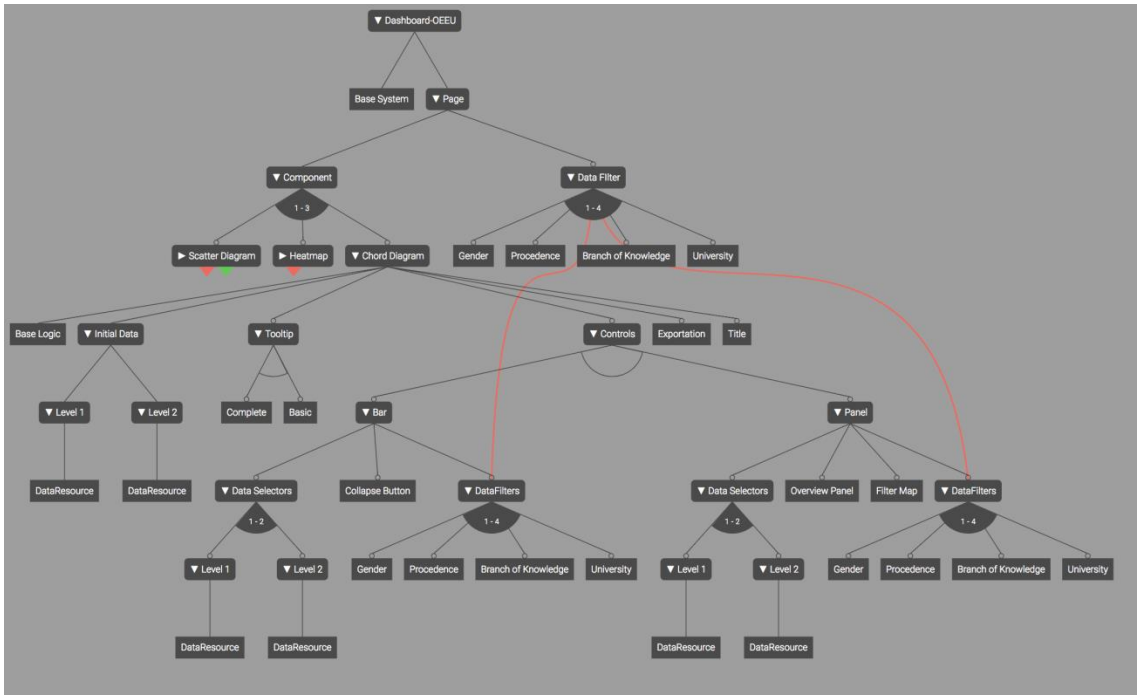

Figura 15. Feature model para los dashboards del OEEU (diagrama de cuerdas)

El diagrama de cuerdas (Figura 15), por último, debe tener una **lógica base** y unos **datos iniciales** al igual que el resto de visualizaciones y por las mismas razones. En este caso los datos iniciales son los dos niveles del diagrama de cuerdas (esto es, los dos conjuntos de variables categóricas a relacionar).

Como puede observarse en la Figura 15, se comparten características con los dos componentes descritos anteriormente, con lo que se aplica la explicación aportada para los mismos.

Filtro de datos (Data Filter)

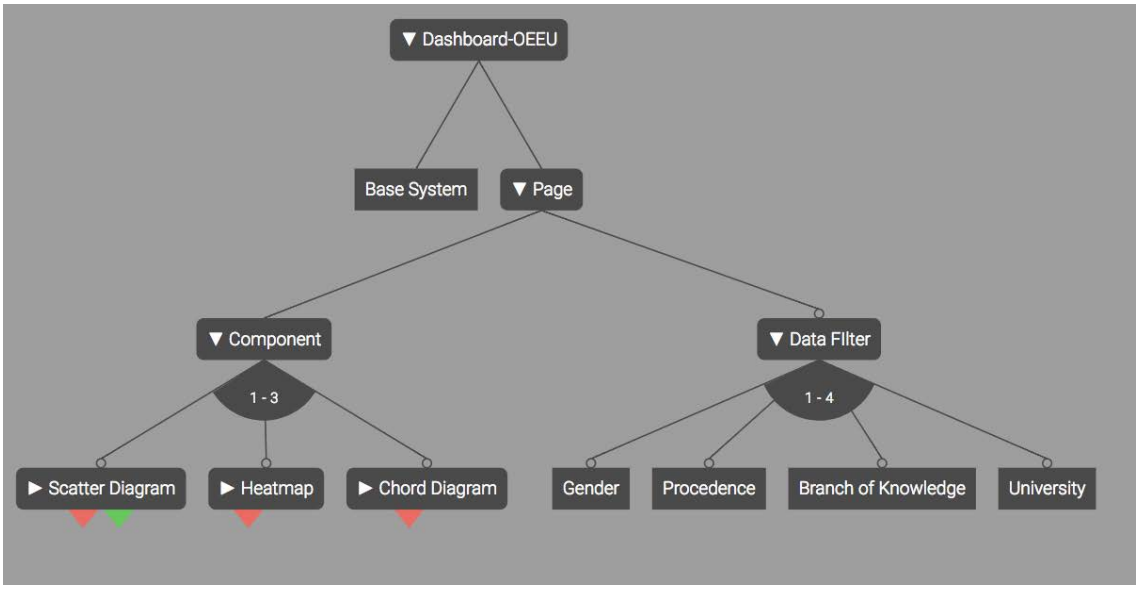

Figura 16. Feature model para los dashboards del OEEU (filtro de datos)

Por último, dentro de la página se puede contar, opcionalmente, con un filtro de datos (Figura 16) que afecta a todos los componentes seleccionados, permitiendo el filtrado y una exploración de datos más profundas a través de diversas perspectivas. Dentro de esta característica se permite la filtración de datos por género, procedencia, rama de conocimiento y universidad (siendo esta última opción seleccionable únicamente por usuarios administradores).

Este *feature model* puede evolucionar a lo largo del tiempo para añadir nuevas características a la línea de productos. Añadir dichas características solo implicaría la creación y/o modificación de los *core assets*, y se aplicaría a todos los miembros de la familia de manera automática, aumentando la mantenibilidad de la familia de *dashboards* del OEEU.

5.4. Validación del DSL

Una vez obtenidos el meta-modelo y el *feature model* para los *dashboards*, se han trasladado las características y reglas especificadas en el mismo a un formato estructurado para implementar el DSL. Como se indicó anteriormente, el DSL está basado en XML, por lo que se ha hecho uso de un esquema (XSD) para facilitar la configuración de los componentes específicos de la línea de productos. En este esquema, se han fusionado los aspectos modelados en el *feature model* así como las características especificadas en el meta-modelo (la estructura del *dashboard*). Así pues, el *feature model* especifica la parte funcional (las funcionalidades que tendrá cada *dashboard*) y el meta-modelo la estructura del mismo.

El esquema XML en el que se basa el DSL, por lo tanto, tiene la siguiente especificación:

- Elemento raíz: **Dashboard**. Se trata de la raíz del esquema y, por tanto, de todos los documentos de configuración, englobando las características que tendrá el producto (Figura 17). Este elemento es complejo y cuenta con los siguientes componentes:
  - ***Title***: opcionalmente, el *dashboard* puede contar con un título.
  - ***NavigationBar***: opcionalmente, el *dashboard* puede contar con una barra de navegación que puede tener uno o más *links*, haciendo referencias a las distintas páginas con las que cuenta el *dashboard*. También cuenta con un elemento (*Brand*) que permitirá especificar un título para la barra de navegación.
  - ***ScreensConfig***: como se verá posteriormente, este elemento complejo engloba la configuración de las diversas páginas del *dashboard*.
  - El elemento *dashboard* también tiene dos atributos, "**name**" y "**user**", los cuales, respectivamente, dan un nombre específico al producto y permiten especificar el usuario al que pertenece la configuración.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Dashboard">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Title" type="xs:string" minOccurs="0" />
                <xs:element name="NavigationBar" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Brand" type="xs:string" />
                            <xs:choice>
                                <xs:element name="Link"...>
                                <xs:element name="LinkGroup"...>
                            </xs:choice>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="ScreensConfig"...>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" />
            <xs:attribute name="user" type="xs:int" />
        </xs:complexType>
    </xs:element>
    <xs:complexType name="LayoutType"...>
</xs:schema>
```

Figura 17. Vista general del esquema XML para la línea de productos dashboard.

- Primer nivel: ***ScreensConfig***. Es la raíz relativa a la configuración de cada una de las posibles páginas que forman el *dashboard*. Este elemento engloba la configuración de una página (***Page***), si es que el *dashboard* solo va a estar compuesto de una sola página, o de un grupo de páginas (***PageGroup***); en este caso, el elemento ***PageGroup*** puede estar compuesto por uno o más elementos ***Page***.

- Configuración de una página del *dashboard* (Figura 18). Como se ha dicho en el punto anterior, gracias al elemento ***Page*** podemos configurar una de las páginas del *dashboard*. Este elemento tiene un identificador de página (único para cada una de las páginas que se configuren). De nuevo, se trata de un tipo complejo que cuenta con los siguientes elementos:

- ***DataFilter***: opcionalmente, se puede configurar un filtro general para la página, que permitirá filtrar datos según una serie de criterios, los cuales afectarán a todas las visualizaciones de la página de manera simultánea. Como se verá posteriormente, este elemento solo se puede configurar si, a nivel de componente, no se ha configurado ningún otro filtro, por razones de coherencia en los datos mostrados.

  Este elemento cuenta, opcionalmente, con un título y con una serie de filtros que provendrán de una determinada fuente de datos. El elemento ***DataResource*** sirve para abstraer las fuentes de datos, por ello se especifica el tipo genérico "anyType". De esta forma, si las especificaciones de las fuentes de datos cambian, no afectará en términos generales al esquema. También es necesario el atributo "**component_id**", que permite especificar de forma unívoca al componente.

- ***Components***. Este elemento es principal, ya que permite seleccionar los componentes visuales que formarán parte de la página del *dashboard*, así como sus funcionalidades.

- ***Layout***. Este elemento está basado en el meta-modelo de *dashboard* (sección 5.1), el cual permite establecer la estructura de la página; esto es, la situación relativa en el espacio de representación de cada componente configurado anteriormente. De esta forma añadimos un grado de personalización en términos de diseño.

```
<xs:element name="ScreensConfig">
    <xs:complexType>
        <xs:choice>
            <xs:element name="PageGroup"...>
            <xs:element name="Page">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="DataFilter" minOccurs="0">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="Title" type="xs:string" minOccurs="0"/>
                                    <xs:element name="FilterGroup">
                                        <xs:complexType>
                                            <xs:sequence>
                                                <xs:element name="DataResource" type="xs:anyType"/>
                                            </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                </xs:sequence>
                                <xs:attribute name="component_id" type="xs:string"/>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="Components"...>
                        <xs:element name="Layout" type="LayoutType"/>
                    </xs:sequence>
                    <xs:attribute name="page_id" type="xs:string"/>
                </xs:complexType>
                <xs:unique name="unique-page_id">
                    <xs:selector xpath="Page"/>
                    <xs:field xpath="@page_id"/>
                </xs:unique>
            </xs:element>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

Figura 18. Detalle del elemento ScreensConfig del esquema XML.

- Configuración de cada componente de la página (Figura 19). En este caso, pueden elegirse una o más instancias de entre los siguientes elementos: ***ScatterDiagram*** (representa el componente "diagrama de dispersión), ***Heatmap*** (representa el componente "mapa de calor") o ***ChordDiagram*** (representa el componente "diagrama de cuerdas"). Cada uno de dichos componentes tiene un identificador único (***component_id***).

```
<xs:element name="Components">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Component" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:choice>
                        <xs:element name="ScatterDiagram">
                            <xs:complexType>
                                <xs:sequence...>
                                <xs:attribute name="component_id"
                                              type="xs:string"/>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="Heatmap">
                            <xs:complexType>
                                <xs:sequence...>
                                <xs:attribute name="component_id"
                                              type="xs:string"/>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="ChordDiagram">
                            <xs:complexType>
                                <xs:sequence...>
                                <xs:attribute name="component_id"
                                              type="xs:string"/>
                            </xs:complexType>
                        </xs:element>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figura 19. Esquema de los componentes disponibles para configurar en cada página del dashboard.

Cada uno de los posibles compontes, a su vez, están formados por diversos elementos. No se entrará en detalle respecto a cada uno de ellos, puesto que están basados en el diagrama de características presentado en la sección 5.3.

A modo de ejemplo se muestran los nodos del componente ***ScatterDiagram***, donde puede observarse la correspondencia con las características modeladas en la fase de ingeniería de dominio (Figura 20).

Al igual que en casos anteriores, las fuentes de datos se abstraen mediante el elemento genérico ***DataResource***. Esto permite especificar las fuentes de datos de las que consumirá la visualización, lo cual también se aplica a las opciones disponibles en los selectores de datos.

```
<xs:element name="ScatterDiagram">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Title" type="xs:string"
                        minOccurs="0"/>
            <xs:element name="Zoom" type="xs:string"
                        minOccurs="0"/>
            <xs:element name="Exportation"
                        type="xs:string" minOccurs="0"/>
            <xs:element name="InitialData">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="xAxis">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element
                                            name="DataResource"
                                            type="xs:anyType"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="yAxis"...>
                        <xs:element name="Category"...>
                        <xs:element name="Radius"...>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Controls" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element
                                name="CollapseButton"
                                type="xs:string"
                                minOccurs="0"/>
```

Figura 20. Detalle del esquema XML para el elemento ScatterDiagram (diagrama de dispersión).

- Configuración de la estructura de la página (Figura 20). Para seguir el meta-modelo y permitir recursividad en la especificación de la estructura de la página, se utiliza un tipo complejo "**LayoutType**". La estructura de la página se especifica a través de grupos de columnas o filas, que estarán formadas por un número ilimitado de las mismas.

La recursividad termina cuando se especifica que el contenedor contiene un elemento "**Component**" o un elemento "**DataFilter**", pues representan los nodos hoja de la estructura.

Estos elementos referencian a los componentes previamente configurados mediante sus identificadores (atributo "**ref**"). Esta especificación permite separar lógicamente la configuración de cada componente en términos de funcionalidad de la especificación del *dashboard* en términos de diseño o estructura. También pueden especificarse la anchura y la altura de cada fila o columna en términos de porcentajes a través de los atributos "**width**" y "**height**".

```
<xs:complexType name="LayoutType">
    <xs:choice>
        <xs:element name="RowGroup" type="LayoutType"/>
        <xs:element name="ColumnGroup" type="LayoutType"/>
        <xs:element name="Row" type="LayoutType" maxOccurs="unbounded"/>
        <xs:element name="Column" type="LayoutType" maxOccurs="unbounded"/>
        <xs:element name="Component">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="xs:anySimpleType">
                        <xs:attribute name="ref" type="xs:string"/>
                    </xs:extension>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="DataFilter">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="xs:anySimpleType">
                        <xs:attribute name="ref" type="xs:string"/>
                    </xs:extension>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
    </xs:choice>
    <xs:attribute name="width" type="xs:string" use="optional"/>
    <xs:attribute name="height" type="xs:string" use="optional"/>
</xs:complexType>
```

Figura 21. Especificación de la estructura del dashboard.

El esquema XML presentado es de gran utilidad para esta propuesta, pues tiene varias funciones:

- Validar la sintaxis del DSL, evitando la generación de código si se encuentra un error sintáctico a la hora de especificar la línea de productos

- Validar la configuración de los productos específicos derivados de la línea de productos. Esta función es interesante, pues al estar el esquema basado en el *feature model*, se cuenta con una herramienta simple de validación de configuraciones de la familia.

Sin embargo, algunas restricciones expresadas en el *feature model* no son posibles de implementar en el esquema XML, debido a su complejidad y al hecho de que se utiliza la versión 1.0, la cual no permite la especificación de ciertas restricciones mediante aserciones. Como ya se mencionó anteriormente, es necesario utilizar la versión 1.0 de XSD por razones de compatibilidad y soporte de las bibliotecas utilizadas, pero en un futuro se espera poder utilizar la versión 1.1 de XSD para hacer dicho esquema más potente. Por estas razones se añade una segunda capa de validación mediante una función de Python que permite recorrer el XML en busca de inconsistencias entre los valores de los diferentes elementos.

Esta función valida que no se incumplan las siguientes restricciones:

- El componente ***DataFilter*** de la página solo puede especificarse si no se ha seleccionado a nivel de componente.

- El componente ***Map*** solo puede especificarse si los controles de la visualización son de tipo "***Panel***". De lo contrario, el espacio disponible no sería suficiente para representar el mapa. A su vez, solo puede seleccionarse si, a nivel de componente, no se ha seleccionado anteriormente el filtro de datos por comunidad autónoma por medio del elemento ***DataFilters***.

- El elemento ***CollapseButton*** solo puede especificarse si los controles de la visualización son de tipo "***Bar***". Esto se debe a que esta funcionalidad es única para este tipo de controles.

- Los identificadores referenciados en la sección de ***Layout*** de la página deben corresponderse con los especificados a la hora de configurar cada componente.

- El filtrado de los datos por universidad solo puede estar disponible para usuarios con rol de administrador y los privilegios asociados.

Antes de comenzar la generación de código, se analiza que el fichero de configuración XML sea estructuralmente correcto mediante su validación contra el XSD, para después revisar que no hay inconsistencias con el *feature model* y sus restricciones mediante la función de Python, evitando problemas posteriores durante la generación del código asociado.

5.5. Diseño e implementación de los *core assets*

Como ya se avanzó en la sección (búsqueda de necesidades), debido a que no hay una especificación clara de las métricas a calcular, se ha optado por la creación de visualizaciones reutilizables en D3.js v4. Estas visualizaciones van a permitir, en función del tipo de variable, explorar ciertas perspectivas de los datos recolectados por el Observatorio.

Así pues, para probar esta aplicación de ingeniería de dominio y poder explotar las diversas variables recogidas por el Observatorio, se han desarrollado tres componentes visuales principales:

- Diagrama de dispersión.
- Mapa de calor.
- Diagrama de cuerdas.

El código básico de funcionamiento de cada uno de estos componentes se ha implementado en plantillas de Jinja2. Dicho código base de cada componente va a permitir instanciarlo tras pasarle una serie de parámetros, como pueden ser los datos iniciales que se van a mostrar.

Por otra parte, las funcionalidades opcionales han sido implementadas también en Jinja2, pero el código asociado a cada una de dichas funcionalidades está encapsulado en macros. Esta metodología permite aislar las características específicas de cada componente para finalmente fusionarlas o inyectarlas en el código base, obteniendo el código fuente específico del componente configurado. De esta manera, si un usuario no quiere que cierto componente ofrezca una característica por determinadas razones (porque puede entorpecer la interacción con el mismo o porque directamente no la contempla en sus necesidades), ésta no se incluirá en el código final, con lo que se proporciona un componente hecho "a medida".

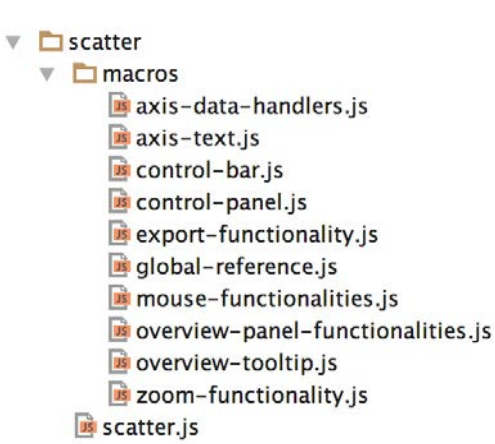

Figura 22. Core assets del componente Scatter Diagram.

En el caso del componente que implementa el diagrama de dispersión, encontramos los *core assets* que se muestran en la Figura 22. Como puede observarse, el código está organizado según las funcionalidades que se implementan a través de macros. El fichero principal, en este caso "scatter.js", implementa la lógica base del componente y contiene la referencia a las diversas funcionalidades codificadas en los macros.

En función de una serie de sentencias condicionales, que toma la configuración del componente a generar, se inyectará o no la característica o funcionalidad según los requisitos expresados.

La llamada a los macros se realiza mediante la sintaxis de Jinja2. Si se cumple la condición, el código asociado será inyectado en el punto donde se llama a la macro; en caso contrario, se ignorará y el componente se generará sin la funcionalidad definida. En la Figura 23 pueden observarse diversas llamadas a macros: código asociado al título del componente, creación de los contenedores necesarios para la barra o el panel de control, funcionalidades asociadas a los ejes de coordenadas, etc.

```
function my(selection) {

    selection.each(function () {
            var tooltipScatterDiagram = d3.select("body").append("div")
                .attr("class", "tooltip")
                .attr("id", "compare-tooltip")
                .style("display", "none")
                .style("opacity", 0);


            {{ chart_title.render_chart_title() }}
            {{ control_bar.render_control_bar() }}
            {{ render_structure.render_component_structure() }}
            {{ control_panel.render_control_panel('query_handler', 'vis_id') }}
            {{ export_functionality.export() }}
            {{ overview_tooltip.create_overview_tooltip('vis_id') }}
            {{ axis_functionality.render_axis_handlers('xText', 'yText', 'vis_id') }}

            xScale = d3.scaleLinear()
                .range([0, width]);

            yScale = d3.scaleLinear()
                .range([height, 0]);

            rScale = d3.scaleLog()
                .range([10, radius]);
```

Figura 23. Ejemplos de llamadas a macros en el código base del componente Scatter Diagram.

El uso de macros hace la inyección de código muy flexible, puesto que las llamadas a los macros pueden realizarse en cualquier punto de la plantilla.

Una vez se realiza la llamada al macro, se evalúa la condición asociada a la funcionalidad que se implementa en la misma. Por ejemplo, en la Figura 24, se observa el código asociado a la funcionalidad de exportación de la visualización a formato PNG. Si en la configuración del componente se ha

especificado la funcionalidad "*Exportation*", el código contenido en la macro se inyectará en el punto definido del código base. En caso contrario, se ignorará.

```
{% macro export() %}
{% if Component|check('Exportation') == 'True' %}
d3.select("#save-{{ Component['@component_id'] }}")
    .on("mouseover", function () {
        d3.select(this).style("cursor", "pointer");
        d3.select(this).style("opacity", 1);
    })
    .on("mouseout", function () {
        d3.select(this).style("cursor", "default");
        d3.select(this).style("opacity", 0.3);
    })
    .on("click", function () {
        d3.select(this).style("opacity", 0);
        saveSvgAsPng(d3.select("#original_svg_{{ Component['@component_id'] }}").node(),
            "{{ Component['@component_id'] }}" + '.png',
            {backgroundColor: 'white', scale: 4});
    });
{% endif %}
{%- endmacro %}
```

Figura 24. Macro con el código asociado a la funcionalidad de exportación de las visualizaciones.

Con esta misma lógica se implementan cada uno de los puntos de variabilidad definidos en el modelo de la línea de productos, enlazando la especificación con la implementación final. En la Figura 25 puede verse un esquema simplificado de cómo funciona este proceso de inyección de código.

A su vez, el uso de plantillas hace el código mucho más mantenible. A parte de separar las funcionalidades en diversas macros y obtener un grado muy fino de personalización a nivel funcional en los componentes, el aislamiento de las mismas respecto al código principal permite modificar, añadir o incluso eliminar (dentro de las restricciones especificadas en la SPL) una funcionalidad sin que afecte al resto. Añadido a esto, con solo modificar el código en las plantillas es posible propagar los cambios en todos los *dashboards* anteriormente generados de forma sencilla, regenerando el código con las actualizaciones introducidas.

El proceso para añadir un nuevo componente sería como sigue:

1. Añadirlo al *feature model* para que quede constancia del mismo a nivel abstracto.
2. Especificar las diversas características (obligatorias, alternativas, opcionales, etc.).
3. Añadirlo al esquema XML para permitir su selección en la fase de ingeniería de aplicación.
4. Programar el código base del componente, así como sus funcionalidades opcionales encapsuladas en macros.
5. Hacer disponible el nuevo componente para el generador de código, indicando dónde se localizan las plantillas que lo implementan.

Al finalizar este proceso se obtendría un nuevo *core asset* para la línea de productos que podría incluirse y configurarse en los *dashboards* de cualquier usuario. El proceso de creación de un *core asset* consume recursos, evidentemente, pues no deja de ser un proceso de programación al que se le añade la fase de ingeniería de dominio y de identificación de características. Sin embargo, y es uno de los puntos fuertes del paradigma de las SPL, una vez creado puede reutilizarse las veces necesarias a lo largo de cualquiera de los productos finales de la línea.

Este enfoque también permite tener funcionalidades compartidas para cada uno de los componentes, es decir, macros con código común respecto a ciertas funcionalidades, lo que aumenta la reutilización de código incluso a nivel de plantilla. Por ejemplo, el código para implementar los filtros de datos a nivel de componente es común para los tres tipos de visualizaciones, dado que los conectores encargados de comunicarse con la API tienen la misma interfaz a la hora de filtrar datos, con lo cual puede utilizarse el mismo código para instancias distintas de cada componente.

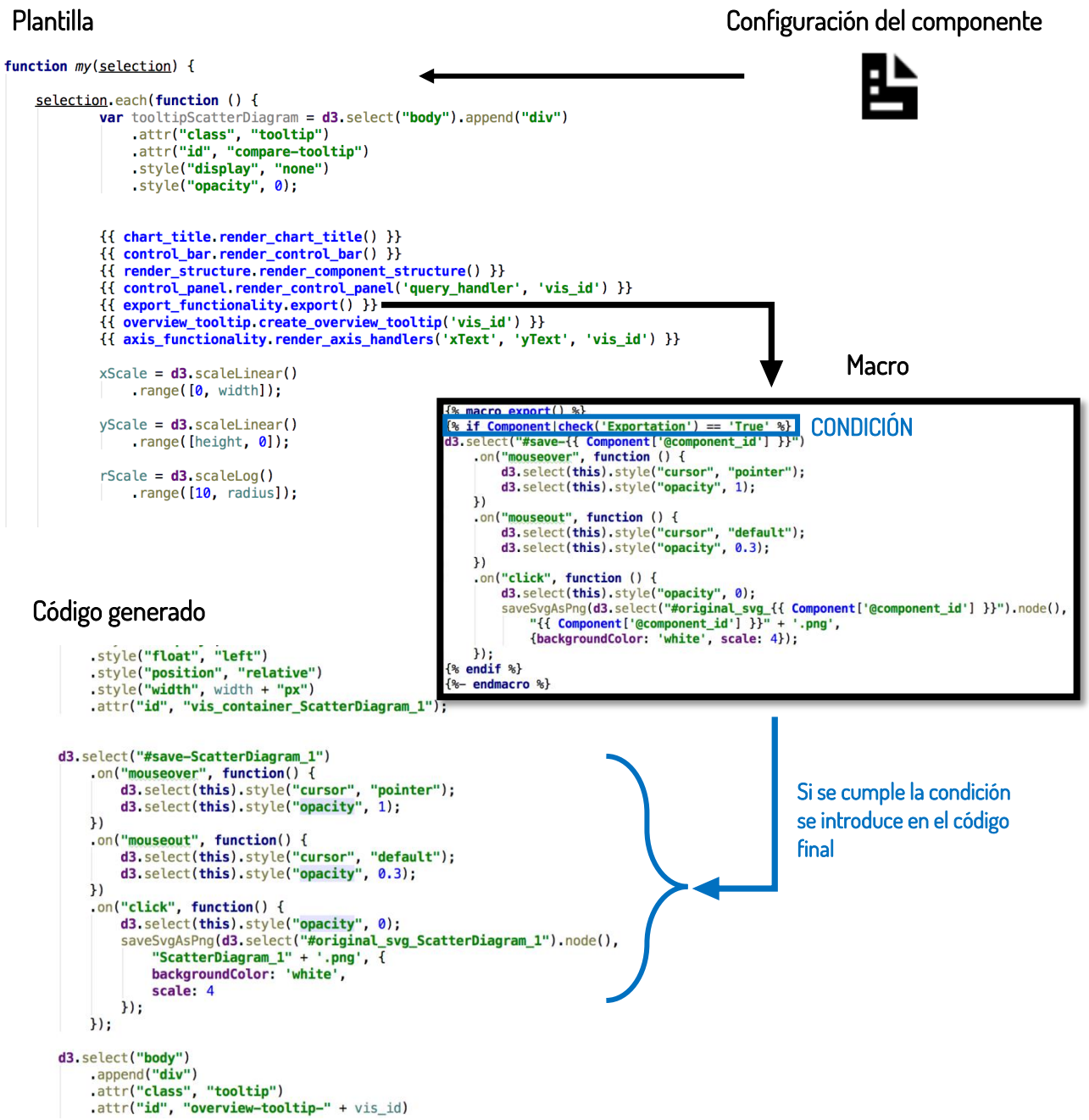

Figura 25. Esquema del funcionamiento de generación de código a través de plantillas Jinja2.

Para instanciar cada uno de los componentes configurados para cada una de las páginas del *dashboard*, también se genera un *script* principal donde se inicializan diversos elementos (visualizaciones, peticiones iniciales a la API, funciones de redimensionado, etc.). Esta tarea se lleva a cabo mediante bucles, también soportados por Jinja2, que permiten recorrer los ficheros de configuración e instanciar los componentes especificados en los mismos (Figura 26).

```
{% for component in Components.Component %}
{% set outer_loop = loop %}
{% for type, configuration in component.items() %}

/**
 * SCATTER DIAGRAM QUERY INITIALIZATION
 */

{% if type == 'ScatterDiagram' %}
{% if configuration.InitialData.Category %}
qh_{{ configuration['@component_id'] }}
    .initC('{{ configuration.InitialData.Category.DataResource.DataSource.code|safe }}');
{% else %}
qh_{{ configuration['@component_id'] }}.initC('TOTAL');
{% endif %}
{% if configuration|check('Controls.DataSelectors') %}
qh_{{ configuration['@component_id'] }}
    .initX('{{ configuration.Controls.DataSelectors.xAxis.DataResource.DataSource.0.code|safe }}',
    '{{ configuration.Controls.DataSelectors.xAxis.DataResource.DataSource.0.metric_code|safe }}', 'x',
    '{{ configuration.Controls.DataSelectors.xAxis.DataResource.DataSource.0.endpoint|safe }}');
qh_{{ configuration['@component_id'] }}.initY('{{ configuration.Controls.DataSelectors.yAxis.DataResource.Da
```

Figura 26. Bucles en la plantilla del script principal para instanciar cada uno de los componentes configurados.

El uso de bucles y este tipo de directivas hace muy potentes a las plantillas, puesto que añadir un nuevo componente e instanciarlo en este *script* principal solo implica añadir una nueva porción de código que se encargue de la instanciación. Además, se permiten instanciaciones de un mismo componente con distintas configuraciones, hecho que potencia la reutilización del código base de los componentes en una misma página (Figura 27).

```
{% elif type == 'Heatmap' %}

/**
 * HEATMAP INSTANTIATION
 */


var c_{{ configuration['@component_id'] }} = ({{ configuration['@component_id'] }})()
    .margin_top(30)
    .margin_bottom(0)
    .query_handler(qh_{{ configuration['@component_id'] }})
    .width($("#" + c_id).width())
    .height($("#" + c_id).height())
    {% if configuration.Dimensions %}
    {% for dim in configuration.Dimensions.Dimension %}
    .data_{{ dim['@dimension_id'] }}(rs['data']['{{ dim.DataResource.DataSource.endpoint }}']
        ['hm_{{ dim['@dimension_id'] }}'])
    {% endfor %}
    {% else %}
    .data(rs['data']['{{ configuration.Dimension.DataResource.DataSource.endpoint }}']
        ['hm_{{ configuration.Dimension['@dimension_id'] }}'])
    {% endif %}
    .cScale(d3.interpolateGnBu);
d3.select("#" + c_id).call(c_{{ configuration['@component_id'] }});
qh_{{ configuration['@component_id'] }}.setViz(c_{{ configuration['@component_id'] }});
```

Figura 27. Instanciación del componente "mapa de calor" mediante directivas Jinja2.

Finalmente, a través de otra plantilla de Jinja2, se genera el documento HTML que contiene la lógica de cada página, donde, además, se importan todos los ficheros de JavaScript necesarios para el funcionamiento del *dashboard*. Para ello, de forma recursiva, se construye el *layout* especificado en los ficheros de configuración y se importa el código JavaScript previamente generado para componente.

```
{% if layout.Component %}
    <div id='{{ layout.Component['@ref'] }}' class="component"></div>
{% elif layout.DataFilter %}
    <div id='{{ layout.DataFilter['@ref'] }}'
         style='...'></div>
{% else %}
    {% for container, children in layout.items() %}
        {% if container == "Row" %}
            <div class="row" style="...">
                {% with layout=children %}
                    {% include include_url+'html/components/layout-constructor.html' %}
                {% endwith %}
            </div>
        {% elif container == "RowGroup" %}
            {% for row in children.Row %}
                <div class="row" style="...">
                    {% with layout=row %}
                        {% include include_url+'html/components/layout-constructor.html' %}
                    {% endwith %}
                </div>
            {% endfor %}
        {% elif container == "Column" %}
            <div class="col-lg col-md col-sm" style="height: {{ children['@height'] }};
                    width: {{ children['@width'] }};">
                {% with layout=children %}
                    {% include include_url+'html/components/layout-constructor.html' %}
                {% endwith %}
            </div>
        {% elif container == "ColumnGroup" %}
            {% for column in children.Column %}
                <div class="col-lg col-md col-sm"
                     style="...">
                    {% with layout=column %}
                        {% include include_url+'html/components/layout-constructor.html' %}
                    {% endwith %}
                </div>
            {% endfor %}
        {% endif %}
    {% endfor %}
{% endif %}
```

Figura 28. Uso de recursividad en plantillas para generar el layout de la página HTML de los dashboard.

El uso de recursividad es muy beneficioso en este caso, puesto que permite generar cualquier estructura en base a combinaciones de filas y columnas (Figura 28). La recursividad tiene su fin cuando se encuentra un "nodo hoja", que en este caso será un componente o el propio filtro de datos general.

5.6. API GraphQL

Uno de los aspectos básicos de un *dashboard* son los datos que muestra. Estos pueden provenir de distintas fuentes y en diversos formatos, así como extraerse de archivos estáticos o provenir de llamadas a APIs externas. Para el ecosistema del Observatorio es fundamental poder recuperar los datos de su banco de conocimiento de una forma desacoplada e interoperable; por ello, se ha propuesto la implementación de una API GraphQL.

La elección de GraphQL como tecnología para la implementación de esta API no es arbitraria, puesto que sus características hacen a este lenguaje de consultas aplicable a la línea de productos de *dashboards* del Observatorio por una serie de razones:

- Las consultas son parametrizables, esto es, permiten variar ciertos campos de consulta que, una vez procesados en el *back-end*, producirán diversos resultados.

- Cada llamada a la API permite especificar exactamente qué atributos se recuperarán. Al contrario que las APIs basadas en REST, por ejemplo, donde una llamada para recuperar un recurso nos devolverá todos los atributos asociados, sean o no necesarios para el usuario que ha ejecutado la llamada. Esto mejora el aprovechamiento del ancho de banda cuando se recupera información.

- Se pueden realizar varias consultas en la misma llamada y especificar un alias para cada una de ellas. De nuevo, esto es beneficioso de cara al rendimiento, siendo necesaria una única llamada para recuperar más de un recurso.

El Observatorio no almacena métricas estáticas en su banco de conocimiento, éstas son calculadas a demanda en función de su necesidad. Esto quiere decir que es necesario procesar los datos antes de visualizarlos. Para hacer el proceso más eficiente, se decide realizar los cálculos de métricas a demanda en el *back-end*, descargando de esta tarea a los navegadores de los usuarios que acceden a la página del *dashboard*.

Sin embargo, esto supone un problema de cara a la recuperación de métricas. GraphQL permite convertir lógicamente cada entidad del almacenamiento persistente en nodos de un grafo, así pues, para recuperar información de una entidad particular, haríamos una consulta contra dicho nodo para recuperar los atributos requeridos.

En la Figura 29 puede observarse un ejemplo de la representación lógica de las entidades mediante GraphQL. Cada entidad sería representada mediante un nodo, el cual puede tener una serie de atributos (nodos hoja) y relaciones con otros nodos.

Para obtener los datos, se utilizaría la sintaxis de GraphQL para indicar exactamente qué elementos del grafo se quieren recuperar, siendo devueltos única y exclusivamente aquellos que se han especificado de forma explícita en la consulta. Puede observarse un ejemplo esquemático en la Figura 30, donde se puede seleccionar una serie de atributos de un nodo en concreto, y otra serie de atributos distintos en otros nodos, ofreciendo grandes libertades al cliente a la hora de recuperar datos.

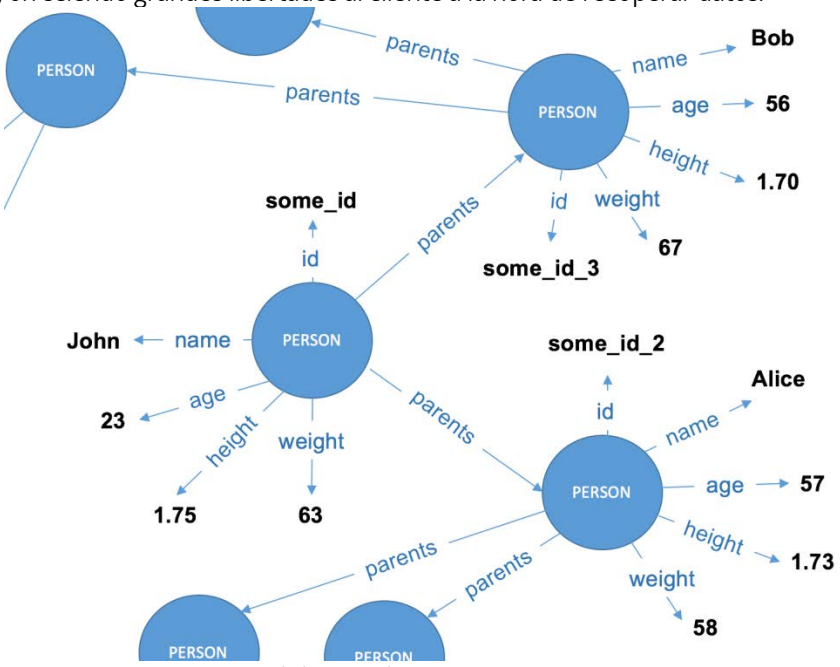

Figura 29. Ejemplo lógico de la representación de datos con GraphQL.

Como ya se ha mencionado, el Observatorio no retiene métricas en su almacenamiento persistente, con lo que esta aproximación no es completamente aplicable.

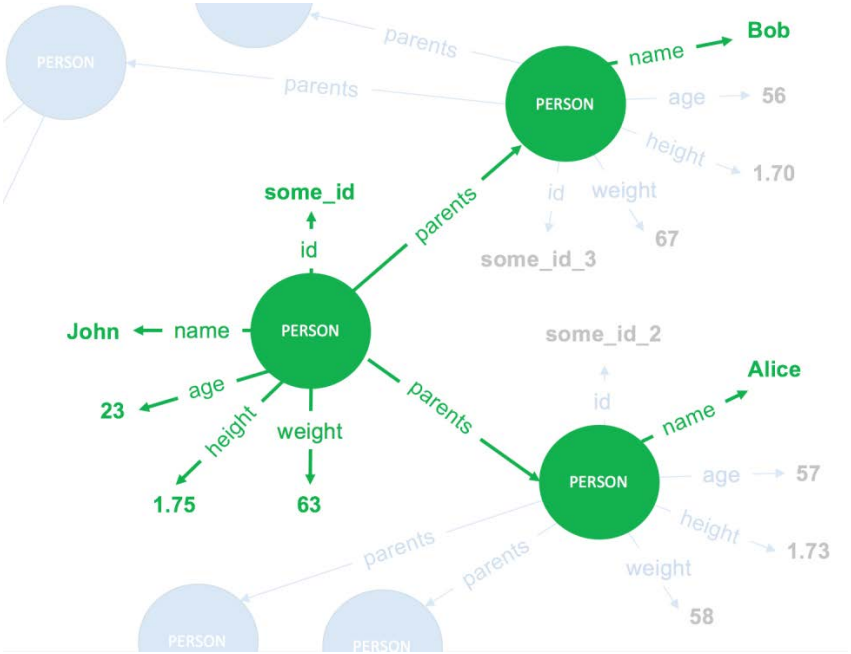

Figura 30. Ejemplo lógico de la recuperación de datos mediante GraphQL.

Por ello, cada uno de los nodos que conforman la API de GraphQL implementada representa un tipo de métrica (un cálculo de porcentajes agregados, relación entre variables categóricas, etc.), a la cual, mediante una serie de parámetros, se le especifica exactamente sobre qué campos del banco de conocimiento del Observatorio se deben aplicar dichos cálculos, para posteriormente devolverlos a la aplicación que ha realizado la llamada.

Los parámetros, en general, son el nivel de agregación y el código del campo o de los campos sobre los que se quiere realizar el cálculo. De esta forma se flexibiliza el cálculo de métricas y se permite la reutilización de la lógica para obtenerlas de cara a futuras ediciones.

Se ha hecho uso de esta metodología para abstraer el *back-end* del *front-end*. Al utilizar códigos que identifican a los campos del almacenamiento persistente se añade una capa de abstracción respecto a cómo se especifican en el *back-end* dichos campos. Este método, aparte de abstraer dicho funcionamiento interno, hace el mantenimiento de la API más sencillo, puesto que en caso de que exista algún cambio en el *backend* a nivel de datos, simplemente es necesario modificar la especificación de los campos asociados a un código, sin afectar a los consumidores de la API, puesto que el código se mantendría tal y como se encontraba.

Debido a la cantidad de variables disponibles, la creación de los distintos códigos se ha realizado de manera automatizada, recuperando del almacenamiento persistente los tipos de variables recogidas y asociándoles el código correspondiente. Así, cualquier cambio en el esquema de la base de datos se puede propagar automáticamente, de forma que los códigos se encuentren permanentemente actualizados.

Para mostrar esta aproximación, se muestran a continuación una serie de ejemplos de cómo funcionaría la recuperación y cálculo de métricas sobre los resultados del Observatorio.

El primer ejemplo muestra una consulta para calcular la media agregada por ramas de conocimiento del nivel de todas las competencias que poseen los titulados de la edición 2017 del estudio (Figura 31).

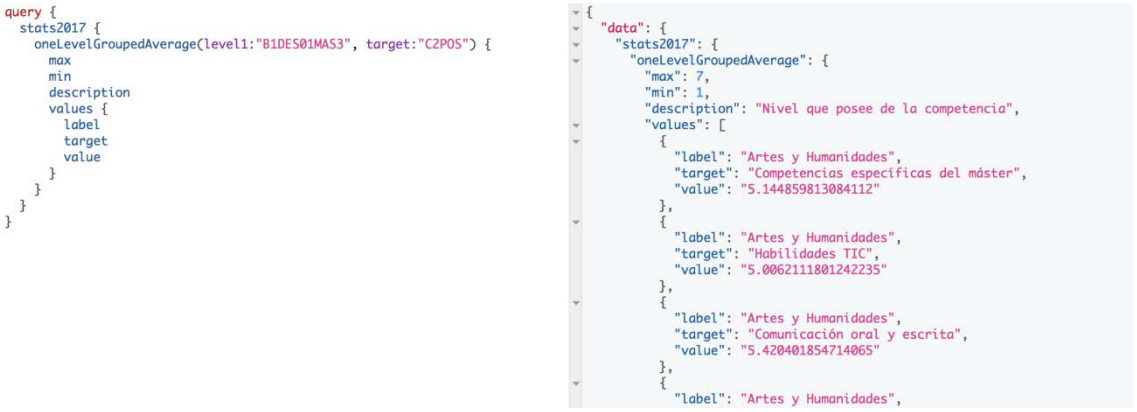

Figura 31. Ejemplo de una consulta a la API GraphQL del Observatorio para calcular la media de competencias poseídas.

En primer lugar, se accede al nodo que representa las métricas de la edición de 2017 (***stats2017***) y se selecciona el nodo correspondiente al cálculo de medias (***oneLevelGroupedAverage***). Mediante los parámetros ***level1*** y ***target*** se especifica la variable por la que se agregarán los datos (en este caso, B1DES01MAS3 se corresponde con el código asociado a las ramas de conocimiento, esto es, los datos vendrán agregados por rama de conocimiento) y el/los campos sobre los que se ejecutarán los cálculos (en este caso, C2POS se corresponde con el código que agrupa todos los campos referentes a las competencias poseídas por los titulados). Finalmente, se especifican los atributos que se recuperarán; en este ejemplo se recuperan el máximo y mínimo valor posible de la métrica calculada, una descripción y la lista de resultados.

Como se señaló anteriormente, pueden calcularse varias métricas a través de una sola consulta. En la Figura 32 se recupera el nivel de todas las competencias de los titulados de la edición 2017, tanto el nivel poseído (C2POS), como el requerido en el trabajo (C1EMP) y el obtenido en la universidad (C3UNI).

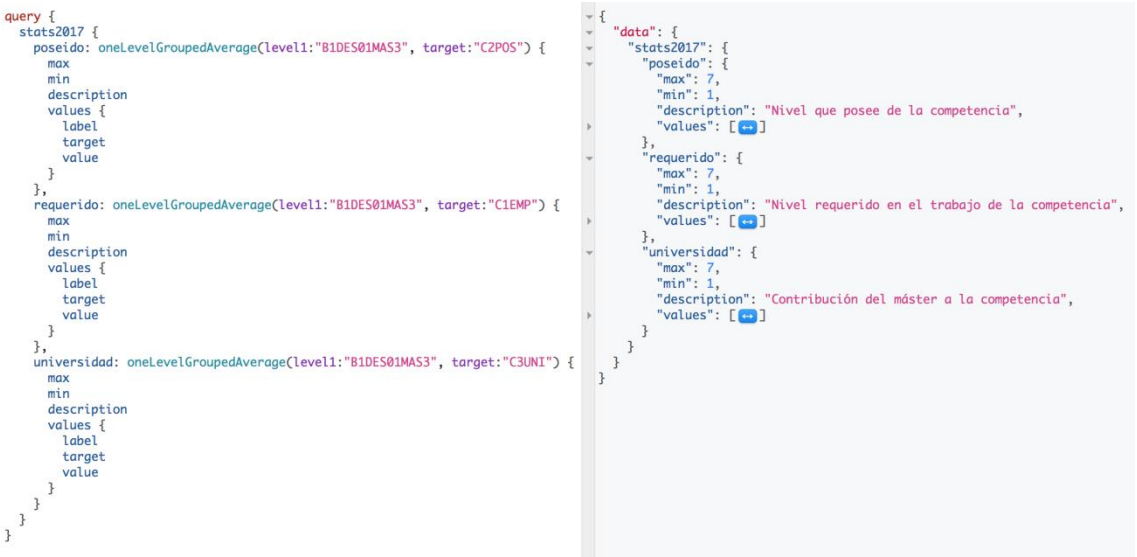

Figura 32. Ejemplo de una consulta a la API GraphQL del Observatorio para calcular la media de competencias poseídas, requeridas en el último empleo y obtenidas en la universidad.

Si se quisieran agregar los resultados por género, solamente sería necesario cambiar el parámetro "**level1**", a través del cual se especifica la variable categórica a través de la que se agregarán los datos (Figura 33). El código A1PER01SEXO se corresponde con dicha variable categórica.

```
query {
  stats2017 {
    oneLevelGroupedAverage(level1:"A1PER01SEXO", target:"C2POS") {
      max
      min
      description
      values {
        label
        target
        value
      }
    }
  }
}
```


```
{
  "data": {
    "stats2017": {
      "oneLevelGroupedAverage": {
        "max": 7,
        "min": 1,
        "description": "Nivel que posee de la competencia",
        "values": [
          {
            "label": "Hombre",
            "target": "Competencias específicas del máster",
            "value": "5.181988742964353"
          },
          {
            "label": "Hombre",
            "target": "Habilidades TIC",
            "value": "5.347269303201506"
          },
          {
            "label": "Hombre",
            "target": "Comunicación oral y escrita",
            "value": "5.406015037593985"
          },
          {
            "label": "Hombre",
            "target": "Comunicación oral y escrita en otros idiomas",
            "value": "4.800449269936354"
          },
          {
```


Figura 33. Especificación de agregación de resultados por sexo en una consulta a la API GraphQL.

Otro aspecto muy beneficioso del uso de GraphQL, como puede observarse en los ejemplos, es que los datos son devueltos con la misma estructura que la especificada en la consulta.

Por otra parte, para filtrar los datos, simplemente es necesario añadir otro parámetro a la consulta que recupere los datos cumpliendo la condición de filtrado especificada. En la Figura 34, se están recuperando únicamente los datos referentes a los estudiantes de Artes y Humanidades (cuyo código de rama de conocimiento es el número 1).

```
query {
  stats2017(RamaConocimiento: 1) {
    oneLevelGroupedAverage(level1:"A1PER01SEXO", target:"C2POS") {
      max
      min
      description
      values {
        label
        target
        value
      }
    }
  }
}
```


```
{
  "data": {
    "stats2017": {
      "oneLevelGroupedAverage": {
        "max": 7,
        "min": 1,
        "description": "Nivel que posee de la competencia",
        "values": [
          {
            "label": "Hombre",
            "target": "Competencias específicas del máster",
            "value": "5.270491803278689"
          },
          {
            "label": "Hombre",
            "target": "Habilidades TIC",
            "value": "5.170124481327801"
          },
          {
            "label": "Hombre",
            "target": "Comunicación oral y escrita",
            "value": "5.520491803278689"
          },
          {
            "label": "Hombre",
            "target": "Comunicación oral y escrita en otros idiomas",
            "value": "4.959349593495935"
          },
          {
            "label": "Hombre",
            "target": "Capacidad de organización",
            "value": "5.493877551020408"
          },
          {
```


Figura 34. Filtrado de datos en una consulta a la API GraphQL.

También se han implementado nodos que calculan la frecuencia de aparición de determinadas variables (Figura 35) en forma de porcentajes, nodos que calculan la población de las diversas categorías dada una variable categórica o nodos que calculan la tabla de contingencia dadas dos variables categóricas.

```
query {
  stats2017(RamaConocimiento: 1) {
    oneLevelGroupedPercentage(level1:"A1PER01SEXO", target:"E2BUS01DIRC") {
      max
      min
      description
      values {
        label
        target
        value
      }
    }
  }
}
```

```
{
  "data": {
    "stats2017": {
      "oneLevelGroupedPercentage": {
        "max": 100,
        "min": 0,
        "description": "% de titulados utilizaron este método",
        "values": [
          {
            "label": "Hombre",
            "target": "Autopresentación y contacto directo",
            "value": "57.534246575342465"
          },
          {
            "label": "Mujer",
            "target": "Autopresentación y contacto directo",
            "value": "60.54054054054055"
          }
        ]
      }
    }
  }
}
```

Figura 35. Ejemplo de cálculo de frecuencias a través de peticiones a la API de GraphQL.

Con estos "nodos" (Figura 36) se cubren las necesidades del Observatorio en cuanto a cálculo de métricas básicas para las dos ediciones de sus estudios, aunque, si en algún momento se necesitase otro tipo de cálculos, bastaría con implementar un nuevo nodo que se encargase de los nuevos cálculos sin que esto afecte al resto. A su vez, este esquema es muy escalable, puesto que introducir los resultados de consecuentes ediciones de los estudios del Observatorio solo implicaría añadir un nuevo nodo al grafo, de nuevo, sin afectar al resto ni modificar el *endpoint* de la API.

Además, cada uno de ellos permite alimentar a los tipos de componentes presentes en los *dashboards*, facilitando la tarea de la visualización de información gracias a los diversos atributos que pueden recuperarse (valores máximos y mínimos, descripciones, unidades, etc.), desacoplando la parte funcional de los datos que representa.

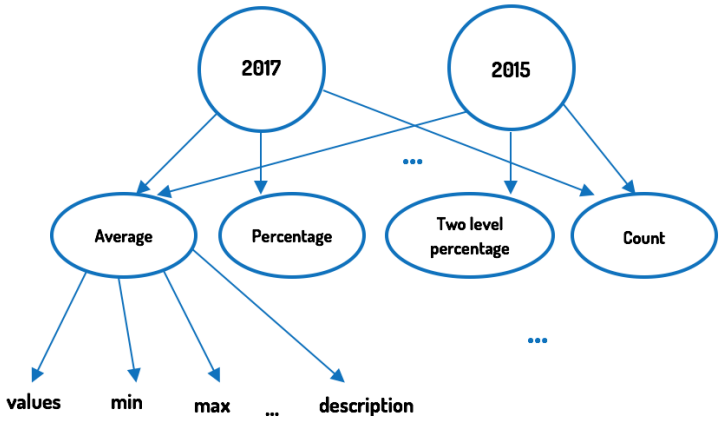

Figura 36. Grafo esquemático de la API GraphQL del Observatorio.

Finalmente, para que los componentes visuales del *dashboard* puedan realizar las peticiones a esta API de forma desacoplada, se han implementado "conectores" específicos que tiene en cuenta sus características. Estos conectores sirven de puente entre la visualización y la API GraphQL, encargándose de realizar las consultas requeridas en cada momento.

Evidentemente cualquier cambio en la API va a acarrear cambios en el conector. Sin embargo, al ser un componente independiente de las visualizaciones, los cambios solo afectarían al código referente al conector, desacoplándolo de los componentes gráficos. En el caso de tener otras fuentes de datos independientes a las internas, sería necesario realizar un nuevo conector basado en las especificaciones de la nueva fuente de datos, el cual pasaría a ser un nuevo *core asset* reutilizable.

Una vez especificado el nexo entre las visualizaciones y la API, es necesario que en los ficheros de configuración se encuentren el/los conjuntos de variables o fuentes de datos de los que van a consumir los componentes. Para ello se ha implementado un *script* que recoge todas las variables disponibles para la exploración, para estructurarlas en ficheros XML (donde se especifica el nombre de la variable y el código de la API GraphQL asociado a la misma). Esto permite tener el catálogo de variables siempre actualizado, dado que, si se produce algún cambio en la capa de datos del Observatorio, solo es necesario regenerar los ficheros XML mediante el script.

```
<DataSource>
    <label>Residencia durante los estudios</label>
    <code>A3RES02ESTU</code>
    <endpoint>stats2017</endpoint>
</DataSource>
<DataSource>
    <label>Rama de conocimiento</label>
    <code>B1DES01MAS3</code>
    <endpoint>stats2017</endpoint>
</DataSource>
<DataSource>
    <label>Especialidad del máster</label>
    <code>B1DES02ESPT</code>
    <endpoint>stats2017</endpoint>
</DataSource>
<DataSource>
    <label>Titularidad del máster</label>
    <code>B1DES04TIT5</code>
    <endpoint>stats2017</endpoint>
```

Figura 37. Ejemplo de variables disponibles para agregar los datos mostrados.

Cuando un desarrollador se disponga a configurar un producto de la línea, accederá a dicho catálogo de variables e irá introduciéndolas como fuentes de datos en los diversos componentes que formarán parte del *dashboard*, como se ejemplifica en la Figura 38, donde los datos iniciales del diagrama de dispersión configurado mostrarán el valor medio del grado de preparación para buscar empleo en su eje X, y la capacidad media para trabajar en un equipo interdisciplinar en su eje Y, agregados por rama de conocimiento. En este ejemplo puede observarse cómo el elemento "**DataResource**" (explicado en la sección 5.4) permite englobar y abstraer las características específicas de las fuentes de datos.

```
<Component>
    <ScatterDiagram component_id="ScatterDiagram_1">
        <Title>Explorador de datos</Title>
        <Zoom>True</Zoom>
        <Exportation>True</Exportation>
        <InitialData>
            <xAxis>
                <DataResource>
                    <DataSource>
                        <label>Grado de preparación para buscar empleo</label>
                        <code>C5BUS01CAPC</code>
                        <metric_code>oneLevelGroupedAverage</metric_code>
                        <endpoint>stats2017</endpoint>
                    </DataSource>
                </DataResource>
            </xAxis>
            <yAxis>
                <DataResource>
                    <DataSource>
                        <label>Capacidad para trabajar en un equipo interdisciplinar</label>
                        <code>C1EMP05IEID</code>
                        <metric_code>oneLevelGroupedAverage</metric_code>
                        <endpoint>stats2017</endpoint>
                    </DataSource>
                </DataResource>
            </yAxis>
            <Category>
                <DataResource>
                    <DataSource>
                        <label>Rama de conocimiento</label>
```

Figura 38. Configuración de un componente de tipo "diagrama de dispersión", donde se inicializan cada una de las dimensiones con una variable del catálogo disponible.

De esta forma se permite también personalizar las fuentes de datos de cada uno de los elementos, dado que introducir todas y cada una de las variables disponibles puede llegar a entorpecer el uso del *dashboard* al contar con demasiadas opciones y variables a explorar. Además, si se diese el caso de que los requisitos de información de un usuario cambiasen, solo sería necesario reconfigurar las fuentes de datos, sin afectar al resto de la configuración.

5.7. Generador de código

Finalmente, se describe el generador de código que orquesta la producción de *dashboards* customizados para los usuarios del Observatorio. Debido a que los principales implicados en el proceso de configuración son los desarrolladores, se ha considerado necesario utilizar dos enfoques para esta generación de archivos fuente:

- **Generación estática:** tomando los ficheros de configuración de cada uno de los usuarios, se generan y despliegan en los directorios correspondientes cada uno de los ficheros fuente generados, para posteriormente ser servidos vía HTTP.

- **Generación dinámica:** dado que la generación estática requiere la ejecución manual de un *script*, los desarrolladores pueden encontrar poco eficiente este proceso, sobre todo si es necesario realizar pruebas de concepto de nuevos componentes. Por ello, se propone también la generación dinámica del código a través de peticiones HTTP, donde se generarán y devolverán a demanda del desarrollador los ficheros necesarios.

Estas dos aproximaciones a la generación de código son similares, sin embargo, cada una reporta ciertos beneficios en función del contexto de aplicación.

En el caso de la generación estática, los ficheros fuentes se sirven de manera más rápida al no necesitar ser generados en el momento en el que se realiza la petición, lo cual beneficia a los usuarios al presentar tiempos de respuesta menores. Cualquier cambio en los requisitos (y, consecuentemente, en los ficheros de configuración), requieren la ejecución del generador de código para actualizar los ficheros fuente del *dashboard*, por lo que se sacrifica flexibilidad a cambio de rendimiento.

Por otra parte, la generación dinámica goza de dicha flexibilidad, ya que cualquier cambio en los ficheros de configuración se hace efectivo al instante, necesitando únicamente recargar la página. Aun así, este enfoque requiere que se generen los ficheros necesarios en el momento en el que se realiza la petición, comportando tiempos de respuesta mayores.

Contar con ambos enfoques permite explotar los beneficios de ambos en diversas situaciones, como se ha comentado anteriormente.

5.8. Despliegue

Una vez realizadas todas las partes anteriores, el sistema del Observatorio con la línea de productos de *dashboards* integrada se ha desplegado en un servidor para que sea accesible de forma externa.

Debido a que ciertos datos individuales son privados en el alcance de este proyecto, se han aleatorizado campos con información sensible, como puede ser información sobre las universidades y titulaciones de cada estudiante, así como sus identificadores. Esta aleatorización no afecta a los resultados a nivel global, pero permite que no se muestre información específica de cada universidad, al haberse aleatorizado y anonimizado sus registros.

Con la base de datos aleatorizada y volcada en el servidor (el servidor utiliza *MariaDB* como tecnología para el almacenamiento persistente), se ha desplegado el sistema a través de ***nginx*** (https://nginx.org/en/), un servidor/proxy ligero, de código abierto y libre. A través de él se atenderán las peticiones para ser redirigidas a la aplicación de Django. Para que dichas peticiones lleguen finalmente a Django se utiliza ***uWSGI*** (https://uwsgi-docs.readthedocs.io/en/latest/). Este proyecto es una implementación de WSGI (Web Server Gateway Interface), una convención para servidores web que permite hacer llegar peticiones externas a aplicaciones web escritas en Python.

A su vez, se ha configurado SSL para asegurar las conexiones contra el servidor y forzar las peticiones para que siempre se realicen mediante HTTPS. Para ello, se ha generado y configurado junto con ***nginx*** un certificado mediante ***Let's Encrypt*** (https://letsencrypt.org/).

El sistema de *dashboards* se encuentra accesible en **https://oeeu-infovis.grial.eu**.

Para probar la aplicación con diversas configuraciones, se han creado tres usuarios de prueba con los que se pueden explorar los datos sobre los estudios del OEEU:

- **Usuario administrador:** revisor **Contraseña:** 8T3GDs
- **Usuario general:** U11 **Contraseña:** pxqyFz
- **Usuario general:** U28 **Contraseña:** KtQbKJ

Las configuraciones XML utilizadas para generar los *dashboards* de estos tres usuarios están disponibles en el directorio "Configuraciones de prueba/Usuarios/" del CD adjunto a esta memoria, para ser consultadas y contrastadas con el *dashboard* generado.

6. Resultados

En este apartado se muestran diversas configuraciones posibles para la posterior generación de los *dashboards* asociado a la mismas, de tal forma que se ilustre el funcionamiento de la línea de productos de *dashboards* del Observatorio de Empleabilidad y Empleo Universitarios.

6.1. Ejemplos de configuración

A continuación, se exploran combinaciones de configuraciones para un *dashboard*, así como las posibilidades que pueden llegar a ofrecer los diversos componentes. Como se verá, es posible generar *dashboards* con distintas características con solo modificar el fichero de configuración correspondiente.

Esta configuración de ejemplo consta de tres páginas que permiten explotar diversas secciones de los resultados. La primera página contendrá cuatro componentes: dos diagramas de dispersión, uno para explorar las variables recolectadas en el estudio realizado en 2015 y otro para el estudio de 2017, así como dos diagramas de cuerdas que siguen la misma lógica, pero que permitirán explorar relaciones entre variables categóricas. Esta página constará de un filtro global que permitirá análisis más profundos de los datos.

Para ello, en el fichero de configuración se especifican y configuran cada uno de los componentes según las preferencias del usuario, especificando las fuentes de datos correspondientes. En la Figura 39 se muestra de forma resumida la configuración de la página de ejemplo que se va a generar.

Posteriormente se profundizará en las características individuales y propias de cada uno de los componentes.

```
<Page page_id="1">
    <DataFilter component_id="Filter"...>
    <Components>
        <Component>
            <ScatterDiagram component_id="ScatterDiagram_1"...>
        </Component>
        <Component>
            <ScatterDiagram component_id="ScatterDiagram_2"...>
        </Component>
        <Component>
            <ChordDiagram component_id="Chord_1"...>
        </Component>
        <Component>
            <ChordDiagram component_id="Chord_2"...>
        </Component>
    </Components>
    <Layout>
        <RowGroup>
            <Row>
                <DataFilter ref="Filter"/>
            </Row>
            <Row>
                <Component ref="ScatterDiagram_1"/>
            </Row>
            <Row>
                <Component ref="ScatterDiagram_2"/>
            </Row>
            <Row>
                <ColumnGroup>
                    <Column>
                        <Component ref="Chord_1"/>
                    </Column>
                    <Column>
                        <Component ref="Chord_2"/>
                    </Column>
                </ColumnGroup>
            </Row>
        </RowGroup>
    </Layout>
</Page>
```

Figura 39. Configuración de la primera página de un dashboard.

Tras ejecutarse el generador de código, se generan los archivos fuentes que darán como resultado el *dashboard* presentado en la Figura 40.

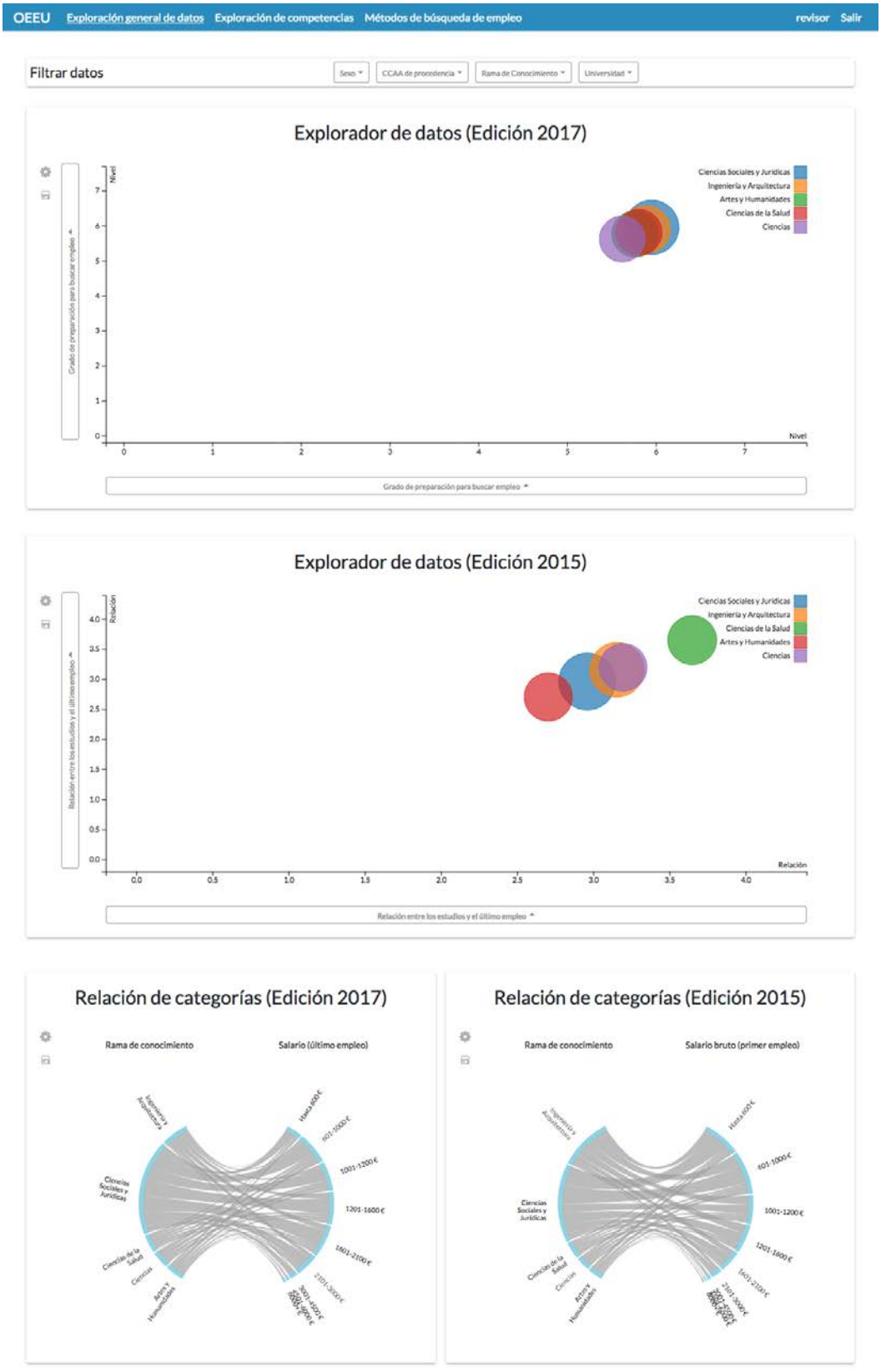

Figura 40. Resultado de la generación de código para la primera página de un dashboard.

Como puede observarse, el *dashboard* sigue la especificación del fichero de configuración mostrado anteriormente. Se describirá la configuración del primer componente (Figura 41) para poder observar la correspondencia entre el DSL y el producto finalmente generado. Para este componente se ha especificado un título concreto (en este caso nos permite especificar que se van a tratar datos de la edición de 2017), se han añadido funcionalidades de zoom y exportación, así como una barra de control con la posibilidad de modificar los datos mostrados en el eje X, el eje Y, el nivel de agregación y el radio de los elementos visualizados.

La localización de los selectores de categorías y radio es la localización por defecto, esto es, se sitúan en la barra de control. Sin embargo, la localización de los selectores de datos de los ejes X e Y se encuentran directamente en dichos ejes, ahorrando espacio en la barra de control.

También se incluye la característica "*GlobalReference*", esto es, la referencia global que permite ver en todo momento los resultados desagregados de las variables mostradas en cada eje.

```
<ScatterDiagram component_id="ScatterDiagram_1">
    <Title>Explorador de datos (Edición 2017)</Title>
    <Zoom>True</Zoom>
    <Exportation>True</Exportation>
    <InitialData>
        <xAxis...>
        <yAxis...>
        <Category...>
        <Radius...>
    </InitialData>
    <Controls type="Bar">
        <CollapseButton>True</CollapseButton>
        <DataSelectors>
            <xAxis location="Axis">
                <Description>Eje X</Description>
                <DataResource>
                    <xi:include
                            href="data-sources/1/ScatterDiagram_1/xAxis_selectors.xml"
                            xpointer="xpointer(/Root/DataSource)"/>
                </DataResource>
            </xAxis>
            <yAxis location="Axis">
                <Description>Eje Y</Description>
                <DataResource>
                    <xi:include
                            href="data-sources/1/ScatterDiagram_1/yAxis_selectors.xml"
                            xpointer="xpointer(/Root/DataSource)"/>
                </DataResource>
            </yAxis>
            <Radius location="Default">
                <Description>Radio</Description>
                <DataResource>
                    <xi:include
                            href="data-sources/1/ScatterDiagram_1/radius_selectors.xml"
                            xpointer="xpointer(/Root/DataSource)"/>
                </DataResource>
            </Radius>
            <Category location="Default">
                <Description>Agrupar por categorías</Description>
                <DataResource>
                    <xi:include
                            href="data-sources/1/ScatterDiagram_1/category_selectors.xml"
                            xpointer="xpointer(/Root/DataSource)"/>
                </DataResource>
            </Category>
        </DataSelectors>
        <GlobalReference>
            <xAxis>True</xAxis>
            <yAxis>True</yAxis>
        </GlobalReference>
    </Controls>
    <Tooltip type="Complete"/>
```

Figura 41. Detalle de la configuración del componente ScatterDiagram_1.

Para especificar las fuentes de datos, se hace uso de *XInclude* (https://www.w3.org/TR/xinclude/), una sintaxis que permite mantener el fichero de configuración más ordenado al poder separar las fuentes de datos en ficheros XML independientes, e incluirlas posteriormente mediante una referencia. De lo contrario las fuentes de datos seleccionadas podrían ocupar demasiado espacio en los ficheros de configuración y entorpecer el proceso.

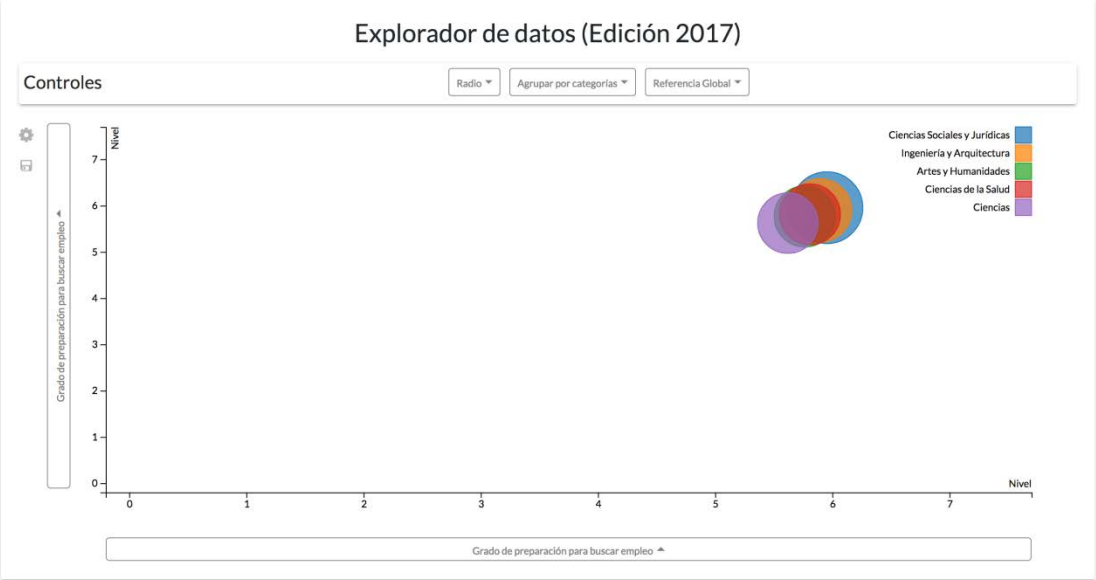

Figura 42. Detalle de la barra de control del componente ScatterDiagram_1.

Si en vez de que los controles se muestren en formato de barra (como puede verse en la Figura 42), se optase por un panel de control más amplio, solo habría que modificar dicha sección en el fichero de configuración siguiendo la sintaxis del DSL (Figura 43).

```
<Component>
    <ScatterDiagram component_id="ScatterDiagram_1">
        <Title>Explorador de datos (Edición 2017)</Title>
        <Zoom>True</Zoom>
        <Exportation>True</Exportation>
        <InitialData...>
        <Controls type="Panel">
            <CollapseButton>True</CollapseButton>
            <OverviewPanel>True</OverviewPanel>
            <DataSelectors>
                <xAxis location="Axis">
                    <Description>Eje X</Description>
                    <DataResource...>
                </xAxis>
                <yAxis location="Axis">
                    <Description>Eje Y</Description>
                    <DataResource...>
                </yAxis>
                <Radius location="Default">
                    <Description>Radio</Description>
                    <DataResource...>
                </Radius>
                <Category location="Default">
                    <Description>Agrupar por categorías</Description>
                    <DataResource...>
                </Category>
            </DataSelectors>
            <GlobalReference>
                <xAxis>True</xAxis>
                <yAxis>True</yAxis>
            </GlobalReference>
        </Controls>
        <Tooltip type="Complete"/>
    </ScatterDiagram>
</Component>
```

Figura 43. Configuración modificada para el componente ScatterDiagram_1.

El resultado de esta nueva configuración generaría un componente donde los controles estarían visibles junto a la propia visualización, como puede observarse en la Figura 44.

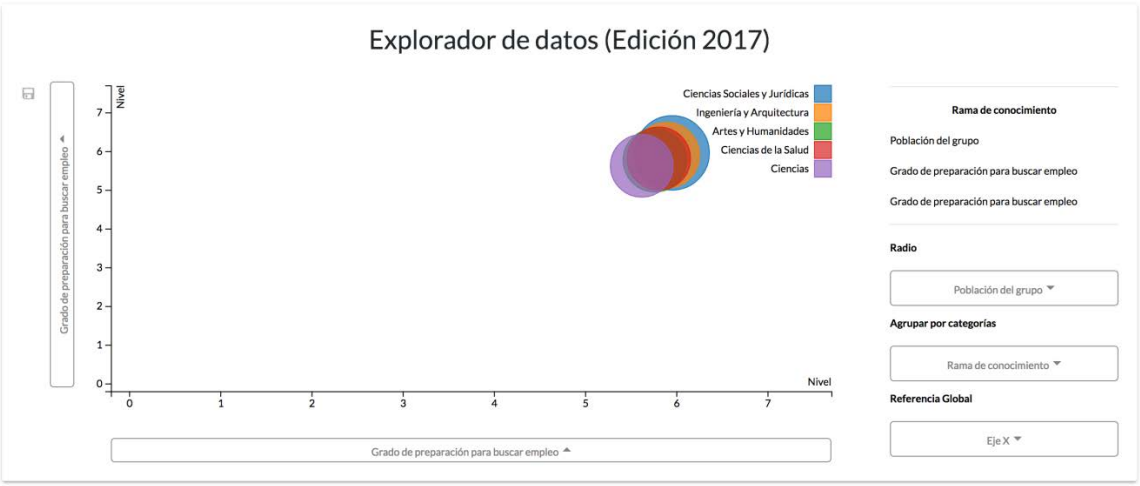

Figura 44. Componente generado con la nueva configuración.

Además, mediante el elemento *"OverviewPanel"* puede especificarse la generación de una sección en el propio panel donde ver en detalle los resultados cuando se interacciona con los mismos (Figura 45).

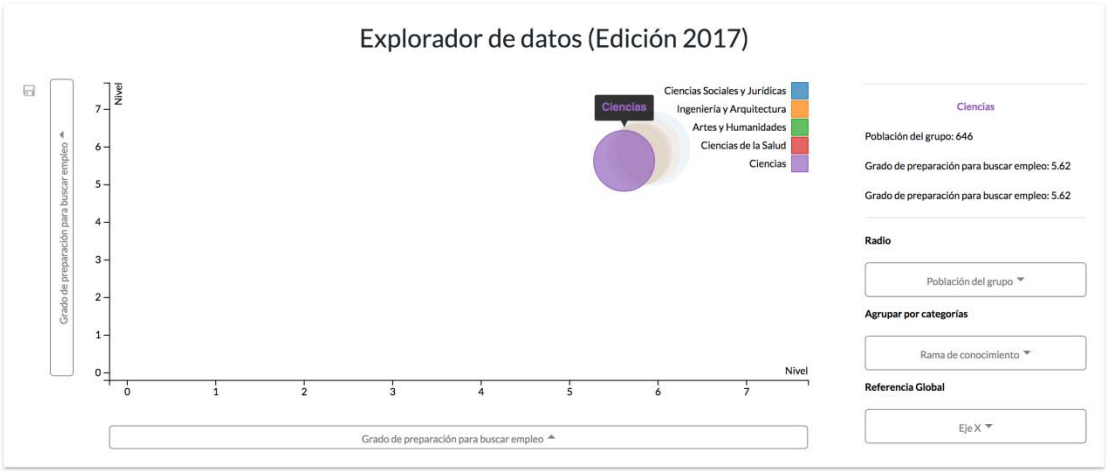

Figura 45. Detalle del panel de control cuando se interactúa con los datos

Por otra parte, también pueden especificarse filtros a nivel de componente, de tal forma que solo afecten a los datos de la visualización asociada. En el caso de que los controles sean de tipo "barra", los filtros se añadirán como una barra adicional (Figura 46).

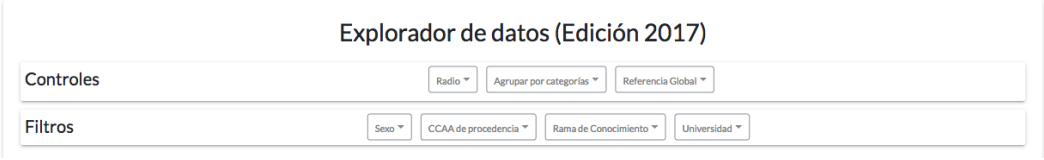

Figura 46. Filtros a nivel de componente.

En el caso de que los controles sean de tipo "panel", se añadirán en el espacio disponible. La barra que implementa los filtros es deslizable, de tal forma que el usuario puede desplazarse horizontalmente para seleccionar el filtro si el espacio disponible no es suficiente (Figura 47).

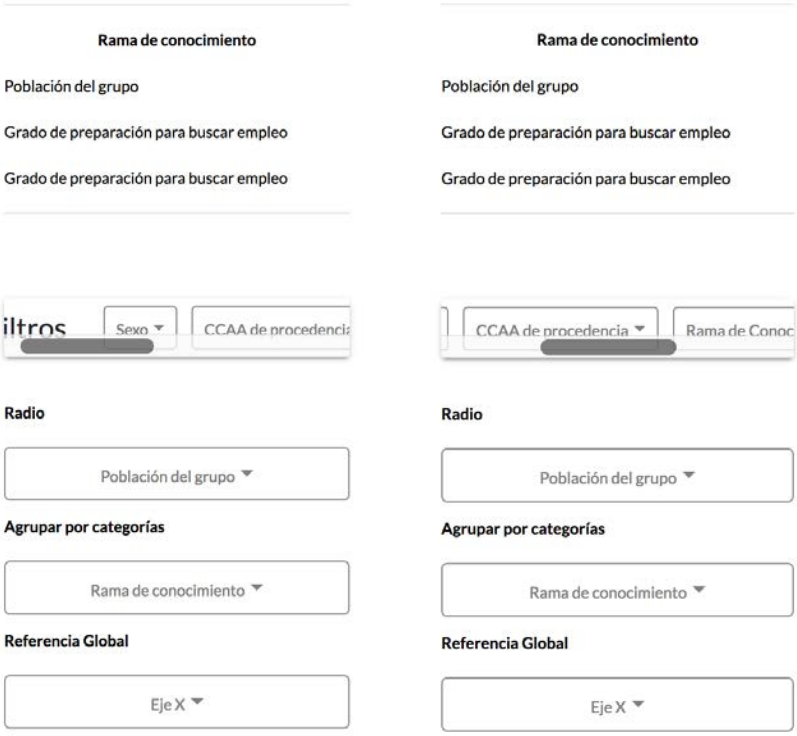

Figura 47. Detalle de los filtros a nivel de componente en el panel de control.

Incluso podría utilizarse un componente que representa un mapa de España para filtrar los datos según la comunidad autónoma de procedencia de los estudiantes mediante el elemento "*Map*" (Figura 48).

```
<Controls type="Panel">
    <OverviewPanel>True</OverviewPanel>
    <Map>
        <Filter>
            <Description>Filtro por comunidad de procedencia de los estudiantes
            </Description>
        </Filter>
    </Map>
    <DataSelectors>
        <xAxis location="Axis"...>
        <yAxis location="Axis"...>
        <Radius location="Default"...>
        <Category location="Default"...>
    </DataSelectors>
    <GlobalReference>
        <xAxis>True</xAxis>
        <yAxis>True</yAxis>
    </GlobalReference>
</Controls>
```

Figura 48. Adición del componente mapa para filtrar datos según la comunidad autónoma de procedencia de los datos mostrados por el componente.

Esta configuración añadirá un componente que representará un mapa de España (Figura 49); al interactuar con el mismo se podrán filtrar los datos según la procedencia de los estudiantes respecto a la comunidad seleccionada.

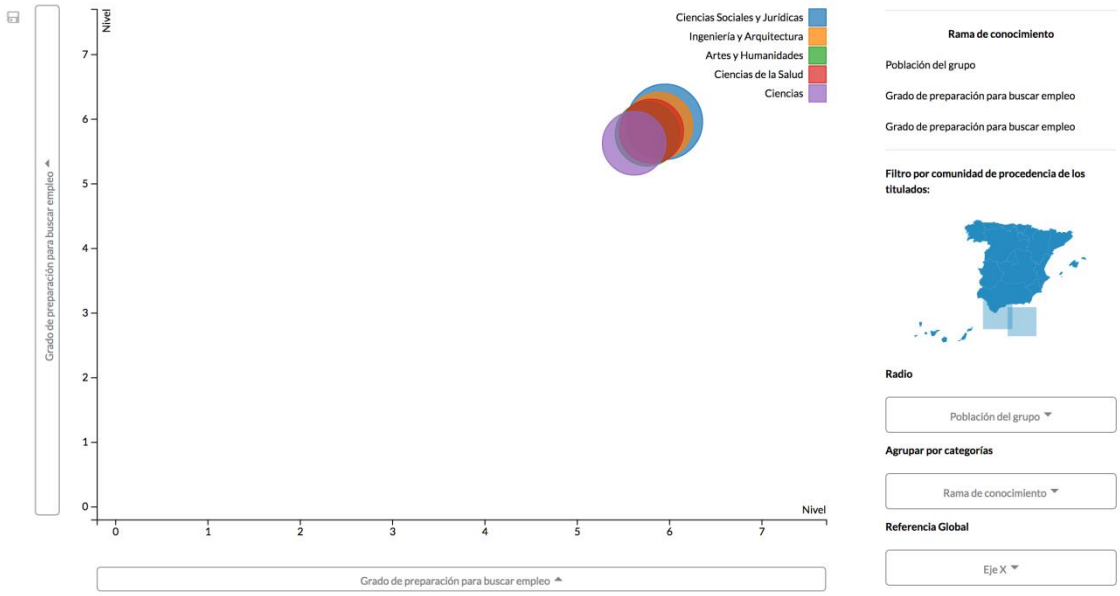

Figura 49. Componente generado tras la adición del elemento "mapa".

Para los diagramas de cuerdas se sigue la misma lógica, pudiendo variar las diversas características y fuentes de datos a través de los ficheros XML de configuración siguiendo el modelo de características.

Como ya se explicó anteriormente, pueden coexistir instancias de un mismo componente con diversas configuraciones, según las necesidades de cada usuario, como puede observarse en la Figura 51. Esto ofrece libertad para personalizar el *dashboard* tanto a nivel global (páginas que formarán parte del *dashboard*), como a nivel de página (componentes y estructura de la página) y a nivel de visualización (funcionalidades y fuentes de datos específicas de la visualización).

Este enfoque también permite mantener diversas visualizaciones con diversos objetivos en páginas independientes. En este ejemplo, existe una página para explorar particularmente las competencias de los titulados para ambas ediciones de los estudios del Observatorio (Figura 52). La configuración de dicha página sigue la misma lógica que la explicada en el ejemplo anterior, la cual puede consultarse a rasgos generales en la Figura 50.

```
<Page page_id="2">
    <DataFilter component_id="Filter"...>
    <Components>
        <Component>
            <Heatmap component_id="HeatMap_1"...>
        </Component>
        <Component>
            <Heatmap component_id="HeatMap_2"...>
        </Component>
    </Components>
    <Layout>
        <RowGroup>
            <Row>
                <DataFilter ref="Filter"/>
            </Row>
            <Row>
                <Component ref="HeatMap_1"/>
            </Row>
            <Row>
                <Component ref="HeatMap_2"/>
            </Row>
        </RowGroup>
    </Layout>
</Page>
```

Figura 50. Configuración de una segunda página para un dashboard específico.

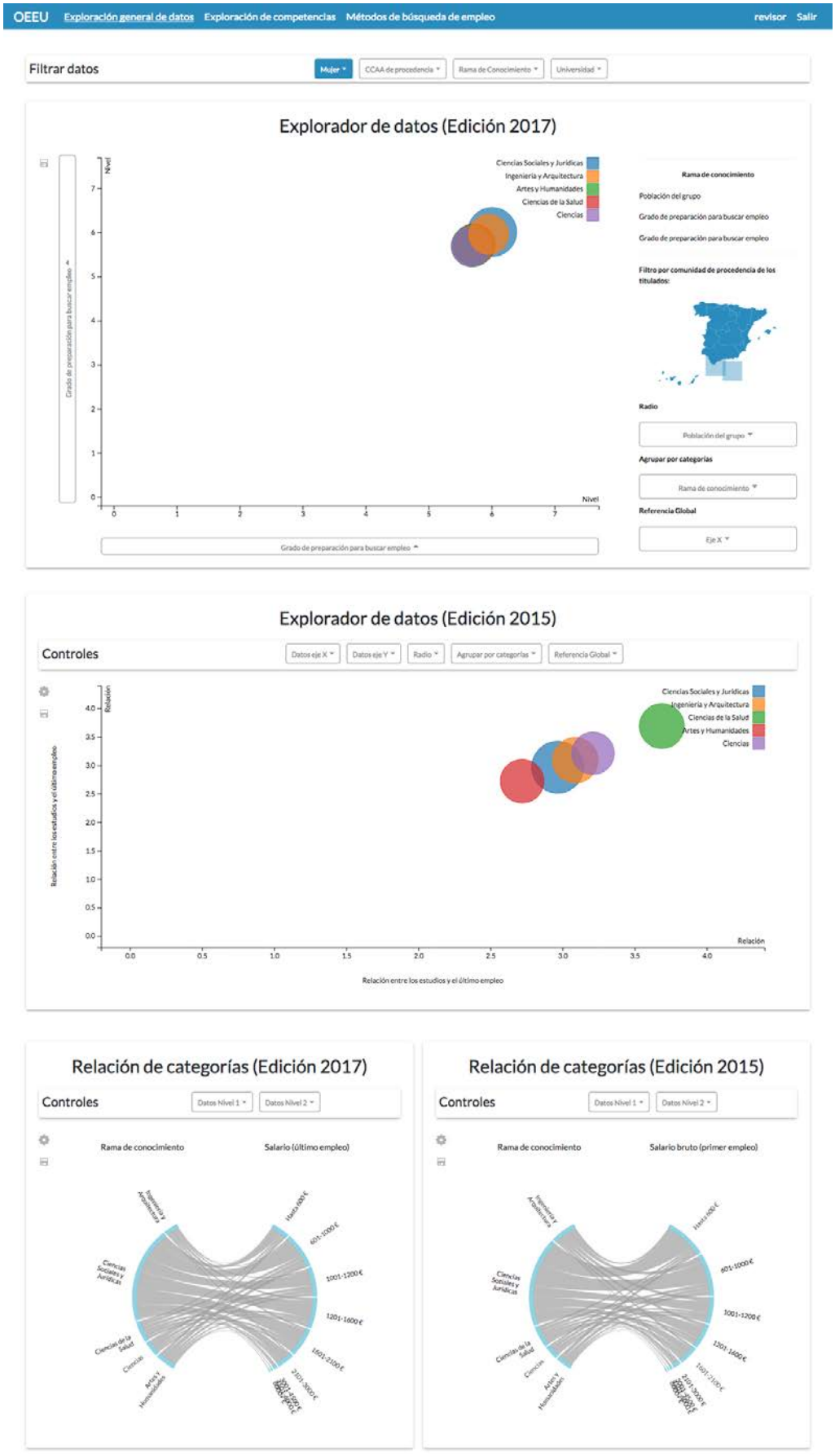

Figura 51. Página de un dashboard generada con diversos componentes configurados de forma individual.

Figura 52. Segunda página configurada del dashboard generado para explorar competencias.

Por último, se muestra un ejemplo de cómo también es posible variar el tamaño de las visualizaciones o componentes de la página a través de la configuración y los atributos "*width*" y "*height*" del esquema XML. Esto nos permite personalizar aún más la estructura de la página. Por ejemplo, la Figura 53 muestra un ejemplo donde cada diagrama de dispersión de la página tiene su altura expresada en proporción de la pantalla donde se muestra el *dashboard*. En este caso, el primer diagrama de dispersión ocupará el 80% del espacio disponible de la pantalla, mientras que el segundo ocupará el 65%.

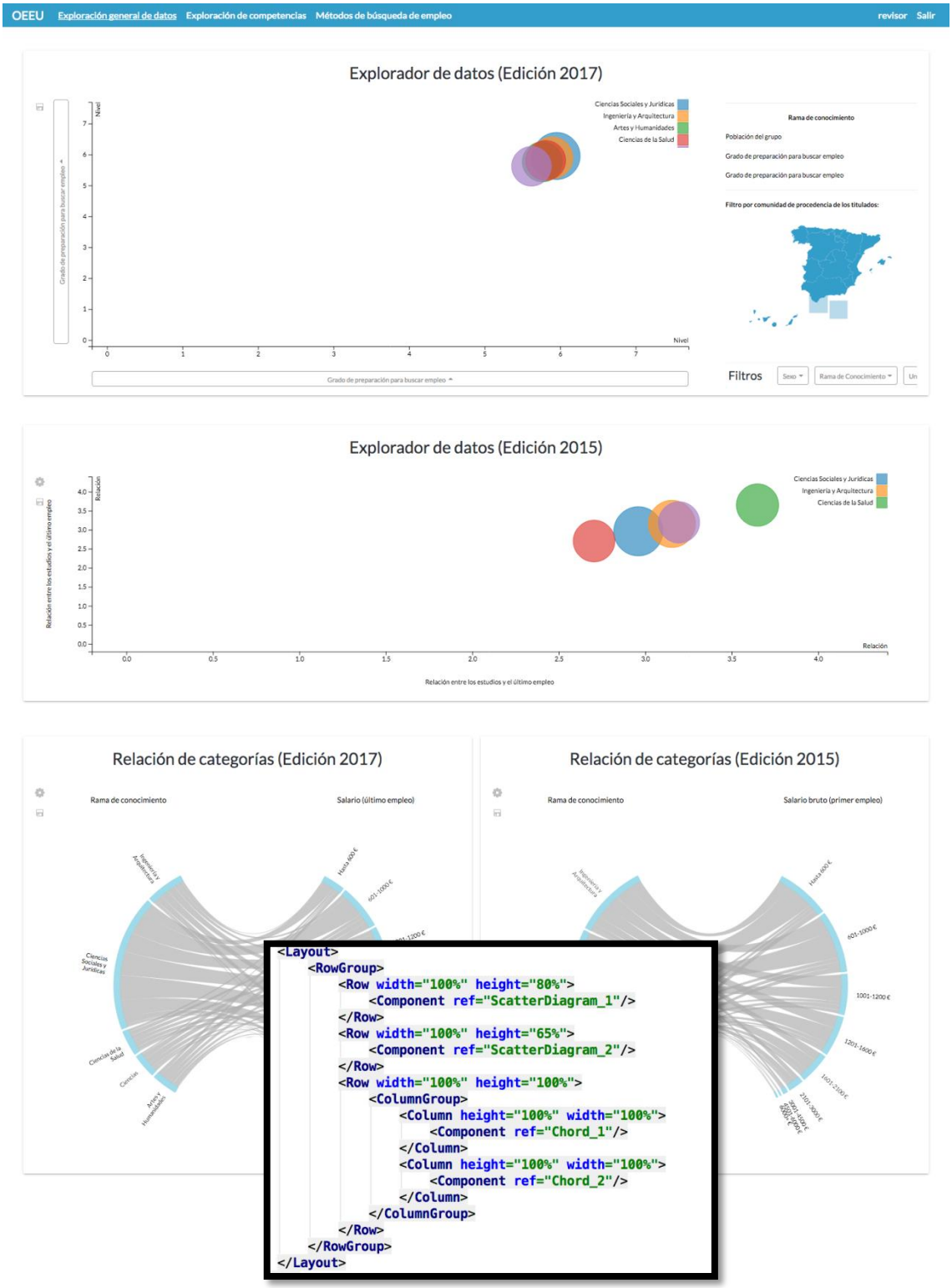

Figura 53. Ejemplo de configuración del layout de una página con ciertos valores de anchura y altura para cada componente.

Puede contrastarse este resultado con la configuración de una página donde cada visualización ocupa todo el espacio disponible (el 100%) de la pantalla (Figura 54).

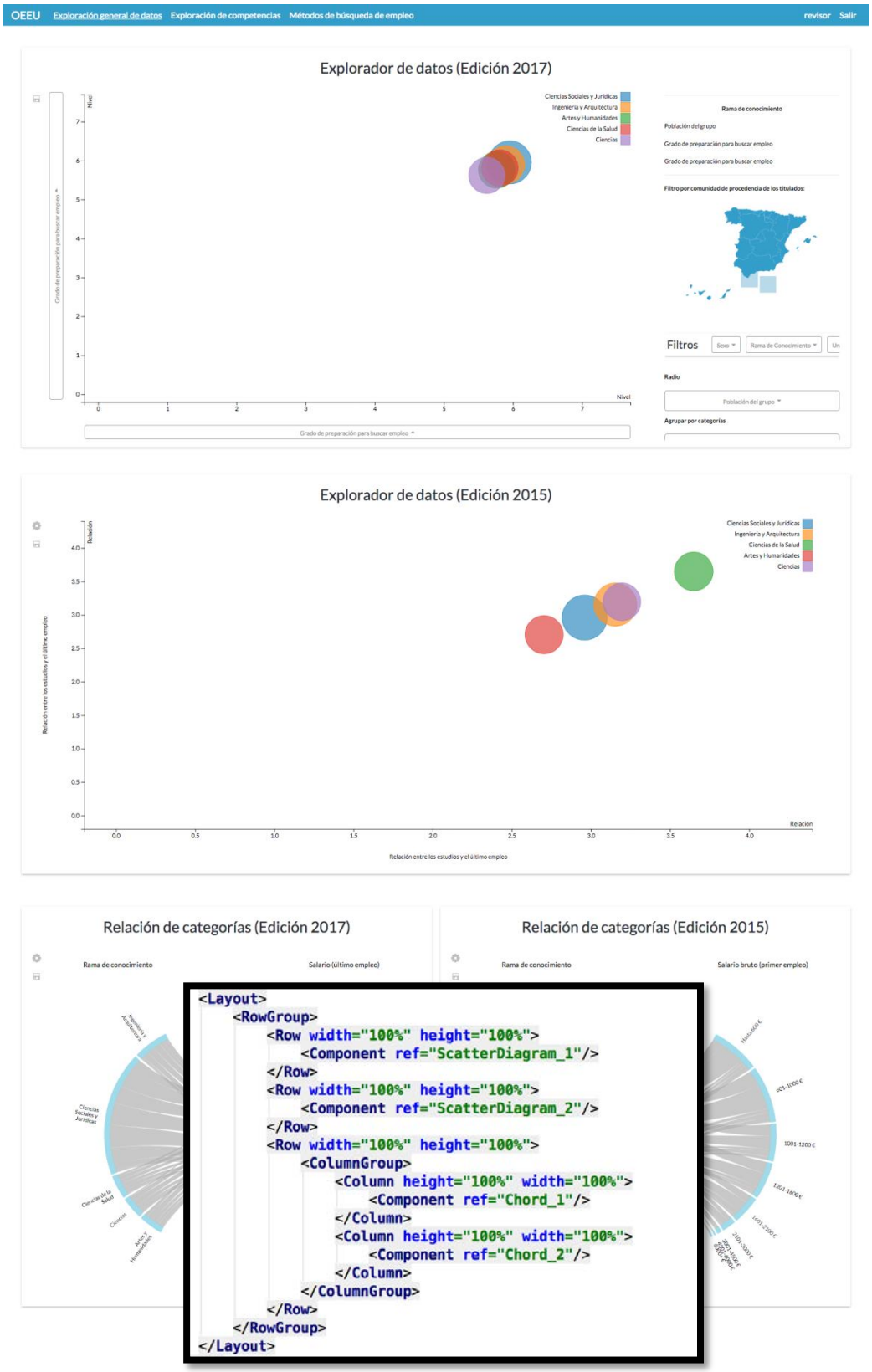

Figura 54. Segundo ejemplo de configuración del layout de una página con ciertos valores de anchura y altura para cada componente

Puede elegirse, como se ha comprobado, una gran cantidad de combinaciones en cuanto a configuraciones para generar diversos *dashboards* que permitan adecuarse a las necesidades de usuarios concretos.

Estas combinaciones pueden centrarse en diversos aspectos del *dashboard*, ya sea:

- La estructura de la página.
- Las funcionalidades de cada componente.
- Las fuentes de datos de cada componente.

De esta forma se obtiene un marco generativo para la personalización de estas herramientas de cara a la obtención y generación de conocimiento sobre la empleabilidad y empleo de estudiantes de diversas universidades españolas.

7. Discusión

A través de la aplicación del paradigma de líneas de producto *software* y el estudio de un dominio específico, como lo es el del Observatorio de Empleabilidad y Empleo Universitarios, se ha conseguido una primera aproximación para la consecución de un *framework* que permita generar *dashboards* personalizables a distintos niveles: a nivel de funcionalidad, a nivel de datos y a nivel de diseño o interfaz.

Las líneas de productos *software* ofrecen un marco de trabajo donde la reutilización de activos *software* se erige como elemento central. A través del estudio y abstracción de dominios concretos, es posible extraer características comunes y puntos de variabilidad de los productos que formarán parte de la familia.

En el caso de los *dashboards*, nos encontramos una serie de retos que han intentado abordarse a lo largo de este proyecto. El primero de ellos, el cual es esencial y nunca debe pasarse por alto, es la búsqueda de necesidades. Los *dashboards* no son una simple colección de recursos gráficos que muestran datos arbitrarios. Los datos se recolectan por una razón y, por tanto, deben visualizarse con sentido. Hay que tener siempre en mente que los *dashboard* buscan dar soporte a la toma de decisiones, de forma que permitan explorar los datos de forma sencilla y flexible para alcanzar conclusiones e identificar patrones o elementos clave. Para ello, hay que tener una noción de los objetivos que se pretenden conseguir con el soporte de los datos.

Esta fue una de las principales dificultades en las fases de conceptualización del trabajo, dado que no existía un objetivo claro más allá de la exploración de datos y resultados recolectados por el Observatorio. Por ello se optó por un enfoque genérico, con visualizaciones que se adaptasen a diversos tipos de variables que recolecta el Observatorio:

- Diagrama de dispersión para una exploración amplia y flexible de datos, permitiendo investigar relaciones entre variables a través de diversas dimensiones.
- Mapa de calor para identificar a simple vista de valores llamativos dentro de un conjunto de datos.
- Diagrama de cuerdas que permita relacionar variables categóricas.

Sin embargo, precisamente por la posibilidad de que apareciesen estas dificultades, la aplicación de la ingeniería de dominio reporta una serie de beneficios; si a posteriori se necesitasen nuevos elementos para explotar nuevas perspectivas de los datos, la adición de nuevos componentes solo supone un esfuerzo inicial en la fase de creación del activo o *core asset*, pudiendo ser reutilizado e incluso modificado tantas veces como fuese necesario de manera eficiente y aislada, sin repercutir en el resto de componentes.

En cuanto a los materiales utilizados, la especificación e implementación de un lenguaje específico de dominio (DSL) ha supuesto la creación de un nexo entre la especificación formal mediante diagramas de características y meta-modelos, y la implementación. Este elemento es clave en cuanto a la generación de código se refiere.

En muchas ocasiones, los modelos formales y diagramas solo tienen utilidad en la fase de conceptualización y diseño del *software*, y no vuelven a ser consultados más allá que por razones de documentación. Estos modelos muchas veces ni siquiera son mantenidos a lo largo del tiempo en conjunción con el código del *software*, rompiendo la trazabilidad de requisitos.

Es por ello que la creación del DSL ha supuesto un aspecto muy beneficioso de cara a la trazabilidad de requisitos y características de la línea de productos, además de obtener un medio más sencillo y abstracto para representar *dashboards*. El DSL se torna como una herramienta para que los desarrolladores y expertos del dominio puedan especificar, de forma estructurada, una serie de necesidades que se materializarán finalmente en la fase de generación de código.

Sin embargo, un aspecto aún por resolver, es la conexión del DSL con los modelos conceptuales. Si bien el lenguaje de dominio refleja los aspectos previamente modelados en fases anteriores, una modificación en los modelos requeriría una actualización del DSL. Precisamente es de vital importancia tener esto en cuenta, debido a que los modelos van a evolucionar a lo largo del tiempo, y más teniendo en cuenta que el dominio en el que se centran (representación de datos y soporte a la toma de decisiones) está en continua evolución, con lo que los modelos conceptuales deben adaptarse a cambios y nuevos requisitos, así como el DSL. Encontrar una forma de conectar los modelos con el DSL, de forma que los cambios se reflejasen automáticamente en este último podría ser posible a través de paradigmas dirigidos por modelos y reglas de transformación.

Otra de las dificultades a la hora de materializar este trabajo se encontró en la propia generación de código. Existen muchas metodologías para generar código a partir de una serie de requisitos, pero en este contexto era importante encontrar un método sencillo, que no consumiese muchos recursos y que permitiese inyectar características a partir de la extracción de requisitos de los ficheros de configuración.

La elección de plantillas de código y macros para materializar los puntos de variabilidad e implementar los *core assets* de la línea de productos se respaldó precisamente en el hecho de que es una aproximación simple, dando la posibilidad de inyectar porciones de código específicas de forma condicional o alternativa y soportando las posibles restricciones especificadas en los modelos conceptuales. Además, esta metodología permite dividir y clasificar las porciones de código según su fin o funcionalidad, obteniendo una serie de *core assets* bien definidos y encapsulados.

Finalmente, se presentan los logros respecto a la personalización de los *dashboards*, los cuales, como se ha mencionado anteriormente, se dividen en tres categorías o niveles: funcional, de datos y de diseño.

7.1. Personalización a nivel funcional

Este nivel se ha constituido como el más sencillo de abordar por una simple de razón: una vez realizada la fase de ingeniería de dominio, añadir funcionalidades en un componente solamente

implica la inyección del código asociado a dicha funcionalidad. Crear componentes con una lógica base (común para todos) a los que se les puede acoplar funcionalidades provee un grano muy fino de personalización en este sentido.

Como se ha visto, es posible tener componentes con diversas características funcionales: filtros, selectores de datos, información detallada, etc. Añadir nuevas características, de nuevo, vuelve a ser un proceso que no requiere grandes esfuerzos a nivel de implementación, además de que pasaría a ser parte de los activos de la línea de productos, pudiendo reutilizarlos en otros contextos.

El paradigma de las líneas de productos *software* ha demostrado ser muy beneficioso y efectivo respecto a la personalización de la funcionalidad de los componentes que formarán parte del producto final.

7.2. Personalización a nivel de datos

Como ya se ha mencionado a lo largo de esta memoria, el dominio de la representación de datos consta de un elemento básico y clave para el correcto funcionamiento de los *dashboards*: los datos. Los datos a representar pueden ser estáticos, provenir de fuentes externas o internas, ser recuperados a través de una API o directamente de una base de datos, etc.

La gran variedad de posibilidades en cuanto a la recuperación de los datos hace de este nivel un reto dentro del contexto de las líneas de productos *software*. Los datos son la base de los *dashboard*, y al igual que es necesario aportar funcionalidades útiles a los componentes, es necesario que los componentes también muestren datos útiles. Los requisitos de información de cada usuario pueden variar, y más en procesos de toma de decisiones, donde ciertos usuarios pueden considerar una serie de datos muy relevantes, mientras que otros pueden considerarlos irrelevantes.

Mediante la construcción de una API GraphQL se ha conseguido dotar al sistema del Observatorio de un método para que los clientes puedan recuperar información de su banco de conocimiento. Las diversas fuentes de datos o métricas pueden especificarse en los ficheros de configuración para representar solo aquellos datos que cierto usuario considera relevantes. Este enfoque permite, además, soportar de forma sencilla la evolución de los requisitos informativos de cada usuario, puesto que dichos requisitos se especifican también en la configuración del producto a generar.

Sin embargo, aún hay una serie de retos a tener en cuenta. La fuente de datos utilizada es interna al Observatorio y se ha construido para facilitar la recuperación de datos por parte de los componentes del *dashboard*. Si en algún momento se quisiera utilizar una fuente externa de datos seguramente no se contaría con tantas facilidades, al no estar la fuente bajo el control de los desarrolladores y necesitar mucho más mantenimiento [71].

Aun así, mediante el modelado de las fuentes de datos podría llegar a ser posible realizar combinaciones de diversas fuentes, obteniendo una potente herramienta de representación de datos sin importar su formato o procedencia. Sería muy interesante seguir investigando para encontrar formas de homogeneizar las peticiones de datos.

7.3. Personalización a nivel de diseño

Sin duda el diseño se convierte en el nivel más complejo de modelar para conseguir su personalización. Ha sido posible abordar ciertos aspectos de diseño, como la estructura de las páginas del *dashboard* y la situación de los diversos componentes. Sin embargo, aún podría alcanzarse un grano más fino de personalización en cuanto a diseño de la interfaz de usuario.

El principal problema de este nivel de personalización es que se trata de un nivel muy complicado de automatizar. Se han propuesto en la literatura diversas aproximaciones para abordar la generación de interfaces gráficas de usuario a través de líneas de producto *software* [42-44], pero en todas ellas se apunta a la necesidad de introducir un proceso manual o semiautomático de personalización, teniendo en cuenta los requisitos de diseño de cada usuario específico. Este proceso manual rompe el ciclo automático que puede conseguirse con la aplicación de este paradigma.

Una vez generado el código para un usuario, podrían modificarse manualmente los archivos generados para adaptarlos a una cierta imagen, pero se perdería la trazabilidad de dichos requisitos, y al regenerar el código en caso de necesitar actualizar las necesidades funcionales o de información, se perderían esos cambios, quitándole sentido al proceso de generación automática.

Sin duda este nivel de personalización es el principal reto en la generación de *dashboards*, puesto que éstos no solo deben proveer al usuario funcionalidades útiles, sino que también han de ser usables para que la experiencia de usuario no se vea comprometida.

8. Conclusiones y líneas de trabajo futuras

La aplicación de ingeniería de dominio para la generación de *dashboards* ha demostrado ser una aproximación interesante para gestionar requisitos variados e incluso excluyentes entre usuarios. Se ha conseguido un marco de trabajo donde generar *dashboards* compuestos por diversos componentes configurados específicamente para suplir una serie de necesidades. Nuevos requisitos pueden ser especificados de forma sencilla e introducidos en el código de forma eficiente a través de generadores automáticos.

La toma de decisiones es un proceso complejo donde es necesario poder contar con recursos informativos para respaldar decisiones de manera fundamentada, de tal forma que puedan reportar beneficios. Es por ello que proveer a los encargados de la toma de decisiones de herramientas potentes y flexibles, que se adapten a las necesidades en cada momento y contexto y que puedan evolucionar de forma rápida se convierte en una tarea clave.

Mediante la creación de líneas de productos *software* de *dashboard* es posible abordar todas estas dificultades y ofrecer productos flexibles cuya modificación no requiere esfuerzos significativos, ofreciendo eficientemente *dashboards* hechos a medida de los usuarios.

Esta aplicación abre muchas puertas y opciones para explorar más profundamente.

Por una parte, como se ha mencionado en la discusión, aunque la generación de código es satisfactoria, el proceso de configuración sigue requiriendo esfuerzos significativos tanto a nivel de recolección de necesidades, como a nivel de construcción de los ficheros de configuración. En un futuro sería muy beneficioso contar con una herramienta que simplifique e incluso automatice este proceso mediante una interfaz gráfica y sugerencias que hagan el proceso de configuración más sencillo.

La introducción de sugerencias también es un aspecto potencialmente muy beneficioso. Como se ha desarrollado a lo largo de este trabajo, el proceso de diseño y construcción de un *dashboard* no es trivial. La elección de los métodos de visualización e interacción [72] utilizados son clave para conseguir una buena experiencia y, consecuentemente, un buen soporte a la toma de decisiones, la cual debe ser siempre el objetivo final de los *dashboards*. Al contar con las configuraciones y características en formatos estructurados, sería posible aplicar técnicas de *machine learning* para

adaptar los *dashboards* de forma dinámica en función de las características de los usuarios [73] para posibilitar la integración de la inteligencia artificial y la natural [74].

Esto permitiría conocer mejor qué configuraciones de *dashboard* son más beneficiosas en términos de alcance de objetivos o qué factores influyen en la efectividad de los *dashboard* según el contexto en el que se utiliza.

También puede aprovecharse este *framework* generativo para realizar test A/B [75-77] de forma sencilla, puesto que las nuevas funcionalidades que quisiesen evaluarse podrían implementarse como un *core asset* de la línea de productos e introducirlo en las diversas versiones del test . Esto permitiría conocer el grado de efectividad y utilidad de nuevas características.

Sin duda, la aplicación de ingeniería de dominio ha reportado una serie de beneficios muy valiosos de cara a la explotación de datos, específicamente en el contexto del Observatorio de Empleabilidad y Empleo Universitarios, ya no solo a nivel de efectividad, personalización, mantenibilidad y evolución de los requisitos, sino también por todos los anteriores beneficios potenciales que podrían llegar a ofrecerse en un futuro.

Apéndice A – Búsqueda de necesidades: Entrevistas

Universidad 1.

1. **¿Cuál es la motivación de la Universidad para la participación en el barómetro del Observatorio?**

Conocer el nivel de empleabilidad y competencias de los estudiantes de la universidad y conocer otros datos como las medias nacionales.

2. **¿Se toman decisiones estratégicas en base a los resultados obtenidos a través de los cuestionarios del Observatorio? ¿En qué ámbito?**

Sí, los datos obtenidos a través del Observatorio influyen en la toma de decisiones, pero no de forma primordial, tienen sus propios sistemas de encuestas e indicadores y los datos del Observatorio los complementan. La toma de decisiones se basaría en el conjunto total de datos una vez analizados y serían los servicios de calidad los encargados de tomarlas.

3. **¿Cuál es el procedimiento (tareas) que se realiza con los datos del Observatorio (uso de Excel, SPSS, otras técnicas)?**

Se descargan los micro-datos (ofrecidos a través de la web del Observatorio por CSV) y se procesan a través de SPSS. Se apunta que llega a ser un procedimiento tedioso por el procesamiento a través de SPSS (normalización de variables, etc.).

4. **¿Qué variables derivadas calcula la Universidad a partir de los datos del Observatorio?**

Medias de satisfacción, competencias y empleabilidad de sus egresados (las que ofreció posteriormente el Observatorio en su *dashboard*)

5. **¿Se realiza un informe o un sumario con los datos del Observatorio?**

Utilizan los datos generales y resultados específicos de la UNED como complemento al resto de sus indicadores propios.

6. **¿Se realiza algún tipo de visualización con los datos del Observatorio?**

Los analistas solo eran encargados de calcular las métricas que se les indican.

7. **¿Hay alguna tarea que podría facilitarse (cálculo de variables derivadas de interés para la Universidad, métricas específicas, visualizaciones específicas)?**

El cálculo de las propias métricas que ellos mismos calculaban a partir de SPSS ya les supondría un gran ahorro.

8. **Otros intereses.**

Mostraban interés en la comparación con las medias nacionales (no se ofrece en los *dashboards* actuales). También indican la posibilidad de descargar las propias visualizaciones y resultados obtenidos del análisis.

Universidad 2.

1. **¿Cuál es la motivación de la Universidad para la participación en el barómetro del Observatorio?**
   Ellos no son consumidores como tal de los datos del Observatorio, ni los utilizan como elemento principal en su toma de decisiones, pero sí que les son de valor a la hora de contrastar sus propios resultados con el barómetro (medias nacionales).

2. **¿Se toman decisiones estratégicas en base a los resultados obtenidos a través de los cuestionarios del Observatorio? ¿En qué ámbito?**
   Sí, pero de nuevo, a través de sus propios estudios, que se envían a las unidades de calidad de la propia universidad.

3. **¿Cuál es el procedimiento (tareas) que se realiza con los datos del Observatorio (uso de Excel, SPSS, otras técnicas)?**
   Cuando necesitan algún dato específico que no encuentran entre sus métricas descargan el CSV del Observatorio, pero les interesa más el poder comparar su situación con la media nacional (sobre todo en Ingeniería y Arquitectura)

4. **¿Qué variables derivadas calcula la Universidad a partir de los datos del Observatorio?**
   Complementan sus propias variables derivadas dependiendo del tipo de estudio: satisfacción, perfiles de ingreso, si su trabajo se ajusta a lo estudiado, contratos permanentes o temporales, vocación.

5. **¿Se realiza un informe o un sumario con los datos del Observatorio?**
   Complementario. El apartado de metodologías y competencias les es el más interesante.

6. **¿Se realiza algún tipo de visualización con los datos del Observatorio?**
   Sí que han utilizado el *dashboard* genérico ofrecido por el Observatorio, pero no lo han utilizado mucho debido a que la mayoría de datos específicos de la UPM ya los recolectan por sus propios medios y sus propios estudios.

7. **¿Hay alguna tarea que podría facilitarse (cálculo de variables derivadas de interés para la Universidad, métricas específicas, visualizaciones específicas)?**
   Les interesaría poder comparar su ingeniería y arquitectura (la rama que tiene peso en su universidad) con el resto de España y, si es posible, con las ramas de ingeniería y arquitectura de otras Comunidades. También les interesa saber la situación de sus egresados en ingeniería y arquitectura frente a otras ramas en su comunidad.

8. **Otros intereses.**
   No expresaron ningún otro interés particular.

Universidad 3.

1. **¿Cuál es la motivación de la Universidad para la participación en el barómetro del Observatorio?**
   Contrastar, complementar su propia información.

2. **¿Se toman decisiones estratégicas en base a los resultados obtenidos a través de los cuestionarios del Observatorio? ¿En qué ámbito?**
   Sí, pero con sus propios estudios, los datos del Observatorio solo les servirían para comparar su situación, la cual ya conocen gracias a sus propios estudios.

3. **¿Cuál es el procedimiento (tareas) que se realiza con los datos del Observatorio (uso de Excel, SPSS, otras técnicas)?**
   Ellos descargan los datos del Observatorio y los procesan mediante SPSS para echarles un vistazo general, pero no para la toma de decisiones. Comparan también sus datos con estudios del Ministerio.

4. **¿Qué variables derivadas calcula la Universidad a partir de los datos del Observatorio?**
   Las propias que ofrece el Observatorio, solo que las calculan a través de SPSS, y como no son un componente principal (ya que tienen sus propios estudios) no se realizan métricas complejas.

5. **¿Se realiza un informe o un sumario con los datos del Observatorio?**
   Son complementarios y sirven a nivel de comparación nacional, los datos propios de la universidad se descargan, pero ya tienen sus propias fuentes y métricas.

6. **¿Se realiza algún tipo de visualización con los datos del Observatorio?**
   No, toman provecho de las visualizaciones propias del barómetro para ver la situación general.

7. **¿Hay alguna tarea que podría facilitarse (cálculo de variables derivadas de interés para la Universidad, métricas específicas, visualizaciones específicas)?**
   Comunidades que más posgrado estudian, que tienen más inserción laboral, etc. (de nuevo, no se sabe si es posible).

8. **Otros intereses.**
   No mostraron ningún otro interés particular.

Universidad 4.

1. **¿Cuál es la motivación de la Universidad para la participación en el barómetro del Observatorio?**
   Utilizar los datos para líneas de investigación o de manera divulgativa para colocarlos en sus webs.

2. **¿Se toman decisiones estratégicas en base a los resultados obtenidos a través de los cuestionarios del Observatorio? ¿En qué ámbito?**
   No, la universidad tiene sus propios cuestionarios de calidad y la toma de decisiones se realiza en base a ellos.

3. **¿Cuál es el procedimiento (tareas) que se realiza con los datos del Observatorio (uso de Excel, SPSS, otras técnicas)?**
   No les dieron uso a los datos de la pasada edición por falta de recursos. Descargaron los micro-datos y los procesaron de forma básica con SPSS.

4. **¿Qué variables derivadas calcula la Universidad a partir de los datos del Observatorio?**
   Ninguna (no se utilizaron).

5. **¿Se realiza un informe o un sumario con los datos del Observatorio?**
   Les habría gustado utilizar los datos de forma divulgativa.

6. **¿Se realiza algún tipo de visualización con los datos del Observatorio?**
   No, pero las visualizaciones ofrecidas por el Observatorio les habrían sido útiles para sus propios intereses.

7. **¿Hay alguna tarea que podría facilitarse (cálculo de variables derivadas de interés para la Universidad, métricas específicas, visualizaciones específicas)?**
   Al no haber utilizado los datos, no llegaron a diseñar nunca métricas específicas. Sí que les habría gustado la posibilidad de descargar las visualizaciones ofrecidas por el Observatorio para poder utilizarlas ellos mismos.

8. **Otros intereses.**
   No mostraron ningún otro interés particular.

Apéndice B – Publicaciones relacionadas

En este apéndice se enumeran diversas publicaciones realizadas relacionadas con el proyecto desarrollado.

- A. Vázquez-Ingelmo, J. Cruz-Benito, and F. J. García-Peñalvo, "Improving the OEEU's data-driven technological ecosystem's interoperability with GraphQL," in *Fifth International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'17) (Cádiz, Spain, October 18-20, 2017)* J. M. Dodero, M. S. Ibarra Sáiz, and I. Ruiz Rube, Eds. (ACM International Conference Proceedings Series (ICPS), New York, NY, USA: ACM, 2017. [78]

- A. Vázquez-Ingelmo, J. Cruz-Benito, F. J. García-Peñalvo, and M. Martín-González, "Scaffolding the OEEU's Data-Driven Ecosystem to Analyze the Employability of Spanish Graduates," in *Global Implications of Emerging Technology Trends*, F. J. García-Peñalvo, Ed. Hershey PA, USA: IGI Global, 2018, pp. 236-255. [28]

- J. Cruz-Benito, A. Vázquez-Ingelmo, J. C. Sánchez-Prieto, R. Therón, F. J. García-Peñalvo, and M. Martín-González, "Enabling adaptability in web forms based on user characteristics detection through A/B testing and machine learning," *IEEE Access,* vol. 6, pp. 2251-2265, 2018. [73]

- J. Cruz-Benito, J. C. Sánchez-Prieto, A. Vázquez-Ingelmo, R. Therón, F. J. García-Peñalvo, and M. Martín-González, "How different versions of layout and complexity of web forms affect users after they start it? A pilot experience," in *Trends and Advances in Information Systems and Technologies*, vol. 2, Á. Rocha, H. Adeli, L. P. Reis, and S. Costanzo, Eds. (Advances in Intelligent Systems and Computing, no. AISC 746) Cham: Springer, 2018, pp. 971-979. [79]

- F. J. García-Peñalvo, J. Cruz-Benito, M. Martín-González, A. Vázquez-Ingelmo, J. C. Sánchez-Prieto, and R. Therón, "Proposing a machine learning approach to analyze and predict employment and its factors," *International Journal of Interactive Multimedia and Artificial Intelligence,* vol. In Press, 2018. [80]

- A. Vázquez-Ingelmo, F. J. García-Peñalvo, and R. Therón, "Generation of customized dashboards through software product line paradigms to analyse university employment and employability data," presented at the Learning Analytics Summer Institute Spain 2018 (LASI 2018), León, Castilla y León, Spain, 2018. [81]

- A. Vázquez-Ingelmo, F. J. García-Peñalvo, and R. Therón, "Application of domain engineering to generate customized information dashboards," presented at the HCI International 2018. 20th Conference on Human-Computer Interaction (15-20 July 2018), Las Vegas, Nevada, USA, 2018. [82]

Referencias

[1] S. C. Albright, W. Winston, and C. Zappe, *Data analysis and decision making*. Cengage Learning, 2010.

[2] D. Patil and H. Mason, *Data Driven*. " O'Reilly Media, Inc.", 2015.

[3] D. Patil, *Data Jujitsu*. " O'Reilly Media, Inc.", 2012.

[4] R. Sharda, D. Delen, and E. Turban, *Business intelligence: a managerial perspective on analytics*. Prentice Hall Press, 2013.

[5] M. Zeleny, "Management support systems: Towards integrated knowledge management," *Human systems management,* vol. 7, no. 1, pp. 59-70, 1987.

[6] M. J. Eppler, "Knowledge communication problems between experts and decision makers: An overview and classification," *Electronic Journal of Knowledge Management,* vol. 5, no. 3, 2007.

[7] S. Few, "Information dashboard design," 2006.

[8] D. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann, *Mastering the information age solving problems with visual analytics*. Eurographics Association, 2010.

[9] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon, "Visual analytics: Definition, process, and challenges," in *Information visualization*: Springer, 2008, pp. 154-175.

[10] E. Tufte and P. Graves-Morris, "The visual display of quantitative information.; 1983," ed, 2014.

[11] A. González Torres, F. J. García-Peñalvo, and R. Therón-Sánchez, "How evolutionary visual software analytics supports knowledge discovery," 2013.

[12] F. J. García-Peñalvo, "Issue on visual analytics," *Journal of Information Technology Research,* vol. 8, no. 2, pp. iv-vi, 2015.

[13] J. J. Thomas and K. A. Cook, *Illuminating the path: The research and development agenda for visual analytics*. USA: National Visualization and Analytics Center, 2005.

[14] R. Therón *et al.*, "Rapid reconstruction of paleoenvironmental features using a new multiplatform program," *Micropaleontology,* vol. 50, no. 4, pp. 391-395, 2004.

[15] R. Theron, "Visual analytics of paleoceanographic conditions," in *IEEE VAST*, 2006, pp. 19-26.

[16] R. Santamaría and R. Therón, "Treevolution: visual analysis of phylogenetic trees," *Bioinformatics,* vol. 25, no. 15, pp. 1970-1971, 2009.

[17] R. Santamaría, R. Therón, and L. Quintales, "BicOverlapper 2.0: visual analysis for gene expression," *Bioinformatics,* vol. 30, no. 12, pp. 1785-1786, 2014.

[18] D. A. Gómez-Aguilar, F. J. García-Peñalvo, and R. Therón, "Analítica Visual en eLearning," *El Profesional de la Información,* vol. 23, no. 3, pp. 236-245, 2014.

[19] R. Theron and L. Fontanillo, "Diachronic-information visualization in historical dictionaries," *Information Visualization,* vol. 14, no. 2, pp. 111-136, 2015.

[20] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[21] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley Longman Publishing Co., Inc., 2004.

[22] K. Pohl, G. n. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., 2005.

[23] A. G. Kleppe, J. Warmer, and W. Bast, "MDA Explained. The Model Driven Architecture: Practice and Promise," ed: Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2003.

[24] C. Gacek and M. Anastasopoules, "Implementing product line variabilities," in *ACM SIGSOFT Software Engineering Notes*, 2001, vol. 26, no. 3, pp. 109-117: ACM.

[25] P. Clements and L. Northrop, *Software product lines*. Addison-Wesley, 2002.

[26] F. Michavila, J. M. Martínez, M. Martín-González, F. J. García-Peñalvo, and J. Cruz-Benito, *Barómetro de empleabilidad y empleo de los universitarios en España, 2015 (Primer informe de resultados)*. Madrid: Observatorio de Empleabilidad y Empleo Universitarios, 2016.

[27] F. Michavila, J. M. Martínez, M. Martín-González, F. J. García-Peñalvo, J. Cruz-Benito, and A. Vázquez-Ingelmo, *Barómetro de empleabilidad y empleo universitarios. Edición Máster 2017*. Madrid, España: Observatorio de Empleabilidad y Empleo Universitarios, 2018.

[28] A. Vázquez-Ingelmo, J. Cruz-Benito, F. J. García-Peñalvo, and M. Martín-González, "Scaffolding the OEEU's Data-Driven Ecosystem to Analyze the Employability of Spanish Graduates," in *Global Implications of Emerging Technology Trends*, F. J. García-Peñalvo, Ed. Hershey PA, USA: IGI Global, 2018, pp. 236-255.

[29] J. Van Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*, 2001, pp. 45-54: IEEE.

[30] A. Metzger and K. Pohl, "Variability management in software product line engineering," in *Companion to the proceedings of the 29th International Conference on Software Engineering*, 2007, pp. 186-187: IEEE Computer Society.

[31] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst1990.

[32] C. Seidl, I. Schaefer, and U. Aßmann, "Capturing variability in space and time with hyper feature models," in *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*, 2014, p. 6: ACM.

[33] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow, "Extending feature diagrams with UML multiplicities," in *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, 2002, vol. 23.

[34] N. Anquetil *et al.*, "Traceability for model driven, software product line engineering," in *ECMDA Traceability Workshop Proceedings*, 2008, vol. 12, pp. 77-86: SINTEF.

[35] S. B. Tajali, J.-P. Corriveau, and W. Shi, "A Template-Based Approach to Modeling Variability," in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, 2013, p. 1: The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[36] I. Magdalenić, D. Radošević, and D. Kermek, "Implementation Model of Source Code Generator," 2011.

[37] T. Greifenberg, K. Müller, A. Roth, B. Rumpe, C. Schulze, and A. Wortmann, "Modeling variability in template-based code generators for product line engineering," *arXiv preprint arXiv:1606.02903,* 2016.

[38] G. Freeman, D. Batory, and G. Lavender, "Lifting transformational models of product lines: A case study," in *International Conference on Theory and Practice of Model Transformations*, 2008, pp. 16-30: Springer.

[39] A. Gómez, M. C. Penadés, J. H. Canós, M. R. Borges, and M. Llavador, "A framework for variable content document generation with multiple actors," *Information and Software Technology,* vol. 56, no. 9, pp. 1101-1121, 2014.

[40] A. EZZAT LABIB AWAD, "Enforcing Customization in e-Learning Systems: an ontology and product line-based approach," 2017.

[41] Y. Gabillon, N. Biri, and B. Otjacques, "Designing an adaptive user interface according to software product line engineering," *Proc. ACHI,* vol. 15, pp. 86-91, 2015.

[42] A. Pleuss, B. Hauptmann, M. Keunecke, and G. Botterweck, "A case study on variability in user interfaces," in *Proceedings of the 16th International Software Product Line Conference-Volume 1*, 2012, pp. 6-10: ACM.

[43] A. Pleuss, S. Wollny, and G. Botterweck, "Model-driven development and evolution of customized user interfaces," in *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, 2013, pp. 13-22: ACM.

[44] D. Kramer, S. Oussena, P. Komisarczuk, and T. Clark, "Graphical user interfaces in dynamic software product lines," in *Product Line Approaches in Software Engineering (PLEASE), 2013 4th International Workshop on*, 2013, pp. 25-28: IEEE.

[45] A. Pleuss, B. Hauptmann, D. Dhungana, and G. Botterweck, "User interface engineering for software product lines: the dilemma between automation and usability," in *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, 2012, pp. 25-34: ACM.

[46] J. García, F. J. García-Peñalvo, R. Therón, and P. O. de Pablos, "Usability evaluation of a visual modelling tool for owl ontologies," *Journal of Universal Computer Science,* vol. 17, no. 9, pp. 1299-1313, 2011.

[47] J. M. Álvarez, A. Evans, and P. Sammut, "Mapping between levels in the metamodel architecture," in *International Conference on the Unified Modeling Language*, 2001, pp. 34-46: Springer.

[48] D. C. Fallside, "XML schema part 0: Primer," *W3C, April 2000,* 2000.

[49] S. U.-J. Lee, "An effective methodology with automated product configuration for software product line development," *Mathematical Problems in Engineering,* vol. 2015, 2015.

[50] A. Holovaty and J. Kaplan-Moss, *The definitive guide to Django: Web development done right*. Apress, 2009.

[51] M. Bostock, V. Ogievetsky, and J. Heer, "D³ data-driven documents," *IEEE transactions on visualization and computer graphics,* vol. 17, no. 12, pp. 2301-2309, 2011.

[52] M. Tatsubori and T. Suzumura, "HTML templates that fly: a template engine approach to automated offloading from server to client," in *Proceedings of the 18th international conference on World wide web*, 2009, pp. 951-960: ACM.

[53] R. J. Rodger, "Jostraca: a template engine for generative programming," in *European Conference for Object-Oriented Programming*, 2002: Citeseer.

[54] E. Kirda and C. Kerer, "MyXML: An XML based template engine for the generation of flexible Web content," in *WebNet World Conference on the WWW and Internet*, 2000, pp. 317-322: Association for the Advancement of Computing in Education (AACE).

[55] A. Ronacher, "Jinja2 Documentation," ed: Welcome to Jinja2—Jinja2 Documentation (2.8-dev), 2008.

[56] Django Software Foundation. (2015, 15/03/2015). *Django Web Framework*. Available: https://www.djangoproject.com/

[57] S. Clark. (2018). *Render your first network configuration template using Python and Jinja2*. Available: https://blogs.cisco.com/developer/network-configuration-template

[58] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology (TOIT),* vol. 2, no. 2, pp. 115-150, 2002.

[59] Facebook. (2016, 24th Jul. 2017). *GraphQL*. Available: https://facebook.github.io/graphql/

[60] D. Pandya, "GraphQL Concepts Visualized," ed, 2016, p. Web log post.

[61] M. Faassen, "GraphQL and REST, Secret Weblog," ed, 2015.

[62] S. Buna. (2017, July 25). *Rest APIs are REST-in-Peace APIs. Long Live GraphQL*. Available: https://medium.freecodecamp.org/rest-apis-are-rest-in-peace-apis-long-live-graphql-d412e559d8e4

[63] W. McKinney, *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*." O'Reilly Media, Inc.", 2012.

[64] W. McKinney, "pandas: a foundational Python library for data analysis and statistics," *Python for High Performance and Scientific Computing,* pp. 1-9, 2011.

[65] W. McKinney. (2017). *Pandas, Python Data Analysis Library. 2017*. Available: http://pandas.pydata.org/

[66] E. Meeks, *D3. js in Action: Data Visualization with JavaScript*. Manning, 2018.

[67] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90-95, 2007.
[68] M. Bostock. (2017). *Towards Reusable Charts*. Available: https://bost.ocks.org/mike/chart/
[69] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *The Craft of Information Visualization*: Elsevier, 2003, pp. 364-371.
[70] M. Weise. (2016, 18/05/2018). *We Need a Better Way to Visualize People's Skills*. Available: https://hbr.org/2016/09/we-need-a-better-way-to-visualize-peoples-skills
[71] S. Wang, I. Keivanloo, and Y. Zou, "How do developers react to restful api evolution?," in *International Conference on Service-Oriented Computing*, 2014, pp. 245-259: Springer.
[72] J. Cruz-Benito, R. Therón, and F. J. García-Peñalvo, "Software architectures supporting human-computer interaction analysis: A literature review," in *International Conference on Learning and Collaboration Technologies*, 2016, pp. 125-136: Springer.
[73] J. Cruz-Benito, A. Vázquez-Ingelmo, J. C. Sánchez-Prieto, R. Therón, F. J. García-Peñalvo, and M. Martín-González, "Enabling adaptability in web forms based on user characteristics detection through A/B testing and machine learning," *IEEE Access*, vol. 6, pp. 2251-2265, 2018.
[74] J. C. Alvarado-Pérez, D. H. Peluffo-Ordóñez, and R. Therón-Sánchez, "Bridging the gap between human knowledge and machine learning," 2015.
[75] D. Siroker and P. Koomen, *A/B testing: The most powerful way to turn clicks into customers*. John Wiley & Sons, 2013.
[76] E. Dixon, E. Enos, and S. Brodmerkle, "A/b testing of a webpage," ed: Google Patents, 2011.
[77] A. C. Kakas, "A/B Testing," 2017.
[78] A. Vázquez-Ingelmo, J. Cruz-Benito, and F. J. García-Peñalvo, "Improving the OEEU's data-driven technological ecosystem's interoperability with GraphQL," in *Fifth International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'17) (Cádiz, Spain, October 18-20, 2017)* J. M. Dodero, M. S. Ibarra Sáiz, and I. Ruiz Rube, Eds. (ACM International Conference Proceedings Series (ICPS), New York, NY, USA: ACM, 2017.
[79] J. Cruz-Benito, J. C. Sánchez-Prieto, A. Vázquez-Ingelmo, R. Therón, F. J. García-Peñalvo, and M. Martín-González, "How different versions of layout and complexity of web forms affect users after they start it? A pilot experience," in *Trends and Advances in Information Systems and Technologies*, vol. 2, Á. Rocha, H. Adeli, L. P. Reis, and S. Costanzo, Eds. (Advances in Intelligent Systems and Computing, no. AISC 746) Cham: Springer, 2018, pp. 971-979.
[80] F. J. García-Peñalvo, J. Cruz-Benito, M. Martín-González, A. Vázquez-Ingelmo, J. C. Sánchez-Prieto, and R. Therón, "Proposing a machine learning approach to analyze and predict employment and its factors," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. In Press, 2018.
[81] A. Vázquez-Ingelmo, F. J. García-Peñalvo, and R. Therón, "Generation of customized dashboards through software product line paradigms to analyse university employment and employability data," presented at the Learning Analytics Summer Institute Spain 2018 (LASI 2018), León, Castilla y León, Spain, 2018.
[82] A. Vázquez-Ingelmo, F. J. García-Peñalvo, and R. Therón, "Application of domain engineering to generate customized information dashboards," presented at the HCI International 2018. 20th Conference on Human-Computer Interaction (15-20 July 2018), Las Vegas, Nevada, USA, 2018.